Land Information System

LIS 7 Reference Manual

October 16, 2014

Revision 1.1

History:

Revision	Summary of Changes	Date
1.1	LIS 7.0 Public Release patch 1	October 16, 2014

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

History:

Revision	Summary of Changes	Date
1.0	LIS 7.0 Public Release	July 18, 2014

Contents

I LIS Overview **47**

1 Overview of LIS **48**

II LIS offline program **49**

2 LIS offline program **50**
2.0.1 lisdrv (Source File: lisdrv.F90) 50

III LIS core structures and methods **51**

3 LIS core structures and methods **52**
3.1 Fortran: Module Interface LIS_DAobservationsMod (Source File: LIS_DAobservationsMod.F90) 52
3.1.1 LIS_initDAobservations (Source File: LIS_DAobservationsMod.F90) 52
3.1.2 LIS_readDAobservations (Source File: LIS_DAobservationsMod.F90) 53
3.1.3 LIS_perturb_DAobservations (Source File: LIS_DAobservationsMod.F90) 54
3.2 Fortran: Module Interface LIS_LMLCMod (Source File: LIS_LMLCMod.F90) 55
3.2.1 Overview 56
3.2.2 LIS_LMLC_init (Source File: LIS_LMLCMod.F90) 56
3.2.3 LIS_diagnoselandmask (Source File: LIS_LMLCMod.F90) 56
3.2.4 LIS_diagnoseLandcover (Source File: LIS_LMLCMod.F90) 57
3.2.5 read_landcover (Source File: LIS_LMLCMod.F90) 58
3.2.6 read_surfacetype (Source File: LIS_LMLCMod.F90) 58
3.3 Fortran: Module Interface LIS_PE_HandlerMod (Source File: LIS_PE_HandlerMod.F90) . . . 59
3.3.1 LIS_PE_init (Source File: LIS_PE_HandlerMod.F90) 60
3.3.2 setupPEOBSPredSpace (Source File: LIS_PE_HandlerMod.F90) 60
3.3.3 setupDecSpaceVars (Source File: LIS_PE_HandlerMod.F90) 61
3.3.4 LIS_readPEobs (Source File: LIS_PE_HandlerMod.F90) 61
3.3.5 LIS_updatePEObjectiveFunc (Source File: LIS_PE_HandlerMod.F90) 61
3.3.6 getPEOBSPred (Source File: LIS_PE_HandlerMod.F90) 62
3.3.7 LIS_computePEObjectiveFunc (Source File: LIS_PE_HandlerMod.F90) 62
3.3.8 LIS_setPEDecisionSpace (Source File: LIS_PE_HandlerMod.F90) 63
3.3.9 setDecisionSpace (Source File: LIS_PE_HandlerMod.F90) 63
3.3.10 LIS_resetPEobjectiveFunc (Source File: LIS_PE_HandlerMod.F90) 64
3.3.11 LIS_resetPEobs (Source File: LIS_PE_HandlerMod.F90) 64
3.3.12 LIS_PE_restart (Source File: LIS_PE_HandlerMod.F90) 64
3.4 Fortran: Module Interface LIS_PRIV_gridMod (Source File: LIS_PRIV_gridMod.F90) 65
3.4.1 Overview 65
3.5 Fortran: Module Interface LIS_PRIV_rcMod (Source File: LIS_PRIV_rcMod.F90) 65
3.5.1 LIS_PRIV_tileMod (Source File: LIS_PRIV_tileMod.F90) 75
3.5.2 Overview 75
3.6 Fortran: Module Interface LIS_RTMMod (Source File: LIS_RTMMod.F90) 75
3.6.1 LIS_RTM_init (Source File: LIS_RTMMod.F90) 75
3.6.2 RTM_run (Source File: LIS_RTMMod.F90) 76
3.6.3 LIS_RTM_output (Source File: LIS_RTMMod.F90) 76
3.6.4 LIS_RTM_final (Source File: LIS_RTMMod.F90) 76
3.7 Fortran: Module Interface LIS_albedoMod (Source File: LIS_albedoMod.F90) 77

3.7.1 Overview . . . 77
3.7.2 LIS_albedo_setup (Source File: LIS_albedoMod.F90) . . . 77
3.7.3 LIS_read_mxsnalb (Source File: LIS_albedoMod.F90) . . . 78
3.7.4 LIS_read_albedo (Source File: LIS_albedoMod.F90) . . . 78
3.7.5 LIS_albedo_finalize (Source File: LIS_albedoMod.F90) . . . 79
3.7.6 LIS_diagnosealbedo (Source File: LIS_albedoMod.F90) . . . 79
3.7.7 LIS_albedo_reset (Source File: LIS_albedoMod.F90) . . . 80
3.7.8 read_albedoclimo (Source File: LIS_albedoMod.F90) . . . 80
3.8 Fortran: Module Interface LIS_appMod (Source File: LIS_appMod.F90) . . . 81
3.8.1 LIS_appModel_init (Source File: LIS_appMod.F90) . . . 81
3.8.2 LIS_runAppModel (Source File: LIS_appMod.F90) . . . 82
3.8.3 LIS_outputAppModel(n) (Source File: LIS_appMod.F90) . . . 82
3.8.4 LIS_appModel_finalize() (Source File: LIS_appMod.F90) . . . 82
3.9 Fortran: Module Interface LIS_constantsMod (Source File: LIS_constantsMod.F90) . . . 83
3.10 Fortran: Module Interface LIS_coreMod (Source File: LIS_coreMod.F90) . . . 84
3.10.1 Overview . . . 84
3.10.2 LIS_config_init (Source File: LIS_coreMod.F90) . . . 86
3.10.3 lisconfig_offline (Source File: LIS_coreMod.F90) . . . 86
3.10.4 lisconfig_coupled1 (Source File: LIS_coreMod.F90) . . . 87
3.10.5 lisconfig_coupled2 (Source File: LIS_coreMod.F90) . . . 88
3.10.6 LIS_core_init (Source File: LIS_coreMod.F90) . . . 88
3.10.7 LIS_ticktime (Source File: LIS_coreMod.F90) . . . 88
3.10.8 LIS_endofrun (Source File: LIS_coreMod.F90) . . . 89
3.10.9 LIS_endofTimeWindow (Source File: LIS_coreMod.F90) . . . 89
3.10.10 LIS_timeToRunNest (Source File: LIS_coreMod.F90) . . . 89
3.10.11 LIS_resetTimeMgr (Source File: LIS_coreMod.F90) . . . 90
3.10.12 LIS_finalize (Source File: LIS_coreMod.F90) . . . 90
3.10.13 spmd_init_offline (Source File: LIS_coreMod.F90) . . . 90
3.10.14 spmd_init_coupled (Source File: LIS_coreMod.F90) . . . 91
3.10.15 spmd_setup (Source File: LIS_coreMod.F90) . . . 91
3.10.16 spmd_finalize (Source File: LIS_coreMod.F90) . . . 91
3.11 Fortran: Module Interface LIS_dataAssimMod (Source File: LIS_dataAssimMod.F90) . . . 91
3.11.1 Overview . . . 92
3.11.2 LIS_dataassim_init (Source File: LIS_dataAssimMod.F90) . . . 92
3.11.3 LIS_dataassim_run (Source File: LIS_dataAssimMod.F90) . . . 93
3.11.4 LIS_dataassim_output (Source File: LIS_dataAssimMod.F90) . . . 93
3.11.5 LIS_dataassim_finalize (Source File: LIS_dataAssimMod.F90) . . . 94
3.11.6 LIS_rescale_with_CDF_matching (Source File: LIS_dataAssimMod.F90) . . . 94
3.11.7 LIS_readCDFdata (Source File: LIS_dataAssimMod.F90) . . . 95
3.11.8 LIS_readMeanSigmaData (Source File: LIS_dataAssimMod.F90) . . . 96
3.12 Fortran: Module Interface LIS_domainMod (Source File: LIS_domainMod.F90) . . . 97
3.12.1 Overview . . . 97
3.12.2 LIS_domain_init (Source File: LIS_domainMod.F90) . . . 97
3.12.3 LIS_domain_setup (Source File: LIS_domainMod.F90) . . . 98
3.12.4 LIS_quilt_domain (Source File: LIS_domainMod.F90) . . . 99
3.12.5 LIS_setParameterDomainSpecs (Source File: LIS_domainMod.F90) . . . 99
3.12.6 LIS_domain_finalize (Source File: LIS_domainMod.F90) . . . 99
3.12.7 make_domain (Source File: LIS_domainMod.F90) . . . 100
3.12.8 calculate_domdistribution (Source File: LIS_domainMod.F90) . . . 100
3.12.9 create_tilespace (Source File: LIS_domainMod.F90) . . . 101
3.12.10 isSurfaceTypeSelected (Source File: LIS_domainMod.F90) . . . 101

3.12.11 ifContainsSelectedSurfaceType (Source File: LIS_domainMod.F90) . . . 102
3.12.12 compute_ntiles_surface (Source File: LIS_domainMod.F90) . . . 102
3.12.13 compute_npatches_surface (Source File: LIS_domainMod.F90) . . . 102
3.12.14 compute_ntiles_soilt (Source File: LIS_domainMod.F90) . . . 103
3.12.15 compute_ntiles_soilf (Source File: LIS_domainMod.F90) . . . 103
3.12.16 compute_ntiles_elev (Source File: LIS_domainMod.F90) . . . 103
3.12.17 compute_ntiles_slope (Source File: LIS_domainMod.F90) . . . 104
3.12.18 compute_ntiles_aspect (Source File: LIS_domainMod.F90) . . . 104
3.12.19 get_vegt_value (Source File: LIS_domainMod.F90) . . . 104
3.12.20 get_surface_value (Source File: LIS_domainMod.F90) . . . 105
3.12.21 get_soilt_value (Source File: LIS_domainMod.F90) . . . 105
3.12.22 get_soilf_value (Source File: LIS_domainMod.F90) . . . 105
3.12.23 get_elev_value (Source File: LIS_domainMod.F90) . . . 106
3.12.24 get_slope_value (Source File: LIS_domainMod.F90) . . . 106
3.12.25 get_aspect_value (Source File: LIS_domainMod.F90) . . . 106
3.13 Fortran: Module Interface LIS_emissMod (Source File: LIS_emissMod.F90) . . . 107
3.13.1 Overview . . . 107
3.13.2 LIS_emiss_setup (Source File: LIS_emissMod.F90) . . . 107
3.13.3 LIS_read_emiss (Source File: LIS_emissMod.F90) . . . 108
3.13.4 LIS_emiss_finalize (Source File: LIS_emissMod.F90) . . . 108
3.13.5 LIS_diagnoseemiss (Source File: LIS_emissMod.F90) . . . 109
3.13.6 LIS_emiss_reset (Source File: LIS_emissMod.F90) . . . 109
3.13.7 read_emissclimo (Source File: LIS_emissMod.F90) . . . 109
3.14 Fortran: Module Interface LIS_fileIOMod (Source File: LIS_fileIOMod.F90) . . . 110
3.14.1 LIS_readData (Source File: LIS_fileIOMod.F90) . . . 111
3.14.2 LIS_create_restart_filename (Source File: LIS_fileIOMod.F90) . . . 111
3.14.3 LIS_putget (Source File: LIS_fileIOMod.F90) . . . 111
3.14.4 LIS_read_param (Source File: LIS_fileIOMod.F90) . . . 112
3.14.5 LIS_convertParamDataToLocalDomain (Source File: LIS_fileIOMod.F90) . . . 112
3.14.6 LIS_create_output_directory (Source File: LIS_fileIOMod.F90) . . . 112
3.14.7 LIS_create_output_filename (Source File: LIS_fileIOMod.F90) . . . 113
3.14.8 LIS_create_dapert_filename (Source File: LIS_fileIOMod.F90) . . . 114
3.14.9 create_restart_filename (Source File: LIS_fileIOMod.F90) . . . 115
3.14.10 create_restart_filename_withtime (Source File: LIS_fileIOMod.F90) . . . 116
3.14.11 LIS_create_stats_filename (Source File: LIS_fileIOMod.F90) . . . 117
3.14.12 LIS_create_innov_filename (Source File: LIS_fileIOMod.F90) . . . 118
3.14.13 LIS_create_daspread_filename (Source File: LIS_fileIOMod.F90) . . . 119
3.14.14 LIS_create_obs_filename (Source File: LIS_fileIOMod.F90) . . . 120
3.14.15 putget_int (Source File: LIS_fileIOMod.F90) . . . 121
3.14.16 putget_real (Source File: LIS_fileIOMod.F90) . . . 121
3.14.17 LIS_create_gain_filename (Source File: LIS_fileIOMod.F90) . . . 122
3.14.18 read2Ddata (Source File: LIS_fileIOMod.F90) . . . 123
3.14.19 LIS_readDomainConfigSpecs (Source File: LIS_fileIOMod.F90) . . . 124
3.14.20 LIS_checkDomainExtents (Source File: LIS_fileIOMod.F90) . . . 124
3.14.21 readparam_real_2d (Source File: LIS_fileIOMod.F90) . . . 125
3.14.22 readparam_int_2d (Source File: LIS_fileIOMod.F90) . . . 125
3.14.23 convertParam_real (Source File: LIS_fileIOMod.F90) . . . 126
3.14.24 convertParam_int (Source File: LIS_fileIOMod.F90) . . . 126
3.15 Fortran: Module Interface LIS_gswpMod (Source File: LIS_gswpMod.F90) . . . 127
3.15.1 getgswp_monindex (Source File: LIS_gswpMod.F90) . . . 127
3.15.2 getgswp_timeindex (Source File: LIS_gswpMod.F90) . . . 127

3.16 Fortran: Module Interface LIS_histDataMod (Source File: LIS_histDataMod.F90) 128
3.16.1 LIS_histDataInit (Source File: LIS_histDataMod.F90) 129
3.16.2 get_moc_attributes (Source File: LIS_histDataMod.F90) 129
3.16.3 allocate_dataEntry (Source File: LIS_histDataMod.F90) 130
3.16.4 convertToCFunits (Source File: LIS_histDataMod.F90) 130
3.16.5 LIS_diagnoseSurfaceOutputVar (Source File: LIS_histDataMod.F90) 130
3.16.6 LIS_diagnoseRoutingOutputVar (Source File: LIS_histDataMod.F90) 131
3.16.7 LIS_diagnoseRTMOutputVar (Source File: LIS_histDataMod.F90) 131
3.16.8 LIS_diagnoseIrrigationOutputVar (Source File: LIS_histDataMod.F90) 132
3.16.9 LIS_diagnoseOutputVar (Source File: LIS_histDataMod.F90) 132
3.16.10 diagnoseDataEntry (Source File: LIS_histDataMod.F90) 133
3.16.11 LIS_resetOutputVars (Source File: LIS_histDataMod.F90) 134
3.16.12 resetOutputVar (Source File: LIS_histDataMod.F90) 134
3.16.13 LIS_rescaleCount (Source File: LIS_histDataMod.F90) 135
3.16.14 register_dataEntry (Source File: LIS_histDataMod.F90) 135
3.16.15 set_ptr_into_list (Source File: LIS_histDataMod.F90) 136
3.17 Fortran: Module Interface LIS_historyMod (Source File: LIS_historyMod.F90) 137
3.17.1 Overview 137
3.17.2 LIS_writevar_bin (Source File: LIS_historyMod.F90) 139
3.17.3 LIS_writevar_restart (Source File: LIS_historyMod.F90) 139
3.17.4 LIS_writevar_reduced_tilespace (Source File: LIS_historyMod.F90) 140
3.17.5 LIS_readvar_reduced_tilespace (Source File: LIS_historyMod.F90) 140
3.17.6 LIS_readvar_restart (Source File: LIS_historyMod.F90) 140
3.17.7 LIS_readvar_gridded (Source File: LIS_historyMod.F90) 141
3.17.8 LIS_writevar_gridded (Source File: LIS_historyMod.F90) 141
3.17.9 LIS_writevar_grib1 (Source File: LIS_historyMod.F90) 141
3.17.10 LIS_grid2patch (Source File: LIS_historyMod.F90) 142
3.17.11 LIS_writevar_netcdf (Source File: LIS_historyMod.F90) 142
3.17.12 LIS_gather_gridded_output (Source File: LIS_historyMod.F90) 142
3.17.13 LIS_tile2grid (Source File: LIS_historyMod.F90) 143
3.17.14 LIS_writeModelOutput (Source File: LIS_historyMod.F90) 143
3.17.15 writeBinaryOutput (Source File: LIS_historyMod.F90) 144
3.17.16 writeSingleBinaryVar (Source File: LIS_historyMod.F90) 145
3.17.17 writeGrib1Output (Source File: LIS_historyMod.F90) 145
3.17.18 writeSingleGrib1Var (Source File: LIS_historyMod.F90) 146
3.17.19 writeNetcdfOutput (Source File: LIS_historyMod.F90) 147
3.17.20 defineNETCDFheaderVar (Source File: LIS_historyMod.F90) 148
3.17.21 writeSingleNETCDFvar (Source File: LIS_historyMod.F90) 148
3.17.22 LIS_writeGlobalHeader_restart (Source File: LIS_historyMod.F90) 149
3.17.23 LIS_writeHeader_restart (Source File: LIS_historyMod.F90) 150
3.17.24 LIS_closeHeader_restart (Source File: LIS_historyMod.F90) 151
3.17.25 writevar_bin_withstats_real (Source File: LIS_historyMod.F90) 151
3.17.26 writevar_bin_real (Source File: LIS_historyMod.F90) 152
3.17.27 writevar_bin_real_direct (Source File: LIS_historyMod.F90) 153
3.17.28 writevar_restart_tile_int (Source File: LIS_historyMod.F90) 153
3.17.29 writevar_restart_tile_real (Source File: LIS_historyMod.F90) 154
3.17.30 writevar_restart_patch_int (Source File: LIS_historyMod.F90) 154
3.17.31 writevar_restart_patch_real (Source File: LIS_historyMod.F90) 155
3.17.32 writevar_tile_noensemble_real (Source File: LIS_historyMod.F90) 155
3.17.33 readvar_tile_noensemble_real (Source File: LIS_historyMod.F90) 156
3.17.34 readvar_restart_tile_int (Source File: LIS_historyMod.F90) 156

3.17.35 readvar_restart_patch_int (Source File: LIS_historyMod.F90) . . . 157
3.17.36 readvar_restart_patch_real (Source File: LIS_historyMod.F90) . . . 157
3.17.37 readvar_restart_tile_real (Source File: LIS_historyMod.F90) . . . 158
3.17.38 readvar_1dgridded_real (Source File: LIS_historyMod.F90) . . . 158
3.17.39 readvar_1dgridded_fromvector_real (Source File: LIS_historyMod.F90) . . . 159
3.17.40 readvar_2dgridded_real (Source File: LIS_historyMod.F90) . . . 160
3.17.41 writevar_gridded_real (Source File: LIS_historyMod.F90) . . . 160
3.17.42 writevar_gridded_real_withstats (Source File: LIS_historyMod.F90) . . . 161
3.17.43 writevar_netcdf_withstats_real (Source File: LIS_historyMod.F90) . . . 161
3.17.44 writevar_grib1_withstats_real (Source File: LIS_historyMod.F90) . . . 162
3.17.45 tile2grid_local (Source File: LIS_historyMod.F90) . . . 163
3.17.46 LIS_patch2tile (Source File: LIS_historyMod.F90) . . . 164
3.17.47 LIS_write_innov (Source File: LIS_historyMod.F90) . . . 164
3.17.48 LIS_write_dastats (Source File: LIS_historyMod.F90) . . . 165
3.17.49 makepdsn (Source File: LIS_historyMod.F90) . . . 166
3.17.50 stats (Source File: LIS_historyMod.F90) . . . 166
3.17.51 stats_da (Source File: LIS_historyMod.F90) . . . 167
3.17.52 write_stats (Source File: LIS_historyMod.F90) . . . 167
3.17.53 LIS_gather_tiled_vector_output (Source File: LIS_historyMod.F90) . . . 168
3.17.54 LIS_gather_tiled_vector_withhalo_output (Source File: LIS_historyMod.F90) . . . 168
3.17.55 LIS_gather_patch_vector_output (Source File: LIS_historyMod.F90) . . . 169
3.17.56 gather_gridded_output_tile (Source File: LIS_historyMod.F90) . . . 170
3.17.57 gather_gridded_output_patch (Source File: LIS_historyMod.F90) . . . 170
3.17.58 LIS_gather_gridded_vector_output (Source File: LIS_historyMod.F90) . . . 171
3.17.59 LIS_convertVarToLocalSpace (Source File: LIS_historyMod.F90) . . . 172
3.17.60 LIS_initialize_registries (Source File: LIS_initialize_registries.F90) . . . 172
3.18 Fortran: Module Interface LIS_irrigationMod (Source File: LIS_irrigationMod.F90) . . . 173
3.18.1 LIS_irrigation_init (Source File: LIS_irrigationMod.F90) . . . 174
3.18.2 LIS_irrigation_run (Source File: LIS_irrigationMod.F90) . . . 174
3.18.3 LIS_irrigation_output (Source File: LIS_irrigationMod.F90) . . . 174
3.19 Fortran: Module Interface LIS_lakemodelMod (Source File: LIS_lakemodelMod.F90) . . . 175
3.19.1 LIS_lakemodel_init (Source File: LIS_lakemodelMod.F90) . . . 176
3.19.2 LIS_setuplakemodel (Source File: LIS_lakemodelMod.F90) . . . 176
3.19.3 LIS_lakemodel_run (Source File: LIS_lakemodelMod.F90) . . . 176
3.19.4 LIS_lakemodel_readrestart (Source File: LIS_lakemodelMod.F90) . . . 177
3.19.5 LIS_lakemodel_output (Source File: LIS_lakemodelMod.F90) . . . 177
3.19.6 LIS_lakemodel_setdynparams (Source File: LIS_lakemodelMod.F90) . . . 178
3.19.7 LIS_lakemodel_f2t (Source File: LIS_lakemodelMod.F90) . . . 178
3.19.8 LIS_lakemodel_writerestart (Source File: LIS_lakemodelMod.F90) . . . 179
3.19.9 LIS_lakemodel_finalize (Source File: LIS_lakemodelMod.F90) . . . 179
3.20 Fortran: Module Interface LIS_landslideMod (Source File: LIS_landslideMod.F90) . . . 179
3.20.1 LIS_initLandSlideModel (Source File: LIS_landslideMod.F90) . . . 180
3.20.2 LIS_runLandSlideModel (Source File: LIS_landslideMod.F90) . . . 180
3.20.3 LIS_outputLandSlideModel (Source File: LIS_landslideMod.F90) . . . 181
3.20.4 LIS_finalizeLandSlideModel (Source File: LIS_landslideMod.F90) . . . 181
3.21 Fortran: Module Interface LIS_logMod (Source File: LIS_logMod.F90) . . . 181
3.21.1 LIS_verify (Source File: LIS_logMod.F90) . . . 182
3.21.2 LIS_log_init (Source File: LIS_logMod.F90) . . . 182
3.21.3 LIS_log_msg (Source File: LIS_logMod.F90) . . . 182
3.21.4 LIS_abort (Source File: LIS_logMod.F90) . . . 183
3.21.5 LIS_alert (Source File: LIS_logMod.F90) . . . 183

3.21.6 LIS_flush (Source File: LIS_logMod.F90) . . . 184
3.21.7 check_error (Source File: LIS_logMod.F90) . . . 184
3.21.8 verify (Source File: LIS_logMod.F90) . . . 184
3.21.9 LIS_warning (Source File: LIS_logMod.F90) . . . 185
3.21.10 LIS_getNextUnitNumber (Source File: LIS_logMod.F90) . . . 185
3.21.11 LIS_releaseUnitNumber (Source File: LIS_logMod.F90) . . . 185
3.21.12 LIS_endrun (Source File: LIS_logMod.F90) . . . 185
3.22 Fortran: Module Interface LIS_lsmMod (Source File: LIS_lsmMod.F90) . . . 186
3.22.1 Overview . . . 186
3.22.2 LIS_lsm_setexport (Source File: LIS_lsmMod.F90) . . . 187
3.22.3 LIS_lsm_init (Source File: LIS_lsmMod.F90) . . . 187
3.22.4 LIS_setuplsm (Source File: LIS_lsmMod.F90) . . . 188
3.22.5 LIS_lsm_run (Source File: LIS_lsmMod.F90) . . . 188
3.22.6 LIS_lsm_perturb_states (Source File: LIS_lsmMod.F90) . . . 189
3.22.7 LIS_lsm_readrestart (Source File: LIS_lsmMod.F90) . . . 189
3.22.8 LIS_setLSMDynparams (Source File: LIS_lsmMod.F90) . . . 190
3.22.9 LIS_lsm_f2t (Source File: LIS_lsmMod.F90) . . . 190
3.22.10 LIS_lsm_writerestart (Source File: LIS_lsmMod.F90) . . . 191
3.22.11 lsm_setexport_noesmf (Source File: LIS_lsmMod.F90) . . . 191
3.22.12 LIS_lsm_finalize (Source File: LIS_lsmMod.F90) . . . 192
3.22.13 LIS_lsm_reset (Source File: LIS_lsmMod.F90) . . . 192
3.22.14 applyLSMPert (Source File: LIS_lsmMod.F90) . . . 192
3.22.15 LIS_mapSoilType (Source File: LIS_mapSoilType.F90) . . . 193
3.22.16 LIS_mapSurfaceModelType (Source File: LIS_mapSurfaceModelType.F90) . . . 193
3.23 Fortran: Module Interface LIS_metforcingMod (Source File: LIS_metforcingMod.F90) . . . 194
3.23.1 Overview . . . 194
3.23.2 LIS_metforcing_init (Source File: LIS_metforcingMod.F90) . . . 195
3.23.3 metforcing_init_offline (Source File: LIS_metforcingMod.F90) . . . 196
3.23.4 metforcing_init_coupled (Source File: LIS_metforcingMod.F90) . . . 196
3.23.5 create_forcing_structures (Source File: LIS_metforcingMod.F90) . . . 197
3.23.6 forcingPerturbSetup (Source File: LIS_metforcingMod.F90) . . . 197
3.23.7 LIS_get_met_forcing (Source File: LIS_metforcingMod.F90) . . . 197
3.23.8 overlayForcings (Source File: LIS_metforcingMod.F90) . . . 198
3.23.9 ensembleForcings (Source File: LIS_metforcingMod.F90) . . . 198
3.23.10 LIS_perturb_forcing (Source File: LIS_metforcingMod.F90) . . . 199
3.23.11 LIS_metforcing_reset (Source File: LIS_metforcingMod.F90) . . . 199
3.23.12 LIS_metforcing_finalize (Source File: LIS_metforcingMod.F90) . . . 199
3.23.13 get_forcingvar_attributes (Source File: LIS_metforcingMod.F90) . . . 200
3.23.14 add_forcing_fields (Source File: LIS_metforcingMod.F90) . . . 200
3.23.15 diagnoseForcingOutput (Source File: LIS_metforcingMod.F90) . . . 200
3.23.16 LIS_microMetCorrection (Source File: LIS_microMetCorrection.F90) . . . 201
3.24 Fortran: Module Interface LIS_mpiMod (Source File: LIS_mpiMod.F90) . . . 201
3.25 Fortran: Module Interface LIS_numerRecipesMod (Source File: LIS_numerRecipesMod.F90) . . . 202
3.25.1 LIS_gasdev (Source File: LIS_numerRecipesMod.F90) . . . 202
3.25.2 ran2 (Source File: LIS_numerRecipesMod.F90) . . . 202
3.26 Fortran: Module Interface LIS_openwatermodelMod (Source File: LIS_openwatermodelMod.F90) 203
3.26.1 LIS_openwatermodel_init (Source File: LIS_openwatermodelMod.F90) . . . 203
3.26.2 LIS_setupopenwatermodel (Source File: LIS_openwatermodelMod.F90) . . . 204
3.26.3 LIS_openwatermodel_run (Source File: LIS_openwatermodelMod.F90) . . . 204
3.26.4 LIS_openwatermodel_readrestart (Source File: LIS_openwatermodelMod.F90) . . . 204
3.26.5 LIS_openwatermodel_output (Source File: LIS_openwatermodelMod.F90) . . . 205

3.26.6 LIS_openwatermodel_setdynparams (Source File: LIS_openwatermodelMod.F90) . . . 205
3.26.7 LIS_openwatermodel_f2t (Source File: LIS_openwatermodelMod.F90) 206
3.26.8 LIS_openwatermodel_writerestart (Source File: LIS_openwatermodelMod.F90) 206
3.26.9 LIS_openwatermodel_finalize (Source File: LIS_openwatermodelMod.F90) 207
3.27 Fortran: Module Interface LIS_optUEMod (Source File: LIS_optUEMod.F90) 207
3.27.1 Overview 207
3.27.2 LIS_optUE_init (Source File: LIS_optUEMod.F90) 207
3.27.3 LIS_objectiveFunc_init (Source File: LIS_optUEMod.F90) 208
3.27.4 LIS_optUEAlg_init (Source File: LIS_optUEMod.F90) 208
3.27.5 LIS_isOptStopCriterionTrue (Source File: LIS_optUEMod.F90) 208
3.27.6 LIS_runoptUE (Source File: LIS_optUEMod.F90) 209
3.27.7 LIS_optUEAlg_readrestart (Source File: LIS_optUEMod.F90) 209
3.28 Fortran: Module Interface LIS_paramsMod (Source File: LIS_paramsMod.F90) 210
3.28.1 Overview 210
3.28.2 LIS_param_init (Source File: LIS_paramsMod.F90) 211
3.28.3 LIS_setDynparams (Source File: LIS_paramsMod.F90) 211
3.28.4 diagnoseOutputparams (Source File: LIS_paramsMod.F90) 212
3.28.5 LIS_param_finalize (Source File: LIS_paramsMod.F90) 213
3.28.6 LIS_param_reset (Source File: LIS_paramsMod.F90) 213
3.29 Fortran: Module Interface LIS_perturbMod (Source File: LIS_perturbMod.F90) 214
3.29.1 LIS_perturb_init (Source File: LIS_perturbMod.F90) 214
3.29.2 LIS_perturb_readrestart (Source File: LIS_perturbMod.F90) 214
3.29.3 LIS_perturb_writerestart (Source File: LIS_perturbMod.F90) 215
3.29.4 LIS_readPertAttributes (Source File: LIS_perturbMod.F90) 215
3.30 Fortran: Module Interface LIS_precisionMod (Source File: LIS_precisionMod.F90) 216
3.30.1 LIS_process_cmd_args (Source File: LIS_process_cmd_args.F90) 217
3.30.2 LIS_readConfig (Source File: LIS_readConfig.F90) 217
3.30.3 LIS_readDAObsAttributes (Source File: LIS_readDAObsAttributes.F90) 218
3.30.4 LIS_readPEDecSpaceAttributes (Source File: LIS_readPEDecSpaceAttributes.F90) . . 219
3.31 Fortran: Module Interface LIS_routingMod (Source File: LIS_routingMod.F90) 219
3.31.1 LIS_routing_init (Source File: LIS_routingMod.F90) 220
3.31.2 LIS_routing_readrestart (Source File: LIS_routingMod.F90) 220
3.31.3 LIS_routing_run (Source File: LIS_routingMod.F90) 221
3.31.4 LIS_routing_writeoutput (Source File: LIS_routingMod.F90) 221
3.31.5 LIS_routing_writerestart (Source File: LIS_routingMod.F90) 221
3.32 Fortran: Module Interface LIS_snowMod (Source File: LIS_snowMod.F90) 221
3.32.1 LIS_snow_setup (Source File: LIS_snowMod.F90) 222
3.32.2 LIS_snow_finalize (Source File: LIS_snowMod.F90) 222
3.33 Fortran: Module Interface LIS_soilsMod (Source File: LIS_soilsMod.F90) 222
3.33.1 Overview 222
3.33.2 LIS_soils_init (Source File: LIS_soilsMod.F90) 223
3.33.3 LIS_diagnosesoils (Source File: LIS_soilsMod.F90) 224
3.33.4 LIS_soils_finalize (Source File: LIS_soilsMod.F90) 225
3.33.5 read_porosity (Source File: LIS_soilsMod.F90) 225
3.33.6 read_soiltexture (Source File: LIS_soilsMod.F90) 226
3.33.7 read_soilfraction (Source File: LIS_soilsMod.F90) 226
3.34 Fortran: Module Interface LIS_spatialDownscalingMod (Source File: LIS_spatialDownscalingMod.F90) 227
3.34.1 LIS_lapseRateCorrection (Source File: LIS_spatialDownscalingMod.F90) 228
3.34.2 LIS_slopeAspectCorrection (Source File: LIS_spatialDownscalingMod.F90) 228
3.34.3 LIS_generatePcpClimoRatioField (Source File: LIS_spatialDownscalingMod.F90) . . . 229
3.34.4 LIS_surfaceModel_setexport (Source File: LIS_surfaceModelMod.F90) 229

3.34.5 LIS_surfaceModel_init (Source File: LIS_surfaceModelMod.F90) 230
3.34.6 LIS_surfaceModel_setup (Source File: LIS_surfaceModelMod.F90) 230
3.34.7 LIS_surfaceModel_readrestart (Source File: LIS_surfaceModelMod.F90) 230
3.34.8 LIS_surfaceModel_run (Source File: LIS_surfaceModelMod.F90) 230
3.34.9 LIS_surfaceModel_f2t (Source File: LIS_surfaceModelMod.F90) 231
3.34.10 LIS_surfaceModel_output (Source File: LIS_surfaceModelMod.F90) 231
3.34.11 LIS_surfaceModel_writerestart (Source File: LIS_surfaceModelMod.F90) 232
3.34.12 LIS_surfaceModel_perturb_states (Source File: LIS_surfaceModelMod.F90) 232
3.34.13 LIS_surfaceModel_diagnoseVarsforDA (Source File: LIS_surfaceModelMod.F90) 232
3.34.14 LIS_surfaceModel_finalize (Source File: LIS_surfaceModelMod.F90) 233
3.34.15 LIS_surfaceModel_reset (Source File: LIS_surfaceModelMod.F90) 233
3.34.16 surfaceModel_setexport_noesmf (Source File: LIS_surfaceModelMod.F90) 233
3.35 Fortran: Module Interface LIS_tbotAdjustMod (Source File: LIS_tbotAdjustMod.F90) 233
3.35.1 Overview 233
3.35.2 LIS_createTmnUpdate (Source File: LIS_tbotAdjustMod.F90) 234
3.35.3 LIS_initTmnUpdateTile (Source File: LIS_tbotAdjustMod.F90) 234
3.35.4 LIS_updateTbot (Source File: LIS_tbotAdjustMod.F90) 234
3.35.5 LIS_writeTmnUpdateRestart (Source File: LIS_tbotAdjustMod.F90) 235
3.35.6 LIS_readTmnUpdateRestart (Source File: LIS_tbotAdjustMod.F90) 235
3.35.7 LIS_tbotTerrainAdjustment (Source File: LIS_tbotAdjustMod.F90) 236
3.36 Fortran: Module Interface LIS_timeMgrMod (Source File: LIS_timeMgrMod.F90) 236
3.36.1 LIS_isAlarmRinging (Source File: LIS_timeMgrMod.F90) 236
3.36.2 LIS_timemgr_init (Source File: LIS_timeMgrMod.F90) 237
3.36.3 LIS_resetClockForTimeWindow (Source File: LIS_timeMgrMod.F90) 237
3.36.4 LIS_resetclock (Source File: LIS_timeMgrMod.F90) 238
3.36.5 LIS_timemgr_set (Source File: LIS_timeMgrMod.F90) 238
3.36.6 LIS_timemgr_print (Source File: LIS_timeMgrMod.F90) 239
3.36.7 LIS_advance_timestep (Source File: LIS_timeMgrMod.F90) 239
3.36.8 LIS_update_timestep (Source File: LIS_timeMgrMod.F90) 240
3.36.9 LIS_update_clock (Source File: LIS_timeMgrMod.F90) 240
3.36.10 LIS_get_step_size (Source File: LIS_timeMgrMod.F90) 240
3.36.11 LIS_get_nstep (Source File: LIS_timeMgrMod.F90) 241
3.36.12 LIS_get_curr_day (Source File: LIS_timeMgrMod.F90) 241
3.36.13 LIS_get_julhr (Source File: LIS_timeMgrMod.F90) 242
3.36.14 LIS_get_julss (Source File: LIS_timeMgrMod.F90) 242
3.36.15 LIS_get_curr_calday (Source File: LIS_timeMgrMod.F90) 243
3.36.16 LIS_is_last_step (Source File: LIS_timeMgrMod.F90) 243
3.36.17 isMonthlyAlarmRinging (Source File: LIS_timeMgrMod.F90) 244
3.36.18 LIS_getDekad (Source File: LIS_timeMgrMod.F90) 245
3.36.19 LIS_isDekadalAlarmRinging (Source File: LIS_timeMgrMod.F90) 245
3.36.20 LIS_getSecsInDekad (Source File: LIS_timeMgrMod.F90) 246
3.36.21 LIS_finishDekadalAlarms (Source File: LIS_timeMgrMod.F90) 246
3.36.22 LIS_computeTimeBookEnds (Source File: LIS_timeMgrMod.F90) 246
3.36.23 LIS_computeTemporalWeights (Source File: LIS_timeMgrMod.F90) 247
3.36.24 LIS_date2time (Source File: LIS_timeMgrMod.F90) 248
3.36.25 LIS_time2date (Source File: LIS_timeMgrMod.F90) 248
3.36.26 LIS_doy2date (Source File: LIS_timeMgrMod.F90) 249
3.36.27 LIS_tick (Source File: LIS_timeMgrMod.F90) 249
3.36.28 LIS_julhr_date (Source File: LIS_timeMgrMod.F90) 250
3.36.29 Method 251
3.36.30 LIS_tmjul4 (Source File: LIS_timeMgrMod.F90) 251

3.36.31 Method . . . 252
3.36.32 LIS_localtime2gmt (Source File: LIS_timeMgrMod.F90) . . . 252
3.36.33 LIS_registerAlarm (Source File: LIS_timeMgrMod.F90) . . . 252
3.36.34 isAlarmRinging (Source File: LIS_timeMgrMod.F90) . . . 253
3.36.35 LIS_parseTimeString (Source File: LIS_timeMgrMod.F90) . . . 254
3.37 Fortran: Module Interface LIS_topoMod (Source File: LIS_topoMod.F90) . . . 254
3.37.1 Overview . . . 254
3.37.2 LIS_topo_init (Source File: LIS_topoMod.F90) . . . 255
3.37.3 LIS_diagnoseTopography (Source File: LIS_topoMod.F90) . . . 255
3.37.4 read_elevation (Source File: LIS_topoMod.F90) . . . 255
3.37.5 read_slope (Source File: LIS_topoMod.F90) . . . 256
3.37.6 read_aspect (Source File: LIS_topoMod.F90) . . . 257
3.38 Fortran: Module Interface LIS_vegDataMod (Source File: LIS_vegDataMod.F90) . . . 257
3.38.1 Overview . . . 258
3.38.2 LIS_greenness_setup (Source File: LIS_vegDataMod.F90) . . . 259
3.38.3 LIS_read_greenness (Source File: LIS_vegDataMod.F90) . . . 259
3.38.4 LIS_greenness_finalize (Source File: LIS_vegDataMod.F90) . . . 260
3.38.5 LIS_diagnosegfrac (Source File: LIS_vegDataMod.F90) . . . 260
3.38.6 LIS_greenness_reset (Source File: LIS_vegDataMod.F90) . . . 261
3.38.7 read_gfracclimo (Source File: LIS_vegDataMod.F90) . . . 261
3.38.8 LIS_read_shdmin (Source File: LIS_vegDataMod.F90) . . . 262
3.38.9 LIS_read_shdmax (Source File: LIS_vegDataMod.F90) . . . 262
3.38.10 LIS_read_laimin (Source File: LIS_vegDataMod.F90) . . . 263
3.38.11 LIS_read_laimax (Source File: LIS_vegDataMod.F90) . . . 263
3.38.12 LIS_roughness_setup (Source File: LIS_vegDataMod.F90) . . . 264
3.38.13 LIS_read_roughness (Source File: LIS_vegDataMod.F90) . . . 264
3.38.14 LIS_roughness_finalize (Source File: LIS_vegDataMod.F90) . . . 265
3.38.15 LIS_diagnoseroughness (Source File: LIS_vegDataMod.F90) . . . 265
3.38.16 LIS_roughness_reset (Source File: LIS_vegDataMod.F90) . . . 265
3.38.17 read_roughnessclimo (Source File: LIS_vegDataMod.F90) . . . 266
3.38.18 LIS_lai_setup (Source File: LIS_vegDataMod.F90) . . . 266
3.38.19 LIS_read_lai (Source File: LIS_vegDataMod.F90) . . . 267
3.38.20 LIS_lai_setup (Source File: LIS_vegDataMod.F90) . . . 267
3.38.21 LIS_read_lai (Source File: LIS_vegDataMod.F90) . . . 268
3.38.22 LIS_lai_finalize (Source File: LIS_vegDataMod.F90) . . . 269
3.38.23 LIS_diagnoseLAI (Source File: LIS_vegDataMod.F90) . . . 269
3.38.24 LIS_lai_reset (Source File: LIS_vegDataMod.F90) . . . 269
3.38.25 LIS_sai_setup (Source File: LIS_vegDataMod.F90) . . . 270
3.38.26 LIS_read_sai (Source File: LIS_vegDataMod.F90) . . . 271
3.38.27 LIS_sai_finalize (Source File: LIS_vegDataMod.F90) . . . 271
3.38.28 LIS_diagnoseSAI (Source File: LIS_vegDataMod.F90) . . . 271
3.38.29 LIS_sai_reset (Source File: LIS_vegDataMod.F90) . . . 272
3.38.30 read_laiclimo (Source File: LIS_vegDataMod.F90) . . . 273
3.38.31 read_saiclimo (Source File: LIS_vegDataMod.F90) . . . 273
3.38.32 ft_check_index (Source File: FTable.c) . . . 274
3.39 Fortran: Module Interface LIS_RTM_FTable (Source File: LIS_RTM_FTable.c) . . . 274
3.39.1 registerrtminit (Source File: LIS_RTM_FTable.c) . . . 274
3.39.2 rtminit (Source File: LIS_RTM_FTable.c) . . . 275
3.39.3 registerrtmf2t (Source File: LIS_RTM_FTable.c) . . . 275
3.39.4 rtmf2t (Source File: LIS_RTM_FTable.c) . . . 275
3.39.5 registergeometry2rtm (Source File: LIS_RTM_FTable.c) . . . 275

3.39.6 geometry2rtm (Source File: LIS_RTM_FTable.c) . . . 276
3.39.7 registerrtmrun (Source File: LIS_RTM_FTable.c) . . . 276
3.39.8 rtmrun (Source File: LIS_RTM_FTable.c) . . . 276
3.39.9 registerrtmpesetdecisionspace (Source File: LIS_RTM_FTable.c) . . . 276
3.39.10 rtmpesetdecisionspace (Source File: LIS_RTM_FTable.c) . . . 277
3.39.11 registerrtmpegetdecisionspace (Source File: LIS_RTM_FTable.c) . . . 277
3.39.12 rtmpegetdecisionspace (Source File: LIS_RTM_FTable.c) . . . 277
3.39.13 registerrtmpesetupdecisionspace (Source File: LIS_RTM_FTable.c) . . . 278
3.39.14 rtmpesetupdecisionspace (Source File: LIS_RTM_FTable.c) . . . 278
3.39.15 registerrtmpesetupobspred (Source File: LIS_RTM_FTable.c) . . . 278
3.39.16 rtmpesetupobspredspace (Source File: LIS_RTM_FTable.c) . . . 279
3.39.17 registerrtmpegetobspred (Source File: LIS_RTM_FTable.c) . . . 279
3.39.18 rtmpegetobspred (Source File: LIS_RTM_FTable.c) . . . 279
3.39.19 registerrtmreset (Source File: LIS_RTM_FTable.c) . . . 279
3.39.20 resetrtm (Source File: LIS_RTM_FTable.c) . . . 280
3.40 Fortran: Module Interface LIS_alb_FTable (Source File: LIS_alb_FTable.c) . . . 280
3.40.1 registeralbedosetup (Source File: LIS_alb_FTable.c) . . . 280
3.40.2 albedosetup (Source File: LIS_alb_FTable.c) . . . 280
3.40.3 registerreadalbedo (Source File: LIS_alb_FTable.c) . . . 281
3.40.4 readalbedo (Source File: LIS_alb_FTable.c) . . . 281
3.41 Fortran: Module Interface LIS_biasestim_FTable (Source File: LIS_biasestim_FTable.c) . . . 281
3.41.1 registerbiasestimationinit (Source File: LIS_biasestim_FTable.c) . . . 281
3.41.2 biasestimationinit (Source File: LIS_biasestim_FTable.c) . . . 282
3.41.3 registerbiasestimationsetup (Source File: LIS_biasestim_FTable.c) . . . 282
3.41.4 biasestimationsetup (Source File: LIS_biasestim_FTable.c) . . . 282
3.41.5 registerbiasestimationcompute (Source File: LIS_biasestim_FTable.c) . . . 282
3.41.6 computebiascorrection (Source File: LIS_biasestim_FTable.c) . . . 283
3.41.7 registerbiasestimationupdate (Source File: LIS_biasestim_FTable.c) . . . 283
3.41.8 applybiascorrection (Source File: LIS_biasestim_FTable.c) . . . 283
3.41.9 registerbiasestimationrestart (Source File: LIS_biasestim_FTable.c) . . . 284
3.41.10 writebiasrestart (Source File: LIS_biasestim_FTable.c) . . . 284
3.41.11 registerbiasestimationfinalize (Source File: LIS_biasestim_FTable.c) . . . 284
3.41.12 biasestimationfinalize (Source File: LIS_biasestim_FTable.c) . . . 284
3.42 Fortran: Module Interface LIS_dataassim_FTable (Source File: LIS_dataassim_FTable.c) . . . 285
3.42.1 registerdainit (Source File: LIS_dataassim_FTable.c) . . . 285
3.42.2 dataassiminit (Source File: LIS_dataassim_FTable.c) . . . 285
3.42.3 registerdasetup (Source File: LIS_dataassim_FTable.c) . . . 285
3.42.4 dataassimsetup (Source File: LIS_dataassim_FTable.c) . . . 286
3.42.5 registerapplyincrements (Source File: LIS_dataassim_FTable.c) . . . 286
3.42.6 applyincrements (Source File: LIS_dataassim_FTable.c) . . . 286
3.42.7 registercomputeincrements (Source File: LIS_dataassim_FTable.c) . . . 287
3.42.8 computeincrements (Source File: LIS_dataassim_FTable.c) . . . 287
3.42.9 registerdafinalize (Source File: LIS_dataassim_FTable.c) . . . 287
3.42.10 dafinalize (Source File: LIS_dataassim_FTable.c) . . . 287
3.42.11 registerdaoutput (Source File: LIS_dataassim_FTable.c) . . . 288
3.42.12 daoutput (Source File: LIS_dataassim_FTable.c) . . . 288
3.42.13 registergetnso (Source File: LIS_dataassim_FTable.c) . . . 288
3.42.14 getselectedobsnumber (Source File: LIS_dataassim_FTable.c) . . . 288
3.42.15 registerreaddaobssetup (Source File: LIS_dataassim_FTable.c) . . . 289
3.42.16 readobsdataconfig (Source File: LIS_dataassim_FTable.c) . . . 289
3.42.17 registerreaddaobs (Source File: LIS_dataassim_FTable.c) . . . 290

3.42.18 readdaobs (Source File: LIS_dataassim_FTable.c) . . . 290
3.42.19 registerwritedaobs (Source File: LIS_dataassim_FTable.c) . . . 290
3.42.20 writedaobs (Source File: LIS_dataassim_FTable.c) . . . 291
3.43 Fortran: Module Interface LIS_domain_FTable (Source File: LIS_domain_FTable.c) . . . 291
3.43.1 registerdomain (Source File: LIS_domain_FTable.c) . . . 291
3.43.2 makedomain (Source File: LIS_domain_FTable.c) . . . 291
3.43.3 registerinput (Source File: LIS_domain_FTable.c) . . . 292
3.43.4 readinput (Source File: LIS_domain_FTable.c) . . . 292
3.44 Fortran: Module Interface LIS_emiss_FTable (Source File: LIS_emiss_FTable.c) . . . 292
3.44.1 registeremissivitysetup (Source File: LIS_emiss_FTable.c) . . . 292
3.44.2 emissivitysetup (Source File: LIS_emiss_FTable.c) . . . 293
3.44.3 registerreademissivity (Source File: LIS_emiss_FTable.c) . . . 293
3.44.4 reademissivity (Source File: LIS_emiss_FTable.c) . . . 293
3.45 Fortran: Module Interface LIS_gfrac_FTable (Source File: LIS_gfrac_FTable.c) . . . 294
3.45.1 registergfracsetup (Source File: LIS_gfrac_FTable.c) . . . 294
3.45.2 gfracsetup (Source File: LIS_gfrac_FTable.c) . . . 294
3.45.3 registerreadgfrac (Source File: LIS_gfrac_FTable.c) . . . 294
3.45.4 readgfrac (Source File: LIS_gfrac_FTable.c) . . . 295
3.46 Fortran: Module Interface LIS_irrigation_FTable (Source File: LIS_irrigation_FTable.c) . . . 295
3.46.1 registerirrigationschemeinit (Source File: LIS_irrigation_FTable.c) . . . 295
3.46.2 irrigationschemeinit (Source File: LIS_irrigation_FTable.c) . . . 295
3.46.3 registerirrigationupdate (Source File: LIS_irrigation_FTable.c) . . . 296
3.46.4 applyirrigationupdates (Source File: LIS_irrigation_FTable.c) . . . 296
3.47 Fortran: Module Interface LIS_laisai_FTable (Source File: LIS_laisai_FTable.c) . . . 296
3.47.1 registerlaisetup (Source File: LIS_laisai_FTable.c) . . . 296
3.47.2 laisetup (Source File: LIS_laisai_FTable.c) . . . 297
3.47.3 registersaisetup (Source File: LIS_laisai_FTable.c) . . . 297
3.47.4 saisetup (Source File: LIS_laisai_FTable.c) . . . 297
3.47.5 registerreadlai (Source File: LIS_laisai_FTable.c) . . . 298
3.47.6 readlai (Source File: LIS_laisai_FTable.c) . . . 298
3.47.7 registerreadsai (Source File: LIS_laisai_FTable.c) . . . 298
3.47.8 readsai (Source File: LIS_laisai_FTable.c) . . . 299
3.48 Fortran: Module Interface LIS_lakemodel_FTable (Source File: LIS_lakemodel_FTable.c) . . . 299
3.48.1 registerlakemodelini (Source File: LIS_lakemodel_FTable.c) . . . 299
3.48.2 lakemodelinit (Source File: LIS_lakemodel_FTable.c) . . . 299
3.48.3 registerlakemodelrun (Source File: LIS_lakemodel_FTable.c) . . . 300
3.48.4 lakemodelrun (Source File: LIS_lakemodel_FTable.c) . . . 300
3.48.5 registerlakemodelfinalize (Source File: LIS_lakemodel_FTable.c) . . . 300
3.48.6 lakemodelfinalize (Source File: LIS_lakemodel_FTable.c) . . . 300
3.48.7 registerlakemodelsetup (Source File: LIS_lakemodel_FTable.c) . . . 301
3.48.8 lakemodelsetup (Source File: LIS_lakemodel_FTable.c) . . . 301
3.48.9 registerlakemodelrestart (Source File: LIS_lakemodel_FTable.c) . . . 301
3.48.10 lakemodelrestart (Source File: LIS_lakemodel_FTable.c) . . . 301
3.48.11 registerlakemodeldynsetup (Source File: LIS_lakemodel_FTable.c) . . . 302
3.48.12 lakemodeldynsetup (Source File: LIS_lakemodel_FTable.c) . . . 302
3.48.13 registerlakemodeloutput (Source File: LIS_lakemodel_FTable.c) . . . 302
3.48.14 lakemodeloutput (Source File: LIS_lakemodel_FTable.c) . . . 302
3.48.15 registerlakemodelf2t (Source File: LIS_lakemodel_FTable.c) . . . 303
3.48.16 lakemodelf2t (Source File: LIS_lakemodel_FTable.c) . . . 303
3.48.17 registerlakemodelwrst (Source File: LIS_lakemodel_FTable.c) . . . 303
3.48.18 lakemodelwrst (Source File: LIS_lakemodel_FTable.c) . . . 304

3.49 Fortran: Module Interface LIS_landslide_FTable (Source File: LIS_landslide_FTable.c) 304
3.49.1 registerlandslidemodelinit (Source File: LIS_landslide_FTable.c) 304
3.49.2 initializelandslidemodel (Source File: LIS_landslide_FTable.c) 304
3.49.3 registerlandslidemodelrun (Source File: LIS_landslide_FTable.c) 304
3.49.4 runlandslidemodel (Source File: LIS_landslide_FTable.c) 305
3.49.5 registerlandslidemodeloutput (Source File: LIS_landslide_FTable.c) 305
3.49.6 outputlandslidemodel (Source File: LIS_landslide_FTable.c) 305
3.49.7 registerlandslidemodelfinalize (Source File: LIS_landslide_FTable.c) 305
3.49.8 landslidemodelfinalize (Source File: LIS_landslide_FTable.c) 306
3.49.9 registersetuplandslidepeobspred (Source File: LIS_landslide_FTable.c) 306
3.49.10 setuplandslidepeobspredspace (Source File: LIS_landslide_FTable.c) 306
3.49.11 registerqclandslidedecisionspace (Source File: LIS_landslide_FTable.c) 307
3.49.12 qclandslidedecisionspace (Source File: LIS_landslide_FTable.c) 307
3.49.13 registersetlandslidedecisionspace (Source File: LIS_landslide_FTable.c) 307
3.49.14 setlandslidedecisionspace (Source File: LIS_landslide_FTable.c) 308
3.49.15 registerlandslidemodelreset (Source File: LIS_landslide_FTable.c) 308
3.49.16 resetlandslidemodel (Source File: LIS_landslide_FTable.c) 308
3.49.17 lis_log_msgC.c (Source File: LIS_log_msgC.c) 308
3.49.18 lis_log_msgC (Source File: LIS_log_msgC.c) 308
3.50 Fortran: Module Interface LIS_lsm_FTable (Source File: LIS_lsm_FTable.c) 309
3.50.1 registerlsmini (Source File: LIS_lsm_FTable.c) 309
3.50.2 lsminit (Source File: LIS_lsm_FTable.c) 309
3.50.3 registerlsmrun (Source File: LIS_lsm_FTable.c) 309
3.50.4 lsmrun (Source File: LIS_lsm_FTable.c) 309
3.50.5 registerlsmfinalize (Source File: LIS_lsm_FTable.c) 310
3.50.6 lsmfinalize (Source File: LIS_lsm_FTable.c) 310
3.50.7 registerlsmreset (Source File: LIS_lsm_FTable.c) 310
3.50.8 lsmreset (Source File: LIS_lsm_FTable.c) 310
3.50.9 registerlsmsetup (Source File: LIS_lsm_FTable.c) 311
3.50.10 lsmsetup (Source File: LIS_lsm_FTable.c) 311
3.50.11 registerlsmrestart (Source File: LIS_lsm_FTable.c) 311
3.50.12 lsmrestart (Source File: LIS_lsm_FTable.c) 311
3.50.13 registerlsmdynsetup (Source File: LIS_lsm_FTable.c) 312
3.50.14 lsmdynsetup (Source File: LIS_lsm_FTable.c) 312
3.50.15 registerlsmf2t (Source File: LIS_lsm_FTable.c) 312
3.50.16 lsmf2t (Source File: LIS_lsm_FTable.c) 312
3.50.17 registerlsmwrst (Source File: LIS_lsm_FTable.c) 313
3.50.18 lsmwrst (Source File: LIS_lsm_FTable.c) 313
3.50.19 registerlsmdainit (Source File: LIS_lsm_FTable.c) 313
3.50.20 lsmdainit (Source File: LIS_lsm_FTable.c) 314
3.50.21 registerlsmdagetstatevar (Source File: LIS_lsm_FTable.c) 314
3.50.22 lsmdagetstatevar (Source File: LIS_lsm_FTable.c) 314
3.50.23 registerlsmdasetstatevar (Source File: LIS_lsm_FTable.c) 315
3.50.24 lsmdasetstatevar (Source File: LIS_lsm_FTable.c) 315
3.50.25 registerlsmdaobstransform (Source File: LIS_lsm_FTable.c) 315
3.50.26 lsmdaobstransform (Source File: LIS_lsm_FTable.c) 316
3.50.27 registerlsmdagetobspred (Source File: LIS_lsm_FTable.c) 316
3.50.28 lsmdagetobspred (Source File: LIS_lsm_FTable.c) 316
3.50.29 registerlsmdadiagnosevars (Source File: LIS_lsm_FTable.c) 317
3.50.30 lsmdadiagnosevars (Source File: LIS_lsm_FTable.c) 317
3.50.31 registerlsmdamapobstolsm (Source File: LIS_lsm_FTable.c) 317

3.50.32 lsmdamapobstolsm (Source File: LIS_lsm_FTable.c) . . . 317
3.50.33 registerlsmdaqcstate (Source File: LIS_lsm_FTable.c) . . . 318
3.50.34 lsmdaqcstate (Source File: LIS_lsm_FTable.c) . . . 318
3.50.35 registerlsmdaqcobsstate (Source File: LIS_lsm_FTable.c) . . . 318
3.50.36 lsmdaqcobsstate (Source File: LIS_lsm_FTable.c) . . . 319
3.50.37 registerlsmdascalestatevar (Source File: LIS_lsm_FTable.c) . . . 319
3.50.38 lsmdascalestatevar (Source File: LIS_lsm_FTable.c) . . . 319
3.50.39 registerlsmdadescalestatevar (Source File: LIS_lsm_FTable.c) . . . 320
3.50.40 lsmdadescalestatevar (Source File: LIS_lsm_FTable.c) . . . 320
3.50.41 registerlsmdaupdatestate (Source File: LIS_lsm_FTable.c) . . . 320
3.50.42 lsmdaupdatestate (Source File: LIS_lsm_FTable.c) . . . 321
3.50.43 registerwritestatevar (Source File: LIS_lsm_FTable.c) . . . 321
3.50.44 writelsmstatevar (Source File: LIS_lsm_FTable.c) . . . 321
3.50.45 registerlsmcplsetexport (Source File: LIS_lsm_FTable.c) . . . 322
3.50.46 lsmcplsetexport (Source File: LIS_lsm_FTable.c) . . . 322
3.50.47 registerlsmpesetdecisionspace (Source File: LIS_lsm_FTable.c) . . . 322
3.50.48 lsmpesetdecisionspace (Source File: LIS_lsm_FTable.c) . . . 322
3.50.49 registerlsmpegetdecisionspace (Source File: LIS_lsm_FTable.c) . . . 323
3.50.50 lsmpegetdecisionspace (Source File: LIS_lsm_FTable.c) . . . 323
3.50.51 registerlsmpesetupdecisionspace (Source File: LIS_lsm_FTable.c) . . . 323
3.50.52 lsmpesetupdecisionspace (Source File: LIS_lsm_FTable.c) . . . 324
3.50.53 registerlsmpesetupobspred (Source File: LIS_lsm_FTable.c) . . . 324
3.50.54 lsmpesetupobspredspace (Source File: LIS_lsm_FTable.c) . . . 324
3.50.55 registerlsmpegetobspred (Source File: LIS_lsm_FTable.c) . . . 325
3.50.56 lsmpegetobspred (Source File: LIS_lsm_FTable.c) . . . 325
3.50.57 registerlsmroutinggetrunoff (Source File: LIS_lsm_FTable.c) . . . 325
3.50.58 lsmroutinggetrunoff (Source File: LIS_lsm_FTable.c) . . . 325
3.50.59 registerlsm2rtm (Source File: LIS_lsm_FTable.c) . . . 326
3.50.60 lsm2rtm (Source File: LIS_lsm_FTable.c) . . . 326
3.50.61 registerlsmirrigationgetstates (Source File: LIS_lsm_FTable.c) . . . 326
3.50.62 getirrigationlsmstates (Source File: LIS_lsm_FTable.c) . . . 327
3.50.63 LIS_memory_managementC (Source File: LIS_memory_managementC.c) . . . 327
3.50.64 lis_calloc (Source File: LIS_memory_managementC.c) . . . 327
3.50.65 lis_malloc (Source File: LIS_memory_managementC.c) . . . 327
3.51 Fortran: Module Interface LIS_metforcing_FTable (Source File: LIS_metforcing_FTable.c) . . . 327
3.51.1 registerinitmetforc (Source File: LIS_metforcing_FTable.c) . . . 328
3.51.2 initmetforc (Source File: LIS_metforcing_FTable.c) . . . 328
3.51.3 registerretrievemetforc (Source File: LIS_metforcing_FTable.c) . . . 328
3.51.4 retrievemetforc (Source File: LIS_metforcing_FTable.c) . . . 328
3.51.5 registertimeinterpmetforc (Source File: LIS_metforcing_FTable.c) . . . 329
3.51.6 timeinterpmetforc (Source File: LIS_metforcing_FTable.c) . . . 329
3.51.7 registerfinalmetforc (Source File: LIS_metforcing_FTable.c) . . . 329
3.51.8 finalmetforc (Source File: LIS_metforcing_FTable.c) . . . 330
3.51.9 registerresetmetforc (Source File: LIS_metforcing_FTable.c) . . . 330
3.51.10 resetmetforc (Source File: LIS_metforcing_FTable.c) . . . 330
3.51.11 LIS_mpDecomp (Source File: LIS_mpDecomp.c) . . . 330
3.52 Fortran: Module Interface LIS_openwater_FTable (Source File: LIS_openwater_FTable.c) . . . 331
3.52.1 registeropenwaterini (Source File: LIS_openwater_FTable.c) . . . 331
3.52.2 openwaterinit (Source File: LIS_openwater_FTable.c) . . . 331
3.52.3 registeropenwaterrun (Source File: LIS_openwater_FTable.c) . . . 331
3.52.4 openwaterrun (Source File: LIS_openwater_FTable.c) . . . 331

3.52.5 registeropenwaterfinalize (Source File: LIS_openwater_FTable.c) . . . 332
3.52.6 openwaterfinalize (Source File: LIS_openwater_FTable.c) . . . 332
3.52.7 registeropenwatersetup (Source File: LIS_openwater_FTable.c) . . . 332
3.52.8 openwatersetup (Source File: LIS_openwater_FTable.c) . . . 332
3.52.9 registeropenwaterrestart (Source File: LIS_openwater_FTable.c) . . . 333
3.52.10 openwaterrestart (Source File: LIS_openwater_FTable.c) . . . 333
3.52.11 registeropenwaterdynsetup (Source File: LIS_openwater_FTable.c) . . . 333
3.52.12 openwaterdynsetup (Source File: LIS_openwater_FTable.c) . . . 333
3.52.13 registeropenwateroutput (Source File: LIS_openwater_FTable.c) . . . 334
3.52.14 openwateroutput (Source File: LIS_openwater_FTable.c) . . . 334
3.52.15 registeropenwaterf2t (Source File: LIS_openwater_FTable.c) . . . 334
3.52.16 openwaterf2t (Source File: LIS_openwater_FTable.c) . . . 334
3.52.17 registeropenwaterwrst (Source File: LIS_openwater_FTable.c) . . . 335
3.52.18 openwaterwrst (Source File: LIS_openwater_FTable.c) . . . 335
3.53 Fortran: Module Interface LIS_optUE_FTable (Source File: LIS_optUE_FTable.c) . . . 335
3.53.1 registeroptuealginit (Source File: LIS_optUE_FTable.c) . . . 336
3.53.2 optuealginit (Source File: LIS_optUE_FTable.c) . . . 336
3.53.3 registeroptuealgsetup (Source File: LIS_optUE_FTable.c) . . . 336
3.53.4 optuealgsetup (Source File: LIS_optUE_FTable.c) . . . 336
3.53.5 registeroptueconvergencecheck (Source File: LIS_optUE_FTable.c) . . . 337
3.53.6 checkconvergence (Source File: LIS_optUE_FTable.c) . . . 337
3.53.7 registeroptuealgrun (Source File: LIS_optUE_FTable.c) . . . 337
3.53.8 runoptue (Source File: LIS_optUE_FTable.c) . . . 337
3.53.9 registergetdecisionspace (Source File: LIS_optUE_FTable.c) . . . 338
3.53.10 (Source File: LIS_optUE_FTable.c) . . . 338
3.53.11 registeroptuesetdecisionspace (Source File: LIS_optUE_FTable.c) . . . 338
3.53.12 (Source File: LIS_optUE_FTable.c) . . . 338
3.53.13 registeroptuegetnparam (Source File: LIS_optUE_FTable.c) . . . 339
3.53.14 (Source File: LIS_optUE_FTable.c) . . . 339
3.53.15 registeroptuereadrestart (Source File: LIS_optUE_FTable.c) . . . 339
3.53.16 (Source File: LIS_optUE_FTable.c) . . . 340
3.53.17 registerpeobssetup (Source File: LIS_optUE_FTable.c) . . . 340
3.53.18 setuppeobsspace (Source File: LIS_optUE_FTable.c) . . . 340
3.53.19 registerpeobsreset (Source File: LIS_optUE_FTable.c) . . . 341
3.53.20 resetpeobsspace (Source File: LIS_optUE_FTable.c) . . . 341
3.53.21 registergetpeobs (Source File: LIS_optUE_FTable.c) . . . 341
3.53.22 getpeobs (Source File: LIS_optUE_FTable.c) . . . 341
3.53.23 registerwritepeobs (Source File: LIS_optUE_FTable.c) . . . 342
3.53.24 writepeobs (Source File: LIS_optUE_FTable.c) . . . 342
3.53.25 registerinitobjfunctype (Source File: LIS_optUE_FTable.c) . . . 342
3.53.26 objectivefunctypeinit (Source File: LIS_optUE_FTable.c) . . . 343
3.53.27 registercomputeobjfunctype (Source File: LIS_optUE_FTable.c) . . . 343
3.53.28 computeobjectivefunctype (Source File: LIS_optUE_FTable.c) . . . 343
3.53.29 registerupdateobjfunctype (Source File: LIS_optUE_FTable.c) . . . 343
3.53.30 updateobjectivefunctype (Source File: LIS_optUE_FTable.c) . . . 344
3.53.31 registerresetobjfunctype (Source File: LIS_optUE_FTable.c) . . . 344
3.53.32 resetobjectivefunctype (Source File: LIS_optUE_FTable.c) . . . 344
3.53.33 registeroptuetypeobjfunceval (Source File: LIS_optUE_FTable.c) . . . 344
3.53.34 evaluateobjfunction (Source File: LIS_optUE_FTable.c) . . . 345
3.54 Fortran: Module Interface LIS_perturb_FTable (Source File: LIS_perturb_FTable.c) . . . 345
3.54.1 registerperturbinit (Source File: LIS_perturb_FTable.c) . . . 345

3.54.2 perturbinit (Source File: LIS_perturb_FTable.c) . . . 345
3.54.3 registerperturbsetup (Source File: LIS_perturb_FTable.c) . . . 346
3.54.4 perturbsetup (Source File: LIS_perturb_FTable.c) . . . 346
3.54.5 registerperturbmethod (Source File: LIS_perturb_FTable.c) . . . 346
3.54.6 perturbmethod (Source File: LIS_perturb_FTable.c) . . . 347
3.54.7 registerperturbwriterst (Source File: LIS_perturb_FTable.c) . . . 347
3.54.8 writepertrestart (Source File: LIS_perturb_FTable.c) . . . 347
3.54.9 registerperturbreadrst (Source File: LIS_perturb_FTable.c) . . . 348
3.54.10 readpertrestart (Source File: LIS_perturb_FTable.c) . . . 348
3.55 Fortran: Module Interface LIS_roughness_FTable (Source File: LIS_roughness_FTable.c) . . . 348
3.55.1 registerroughnesssetup (Source File: LIS_roughness_FTable.c) . . . 348
3.55.2 roughnesssetup (Source File: LIS_roughness_FTable.c) . . . 349
3.55.3 registerreadroughness (Source File: LIS_roughness_FTable.c) . . . 349
3.55.4 readroughness (Source File: LIS_roughness_FTable.c) . . . 349
3.56 Fortran: Module Interface LIS_routing_FTable (Source File: LIS_routing_FTable.c) . . . 350
3.56.1 registerroutinginit (Source File: LIS_routing_FTable.c) . . . 350
3.56.2 routinginit (Source File: LIS_routing_FTable.c) . . . 350
3.56.3 registerroutingreadrestart (Source File: LIS_routing_FTable.c) . . . 350
3.56.4 routingreadrestart (Source File: LIS_routing_FTable.c) . . . 350
3.56.5 registerroutingrun (Source File: LIS_routing_FTable.c) . . . 351
3.56.6 routingrun (Source File: LIS_routing_FTable.c) . . . 351
3.56.7 registerroutingoutput (Source File: LIS_routing_FTable.c) . . . 351
3.56.8 routingoutput (Source File: LIS_routing_FTable.c) . . . 351
3.56.9 registerroutingwriterestart (Source File: LIS_routing_FTable.c) . . . 351
3.56.10 routingwriterestart (Source File: LIS_routing_FTable.c) . . . 352
3.57 Fortran: Module Interface LIS_runmode_FTable (Source File: LIS_runmode_FTable.c) . . . 352
3.57.1 registerlisinit (Source File: LIS_runmode_FTable.c) . . . 352
3.57.2 lisinit (Source File: LIS_runmode_FTable.c) . . . 352
3.57.3 registerlisrun (Source File: LIS_runmode_FTable.c) . . . 353
3.57.4 lisrun (Source File: LIS_runmode_FTable.c) . . . 353
3.57.5 registerlisfinalize (Source File: LIS_runmode_FTable.c) . . . 353
3.57.6 lisfinalize (Source File: LIS_runmode_FTable.c) . . . 353

IV Plugin Interfaces in LIS 354

4 Plugin interfaces in LIS 355
4.1 Fortran: Module Interface LIS_DAobs_pluginMod (Source File: LIS_DAobs_pluginMod.F90) . 355
4.1.1 LIS_DAobs_plugin (Source File: LIS_DAobs_pluginMod.F90) . . . 355
4.2 Fortran: Module Interface LIS_ObjFunc_pluginMod (Source File: LIS_ObjFunc_pluginMod.F90) 356
4.2.1 LIS_ObjFunc_plugin (Source File: LIS_ObjFunc_pluginMod.F90) . . . 356
4.3 Fortran: Module Interface LIS_PEobs_pluginMod (Source File: LIS_PEobs_pluginMod.F90) . 356
4.3.1 LIS_PEobs_plugin (Source File: LIS_PEobs_pluginMod.F90) . . . 356
4.4 Fortran: Module Interface LIS_RTM_pluginMod (Source File: LIS_RTM_pluginMod.F90) . . . 357
4.4.1 LIS_RTM_plugin (Source File: LIS_RTM_pluginMod.F90) . . . 357
4.5 Fortran: Module Interface LIS_RTMoptue_pluginMod (Source File: LIS_RTMoptue_pluginMod.F90) 357
4.5.1 LIS_RTMoptue_plugin (Source File: LIS_RTMoptue_pluginMod.F90) . . . 357
4.6 Fortran: Module Interface LIS_biasEstimation_pluginMod (Source File: LIS_biasEstimation_pluginMod.F90) 358
4.6.1 LIS_biasEstimation_plugin (Source File: LIS_biasEstimation_pluginMod.F90) . . . 358
4.7 Fortran: Module Interface LIS_dataassim_pluginMod.F90 (Source File: LIS_dataassim_pluginMod.F90) 359
4.7.1 LIS_dataassim_plugin (Source File: LIS_dataassim_pluginMod.F90) . . . 359

4.8 Fortran: Module Interface LIS_domain_pluginMod (Source File: LIS_domain_pluginMod.F90) 360
4.8.1 LIS_domain_plugin (Source File: LIS_domain_pluginMod.F90) 360
4.9 Fortran: Module Interface LIS_irrigationmodel_pluginMod (Source File: LIS_irrigationmodel_pluginMod.F90) 360
4.9.1 LIS_irrigationmodel_plugin (Source File: LIS_irrigationmodel_pluginMod.F90) 361
4.10 Fortran: Module Interface LIS_lakemodel_pluginMod (Source File: LIS_lakemodel_pluginMod.F90) 361
4.10.1 LIS_lakemodel_plugin (Source File: LIS_lakemodel_pluginMod.F90) 361
4.11 Fortran: Module Interface LIS_landslidemodel_pluginMod (Source File: LIS_landslidemodel_pluginMod.F90) 362
4.11.1 LIS_landslidemodel_plugin (Source File: LIS_landslidemodel_pluginMod.F90) 362
4.12 Fortran: Module Interface LIS_lsm_pluginMod (Source File: LIS_lsm_pluginMod.F90) 363
4.12.1 LIS_lsm_plugin (Source File: LIS_lsm_pluginMod.F90) 363
4.13 Fortran: Module Interface LIS_lsmcpl_pluginMod (Source File: LIS_lsmcpl_pluginMod.F90) . 365
4.13.1 LIS_lsmcpl_plugin (Source File: LIS_lsmcpl_pluginMod.F90) 365
4.14 Fortran: Module Interface LIS_lsmda_pluginMod (Source File: LIS_lsmda_pluginMod.F90) . . 365
4.14.1 LIS_lsmda_plugin (Source File: LIS_lsmda_pluginMod.F90) 366
4.15 Fortran: Module Interface LIS_lsmirrigation_pluginMod (Source File: LIS_lsmirrigation_pluginMod.F90) 367
4.15.1 LIS_lsmirrigation_plugin (Source File: LIS_lsmirrigation_pluginMod.F90) 367
4.16 Fortran: Module Interface LIS_lsmoptue_pluginMod (Source File: LIS_lsmoptue_pluginMod.F90) 367
4.16.1 LIS_lsmoptue_plugin (Source File: LIS_lsmoptue_pluginMod.F90) 368
4.17 Fortran: Module Interface LIS_lsmrouting_pluginMod (Source File: LIS_lsmrouting_pluginMod.F90) 368
4.17.1 LIS_lsmrouting_plugin (Source File: LIS_lsmrouting_pluginMod.F90) 368
4.18 Fortran: Module Interface LIS_lsmrtm_pluginMod (Source File: LIS_lsmrtm_pluginMod.F90) 369
4.18.1 LIS_lsmtm_plugin (Source File: LIS_lsmrtm_pluginMod.F90) 369
4.19 Fortran: Module Interface LIS_metforcing_pluginMod (Source File: LIS_metforcing_pluginMod.F90) 369
4.19.1 LIS_metforcing_plugin (Source File: LIS_metforcing_pluginMod.F90) 369
4.20 Fortran: Module Interface LIS_openwatermodel_pluginMod (Source File: LIS_openwatermodel_pluginMod.F90) 370
4.20.1 LIS_openwatermodel_plugin (Source File: LIS_openwatermodel_pluginMod.F90) 370
4.21 Fortran: Module Interface LIS_optUEAlgorithm_pluginMod (Source File: LIS_optUEAlgorithm_pluginMod.F90) 372
4.21.1 LIS_optUEAlgorithm_plugin (Source File: LIS_optUEAlgorithm_pluginMod.F90) . . . 372
4.22 Fortran: Module Interface LIS_param_pluginMod (Source File: LIS_param_pluginMod.F90) . 372
4.22.1 LIS_laisai_plugin (Source File: LIS_param_pluginMod.F90) 372
4.22.2 LIS_gfrac_plugin (Source File: LIS_param_pluginMod.F90) 373
4.22.3 LIS_roughness_plugin (Source File: LIS_param_pluginMod.F90) 374
4.22.4 LIS_emissivity_plugin (Source File: LIS_param_pluginMod.F90) 374
4.22.5 LIS_alb_plugin (Source File: LIS_param_pluginMod.F90) 375
4.23 Fortran: Module Interface LIS_perturb_pluginMod (Source File: LIS_perturb_pluginMod.F90) 375
4.23.1 LIS_perturb_plugin (Source File: LIS_perturb_pluginMod.F90) 376
4.24 Fortran: Module Interface LIS_pluginIndices (Source File: LIS_pluginIndices.F90) 376
4.25 Fortran: Module Interface LIS_routing_pluginMod (Source File: LIS_routing_pluginMod.F90) 381
4.25.1 LIS_routing_plugin (Source File: LIS_routing_pluginMod.F90) 381
4.26 Fortran: Module Interface LIS_runmode_pluginMod (Source File: LIS_runmode_pluginMod.F90) 381
4.26.1 LIS_runmode_plugin (Source File: LIS_runmode_pluginMod.F90) 381

V Interpolation Tools in LIS 383

5 Interpolation tools in LIS 384
5.1 Fortran: Module Interface UTM_utils (Source File: UTM_utils.F90) 384
5.1.1 bilinear_interp (Source File: bilinear_interp.F90) 384
5.1.2 bilinear_interp_input (Source File: bilinear_interp_input.F90) 386
5.1.3 compute_earth_coord (Source File: compute_earth_coord.F90) 387
5.1.4 compute_earth_coord_UTM (Source File: compute_earth_coord_UTM.F90) 388

5.1.5 compute_earth_coord_ease (Source File: compute_earth_coord_ease.F90) 389
5.1.6 compute_earth_coord_gauss (Source File: compute_earth_coord_gauss.F90) 389
5.1.7 compute_earth_coord_hrap (Source File: compute_earth_coord_hrap.F90) 390
5.1.8 compute_earth_coord_lambert (Source File: compute_earth_coord_lambert.F90) 391
5.1.9 compute_earth_coord_latlon (Source File: compute_earth_coord_latlon.F90) 392
5.1.10 compute_earth_coord_latlon_ll (Source File: compute_earth_coord_latlon_ll.F90) 393
5.1.11 compute_earth_coord_ll (Source File: compute_earth_coord_ll.F90) 394
5.1.12 compute_earth_coord_merc (Source File: compute_earth_coord_merc.F90) 395
5.1.13 compute_earth_coord_polar (Source File: compute_earth_coord_polar.F90) 396
5.1.14 compute_grid_coord (Source File: compute_grid_coord.F90) 397
5.1.15 compute_grid_coord_ease (Source File: compute_grid_coord_ease.F90) 398
5.1.16 compute_grid_coord_gauss (Source File: compute_grid_coord_gauss.F90) 399
5.1.17 compute_grid_coord_hrap (Source File: compute_grid_coord_hrap.F90) 399
5.1.18 compute_grid_coord_lambert (Source File: compute_grid_coord_lambert.F90) 400
5.1.19 compute_grid_coord_latlon (Source File: compute_grid_coord_latlon.F90) 401
5.1.20 compute_grid_coord_merc (Source File: compute_grid_coord_merc.F90) 402
5.1.21 compute_grid_coord_polar (Source File: compute_grid_coord_polar.F90) 403
5.1.22 compute_stnwts (Source File: compute_stnwts.F90) 404
5.1.23 conserv_interp (Source File: conserv_interp.F90) 405
5.1.24 conserv_interp_input (Source File: conserv_interp_input.F90) 406
5.1.25 defineLocalDomain (Source File: defineLocalDomain.F90) 407
5.1.26 ezlh_convert (Source File: ezlh_convert.F90) 408
5.1.27 ezlh_inverse (Source File: ezlh_inverse.F90) 409
5.2 Fortran: Module Interface gaussian_mod (Source File: gaussian_mod.F90) 410
5.2.1 gaussian_find_row (Source File: gaussian_mod.F90) 410
5.2.2 gaussian_find_col (Source File: gaussian_mod.F90) 410
5.2.3 gaussian_comp_lats (Source File: gaussian_mod.F90) 411
5.2.4 gaussian_comp_lons (Source File: gaussian_mod.F90) 411
5.2.5 gaussian_read_grid (Source File: gaussian_mod.F90) 412
5.2.6 get_fieldpos (Source File: get_fieldpos.F90) 413
5.2.7 hrapToLatLon (Source File: hrapToLatLon.F90) 413
5.2.8 interp_stndata (Source File: interp_stndata.F90) 414
5.2.9 latlonTohrap (Source File: latlonTohrap.F90) 415
5.2.10 latlonTopolar (Source File: latlonTopolar.F90) 415
5.2.11 lltops (Source File: lltops.F90) 416
5.3 Fortran: Module Interface map_utils (Source File: map_utils.F90) 417
5.3.1 Supported Projections 417
5.3.2 Remarks 417
5.3.3 Assumptions 417
5.3.4 Data Definitions: 418
5.3.5 Usage 418
5.3.6 map_init (Source File: map_utils.F90) 420
5.3.7 map_set (Source File: map_utils.F90) 420
5.3.8 latlon_to_ij (Source File: map_utils.F90) 421
5.3.9 ij_to_latlon (Source File: map_utils.F90) 422
5.3.10 neighbor_interp (Source File: neighbor_interp.F90) 422
5.3.11 neighbor_interp_input (Source File: neighbor_interp_input.F90) 424
5.3.12 normalize_stnwts (Source File: normalize_stnwts.F90) 425
5.3.13 polarToLatLon (Source File: polarToLatLon.F90) 425
5.3.14 polfixs (Source File: polfixs.F90) 426
5.3.15 pstoll (Source File: pstoll.F90) 427

5.3.16 snsoidxyTolatlon (Source File: snsoidxyTolatlon.F90) 428
5.3.17 upscaleByAveraging (Source File: upscaleByAveraging.F90) 428
5.3.18 upscaleByAveraging_input (Source File: upscaleByAveraging_input.F90) 429
5.3.19 zterp (Source File: zterp.F90) 430
5.3.20 coszenith (Source File: zterp.F90) 432
5.3.21 LIS_localtime (Source File: zterp.F90) 432

VI User-defined extensible components in LIS 434

6 User-defined extensible components in LIS 435

VII Running Modes in LIS 436

7 Running Modes in LIS 437

8 optUE Running Mode 437
8.1 Fortran: Module Interface paramEstim_runMod (Source File: paramEstim_runMod.F90) . . . 437
8.1.1 lis_init_paramEstim (Source File: paramEstim_runMod.F90) 437
8.1.2 lis_run_paramEstim (Source File: paramEstim_runMod.F90) 438
8.1.3 lis_final_paramEstim (Source File: paramEstim_runMod.F90) 439

9 Retrospective Running Mode 439
9.1 Fortran: Module Interface retrospective_runMod (Source File: retrospective_runMod.F90) . . 439
9.1.1 LIS_init_retrospective (Source File: retrospective_runMod.F90) 440
9.1.2 lis_run_retrospective (Source File: retrospective_runMod.F90) 441
9.1.3 lis_final_retrospective (Source File: retrospective_runMod.F90) 443
9.2 Fortran: Module Interface smootherDA_runMod (Source File: smootherDA_runMod.F90) . . 443
9.2.1 lis_init_smootherDA (Source File: smootherDA_runMod.F90) 444
9.2.2 lis_run_smootherDA (Source File: smootherDA_runMod.F90) 445
9.2.3 lis_final_smootherDA (Source File: smootherDA_runMod.F90) 445
9.3 Fortran: Module Interface LISWRFGridCompMod (Source File: LISWRFGridCompMod.F90) 446
9.4 Fortran: Module Interface LISWRFexport_module (Source File: LISWRFexport_module.F90) 446
9.5 Fortran: Module Interface LISWRFimport_module (Source File: LISWRFimport_module.F90) 446
9.5.1 liswrffinalize_coupled (Source File: liswrffinalize_coupled.F90) 447
9.5.2 liswrfinit_coupled (Source File: liswrfinit_coupled.F90) 447
9.5.3 liswrfrun_coupled (Source File: liswrfrun_coupled.F90) 447

VIII Domain implementations in LIS 449

10 Domain implementations in LIS 450

11 UTM domain 450
11.0.4 readinput_UTM (Source File: readinput_UTM.F90) 450
11.0.5 readinput_hrap (Source File: readinput_hrap.F90) 450

12 Lambert Conformal domain 451
12.0.6 readinput_lambert (Source File: readinput_lambert.F90) 451

13 Lat/Lon domain 452
13.0.7 readinput_latlon (Source File: readinput_latlon.F90) 452

14 Mercator domain **453**
14.0.8 readinput_merc (Source File: readinput_merc.F90) 453

15 Polar stereographic domain **454**
15.0.9 readinput_polar (Source File: readinput_polar.F90) 454

IX Land Surface Parameters in LIS 455

16 Land Surface Parameters in LIS **456**

17 Greenness Fraction Data **456**
17.0.10 read_NESDISgfrac (Source File: read_NESDISgfrac.F90) 456
17.0.11 read_SPORTgfrac (Source File: read_SPORTgfrac.F90) 457
17.0.12 setup_SPORTgfrac (Source File: setup_SPORTgfrac.F90) 457

18 Leaf Area Index Data **458**
18.0.13 read_MODISlai (Source File: read_MODISlai.F90) 458
18.0.14 read_MODISsai (Source File: read_MODISsai.F90) 459
18.0.15 setup_MODISlai (Source File: setup_MODISlai.F90) 460

X Meteorological forcing analyses in LIS 461

19 TRMM 3B42RT **462**
19.1 Fortran: Module Interface TRMM3B42RT_forcingMod (Source File: TRMM3B42RT_forcingMod.F90) 462
19.1.1 init_TRMM3B42RT (Source File: TRMM3B42RT_forcingMod.F90) 463
19.1.2 finalize_TRMM3B42RT (Source File: finalize_TRMM3B42RT.F90) 464
19.1.3 get_TRMM3B42RT (Source File: get_TRMM3B42RT.F90) 464
19.1.4 TRMM3B42RTfile (Source File: get_TRMM3B42RT.F90) 465
19.1.5 interp_TRMM3B42RT (Source File: interp_TRMM3B42RT.F90) 465
19.1.6 read_TRMM3B42RT (Source File: read_TRMM3B42RT.F90) 466
19.1.7 readcrd_TRMM3B42RT (Source File: readcrd_TRMM3B42RT.F90) 467
19.1.8 timeinterp_TRMM3B42RT (Source File: timeinterp_TRMM3B42RT.F90) 468

20 TRMM 3B42V6 **468**
20.1 Fortran: Module Interface TRMM3B42V6_forcingMod (Source File: TRMM3B42V6_forcingMod.F90) 468
20.1.1 init_TRMM3B42V6 (Source File: TRMM3B42V6_forcingMod.F90) 469
20.1.2 finalize_TRMM3B42V6 (Source File: finalize_TRMM3B42V6.F90) 470
20.1.3 get_TRMM3B42V6 (Source File: get_TRMM3B42V6.F90) 470
20.1.4 TRMM3B42V6file (Source File: get_TRMM3B42V6.F90) 471
20.1.5 interp_TRMM3B42V6.F90 (Source File: interp_TRMM3B42V6.F90) 472
20.1.6 read_TRMM3B42V6.F90 (Source File: read_TRMM3B42V6.F90) 473
20.1.7 readcrd_TRMM3B42V6 (Source File: readcrd_TRMM3B42V6.F90) 474
20.1.8 timeinterp_TRMM3B42V6 (Source File: timeinterp_TRMM3B42V6.F90) 474

21 TRMM 3B42V7 **475**
21.1 Fortran: Module Interface TRMM3B42V7_forcingMod (Source File: TRMM3B42V7_forcingMod.F90) 475
21.1.1 init_TRMM3B42V7 (Source File: TRMM3B42V7_forcingMod.F90) 476
21.1.2 finalize_TRMM3B42V7 (Source File: finalize_TRMM3B42V7.F90) 477
21.1.3 get_TRMM3B42V7 (Source File: get_TRMM3B42V7.F90) 477
21.1.4 TRMM3B42V7file (Source File: get_TRMM3B42V7.F90) 478

21.1.5 interp_TRMM3B42V7.F90 (Source File: interp_TRMM3B42V7.F90) 478
21.1.6 read_TRMM3B42V7.F90 (Source File: read_TRMM3B42V7.F90) 479
21.1.7 readcrd_TRMM3B42V7 (Source File: readcrd_TRMM3B42V7.F90) 480
21.1.8 timeinterp_TRMM3B42V7 (Source File: timeinterp_TRMM3B42V7.F90) 481

22 Noah3.1 Bondville forcing 481
22.1 Fortran: Module Interface Bondville_forcingMod (Source File: Bondville_forcingMod.F90) . . 481
22.1.1 init_Bondville (Source File: Bondville_forcingMod.F90) 482
22.1.2 finalize_Bondville (Source File: finalize_Bondville.F90) 483
22.1.3 get_Bondville (Source File: get_Bondville.F90) 483
22.1.4 read_Bondville (Source File: read_Bondville.F90) 484
22.1.5 readcrd_Bondville (Source File: readcrd_Bondville.F90) 484
22.1.6 timeinterp_Bondville (Source File: timeinterp_Bondville.F90) 485
22.2 Fortran: Module Interface PALSmetdata_forcingMod (Source File: PALSmetdata_forcingMod.F90) 485
22.2.1 init_PALSmetdata (Source File: PALSmetdata_forcingMod.F90) 486
22.3 Fortran: Module Interface finalize_PALSmetdata (Source File: finalize_PALSmetdata.F90) . . 486
22.3.1 get_PALSmetdata (Source File: get_PALSmetdata.F90) 487
22.3.2 PALSmetdatafile (Source File: get_PALSmetdata.F90) 488
22.3.3 read_PALSvar (Source File: get_PALSmetdata.F90) 488
22.3.4 readcrd_PALSmetdata (Source File: readcrd_PALSmetdata.F90) 489
22.4 Fortran: Module Interface reset_PALSmetdata (Source File: reset_PALSmetdata.F90) 489
22.4.1 timeinterp_PALSmetdata (Source File: timeinterp_PALSmetdata.F90) 490
22.4.2 finalize_pildas (Source File: finalize_pildas.F90) 490
22.4.3 get_pildas (Source File: get_pildas.F90) 491
22.4.4 pildasfile (Source File: get_pildas.F90) 492
22.4.5 interp_pildas (Source File: interp_pildas.F90) 492
22.5 Fortran: Module Interface pildas_forcingMod (Source File: pildas_forcingMod.F90) 493
22.5.1 init_pildas (Source File: pildas_forcingMod.F90) 494
22.5.2 read_pildas (Source File: read_pildas.F90) 494
22.5.3 readcrd_pildas (Source File: readcrd_pildas.F90) 495
22.5.4 timeinterp_pildas (Source File: timeinterp_pildas.F90) 496

23 RFE2Daily 496
23.1 Fortran: Module Interface RFE2Daily_forcingMod (Source File: RFE2Daily_forcingMod.F90) 497
23.1.1 init_RFE2Daily (Source File: RFE2Daily_forcingMod.F90) 498
23.2 Fortran: Module Interface finalize_RFE2Daily (Source File: finalize_RFE2Daily.F90) 499
23.2.1 get_RFE2Daily (Source File: get_RFE2Daily.F90) 499
23.2.2 RFE2Dailyfile (Source File: get_RFE2Daily.F90) 500
23.2.3 readcrd_RFE2Daily (Source File: readcrd_RFE2Daily.F90) 501
23.2.4 readprecip_RFE2Daily (Source File: readprecip_RFE2Daily.F90) 501
23.2.5 reproject_RFE2Daily (Source File: reproject_RFE2Daily.F90) 502
23.2.6 timeinterp_RFE2Daily (Source File: timeinterp_RFE2Daily.F90) 503

24 RFE2gdas 504
24.1 Fortran: Module Interface RFE2gdas_forcingMod (Source File: RFE2gdas_forcingMod.F90) . 504
24.1.1 init_RFE2gdas (Source File: RFE2gdas_forcingMod.F90) 506
24.2 Fortran: Module Interface finalize_RFE2gdas (Source File: finalize_RFE2gdas.F90) 506
24.2.1 get_RFE2gdas (Source File: get_RFE2gdas.F90) 507
24.2.2 RFE2gdasfile (Source File: get_RFE2gdas.F90) 508
24.2.3 readRFE2gdascrd (Source File: readRFE2gdascrd.F90) 508
24.2.4 readprecip_RFE2gdas (Source File: readprecip_RFE2gdas.F90) 509

24.2.5 reproject_RFE2gdas (Source File: reproject_RFE2gdas.F90) 510
24.2.6 timeinterp_RFE2gdas (Source File: timeinterp_RFE2gdas.F90) 511

25 WRF output **512**
25.1 Fortran: Module Interface WRFout_forcingMod (Source File: WRFout_forcingMod.F90) . . . 512
25.1.1 init_WRFout (Source File: WRFout_forcingMod.F90) 512
25.2 Fortran: Module Interface finalize_WRFout (Source File: finalize_WRFout.F90) 513
25.2.1 get_WRFout (Source File: get_WRFout.F90) 513
25.2.2 WRFoutfile (Source File: get_WRFout.F90) 514
25.2.3 read_WRFout (Source File: read_WRFout.F90) 515
25.2.4 readcrd_WRFout (Source File: readcrd_WRFout.F90) 515
25.3 Fortran: Module Interface reset_WRFout (Source File: reset_WRFout.F90) 516
25.3.1 timeinterp_WRFout (Source File: timeinterp_WRFout.F90) 516

26 AGRMET radiation **517**
26.1 Fortran: Module Interface agrrad_forcingMod (Source File: agrrad_forcingMod.F90) 517
26.1.1 init_agrrad (Source File: agrrad_forcingMod.F90) 518
26.1.2 finalize_agrrad (Source File: finalize_agrrad.F90) 519
26.1.3 get_agrrad (Source File: get_agrrad.F90) 519
26.1.4 read_agrrad (Source File: read_agrrad.F90) 520
26.1.5 agrrad_filename (Source File: read_agrrad.F90) 521
26.1.6 agrrad_fillgaps (Source File: read_agrrad.F90) 522
26.1.7 retrieve_agrrad_variables (Source File: read_agrrad.F90) 522
26.1.8 readcrd_agrrad (Source File: readcrd_agrrad.F90) 523
26.1.9 timeinterp_agrrad (Source File: timeinterp_agrrad.F90) 524

27 AGRMET polar stereographic radiation **524**
27.0.10 agrradps_filename_sw (Source File: agrradps_filenames.F90) 524
27.0.11 agrradps_filename_cloud (Source File: agrradps_filenames.F90) 525
27.1 Fortran: Module Interface agrradps_forcingMod (Source File: agrradps_forcingMod.F90) . . . 526
27.1.1 init_agrradps (Source File: agrradps_forcingMod.F90) 527
27.1.2 finalize_agrradps (Source File: finalize_agrradps.F90) 528
27.1.3 get_agrradps (Source File: get_agrradps.F90) 528
27.1.4 interp_agrradpsvar (Source File: interp_agrradpsvar.F90) 529
27.1.5 read_agrradps (Source File: read_agrradps.F90) 530
27.1.6 fillgaps_agrradps (Source File: read_agrradps.F90) 531
27.1.7 agrmet2latlon1 (Source File: read_agrradps.F90) 531
27.1.8 readcrd_agrradps (Source File: readcrd_agrradps.F90) 532
27.1.9 timeinterp_agrradps (Source File: timeinterp_agrradps.F90) 532

28 CEOP forcing **533**
28.1 Fortran: Module Interface ceop_forcingMod (Source File: ceop_forcingMod.F90) 533
28.1.1 init_CEOP (Source File: ceop_forcingMod.F90) 533
28.1.2 finalize_ceop (Source File: finalize_ceop.F90) 534
28.1.3 get_ceop (Source File: get_ceop.F90) 534
28.1.4 read_ceop (Source File: read_ceop.F90) 535
28.1.5 readbon (Source File: readbon.F90) 536
28.1.6 readcrd_ceop (Source File: readcrd_ceop.F90) 537
28.1.7 readfpk (Source File: readfpk.F90) 538
28.1.8 readsgp (Source File: readsgp.F90) 539
28.1.9 timeinterp_ceop (Source File: timeinterp_ceop.F90) 540

29 CMAP **540**
29.1 Fortran: Module Interface cmap_forcingMod (Source File: cmap_forcingMod.F90) 541
29.1.1 init_CMAP (Source File: cmap_forcingMod.F90) 542
29.1.2 finalize_cmap (Source File: finalize_cmap.F90) 542
29.1.3 get_cmap (Source File: get_cmap.F90) 543
29.1.4 cmapfile (Source File: get_cmap.F90) 543
29.1.5 cmapfile_old (Source File: get_cmap.F90) 544
29.1.6 interp_cmap (Source File: interp_cmap.F90) 544
29.1.7 read_cmap (Source File: read_cmap.F90) 545
29.1.8 readcrd_cmap (Source File: readcrd_cmap.F90) 546
29.1.9 timeinterp_cmap (Source File: timeinterp_cmap.F90) 546

30 CMORPH **547**
30.1 Fortran: Module Interface cmorph_forcingMod (Source File: cmorph_forcingMod.F90) 547
30.1.1 init_cmorph (Source File: cmorph_forcingMod.F90) 548
30.1.2 finalize_cmorph (Source File: finalize_cmorph.F90) 549
30.1.3 get_cmorph (Source File: get_cmorph.F90) 549
30.1.4 cmorphfile (Source File: get_cmorph.F90) 550
30.1.5 interp_cmorph (Source File: interp_cmorph.F90) 551
30.1.6 read_cmorph (Source File: read_cmorph.F90) 551
30.1.7 readcrd_cmorph (Source File: readcrd_cmorph.F90) 552
30.1.8 timeinterp_cmorph (Source File: timeinterp_cmorph.F90) 553
30.2 Fortran: Module Interface coop_forcingMod (Source File: coop_forcingMod.F90) 553
30.2.1 init_COOP (Source File: coop_forcingMod.F90) 554
30.2.2 finalize_coop (Source File: finalize_coop.F90) 554
30.2.3 get_coop (Source File: get_coop.F90) 555
30.2.4 read_coop (Source File: read_coop.F90) 555
30.2.5 readcrd_coop (Source File: readcrd_coop.F90) 556
30.2.6 timeinterp_coop (Source File: timeinterp_coop.F90) 557

31 ECMWF **557**
31.1 Fortran: Module Interface ecmwf_forcingMod (Source File: ecmwf_forcingMod.F90) 557
31.1.1 init_ecmwf (Source File: ecmwf_forcingMod.F90) 559
31.1.2 finalize_ecmwf (Source File: finalize_ecmwf.F90) 559
31.1.3 get_ecmwf (Source File: get_ecmwf.F90) 560
31.1.4 create_ecmwf_filename (Source File: get_ecmwf.F90) 560
31.1.5 read_ecmwf (Source File: read_ecmwf.F90) 561
31.1.6 retrieve_inst_ecmwfvars (Source File: read_ecmwf.F90) 562
31.1.7 retrieve_accum_ecmwfvars (Source File: read_ecmwf.F90) 563
31.1.8 interp_ecmwf (Source File: read_ecmwf.F90) 563
31.1.9 read_ecmwf_elev (Source File: read_ecmwf_elev.F90) 564
31.1.10 readcrd_ecmwf (Source File: readcrd_ecmwf.F90) 565
31.1.11 timeinterp_ecmwf (Source File: timeinterp_ecmwf.F90) 565

32 ECMWF Reanalysis **566**
32.1 Fortran: Module Interface ecmwfreanal_forcingMod (Source File: ecmwfreanal_forcingMod.F90) 566
32.1.1 init_ECMWFREANAL (Source File: ecmwfreanal_forcingMod.F90) 567
32.1.2 finalize_ecmwfreanal (Source File: finalize_ecmwfreanal.F90) 568
32.1.3 geogfill2_ecmwfreanal (Source File: geogfill2_ecmwfreanal.F90) 569
32.1.4 get_ecmwfreanal (Source File: get_ecmwfreanal.F90) 570
32.1.5 read_ecmwfreanal (Source File: read_ecmwfreanal.F90) 570

32.1.6 ecmwfreanalgrid_2_lisgrid (Source File: read_ecmwfreanal.F90) 572
32.1.7 fillgaps (Source File: read_ecmwfreanal.F90) 572
32.1.8 read_ecmwfreanal_elev (Source File: read_ecmwfreanal_elev.F90) 573
32.1.9 readcrd_ecmwfreanal (Source File: readcrd_ecmwfreanal.F90) 573
32.1.10 readmask_ecmwfreanal (Source File: readmask_ecmwfreanal.F90) 573
32.1.11 timeinterp_ecmwfreanal (Source File: timeinterp_ecmwfreanal.F90) 574

33 GDAS **575**
33.0.12 create_gdasf9_filename (Source File: create_gdasf9_filename.F90) 575
33.0.13 create_gdasfilename (Source File: create_gdasfilename.F90) 576
33.1 Fortran: Module Interface finalize_gdas (Source File: finalize_gdas.F90) 578
33.2 Fortran: Module Interface gdas_forcingMod (Source File: gdas_forcingMod.F90) 578
33.2.1 init_GDAS (Source File: gdas_forcingMod.F90) 580
33.2.2 get_gdas (Source File: get_gdas.F90) 580
33.2.3 read_gdas (Source File: read_gdas.F90) 582
33.2.4 retrieve_gdas_variables (Source File: read_gdas.F90) 583
33.2.5 interp_gdas (Source File: read_gdas.F90) 583
33.2.6 read_gdas_elev (Source File: read_gdas_elev.F90) 584
33.2.7 readcrd_gdas (Source File: readcrd_gdas.F90) 585
33.3 Fortran: Module Interface reset_gdas (Source File: reset_gdas.F90) 585
33.3.1 timeinterp_gdas (Source File: timeinterp_gdas.F90) 586

34 GDAS LSWG forcing **587**
34.1 Fortran: Module Interface finalize_gdasLSWG (Source File: finalize_gdasLSWG.F90) 587
34.2 Fortran: Module Interface gdasLSWG_forcingMod (Source File: gdasLSWG_forcingMod.F90) 587
34.2.1 init_gdasLSWG (Source File: gdasLSWG_forcingMod.F90) 588
34.2.2 get_gdasLSWG (Source File: get_gdasLSWG.F90) 588
34.2.3 retgdasLSWG (Source File: read_gdasLSWG.F90) 589
34.2.4 readcrd_gdasLSWG (Source File: readcrd_gdasLSWG.F90) 589
34.2.5 timeinterp_gdasLSWG (Source File: timeinterp_gdasLSWG.F90) 590

35 GEOS **590**
35.0.6 finalize_geos (Source File: finalize_geos.F90) 591
35.1 Fortran: Module Interface geos_forcingMod (Source File: geos_forcingMod.F90) 591
35.1.1 init_geos (Source File: geos_forcingMod.F90) 592
35.1.2 get_geos (Source File: get_geos.F90) 593
35.1.3 geosfile (Source File: get_geos.F90) 594
35.1.4 read_geos (Source File: read_geos.F90) 594
35.1.5 readcrd_geos (Source File: readcrd_geos.F90) 596
35.1.6 timeinterp_geos (Source File: timeinterp_geos.F90) 596
35.2 Fortran: Module Interface finalize_geos5fcst (Source File: finalize_geos5fcst.F90) 597
35.3 Fortran: Module Interface geos5fcst_forcingMod (Source File: geos5fcst_forcingMod.F90) . . . 597
35.3.1 init_GEOS5FCST (Source File: geos5fcst_forcingMod.F90) 598
35.3.2 get_geos5fcst (Source File: get_geos5fcst.F90) 598
35.3.3 geos5fcstfile (Source File: get_geos5fcst.F90) 599
35.3.4 interp_geos5fcst (Source File: interp_geos5fcst.F90) 600
35.3.5 read_geos5fcst (Source File: read_geos5fcst.F90) 600
35.3.6 readcrd_geos5fcst (Source File: readcrd_geos5fcst.F90) 601
35.4 Fortran: Module Interface reset_geos5fcst (Source File: reset_geos5fcst.F90) 602
35.4.1 timeinterp_geos5fcst (Source File: timeinterp_geos5fcst.F90) 602

36 GFS **603**
36.0.2 create_gfs_f0backup_filename (Source File: create_gfs_f0backup_filename.F90) 603
36.0.3 create_gfsfilename (Source File: create_gfsfilename.F90) 604
36.1 Fortran: Module Interface finalize_gfs (Source File: finalize_gfs.F90) 605
36.1.1 get_gfs (Source File: get_gfs.F90) 605
36.2 Fortran: Module Interface gfs_forcingMod (Source File: gfs_forcingMod.F90) 606
36.2.1 init_GFS (Source File: gfs_forcingMod.F90) 608
36.2.2 read_gfs (Source File: read_gfs.F90) 608
36.2.3 retrieve_gfs_variables (Source File: read_gfs.F90) 609
36.2.4 interp_gfs (Source File: read_gfs.F90) 610
36.2.5 read_gfs_elev (Source File: read_gfs_elev.F90) 611
36.2.6 readcrd_gfs (Source File: readcrd_gfs.F90) 612
36.2.7 timeinterp_gfs (Source File: timeinterp_gfs.F90) 612

37 GLDAS **613**
37.0.8 fillgaps_gldas (Source File: fillgaps_gldas.F90) 613
37.1 Fortran: Module Interface finalize_gldas (Source File: finalize_gldas.F90) 614
37.1.1 get_gldas (Source File: get_gldas.F90) 614
37.1.2 gldasfile (Source File: get_gldas.F90) 615
37.2 Fortran: Module Interface gldas_forcingMod (Source File: gldas_forcingMod.F90) 615
37.2.1 init_gldas (Source File: gldas_forcingMod.F90) 616
37.2.2 read_gldas (Source File: read_gldas.F90) 617
37.2.3 interp_gldas (Source File: read_gldas.F90) 618
37.2.4 readcrd_gldas (Source File: readcrd_gldas.F90) 619
37.2.5 timeinterp_gldas (Source File: timeinterp_gldas.F90) 619

38 GSWP1 **620**
38.1 Fortran: Module Interface finalize_gswp1 (Source File: finalize_gswp1.F90) 620
38.1.1 get_gswp1 (Source File: get_gswp1.F90) 621
38.1.2 gswp1file (Source File: get_gswp1.F90) 621
38.2 Fortran: Module Interface gswp1_forcingMod (Source File: gswp1_forcingMod.F90) 622
38.2.1 init_gswp1 (Source File: gswp1_forcingMod.F90) 622
38.2.2 read_gswp1 (Source File: read_gswp1.F90) 623
38.3 Core Functions of read_gswp1 623
38.3.1 readcrd_gswp1 (Source File: readcrd_gswp1.F90) 624
38.3.2 timeinterp_gswp1 (Source File: timeinterp_gswp1.F90) 624

39 GSWP **625**
39.0.3 fillgaps_gswp2 (Source File: fillgaps_gswp2.F90) 625
39.1 Fortran: Module Interface finalize_gswp2 (Source File: finalize_gswp2.F90) 625
39.1.1 finterp_gswp2 (Source File: finterp_gswp2.F90) 626
39.1.2 get_gswp2 (Source File: get_gswp2.F90) 627
39.2 Fortran: Module Interface gswp2_forcingMod (Source File: gswp2_forcingMod.F90) 627
39.2.1 init_GSWP2 (Source File: gswp2_forcingMod.F90) 628
39.2.2 read_gswp2 (Source File: read_gswp2.F90) 629
39.2.3 gswp2_process_forcing (Source File: read_gswp2.F90) 630
39.2.4 interp_gswp2 (Source File: read_gswp2.F90) 631
39.2.5 gswp2_expand_to_grid (Source File: read_gswp2.F90) 632
39.2.6 gswp2_2d_to_1d (Source File: read_gswp2.F90) 632
39.2.7 gswp2_mask (Source File: read_gswp2.F90) 633
39.2.8 readcrd_gswp2 (Source File: readcrd_gswp2.F90) 633

39.2.9 timeinterp_gswp2 (Source File: timeinterp_gswp2.F90) 633

40 MERRA-Land **634**
40.0.10 finalize_merraland (Source File: finalize_merraland.F90) 634
40.0.11 get_merraland (Source File: get_merraland.F90) 635
40.0.12 merralandfiles (Source File: get_merraland.F90) 635
40.1 Fortran: Module Interface merraland_forcingMod (Source File: merraland_forcingMod.F90) . 636
40.1.1 init_MERRALAND (Source File: merraland_forcingMod.F90) 638
40.1.2 read_merraland (Source File: read_merraland.F90) 638
40.1.3 interp_merraland_var (Source File: read_merraland.F90) 640
40.1.4 readcrd_merraland (Source File: readcrd_merraland.F90) 640
40.2 Fortran: Module Interface reset_merraland (Source File: reset_merraland.F90) 641
40.2.1 timeinterp_merraland (Source File: timeinterp_merraland.F90) 641

41 NAM242 **642**
41.0.2 create_nam242f9_filename (Source File: create_nam242f9_filename.F90) 642
41.0.3 create_nam242filename (Source File: create_nam242filename.F90) 643
41.1 Fortran: Module Interface finalize_nam242 (Source File: finalize_nam242.F90) 644
41.1.1 get_nam242 (Source File: get_nam242.F90) 644
41.2 Fortran: Module Interface nam242_forcingMod (Source File: nam242_forcingMod.F90) 645
41.2.1 init_nam242 (Source File: nam242_forcingMod.F90) 646
41.2.2 read_nam242 (Source File: read_nam242.F90) 647
41.2.3 interp_nam242 (Source File: read_nam242.F90) 648
41.2.4 read_nam242_elev (Source File: read_nam242_elev.F90) 649
41.2.5 readcrd_nam242 (Source File: readcrd_nam242.F90) 650
41.3 Fortran: Module Interface reset_nam242 (Source File: reset_nam242.F90) 650
41.3.1 timeinterp_nam242 (Source File: timeinterp_nam242.F90) 651

42 NARR forcing **652**
42.1 Fortran: Module Interface finalize_narr (Source File: finalize_narr.F90) 652
42.1.1 get_narr (Source File: get_narr.F90) 652
42.1.2 interp_narrfield (Source File: interp_narrfield.F90) 653
42.2 Fortran: Module Interface narr_forcingMod (Source File: narr_forcingMod.F90) 653
42.2.1 init_narr (Source File: narr_forcingMod.F90) 654
42.2.2 read_narr (Source File: read_narr.F90) 654
42.2.3 readcrd_narr (Source File: readcrd_narr.F90) 655
42.2.4 timeinterp_narr (Source File: timeinterp_narr.F90) 655

43 NLDAS1 **656**
43.0.5 apply_conusmask (Source File: apply_conusmask.F90) 656
43.1 Fortran: Module Interface finalize_nldas1 (Source File: finalize_nldas1.F90) 656
43.1.1 get_nldas1 (Source File: get_nldas1.F90) 657
43.1.2 get_nldas1_filename (Source File: get_nldas1_filename.F90) 658
43.1.3 interp_nldas1 (Source File: interp_nldas1.F90) 658
43.1.4 nldas1_ec_removal (Source File: nldas1_ec_removal.F90) 659
43.2 Fortran: Module Interface nldas1_forcingMod (Source File: nldas1_forcingMod.F90) 660
43.2.1 init_NLDAS1 (Source File: nldas1_forcingMod.F90) 662
43.2.2 orig_elevdiff (Source File: orig_elevdiff.F90) 663
43.2.3 read_edas_elev (Source File: read_edas_elev.F90) 663
43.2.4 read_nldas1 (Source File: read_nldas1.F90) 664
43.2.5 readcrd_nldas1 (Source File: readcrd_nldas1.F90) 665
43.2.6 reset_nldas1 (Source File: reset_nldas1.F90) 666

43.2.7 timeinterp_nldas1 (Source File: timeinterp_nldas1.F90) 666

44 NLDAS2 **667**
44.1 Fortran: Module Interface finalize_nldas2 (Source File: finalize_nldas2.F90) 667
44.1.1 gesdisc_nldas2filea (Source File: get_gesdisc_filenames.F90) 668
44.1.2 gesdisc_nldas2fileb (Source File: get_gesdisc_filenames.F90) 668
44.1.3 ncep_nldas2filea (Source File: get_ncep_filenames.F90) 669
44.1.4 ncep_nldas2fileb (Source File: get_ncep_filenames.F90) 670
44.1.5 get_nldas2 (Source File: get_nldas2.F90) 671
44.1.6 nldas2_ec_removal (Source File: nldas2_ec_removal.F90) 672
44.2 Fortran: Module Interface nldas2_forcingMod (Source File: nldas2_forcingMod.F90) 672
44.2.1 init_NLDAS2 (Source File: nldas2_forcingMod.F90) 674
44.2.2 read_nldas2_elev (Source File: read_nldas2_elev.F90) 674
44.2.3 read_nldas2a (Source File: read_nldas2a.F90) 675
44.2.4 interp_nldas2 (Source File: read_nldas2a.F90) 677
44.2.5 read_nldas2b (Source File: read_nldas2b.F90) 678
44.2.6 read_orig_nldas2_elevdiff (Source File: read_orig_nldas2_elevdiff.F90) 679
44.2.7 readcrd_nldas2 (Source File: readcrd_nldas2.F90) 680
44.3 Fortran: Module Interface reset_nldas2 (Source File: reset_nldas2.F90) 680
44.3.1 timeinterp_nldas2 (Source File: timeinterp_nldas2.F90) 680

45 PET_USGS **681**
45.0.2 finalize_petusgs (Source File: finalize_petusgs.F90) 681
45.0.3 get_petusgs (Source File: get_petusgs.F90) 682
45.0.4 petusgsfile (Source File: get_petusgs.F90) 683
45.0.5 interp_petusgs (Source File: interp_petusgs.F90) 683
45.1 Fortran: Module Interface petusgs_forcingMod (Source File: petusgs_forcingMod.F90) 684
45.1.1 init_petusgs (Source File: petusgs_forcingMod.F90) 685
45.1.2 read_petusgs (Source File: read_petusgs.F90) 686
45.1.3 readpetusgscrd (Source File: readpetusgscrd.F90) 687
45.1.4 timeinterp_petusgs (Source File: timeinterp_petusgs.F90) 687

46 Princeton **688**
46.0.5 finalize_princeton (Source File: finalize_princeton.F90) 688
46.0.6 get_princeton (Source File: get_princeton.F90) 688
46.1 Fortran: Module Interface princeton_forcingMod (Source File: princeton_forcingMod.F90) . . 689
46.1.1 init_PRINCETON (Source File: princeton_forcingMod.F90) 690
46.1.2 read_princeton (Source File: read_princeton.F90) 691
46.1.3 princetongrid_2_lisgrid (Source File: read_princeton.F90) 692
46.1.4 read_princeton_elev (Source File: read_princeton_elev.F90) 693
46.1.5 readcrd_princeton (Source File: readcrd_princeton.F90) 693
46.2 Fortran: Module Interface reset_princeton (Source File: reset_princeton.F90) 694
46.2.1 timeinterp_princeton (Source File: timeinterp_princeton.F90) 694

47 Rhone-AGG **695**
47.0.2 finalize_rhoneAGG (Source File: finalize_rhoneAGG.F90) 695
47.0.3 get_rhoneAGG (Source File: get_rhoneAGG.F90) 696
47.0.4 rhoneAGGfile (Source File: get_rhoneAGG.F90) 696
47.0.5 readrhoneAGG (Source File: read_rhoneAGG.F90) 697
47.1 Core Functions of readrhoneAGG 697
47.1.1 readcrd_rhoneAGG (Source File: readcrd_rhoneAGG.F90) 697
47.2 Fortran: Module Interface rhoneAGG_forcingMod (Source File: rhoneAGG_forcingMod.F90) 698

47.2.1 init_RHONEAGG (Source File: rhoneAGG_forcingMod.F90) 698
47.2.2 timeinterp_rhoneAGG (Source File: timeinterp_rhoneAGG.F90) 698
47.3 Core Functions of time_interp_rhoneAGG 699

48 SCAN forcing **699**
48.0.1 finalize_scan (Source File: finalize_scan.F90) 699
48.0.2 get_scan (Source File: get_scan.F90) 700
48.0.3 read_scan (Source File: read_scan.F90) 700
48.0.4 readcrd_scan (Source File: readcrd_scan.F90) 701
48.1 Fortran: Module Interface scan_forcingMod (Source File: scan_forcingMod.F90) 701
48.1.1 init_SCAN (Source File: scan_forcingMod.F90) 702
48.1.2 timeinterp_scan (Source File: timeinterp_scan.F90) 702
48.1.3 finalize_snotel (Source File: finalize_snotel.F90) 703
48.1.4 get_snotel (Source File: get_snotel.F90) 703
48.1.5 read_snotel (Source File: read_snotel.F90) 704
48.1.6 readcrd_snotel (Source File: readcrd_snotel.F90) 705
48.2 Fortran: Module Interface snotel_forcingMod (Source File: snotel_forcingMod.F90) 705
48.2.1 init_SNOTEL (Source File: snotel_forcingMod.F90) 706
48.2.2 timeinterp_snotel (Source File: timeinterp_snotel.F90) 706

49 STAGEII **707**
49.1 Fortran: Module Interface finalize_stg2 (Source File: finalize_stg2.F90) 707
49.1.1 get_stg2 (Source File: get_stg2.F90) 707
49.1.2 interp_stg2 (Source File: interp_stg2.F90) 708
49.1.3 read_stg2 (Source File: read_stg2.F90) 709
49.1.4 readcrd_stg2 (Source File: readcrd_stg2.F90) 710
49.2 Fortran: Module Interface stg2_forcingMod (Source File: stg2_forcingMod.F90) 711
49.2.1 init_STG2 (Source File: stg2_forcingMod.F90) 712
49.2.2 stg2file (Source File: stg2file.F90) 712
49.2.3 timeinterp_stg2 (Source File: timeinterp_stg2.F90) 713

50 STAGEIV **713**
50.1 Fortran: Module Interface finalize_stg4 (Source File: finalize_stg4.F90) 713
50.1.1 get_stg4 (Source File: get_stg4.F90) 714
50.1.2 interp_stg4 (Source File: interp_stg4.F90) 715
50.1.3 read_stg4 (Source File: read_stg4.F90) 716
50.1.4 readcrd_stg4 (Source File: readcrd_stg4.F90) 717
50.2 Fortran: Module Interface reset_stg4 (Source File: reset_stg4.F90) 717
50.3 Fortran: Module Interface stg4_forcingMod (Source File: stg4_forcingMod.F90) 718
50.3.1 init_STG4 (Source File: stg4_forcingMod.F90) 719
50.3.2 stg4file (Source File: stg4file.F90) 719
50.3.3 timeinterp_stg4 (Source File: timeinterp_stg4.F90) 720

51 Template **720**
51.1 Fortran: Module Interface finalize_metForcTemplate (Source File: finalize_metForcTemplate.F90) 720
51.1.1 get_metForcTemplate (Source File: get_metForcTemplate.F90) 721
51.2 Fortran: Module Interface metForcTemplate_forcingMod (Source File: metForcTemplate_forcingMod.F90) 721
51.2.1 init_metForcTemplate (Source File: metForcTemplate_forcingMod.F90) 722
51.3 Fortran: Module Interface reset_metForcTemplate (Source File: reset_metForcTemplate.F90) 722
51.3.1 timeinterp_metForcTemplate (Source File: timeinterp_metForctemplate.F90) 722

52 VIC-processed Forcing **723**
52.0.2 getvicforcing (Source File: getvicforcing.F90) 723
52.0.3 get_vicforcing_filename (Source File: getvicforcing.F90) 724
52.0.4 readviccrd (Source File: readviccrd.F90) 724
52.0.5 time_interp_vicforcing (Source File: time_interp_vicforcing.F90) 725
52.0.6 vic411_read_gridded_forcing_data (Source File: vic411_read_gridded_forcing_data.F90) . 725
52.0.7 interp_vic (Source File: vic411_read_gridded_forcing_data.F90) 726
52.1 Fortran: Module Interface vic_forcingMod (Source File: vic_forcingMod.F90) 727
52.1.1 defineNativevicforcing (Source File: vic_forcingMod.F90) 728
52.2 Fortran: Module Interface vicforcing_finalize (Source File: vicforcing_finalize.F90) 728
52.3 Fortran: Module Interface vicforcing_reset (Source File: vicforcing_reset.F90) 728

53 VIC-processed Forcing **729**
53.0.1 getvicforcing (Source File: getvicforcing.F90) 729
53.0.2 get_vicforcing_filename (Source File: getvicforcing.F90) 730
53.0.3 readviccrd (Source File: readviccrd.F90) 730
53.0.4 time_interp_vicforcing (Source File: time_interp_vicforcing.F90) 731
53.0.5 vic412_read_gridded_forcing_data (Source File: vic412_read_gridded_forcing_data.F90) . 731
53.0.6 interp_vic (Source File: vic412_read_gridded_forcing_data.F90) 732
53.1 Fortran: Module Interface vic_forcingMod (Source File: vic_forcingMod.F90) 733
53.1.1 defineNativevicforcing (Source File: vic_forcingMod.F90) 734
53.2 Fortran: Module Interface vicforcing_finalize (Source File: vicforcing_finalize.F90) 734
53.3 Fortran: Module Interface vicforcing_reset (Source File: vicforcing_reset.F90) 734

54 rdhm356 forcing **734**
54.1 Fortran: Module Interface LIS_XMRG_Reader (Source File: LIS_XMRG_Reader.F90) 735
54.1.1 xmrg_read_header (Source File: LIS_XMRG_Reader.F90) 735
54.1.2 xmrg_read_data (Source File: LIS_XMRG_Reader.F90) 736
54.1.3 xmrg_hrap_xy (Source File: LIS_XMRG_Reader.F90) 737
54.1.4 xmrg_grid_latlon (Source File: LIS_XMRG_Reader.F90) 737
54.1.5 latlon_to_hrap (Source File: LIS_XMRG_Reader.F90) 738
54.1.6 hrap_to_latlon (Source File: LIS_XMRG_Reader.F90) 738
54.1.7 rdhm356_temper_file (Source File: rdhm356_file_name.F90) 738
54.1.8 rdhm356_precip_file (Source File: rdhm356_file_name.F90) 739
54.2 Fortran: Module Interface finalize_rdhm356 (Source File: rdhm356_finalize.F90) 739
54.3 Fortran: Module Interface rdhm356_forcingMod (Source File: rdhm356_forcingMod.F90) . . . 740
54.3.1 init_rdhm356 (Source File: rdhm356_forcingMod.F90) 741
54.3.2 getrdhm356 (Source File: rdhm356_get_forcing.F90) 741
54.3.3 interp_rdhm356_precip (Source File: rdhm356_interp_precip.F90) 742
54.3.4 interp_rdhm356_temper (Source File: rdhm356_interp_temper.F90) 744
54.3.5 rdhm356_read_precip (Source File: rdhm356_read_precip.F90) 745
54.3.6 rdhm356_read_temper (Source File: rdhm356_read_temper.F90) 746
54.3.7 readcrd_rdhm356 (Source File: rdhm356_readcrd.F90) 746
54.3.8 timeinterp_rdhm356 (Source File: timeinterp_rdhm356.F90) 747
54.3.9 xmrg_read_data_c (Source File: xmrg_read_data_c.c) 748
54.3.10 xmrg_read_header_c (Source File: xmrg_read_header_c.c) 748
54.3.11 reverse_byte_order (Source File: xmrg_reverse_byte.c) 749
54.3.12 reverse_byte_order_short (Source File: xmrg_reverse_byte.c) 749

XI Data Assimilation in LIS **750**

55 Data Assimilation Algorithms in LIS **751**

56 Direct Insertion **751**
56.1 Fortran: Module Interface directInsertion_Mod (Source File: directInsertion_Mod.F90) 751
56.1.1 di_init (Source File: directInsertion_Mod.F90) 751
56.1.2 di_increments (Source File: directInsertion_Mod.F90) 752
56.1.3 di_update (Source File: directInsertion_Mod.F90) 752
56.1.4 di_diagnostics (Source File: directInsertion_Mod.F90) 753
56.1.5 di_final (Source File: directInsertion_Mod.F90) 753

57 GMAO Ensemble Kalman Filter **753**
57.1 Fortran: Module Interface enkf_Mod (Source File: enkf_Mod.F90) 753
57.1.1 enkf_init (Source File: enkf_Mod.F90) 754
57.1.2 enkf_setup (Source File: enkf_Mod.F90) 754
57.1.3 enkf_increments (Source File: enkf_Mod.F90) 755
57.1.4 enkf_update (Source File: enkf_Mod.F90) 756
57.1.5 enkf_diagnostics (Source File: enkf_Mod.F90) 756
57.1.6 pruneVarname (Source File: enkf_Mod.F90) 757
57.1.7 assemble_obs_cov (Source File: enkf_Mod.F90) 757
57.1.8 enkf_final (Source File: enkf_Mod.F90) 758
57.1.9 getLSMvarnames (Source File: enkf_Mod.F90) 758
57.1.10 getObsPert (Source File: enkf_Mod.F90) 759
57.1.11 generateObservations (Source File: enkf_Mod.F90) 759
57.1.12 generateObsparam (Source File: enkf_Mod.F90) 760
57.1.13 getLSMData (Source File: enkf_Mod.F90) 760
57.2 Fortran: Module Interface enkf_general (Source File: enkf_general.F90) 761
57.2.1 enkf_analysis (Source File: enkf_general.F90) 761
57.2.2 hadamard_for_repr_matrix (Source File: enkf_general.F90) 762
57.2.3 gaspari_cohn (Source File: enkf_general.F90) 762
57.2.4 get_gaspari_cohn (Source File: enkf_general.F90) 763
57.3 Fortran: Module Interface enkf_types (Source File: enkf_types.F90) 763
57.3.1 ludcmp (Source File: my_lu_decomp.F90) 763
57.3.2 lubksb (Source File: my_lu_decomp.F90) 764
57.4 Fortran: Module Interface my_matrix_functions (Source File: my_matrix_functions.F90) . . . 764
57.4.1 (Source File: my_matrix_functions.F90) 764
57.4.2 row_variance (Source File: my_matrix_functions.F90) 764
57.4.3 row_third_moment (Source File: my_matrix_functions.F90) 764
57.4.4 five_number_summary (Source File: my_matrix_functions.F90) 765
57.4.5 adjust_mean (Source File: my_matrix_functions.F90) 765
57.4.6 adjust_std (Source File: my_matrix_functions.F90) 766
57.4.7 matrix_std (Source File: my_matrix_functions.F90) 766
57.4.8 nr_sort (Source File: nr_sort.F90) 766
57.5 Fortran: Module Interface enks_general (Source File: enks_general.F90) 767
57.5.1 enks_analysis (Source File: enks_general.F90) 767
57.5.2 hadamard_for_repr_matrix (Source File: enks_general.F90) 768
57.5.3 gaspari_cohn (Source File: enks_general.F90) 768
57.5.4 get_gaspari_cohn (Source File: enks_general.F90) 769
57.5.5 enks_ludcmp (Source File: enks_lu_decomp.F90) 769
57.5.6 enks_lubksb (Source File: enks_lu_decomp.F90) 769
57.6 Fortran: Module Interface enks_matrix_functions (Source File: enks_matrix_functions.F90) . . 770
57.6.1 (Source File: enks_matrix_functions.F90) 770

57.6.2 row_variance (Source File: enks_matrix_functions.F90) . . . 770
57.6.3 row_third_moment (Source File: enks_matrix_functions.F90) . . . 770
57.6.4 five_number_summary (Source File: enks_matrix_functions.F90) . . . 771
57.6.5 adjust_mean (Source File: enks_matrix_functions.F90) . . . 771
57.6.6 adjust_std (Source File: enks_matrix_functions.F90) . . . 772
57.6.7 matrix_std (Source File: enks_matrix_functions.F90) . . . 772
57.6.8 nr_sort (Source File: enks_matrix_functions.F90) . . . 772
57.7 Fortran: Module Interface enks_types (Source File: enks_types.F90) . . . 772
57.8 Fortran: Module Interface enksgrace_Mod (Source File: enksgrace_Mod.F90) . . . 773
57.8.1 enksgrace_init (Source File: enksgrace_Mod.F90) . . . 773
57.8.2 enksgrace_setup (Source File: enksgrace_Mod.F90) . . . 774
57.8.3 enksgrace_increments (Source File: enksgrace_Mod.F90) . . . 774
57.8.4 enksgrace_update (Source File: enksgrace_Mod.F90) . . . 775
57.8.5 enksgrace_diagnostics (Source File: enksgrace_Mod.F90) . . . 776
57.8.6 pruneVarname (Source File: enksgrace_Mod.F90) . . . 777
57.8.7 assemble_obs_cov (Source File: enksgrace_Mod.F90) . . . 777
57.8.8 enksgrace_final (Source File: enksgrace_Mod.F90) . . . 778
57.8.9 getLSMvarnames (Source File: enksgrace_Mod.F90) . . . 778
57.8.10 getObsPert (Source File: enksgrace_Mod.F90) . . . 778
57.8.11 generateObservations (Source File: enksgrace_Mod.F90) . . . 779
57.8.12 generateObsparam (Source File: enksgrace_Mod.F90) . . . 779
57.8.13 getLSMData (Source File: enksgrace_Mod.F90) . . . 780

58 Perturbation Algorithms in LIS **780**

59 GMAO Perturbation Algorithm **780**
59.1 Fortran: Module Interface gmaopert_Mod (Source File: gmaopert_Mod.F90) . . . 780
59.1.1 gmaopert_init (Source File: gmaopert_Mod.F90) . . . 781
59.1.2 gmaopert_setup (Source File: gmaopert_Mod.F90) . . . 781
59.1.3 gmaoperturb (Source File: gmaopert_Mod.F90) . . . 781
59.1.4 gmaopert_writerestart (Source File: gmaopert_Mod.F90) . . . 782
59.1.5 gmaopert_readrestart (Source File: gmaopert_Mod.F90) . . . 782
59.2 Fortran: Module Interface landpert_module.F90 (Source File: landpert_module.F90) . . . 783
59.2.1 allocate_pert_param (Source File: landpert_module.F90) . . . 783
59.3 Fortran: Module Interface landpert_routines.F90 (Source File: landpert_routines.F90) . . . 783
59.3.1 assemble_pert_param (Source File: landpert_routines.F90) . . . 783
59.3.2 get_sqrt_corr_matrix (Source File: landpert_routines.F90) . . . 784
59.3.3 get_init_Pert_rseed (Source File: landpert_routines.F90) . . . 785
59.3.4 get_pert (Source File: landpert_routines.F90) . . . 785
59.3.5 propagate_pert (Source File: landpert_routines.F90) . . . 787
59.3.6 truncate_std_normal (Source File: landpert_routines.F90) . . . 788
59.3.7 adjust_mean (Source File: landpert_routines.F90) . . . 788
59.4 Fortran: Module Interface nr_jacobi.F90 (Source File: nr_jacobi.F90) . . . 788
59.4.1 jacobi (Source File: nr_jacobi.F90) . . . 789
59.5 Fortran: Module Interface nr_ran2_gasdev.F90 (Source File: nr_ran2_gasdev.F90) . . . 789
59.6 Fortran: Module Interface random_fields.F90 (Source File: random_fields.F90) . . . 789
59.6.1 sqrt_gauss_spectrum_2d (Source File: random_fields.F90) . . . 790

60 Bias Estimation Algorithms in LIS **790**
60.1 Fortran: Module Interface gmaobias_estimationMod (Source File: gmaobias_estimationMod.F90) 790
60.1.1 gmaobiasestimation_calc (Source File: gmaobias_estimationMod.F90) 791
60.1.2 gmaobiasestimation_update (Source File: gmaobias_estimationMod.F90) 791
60.1.3 gmaobiasestimation_restart (Source File: gmaobias_estimationMod.F90) 792
60.1.4 gmaobias_restart_filename (Source File: gmaobias_estimationMod.F90) 792
60.1.5 bias_update (Source File: gmaobias_estimationMod.F90) 792
60.1.6 bias_update_helper (Source File: gmaobias_estimationMod.F90) 793
60.1.7 get_ens_avg (Source File: gmaobias_estimationMod.F90) 793
60.1.8 bias_corr (Source File: gmaobias_estimationMod.F90) 794
60.1.9 bias_corr_helper (Source File: gmaobias_estimationMod.F90) 794
60.1.10 bias_get_const_param (Source File: gmaobias_estimationMod.F90) 794
60.1.11 bias_time_of_day_index (Source File: gmaobias_estimationMod.F90) 795
60.1.12 bias_options_helper (Source File: gmaobias_estimationMod.F90) 795

XII Observation data handling for assimilation in LIS **797**

61 Observation data handling **798**

62 ANSA Snow Cover Fraction **798**
62.1 Fortran: Module Interface ANSASCFsnow_Mod (Source File: ANSASCFsnow_Mod.F90) . . . 798
62.1.1 ANSASCFsnow_setup (Source File: ANSASCFsnow_Mod.F90) 798
62.1.2 getNSO_ANSASCFsnow (Source File: getNSO_ANSASCFsnow.F90) 799
62.1.3 read_ANSASCFsnow (Source File: read_ANSASCFsnow.F90) 799
62.1.4 ANSAscf_filename (Source File: read_ANSASCFsnow.F90) 800
62.1.5 write_ANSASCFsnowobs (Source File: write_ANSASCFsnow.F90) 801
62.1.6 ANSASCFsnow_obsname (Source File: write_ANSASCFsnow.F90) 801
62.2 Fortran: Module Interface ECV_sm_Mod (Source File: ECV_sm_Mod.F90) 802
62.2.1 ECV_sm_setup (Source File: ECV_sm_Mod.F90) 802
62.2.2 getNSO_ECVsm.F90 (Source File: getNSO_ECVsm.F90) 803
62.2.3 read_ECVsm (Source File: read_ECVsm.F90) 804
62.2.4 read_ECV_data (Source File: read_ECVsm.F90) 804
62.2.5 create_ECVsm_filename (Source File: read_ECVsm.F90) 805
62.2.6 write_ECVsmobs (Source File: write_ECVsmobs.F90) 806
62.2.7 ECV_smobsname (Source File: write_ECVsmobs.F90) 806
62.3 Fortran: Module Interface GRACEobs_module (Source File: GRACEobs_module.F90) 806
62.3.1 GRACEobs_setup (Source File: GRACEobs_module.F90) 807
62.3.2 getNSO_GRACE.F90 (Source File: getNSO_GRACE.F90) 807
62.3.3 read_GRACEobs (Source File: read_GRACEobs.F90) 808
62.3.4 write_GRACEobs (Source File: write_GRACEobs.F90) 809
62.3.5 GRACE_twsobsname (Source File: write_GRACEobs.F90) 809
62.4 Fortran: Module Interface LPRM_AMSREsm_Mod (Source File: LPRM_AMSREsm_Mod.F90) 809
62.4.1 LPRM_AMSREsm_setup (Source File: LPRM_AMSREsm_Mod.F90) 810
62.4.2 getNSO_LPRM_AMSREsm.F90 (Source File: getNSO_LPRM_AMSREsm.F90) 811
62.4.3 read_LPRM_AMSREsm (Source File: read_LPRM_AMSREsm.F90) 811
62.4.4 read_LPRM_data (Source File: read_LPRM_AMSREsm.F90) 812
62.4.5 create_LPRM_AMSREsm_filename (Source File: read_LPRM_AMSREsm.F90) 813
62.4.6 write_LPRM_AMSREsmobs (Source File: write_LPRM_AMSREsmobs.F90) 813
62.4.7 LPRM_AMSRE_smobsname (Source File: write_LPRM_AMSREsmobs.F90) 814

63 MODIS Snow Cover Area **814**
63.1 Fortran: Module Interface MODISscaobs_module (Source File: MODISscaobs_module.F90) . 814
63.1.1 MODISscaobs_setup (Source File: MODISscaobs_module.F90) 815
63.1.2 getNSO_MODISsca.F90 (Source File: getNSO_MODISsca.F90) 816
63.1.3 read_MODISscaobs (Source File: read_MODISscaobs.F90) 816
63.1.4 MODISscaGF_filename (Source File: read_MODISscaobs.F90) 817
63.1.5 MODISsca_filename (Source File: read_MODISscaobs.F90) 817
63.1.6 getMODISsca (Source File: read_MODISscaobs.F90) 818
63.1.7 calculate_sca (Source File: read_MODISscaobs.F90) 818
63.1.8 write_MODISsca (Source File: write_MODISsca.F90) 819
63.1.9 MODIS_scaobsname (Source File: write_MODISsca.F90) 819

64 NASA AMSR-E Soil Moisture **820**
64.1 Fortran: Module Interface NASA_AMSREsm_Mod (Source File: NASA_AMSREsm_Mod.F90) 820
64.1.1 NASA_AMSREsm_setup (Source File: NASA_AMSREsm_Mod.F90) 820
64.1.2 NASA_AMSREsm_computeInterpWeights (Source File: NASA_AMSREsm_Mod.F90) . 821
64.1.3 getNSO_NASA_AMSREsm.F90 (Source File: getNSO_NASA_AMSREsm.F90) 821
64.1.4 read_NASA_AMSREsm (Source File: read_NASA_AMSREsm.F90) 822
64.1.5 read_AMSREsm (Source File: read_NASA_AMSREsm.F90) 823
64.1.6 maskObs_basedonQC (Source File: read_NASA_AMSREsm.F90) 823
64.1.7 maskObs_basedonTime (Source File: read_NASA_AMSREsm.F90) 824
64.1.8 NASA_AMSREsm_filename (Source File: read_NASA_AMSREsm.F90) 824
64.1.9 write_NASA_AMSREsmobs (Source File: write_NASA_AMSREsmobs.F90) 825
64.1.10 NASA_AMSRE_smobsname (Source File: write_NASA_AMSREsmobs.F90) 825

65 PMW SWE or snow depth Observations **825**
65.1 Fortran: Module Interface PMW_snow_Mod (Source File: PMW_snow_Mod.F90) 826
65.1.1 PMW_snow_setup (Source File: PMW_snow_Mod.F90) 826
65.1.2 readPMWSnowDataConfig (Source File: PMW_snow_Mod.F90) 827
65.1.3 computeInterpWeights_latlon (Source File: PMW_snow_Mod.F90) 827
65.1.4 computeInterpWeights_ease (Source File: PMW_snow_Mod.F90) 828
65.1.5 getNSO_PMW_snow (Source File: getNSO_PMW_snow.F90) 828
65.1.6 read_PMW_snow (Source File: read_PMW_snow.F90) 828
65.1.7 read_PMWSnow_HDF4 (Source File: read_PMW_snow.F90) 829
65.1.8 read_PMWSnow_HDFEOS (Source File: read_PMW_snow.F90) 830
65.1.9 read_PMWSnow_HDF5 (Source File: read_PMW_snow.F90) 830
65.1.10 PMW_snow_filename (Source File: read_PMW_snow.F90) 831
65.1.11 qc_PMWSnow (Source File: read_PMW_snow.F90) 831
65.1.12 write_PMW_snowobs (Source File: write_PMW_snowobs.F90) 832
65.1.13 PMW_snow_obsname (Source File: write_PMW_snowobs.F90) 832
65.2 Fortran: Module Interface RT_SMOPSsm_Mod (Source File: RT_SMOPSsm_Mod.F90) 833
65.2.1 RT_SMOPSsm_setup (Source File: RT_SMOPSsm_Mod.F90) 833
65.2.2 getNSO_RT_SMOPSsm.F90 (Source File: getNSO_RT_SMOPSsm.F90) 834
65.2.3 read_RT_SMOPSsm (Source File: read_RT_SMOPSsm.F90) 834
65.2.4 read_RTSMOPS_data (Source File: read_RT_SMOPSsm.F90) 835
65.2.5 create_RT_SMOPSsm_filename (Source File: read_RT_SMOPSsm.F90) 836
65.2.6 write_RT_SMOPSsmobs (Source File: write_RT_SMOPSsmobs.F90) 837
65.2.7 RT_SMOPS_smobsname (Source File: write_RT_SMOPSsmobs.F90) 838

66 WindSat Soil Moisture **838**
66.1 Fortran: Module Interface WindSatsm_Mod (Source File: WindSatsm_Mod.F90) 838
66.1.1 WindSatsm_setup (Source File: WindSatsm_Mod.F90) 838
66.1.2 getNSO_WindSatsm (Source File: getNSO_WindSatsm.F90) 839
66.1.3 read_WindSatsm (Source File: read_WindSatsm.F90) 840
66.1.4 WindSatsm_filename (Source File: read_WindSatsm.F90) 840
66.1.5 write_WindSatsmobs (Source File: write_WindSatsmobs.F90) 841
66.1.6 WindSat_smobsname (Source File: write_WindSatsmobs.F90) 842

XIII Parameter Estimation in LIS **843**

67 Parameter Estimation in LIS **844**

68 DREAM **844**
68.1 Fortran: Module Interface DEMCzAlgorithm (Source File: DEMCzAlgorithm.F90) 844
68.1.1 DEMCz_init (Source File: DEMCzAlgorithm.F90) 844
68.1.2 DEMCz_setup (Source File: DEMCzAlgorithm.F90) 845
68.1.3 DEMCz_checkConvergence (Source File: DEMCzAlgorithm.F90) 845
68.1.4 DEMCz_run (Source File: DEMCzAlgorithm.F90) 845
68.1.5 DEMCz_setPreLnZ (Source File: DEMCzAlgorithm.F90) 846
68.1.6 DEMCz_drawCandidates (Source File: DEMCzAlgorithm.F90) 846
68.1.7 DEMCz_acceptReject (Source File: DEMCzAlgorithm.F90) 846
68.1.8 writeDEMCzrestart (Source File: DEMCzAlgorithm.F90) 846
68.1.9 DEMCz_readrestart (Source File: DEMCzAlgorithm.F90) 847
68.1.10 DEMCz_getNparam (Source File: DEMCzAlgorithm.F90) 847
68.1.11 DEMCz_getdecSpaceValues (Source File: DEMCzAlgorithm.F90) 847
68.2 Fortran: Module Interface DEMCz_varctl (Source File: DEMCz_varctl.F90) 847

69 GeneticAlgorithm **847**
69.1 Fortran: Module Interface GA_varctl (Source File: GA_varctl.F90) 848
69.2 Fortran: Module Interface GeneticAlgorithm (Source File: GeneticAlgorithm.F90) 848
69.2.1 GAOpt_init (Source File: GeneticAlgorithm.F90) 848
69.2.2 GAOpt_setup (Source File: GeneticAlgorithm.F90) 849
69.2.3 GAOpt_checkConvergence (Source File: GeneticAlgorithm.F90) 849
69.2.4 GAOpt_run (Source File: GeneticAlgorithm.F90) 849
69.2.5 GAOpt_getdecSpaceValues (Source File: GeneticAlgorithm.F90) 850
69.2.6 GAOpt_getNparam (Source File: GeneticAlgorithm.F90) 850
69.2.7 GA_recordFitness (Source File: GeneticAlgorithm.F90) 850
69.2.8 GA_selection (Source File: GeneticAlgorithm.F90) 851
69.2.9 GA_crossover (Source File: GeneticAlgorithm.F90) 851
69.2.10 GA_mutate (Source File: GeneticAlgorithm.F90) 851
69.2.11 GA_newGeneration (Source File: GeneticAlgorithm.F90) 851
69.2.12 select (Source File: GeneticAlgorithm.F90) 852
69.2.13 encode (Source File: GeneticAlgorithm.F90) 852
69.2.14 decode (Source File: GeneticAlgorithm.F90) 852
69.2.15 writeGArestart (Source File: GeneticAlgorithm.F90) 853
69.2.16 GAopt_readrestart (Source File: GeneticAlgorithm.F90) 853
69.2.17 writeGAoutput (Source File: GeneticAlgorithm.F90) 853
69.3 Fortran: Module Interface MCSIMAlgorithm (Source File: MCSIMAlgorithm.F90) 854
69.3.1 MCSIM_init (Source File: MCSIMAlgorithm.F90) 854

69.3.2 MCSIM_setup (Source File: MCSIMAlgorithm.F90) 855
69.3.3 MCSIM_checkConvergence (Source File: MCSIMAlgorithm.F90) 855
69.3.4 MCSIM_run (Source File: MCSIMAlgorithm.F90) 855
69.3.5 writeMCSIMrestart (Source File: MCSIMAlgorithm.F90) 856
69.3.6 MCSIM_readrestart (Source File: MCSIMAlgorithm.F90) 856
69.3.7 MCSIM_getNparam (Source File: MCSIMAlgorithm.F90) 856
69.3.8 MCSIM_getdecSpaceValues (Source File: MCSIMAlgorithm.F90) 856
69.4 Fortran: Module Interface MCSIM_varctl (Source File: MCSIM_varctl.F90) 857
69.5 Fortran: Module Interface LLobjFunc_Mod (Source File: LLobjFunc_Mod.F90) 857
69.5.1 intializeLLObjFunc (Source File: LLobjFunc_Mod.F90) 857
69.5.2 computeLLestimate (Source File: computeLLestimate.F90) 858
69.5.3 updateLLestimate (Source File: updateLLestimate.F90) 858
69.6 Fortran: Module Interface LSobjFunc_Mod (Source File: LSobjFunc_Mod.F90) 858
69.6.1 intializeLSObjFunc (Source File: LSobjFunc_Mod.F90) 859
69.6.2 computeLSestimate (Source File: computeLSestimate.F90) 859
69.6.3 updateLSestimate (Source File: updateLSestimate.F90) 860
69.7 Fortran: Module Interface PobjFunc_Mod (Source File: PobjFunc_Mod.F90) 860
69.7.1 intializePObjFunc (Source File: PobjFunc_Mod.F90) 861
69.7.2 computePestimate (Source File: computePestimate.F90) 862
69.7.3 updatePestimate (Source File: updatePestimate.F90) 862
69.8 Fortran: Module Interface AMSRE_SR_em_obsMod (Source File: AMSRE_SR_em_obsMod.F90) 863
69.8.1 AMSRE_SR_em_obs_setup (Source File: AMSRE_SR_em_obsMod.F90) 863
69.8.2 read_AMSRE_SR_em_obsdata (Source File: read_AMSRE_SRdata.F90) 864
69.8.3 reset_AMSRE_SR_em_obs_data (Source File: reset_AMSRE_SRObsdata.F90) 864
69.8.4 write_AMSRE_SR_em_obs_data (Source File: write_AMSRE_SR_em_obs_data.F90) . . 865
69.8.5 AMSRE_SR_obsname (Source File: write_AMSRE_SR_em_obs_data.F90) 865
69.9 Fortran: Module Interface Empty_obsMod (Source File: Empty_obsMod.F90) 866
69.9.1 Empty_obs_setup (Source File: Empty_obsMod.F90) 866
69.10 Fortran: Module Interface LPRM_AMSREsm_obsMod (Source File: LPRM_AMSREsm_obsMod.F90) 867
69.10.1 LPRM_AMSREsm_obs_setup (Source File: LPRM_AMSREsm_obsMod.F90) 867
69.10.2 read_LPRM_AMSREsm_obsdata (Source File: read_LPRM_AMSREsmdata.F90) . . . 867
69.10.3 read_LPRM_obs_data (Source File: read_LPRM_AMSREsmdata.F90) 868
69.10.4 create_LPRM_AMSREsm_obs_filename (Source File: read_LPRM_AMSREsmdata.F90) 869
69.10.5 reset_LPRM_AMSREsm_obs_data (Source File: reset_LPRM_AMSREsmObsdata.F90) 870
69.10.6 write_LPRM_AMSREsm_obs_data (Source File: write_LPRM_AMSREsm_obs_data.F90) 870
69.10.7 read_templateObs (Source File: read_templateObs.F90) 871
69.10.8 reset_templateObs (Source File: reset_templateObs.F90) 871
69.11 Fortran: Module Interface templateObs_module (Source File: templateObs_module.F90) . . . 872
69.11.1 templateObs_setup (Source File: templateObs_module.F90) 872
69.11.2 write_templateObs (Source File: write_templateObs.F90) 873

XIV Land Surface Models in LIS 874

70 Land Surface Models in LIS 875

71 Community Land Model version 2.0 875
71.0.3 canhtset (Source File: canhtset.F90) 875
71.0.4 clm2_dynsetup (Source File: clm2_dynsetup.F90) 875
71.0.5 clm2_finalize (Source File: clm2_finalize.F90) 876
71.0.6 clm2_lairead.F90: (Source File: clm2_lairead.F90) 876

71.1 Fortran: Module Interface clm2_lsmMod (Source File: clm2_lsmMod.F90) 877
71.1.1 clm2_lsm_ini (Source File: clm2_lsmMod.F90) 877
71.1.2 clm2_lsm_init (Source File: clm2_lsmMod.F90) 878
71.1.3 clm2_main (Source File: clm2_main.F90) 878
71.1.4 clm2_readrestart (Source File: clm2_readrestart.F90) 880
71.1.5 clm2_setup (Source File: clm2_setup.F90) 880
71.1.6 clm2_writerestart (Source File: clm2_writerestart.F90) 881
71.1.7 clm2_dump_restart (Source File: clm2_writerestart.F90) 881
71.2 Fortran: Module Interface clm2type (Source File: clm2type.F90) 882
71.2.1 clm2_setwrfesmfexport (Source File: clm2_setwrfesmfexport.F90) 887
71.2.2 clm2_wrf_esmff2t (Source File: clm2_wrf_esmff2t.F90) 887
71.2.3 clm2_setwrfexport.F90 (Source File: clm2_setwrfexport.F90) 888
71.2.4 clm2_wrf_f2t (Source File: clm2_wrf_f2t.F90) 888
71.2.5 clm2_descale_lst (Source File: clm2_descale_lst.F90) 889
71.2.6 clm2_getlst (Source File: clm2_getlst.F90) 890
71.2.7 clm2_getsynlstpred (Source File: clm2_getsynlstpred.F90) 890
71.2.8 clm2_qclst (Source File: clm2_qclst.F90) 891
71.2.9 clm2_scale_lst (Source File: clm2_scale_lst.F90) 891
71.2.10 clm2_setlst (Source File: clm2_setlst.F90) 892
71.2.11 clm2_descale_snow (Source File: clm2_descale_snow.F90) 892
71.2.12 clm2_getsnowvars (Source File: clm2_getsnowvars.F90) 893
71.2.13 clm2_getsnwdpred (Source File: clm2_getsnwdpred.F90) 894
71.2.14 clm2_getswepred (Source File: clm2_getswepred.F90) 894
71.2.15 clm2_map_sca (Source File: clm2_map_sca.F90) 894
71.2.16 di_method_rh04 (Source File: clm2_map_sca.F90) 895
71.2.17 clm2_qc_snowobs (Source File: clm2_qc_snowobs.F90) 896
71.2.18 clm2_qcsnow (Source File: clm2_qcsnow.F90) 896
71.2.19 clm2_scale_snow (Source File: clm2_scale_snow.F90) 897
71.2.20 clm2_setsnowvars (Source File: clm2_setsnowvars.F90) 898
71.2.21 clm2_transform_obssca (Source File: clm2_transform_obssca.F90) 898
71.2.22 clm2_updatesnowvars (Source File: clm2_updatesnowvars.F90) 899
71.2.23 clm2_descale_soilm (Source File: clm2_descale_soilm.F90) 899
71.2.24 clm2_getsoilm (Source File: clm2_getsoilm.F90) 900
71.2.25 clm2_getsynsmpred (Source File: clm2_getsynsmpred.F90) 901
71.2.26 clm2_qcsoilm (Source File: clm2_qcsoilm.F90) 901
71.2.27 clm2_scale_soilm (Source File: clm2_scale_soilm.F90) 902
71.2.28 clm2_setsoilm (Source File: clm2_setsoilm.F90) 902
71.2.29 clm2_updatesoilm (Source File: clm2_updatesoilm.F90) 903
71.2.30 clm2_write_soilm (Source File: clm2_write_soilm.F90) 903
71.2.31 iniTimeConst (Source File: iniTimeConst.F90) 904
71.2.32 iniTimeVar (Source File: iniTimeVar.F90) 905
71.3 Fortran: Module Interface clm2_atmdrvMod (Source File: clm2_atmdrvMod.F90) 905
71.3.1 clm2_atmdrv (Source File: clm2_atmdrvMod.F90) 906
71.3.2 bufferunpack3d 908
71.3.3 bufferunpack4d 909
71.4 Fortran: Module Interface debugutilitiesmodule (Source File: debugutilitiesmodule.F90) . . . 910
71.4.1 DumAssert 911
71.4.2 DumEnter 912
71.4.3 DumLeave 913
71.5 Fortran: Module Interface decompmodule (Source File: decompmodule.F90) 914
71.5.1 DecompFree 915

71.5.2 DecompCopy . . . 916
71.5.3 DecompPermute . . . 918
71.5.4 DecompRegular1D . . . 919
71.5.5 DecompRegular2D . . . 921
71.5.6 DecompRegular3D . . . 923
71.5.7 DecompRegular3Dorder . . . 925
71.5.8 DecompCreateIrr . . . 928
71.5.9 DecompCreateTags . . . 930
71.5.10 DecompGlobalToLocal . . . 932
71.5.11 DecompLocalToGlobal . . . 933
71.5.12 DecompInfo . . . 935
71.6 Fortran: Module Interface ghostmodule (Source File: ghostmodule.F90) . . . 936
71.6.1 GhostFree . . . 937
71.6.2 GhostCopy . . . 938
71.6.3 GhostIrregular . . . 938
71.6.4 GhostRegular1D . . . 940
71.6.5 GhostRegular2D . . . 942
71.6.6 GhostRegular3D . . . 945
71.6.7 GhostInfo . . . 948
71.7 Fortran: Module Interface parutilitiesmodule (Source File: parutilitiesmodule.F90) . . . 949
71.7.1 ParInit . . . 953
71.7.2 ParExit . . . 954
71.7.3 ParSplit . . . 955
71.7.4 ParFree . . . 956
71.7.5 ParPatternGhost . . . 957
71.7.6 ParPatternDecompToDecomp . . . 960
71.7.7 ParPatternDecompToGhost . . . 965
71.7.8 ParPatternFree . . . 969
71.7.9 ParScatterReal . . . 970
71.7.10 ParScatterInt . . . 973
71.7.11 ParGatherReal . . . 976
71.7.12 ParGatherInt . . . 979
71.7.13 ParBeginTransferReal . . . 982
71.7.14 ParBeginTransferPattern . . . 985
71.7.15 ParEndTransferReal . . . 987
71.7.16 ParEndTransferPattern . . . 989
71.7.17 ParExchangeVectorReal . . . 991
71.7.18 ParExchangeVectorInt . . . 993
71.7.19 ParCollectiveBarrier — Barrier: Simplest collective op. (Source File: parutilitiesmodule.F90) . . . 996
71.7.20 ParCollective0D — Perform global Collective of a scalar (Source File: parutilitiesmodule.F90) . . . 997
71.7.21 ParCollective1D — Perform component-wise global Collective of a vector (Source File: parutilitiesmodule.F90) . . . 998
71.7.22 ParCollective2D — Perform component-wise collective operation (Source File: parutilitiesmodule.F90) . . . 1000
71.7.23 ParCollective3D — Perform component-wise global Collective of a vector (Source File: parutilitiesmodule.F90) . . . 1002
71.7.24 ParCollective0DInt — Perform global Collective of a scalar (Source File: parutilitiesmodule.F90) . . . 1004

71.7.25 ParCollective1DInt — Perform component-wise global (Source File: parutilitiesmodule.F90) . . . 1006
71.7.26 scatter — wrapper for PILGRIM utility (Source File: puminterfaces.F90) . . . 1007
71.7.27 gather — wrapper for PILGRIM utility (Source File: puminterfaces.F90) . . . 1008
71.7.28 begintransfer — wrapper for PILGRIM utility (Source File: puminterfaces.F90) . . . 1009
71.7.29 endtransfer — wrapper for PILGRIM utility (Source File: puminterfaces.F90) . . . 1010
71.7.30 exchangevector — wrapper for PILGRIM utility (Source File: puminterfaces.F90) . . . 1011
71.8 Fortran: Module Interface redistributemodule (Source File: redistributemodule.F90) . . . 1011
71.8.1 RedistributeCreate . . . 1012
71.8.2 RedistributePerform . . . 1015
71.8.3 RedistributeFree . . . 1016
71.8.4 RedistributeStart . . . 1017
71.8.5 RedistributeFinish . . . 1019

72 Catchment Land Surface Model 1020
72.1 Fortran: Module Interface (Source File: clsmf25_DragCoefficientsMod.F90) . . . 1020
72.1.1 GETCDS . . . 1021
72.2 Fortran: Module Interface – A container module for global constants (Source File: clsmf25_MAPL_constants.F90) 1023
72.2.1 clsmf25_read_land_parameters3 (Source File: clsmf25_compute_land_parameters.F90) . 1023
72.3 Fortran: Module Interface clsmf25_diagn_routines (Source File: clsmf25_diagn_routines.F90) . 1024
72.3.1 calc_tsurf (Source File: clsmf25_diagn_routines.F90) . . . 1024
72.3.2 calc_arX (Source File: clsmf25_diagn_routines.F90) . . . 1025
72.3.3 calc_asnow (Source File: clsmf25_diagn_routines.F90) . . . 1025
72.4 Fortran: Module Interface clsmf25_drv_types (Source File: clsmf25_drv_types.F90) . . . 1025
72.4.1 clsmf25_dynsetup.F90 (Source File: clsmf25_dynsetup.F90) . . . 1025
72.5 Fortran: Module Interface clsmf25_esat_qsat (Source File: clsmf25_esat_qsat.F90) . . . 1026
72.5.1 clsmf25_f2t (Source File: clsmf25_f2t.F90) . . . 1026
72.5.2 clsmf25_finalize (Source File: clsmf25_finalize.F90) . . . 1026
72.5.3 clsmf25_lsmMod.F90 (Source File: clsmf25_lsmMod.F90) . . . 1027
72.5.4 clsmf25_varalloc (Source File: clsmf25_lsmMod.F90) . . . 1028
72.5.5 clsmf25_main (Source File: clsmf25_main.F90) . . . 1028
72.5.6 clsmf25_pmonth (Source File: clsmf25_pmonth_stage4.F90) . . . 1029
72.5.7 clsmf25_readcrd (Source File: clsmf25_readcrd.F90) . . . 1029
72.5.8 clsmf25_readrst (Source File: clsmf25_readrst.F90) . . . 1030
72.5.9 clsmf25_setup (Source File: clsmf25_setup.F90) . . . 1030
72.5.10 clsmf25_read_land_parameter (Source File: clsmf25_setup.F90) . . . 1031
72.5.11 clsmf25_read_land3d_parameter (Source File: clsmf25_setup.F90) . . . 1032
72.5.12 clsmf25_turb (Source File: clsmf25_turb.F90) . . . 1033
72.6 Fortran: Module Interface clsmf25_types (Source File: clsmf25_types.F90) . . . 1033
72.6.1 clsmf25_writerst (Source File: clsmf25_writerst.F90) . . . 1033
72.6.2 clsmf25_dump_restart (Source File: clsmf25_writerst.F90) . . . 1034
72.7 Fortran: Module Interface clsmf25_dasnow_Mod (Source File: clsmf25_dasnow_Mod.F90) . . . 1035
72.7.1 clsmf25_dasnow_init (Source File: clsmf25_dasnow_Mod.F90) . . . 1035
72.7.2 clsmf25_descale_snow (Source File: clsmf25_descale_snow.F90) . . . 1036
72.7.3 clsmf25_getsnowvars (Source File: clsmf25_getsnowvars.F90) . . . 1036
72.7.4 clsmf25_getsnwdpred (Source File: clsmf25_getsnwdpred.F90) . . . 1037
72.7.5 clsmf25_qc_snowobs (Source File: clsmf25_qc_snowobs.F90) . . . 1037
72.7.6 clsmf25_qcsnow (Source File: clsmf25_qcsnow.F90) . . . 1038
72.7.7 clsmf25_scale_snow (Source File: clsmf25_scale_snow.F90) . . . 1039
72.7.8 clsmf25_setsnowvars (Source File: clsmf25_setsnowvars.F90) . . . 1039
72.7.9 clsmf25_updatesnowvars (Source File: clsmf25_updatesnowvars.F90) . . . 1040

72.7.10 clsmf25_wrtswe (Source File: clsmf25_wrtswe.F90) 1041
72.8 Fortran: Module Interface clsmf25_dasoilm_Mod (Source File: clsmf25_dasoilm_Mod.F90) . . 1041
72.8.1 clsmf25_dasoilm_init (Source File: clsmf25_dasoilm_Mod.F90) 1042
72.8.2 clsmf25_descale_soilm (Source File: clsmf25_descale_soilm.F90) 1042
72.8.3 clsmf25_getsmpred (Source File: clsmf25_getsmpred.F90) 1043
72.8.4 clsmf25_getsoilm (Source File: clsmf25_getsoilm.F90) 1043
72.8.5 clsmf25_qc_soilmobs (Source File: clsmf25_qc_soilmobs.F90) 1044
72.8.6 clsmf25_qcsoilm (Source File: clsmf25_qcsoilm.F90) 1044
72.8.7 clsmf25_scale_soilm (Source File: clsmf25_scale_soilm.F90) 1045
72.8.8 clsmf25_setsoilm (Source File: clsmf25_setsoilm.F90) 1046
72.8.9 clsmf25_updatesoilm (Source File: clsmf25_updatesoilm.F90) 1046
72.8.10 clsmf25_write_soilm (Source File: clsmf25_write_soilm.F90) 1047
72.9 Fortran: Module Interface clsmf25_datws_Mod (Source File: clsmf25_datws_Mod.F90) 1047
72.9.1 clsmf25_datws_init (Source File: clsmf25_datws_Mod.F90) 1048
72.9.2 clsmf25_descale_tws (Source File: clsmf25_descale_tws.F90) 1048
72.9.3 clsmf25_gettws (Source File: clsmf25_gettws.F90) 1048
72.9.4 clsmf25_gettwspred (Source File: clsmf25_gettwspred.F90) 1049
72.9.5 clsmf25_qc_twsobs (Source File: clsmf25_qc_twsobs.F90) 1050
72.9.6 clsmf25_qctws (Source File: clsmf25_qctws.F90) 1050
72.9.7 clsmf25_scale_tws (Source File: clsmf25_scale_tws.F90) 1051
72.9.8 clsmf25_settws (Source File: clsmf25_settws.F90) 1052
72.9.9 clsmf25_update_tws (Source File: clsmf25_update_tws.F90) 1052
72.9.10 clsmf25_write_tws (Source File: clsmf25_write_tws.F90) 1053
72.9.11 clsmf25_getirrigationstates (Source File: clsmf25_getirrigationstates.F90) 1053
72.9.12 clsmf25_getrunoffs (Source File: clsmf25_getrunoffs.F90) 1054
72.9.13 clsmf25_getrunoffs_mm (Source File: clsmf25_getrunoffs_mm.F90) 1054

73 WRSI Model **1055**
73.0.14 geowrsi2_dynsetup (Source File: geowrsi2_dynsetup.F90) 1055
73.0.15 geowrsi2_f2t (Source File: geowrsi2_f2t.F90) 1056
73.0.16 geowrsi2_finalize (Source File: geowrsi2_finalize.F90) 1056
73.1 Fortran: Module Interface geowrsi2_lsmMod (Source File: geowrsi2_lsmMod.F90) 1057
73.1.1 geowrsi2_lsm_ini (Source File: geowrsi2_lsmMod.F90) 1057
73.1.2 geowrsi2_main (Source File: geowrsi2_main.F90) 1058
73.2 Fortran: Module Interface geowrsi2_module.F90 (Source File: geowrsi2_module.F90) 1059
73.3 Fortran: Module Interface geowrsi2_physics_module (Source File: geowrsi2_physics_module.F90) 1059
73.3.1 geowrsi2_readInputSettings (Source File: geowrsi2_readInputSettings.F90) 1060
73.3.2 geowrsi2_readSOS_Bilfile (Source File: geowrsi2_readSOS_Bilfile.F90) 1060
73.3.3 geowrsi2_readcrd (Source File: geowrsi2_readcrd.F90) 1061
73.3.4 geowrsi2_readrst (Source File: geowrsi2_readrst.F90) 1061
73.3.5 geowrsi2_set_gcoords (Source File: geowrsi2_set_gcoords.F90) 1062
73.3.6 geowrsi2_setup (Source File: geowrsi2_setup.F90) 1063
73.3.7 geowrsi2_writeSOS_Bilfile (Source File: geowrsi2_writeSOS_Bilfile.F90) 1063
73.3.8 geowrsi2_writerst (Source File: geowrsi2_writerst.F90) 1064

74 HySSIB Land Surface Model **1065**
74.0.9 hyssib_coldstart (Source File: hyssib_coldstart.F90) 1065
74.0.10 hyssib_dynsetup (Source File: hyssib_dynsetup.F90) 1065
74.0.11 hyssib_f2t (Source File: hyssib_f2t.F90) 1065
74.0.12 hyssib_finalize (Source File: hyssib_finalize.F90) 1066
74.0.13 hyssib_gfrac.F90 (Source File: hyssib_gfrac.F90) 1066

74.1 Fortran: Module Interface hyssib_lsmMod (Source File: hyssib_lsmMod.F90) 1067
74.1.1 hyssib_lsm_ini (Source File: hyssib_lsmMod.F90) 1068
74.1.2 hyssib_main (Source File: hyssib_main.F90) 1068
74.1.3 hyssib_mapvegc.F90 (Source File: hyssib_mapvegc.F90) 1069
74.2 Fortran: Module Interface hyssib_module (Source File: hyssib_module.F90) 1070
74.2.1 hyssib_readcrd (Source File: hyssib_readcrd.F90) 1073
74.2.2 hyssib_readrst (Source File: hyssib_readrst.F90) 1074
74.2.3 hyssib_settbot (Source File: hyssib_settbot.F90) 1074
74.2.4 hyssib_settopostd (Source File: hyssib_settopostd.F90) 1075
74.2.5 hyssib_setup (Source File: hyssib_setup.F90) 1075
74.2.6 hyssib_setvegparms (Source File: hyssib_setvegparms.F90) 1076
74.2.7 hyssib_totinit (Source File: hyssib_totinit.F90) 1077
74.2.8 hyssib_writerst (Source File: hyssib_writerst.F90) 1077
74.2.9 hyssib_dump_restart (Source File: hyssib_writerst.F90) 1078
74.3 Fortran: Module Interface hyssibalb_module (Source File: hyssibalb_module.F90) 1079
74.3.1 hyssibalb_ini (Source File: hyssibalb_module.F90) 1080
74.4 Fortran: Module Interface hyssibveg_module (Source File: hyssibveg_module.F90) 1081
74.4.1 hyssibveg_ini (Source File: hyssibveg_module.F90) 1082

75 Mosaic Land Surface Model 1082
75.0.2 mos_descale_soilm (Source File: mos_descale_soilm.F90) 1082
75.0.3 mos_getsoilm (Source File: mos_getsoilm.F90) 1083
75.0.4 mos_getsynsmpred (Source File: mos_getsynsmpred.F90) 1084
75.0.5 mos_qcsoilm (Source File: mos_qcsoilm.F90) 1084
75.0.6 mos_scale_soilm (Source File: mos_scale_soilm.F90) 1085
75.0.7 mos_setsoilm (Source File: mos_setsoilm.F90) 1085
75.0.8 mos_updatesoilm (Source File: mos_updatesoilm.F90) 1086
75.0.9 mos_write_soilm (Source File: mos_write_soilm.F90) 1086
75.0.10 mapsib2umd (Source File: mapsib2umd.F90) 1087
75.0.11 mos_coldstart (Source File: mos_coldstart.F90) 1087
75.0.12 mos_f2t (Source File: mos_f2t.F90) 1088
75.0.13 mos_finalize (Source File: mos_finalize.F90) 1088
75.0.14 mos_lsmMod (Source File: mos_lsmMod.F90) 1089
75.0.15 mos_lsm_ini (Source File: mos_lsmMod.F90) 1090
75.0.16 mos_main (Source File: mos_main.F90) 1090
75.1 Fortran: Module Interface mos_module (Source File: mos_module.F90) 1092
75.1.1 mos_readrestart (Source File: mos_readrestart.F90) 1094
75.1.2 mos_setup (Source File: mos_setup.F90) 1094
75.1.3 mos_writerst (Source File: mos_writerst.F90) 1095
75.1.4 mos_dump_restart (Source File: mos_writerst.F90) 1096
75.1.5 mosdynp (Source File: mosdynp.F90) 1097
75.1.6 paramb (Source File: paramb.F90) 1098
75.1.7 readmoscrd (Source File: readmoscrd.F90) 1098
75.1.8 setmosp (Source File: setmosp.F90) 1099
75.1.9 mos_sfc2cmem3 (Source File: mos_sfc2cmem3.F90) 1099
75.2 Fortran: Module Interface sibalb_module (Source File: sibalb_module.F90) 1100
75.2.1 texture (Source File: texture.F90) 1100

76 Noah land surface model version 2.7.1 **1101**
76.0.2 noah271_gce_f2t (Source File: noah271_gce_f2t.F90) 1101
76.0.3 noah271_setgceexport.F90 (Source File: noah271_setgceexport.F90) 1102
76.0.4 noah271_setwrfesmfexport.F90 (Source File: noah271_setwrfesmfexport.F90) 1102
76.0.5 noah271_wrf_esmff2t (Source File: noah271_wrf_esmff2t.F90) 1103
76.0.6 noah271_setwrfexport.F90 (Source File: noah271_setwrfexport.F90) 1103
76.0.7 noah271_wrf_f2t (Source File: noah271_wrf_f2t.F90) 1104
76.0.8 noah271_descale_multism (Source File: noah271_descale_multism.F90) 1105
76.0.9 noah271_getmultism (Source File: noah271_getmultism.F90) 1105
76.0.10 noah271_getmultismpred (Source File: noah271_getmultismpred.F90) 1106
76.0.11 noah271_qcmultism (Source File: noah271_qcmultism.F90) 1106
76.0.12 noah271_scale_multism (Source File: noah271_scale_multism.F90) 1107
76.0.13 noah271_setmultism (Source File: noah271_setmultism.F90) 1108
76.0.14 noah271_updatemultism (Source File: noah271_updatemultism.F90) 1108
76.0.15 noah271_write_multism (Source File: noah271_write_multism.F90) 1109
76.0.16 noah271_descale_scf (Source File: noah271_descale_scf.F90) 1109
76.0.17 noah271_getscfpred (Source File: noah271_getscfpred.F90) 1110
76.0.18 noah271_getscfvars (Source File: noah271_getscfvars.F90) 1110
76.0.19 noah271_map_scf (Source File: noah271_map_scf.F90) 1111
76.0.20 noah271_qc_scf (Source File: noah271_qc_scf.F90) 1112
76.0.21 noah271_scale_scf (Source File: noah271_scale_scf.F90) 1112
76.0.22 noah271_setscfvars (Source File: noah271_setscfvars.F90) 1113
76.0.23 noah271_transform_scf (Source File: noah271_transform_scf.F90) 1113
76.0.24 noah271_updatescfvars (Source File: noah271_updatescfvars.F90) 1114
76.0.25 noah271_descale_snow (Source File: noah271_descale_snow.F90) 1115
76.0.26 noah271_getscapred (Source File: noah271_getscapred.F90) 1115
76.0.27 noah271_getsnowvars (Source File: noah271_getsnowvars.F90) 1116
76.0.28 noah271_getsynswepred (Source File: noah271_getsynswepred.F90) 1116
76.0.29 noah271_map_snow (Source File: noah271_map_snow.F90) 1117
76.0.30 noah271_map_swe (Source File: noah271_map_swe.F90) 1117
76.0.31 noah271_qc_snowobs (Source File: noah271_qc_snowobs.F90) 1118
76.0.32 noah271_qcsnow (Source File: noah271_qcsnow.F90) 1119
76.0.33 noah271_scale_snow (Source File: noah271_scale_snow.F90) 1119
76.0.34 noah271_setsnowvars (Source File: noah271_setsnowvars.F90) 1120
76.0.35 noah271_transform_snow (Source File: noah271_transform_snow.F90) 1121
76.0.36 noah271_updatesnowvars (Source File: noah271_updatesnowvars.F90) 1121
76.0.37 noah271_wrtswe (Source File: noah271_wrtswe.F90) 1122
76.0.38 noah271_descale_snwd (Source File: noah271_descale_snwd.F90) 1122
76.0.39 noah271_getsnwdpred (Source File: noah271_getsnwdpred.F90) 1123
76.0.40 noah271_getsnwdvars (Source File: noah271_getsnwdvars.F90) 1123
76.0.41 noah271_qcsnwd (Source File: noah271_qcsnwd.F90) 1124
76.0.42 noah271_scale_snwd (Source File: noah271_scale_snwd.F90) 1125
76.0.43 noah271_setsnwdvars (Source File: noah271_setsnwdvars.F90) 1125
76.0.44 noah271_updatesnwdvars (Source File: noah271_updatesnwdvars.F90) 1126
76.0.45 noah271_descale_soilm (Source File: noah271_descale_soilm.F90) 1126
76.0.46 noah271_getsoilm (Source File: noah271_getsoilm.F90) 1127
76.0.47 noah271_getsynsmpred (Source File: noah271_getsynsmpred.F90) 1128
76.0.48 noah271_map_sm (Source File: noah271_map_sm.F90) 1128
76.0.49 noah271_qc_soilmobs (Source File: noah271_qc_soilmobs.F90) 1129
76.0.50 noah271_qcsoilm (Source File: noah271_qcsoilm.F90) 1129
76.0.51 noah271_scale_soilm (Source File: noah271_scale_soilm.F90) 1130

76.0.52 noah271_setsoilm (Source File: noah271_setsoilm.F90) 1131
76.0.53 noah271_transform_synsm (Source File: noah271_transform_synsm.F90) 1131
76.0.54 noah271_updatesoilm (Source File: noah271_updatesoilm.F90) 1132
76.0.55 noah271_write_soilm (Source File: noah271_write_soilm.F90) 1132
76.0.56 noah271_descale_swe (Source File: noah271_descale_swe.F90) 1133
76.0.57 noah271_getswepred (Source File: noah271_getswepred.F90) 1134
76.0.58 noah271_getswevars (Source File: noah271_getswevars.F90) 1134
76.0.59 noah271_qcswe (Source File: noah271_qcswe.F90) 1135
76.0.60 noah271_scale_swe (Source File: noah271_scale_swe.F90) 1135
76.0.61 noah271_setswevars (Source File: noah271_setswevars.F90) 1136
76.0.62 noah271_updateswevars (Source File: noah271_updateswevars.F90) 1136
76.0.63 noah271_descale_tskin (Source File: noah271_descale_tskin.F90) 1137
76.0.64 noah271_getTskinPred (Source File: noah271_getTskinPred.F90) 1138
76.0.65 noah271_gettskin (Source File: noah271_gettskin.F90) 1138
76.0.66 noah271_qc_tskinobs (Source File: noah271_qc_tskinobs.F90) 1139
76.0.67 noah271_qctskin (Source File: noah271_qctskin.F90) 1139
76.0.68 noah271_scale_tskin (Source File: noah271_scale_tskin.F90) 1140
76.0.69 noah271_settskin (Source File: noah271_settskin.F90) 1140
76.0.70 noah271_update_tskin (Source File: noah271_update_tskin.F90) 1141
76.0.71 noah271_wrttskin (Source File: noah271_wrttskin.F90) 1142
76.0.72 noah_physics.F90 (Source File: module_sf_noah271lsm.F90) 1142
76.0.73 noah271_calc_tskin (Source File: noah271_calc_tskin.F90) 1143
76.0.74 noah271_coldstart (Source File: noah271_coldstart.F90) 1144
76.0.75 noah271_dynsetup (Source File: noah271_dynsetup.F90) 1144
76.0.76 noah271_f2t (Source File: noah271_f2t.F90) 1145
76.0.77 noah271_finalize (Source File: noah271_finalize.F90) 1145
76.1 Fortran: Module Interface noah271_lsmMod (Source File: noah271_lsmMod.F90) 1146
76.1.1 noah271_lsm_ini (Source File: noah271_lsmMod.F90) 1147
76.1.2 noah271_main (Source File: noah271_main.F90) 1148
76.1.3 sh2o_init.F90 (Source File: noah271_main.F90) 1148
76.2 Fortran: Module Interface noah271_module (Source File: noah271_module.F90) 1149
76.2.1 noah271_readcrd (Source File: noah271_readcrd.F90) 1150
76.2.2 noah271_readrst (Source File: noah271_readrst.F90) 1151
76.2.3 noah271_read_paramrst (Source File: noah271_readrst.F90) 1152
76.2.4 noah271_setsoils (Source File: noah271_setsoils.F90) 1152
76.2.5 noah271_settbot (Source File: noah271_settbot.F90) 1153
76.2.6 noah271_setup (Source File: noah271_setup.F90) 1153
76.2.7 noah271_setvegparms (Source File: noah271_setvegparms.F90) 1154
76.2.8 noah271_writerst (Source File: noah271_writerst.F90) 1154
76.2.9 noah271_dump_restart (Source File: noah271_writerst.F90) 1155
76.2.10 noah271_getpeobspred_wgPBMRsm (Source File: noah271_getpeobspred_wgPBMRsm.F90)1156
76.3 Fortran: Module Interface noah271_peMod (Source File: noah271_peMod.F90) 1157
76.3.1 noah271_setup_pedecvars (Source File: noah271_peMod.F90) 1157
76.3.2 initializeDecSpace (Source File: noah271_peMod.F90) 1158
76.3.3 noah271_set_pedecvars (Source File: noah271_set_pedecvars.F90) 1158
76.3.4 noah271_setupobspred_wgPBMRsm (Source File: noah271_setupobspred_wgPBMRsm.F90)1158
76.3.5 noah271_getrunoffs (Source File: noah271_getrunoffs.F90) 1159
76.3.6 noah271_sfc2cmem3 (Source File: noah271_sfc2cmem3.F90) 1159
76.3.7 noah271_sfc2crtm (Source File: noah271_sfc2crtm.F90) 1160
76.3.8 noah32_setwrfexport.F90 (Source File: noah32_setwrfexport.F90) 1161
76.3.9 noah32_wrf_f2t (Source File: noah32_wrf_f2t.F90) 1161

76.4 Fortran: Module Interface noah32_dascf_Mod (Source File: noah32_dascf_Mod.F90) 1162
76.4.1 noah32_dascf_init (Source File: noah32_dascf_Mod.F90) 1162
76.4.2 noah32_descale_snow (Source File: noah32_descale_snow.F90) 1163
76.4.3 noah32_getsnowvars (Source File: noah32_getsnowvars.F90) 1163
76.4.4 noah32_getsnwdpred (Source File: noah32_getsnwdpred.F90) 1164
76.4.5 noah32_getswepred (Source File: noah32_getswepred.F90) 1164
76.4.6 noah32_map_snow (Source File: noah32_map_snow.F90) 1165
76.4.7 noah32_map_swe (Source File: noah32_map_swe.F90) 1166
76.4.8 noah32_qc_snowobs (Source File: noah32_qc_snowobs.F90) 1166
76.4.9 noah32_qcsnow (Source File: noah32_qcsnow.F90) 1167
76.4.10 noah32_scale_snow (Source File: noah32_scale_snow.F90) 1168
76.4.11 noah32_setsnowvars (Source File: noah32_setsnowvars.F90) 1168
76.4.12 noah32_transform_snow (Source File: noah32_transform_snow.F90) 1169
76.4.13 noah32_updatesnowvars (Source File: noah32_updatesnowvars.F90) 1169
76.4.14 noah32_wrtswe (Source File: noah32_wrtswe.F90) 1170
76.4.15 noah32_descale_soilm (Source File: noah32_descale_soilm.F90) 1171
76.4.16 noah32_getsmpred (Source File: noah32_getsmpred.F90) 1171
76.4.17 noah32_getsoilm (Source File: noah32_getsoilm.F90) 1172
76.4.18 noah32_qc_soilmobs (Source File: noah32_qc_soilmobs.F90) 1172
76.4.19 noah32_qcsoilm (Source File: noah32_qcsoilm.F90) 1173
76.4.20 noah32_scale_soilm (Source File: noah32_scale_soilm.F90) 1174
76.4.21 noah32_setsoilm (Source File: noah32_setsoilm.F90) 1174
76.4.22 noah32_updatesoilm (Source File: noah32_updatesoilm.F90) 1175
76.4.23 noah32_write_soilm (Source File: noah32_write_soilm.F90) 1175
76.4.24 noah32_coldstart (Source File: noah32_coldstart.F90) 1176
76.4.25 noah32_dynsetup (Source File: noah32_dynsetup.F90) 1176
76.4.26 noah32_f2t (Source File: noah32_f2t.F90) 1177
76.4.27 noah32_finalize (Source File: noah32_finalize.F90) 1178
76.5 Fortran: Module Interface noah32_lsmMod (Source File: noah32_lsmMod.F90) 1178
76.5.1 noah32_lsm_ini (Source File: noah32_lsmMod.F90) 1179
76.5.2 noah32_main (Source File: noah32_main.F90) 1180
76.6 Fortran: Module Interface noah32_module (Source File: noah32_module.F90) 1180
76.6.1 noah32_readcrd (Source File: noah32_readcrd.F90) 1182
76.6.2 noah32_readrst (Source File: noah32_readrst.F90) 1182
76.6.3 noah32_setsoils (Source File: noah32_setsoils.F90) 1183
76.6.4 noah32_settbot (Source File: noah32_settbot.F90) 1184
76.6.5 noah32_setup (Source File: noah32_setup.F90) 1185
76.6.6 noah32_setvegparms (Source File: noah32_setvegparms.F90) 1185
76.6.7 noah32_writerst (Source File: noah32_writerst.F90) 1186
76.6.8 noah32_dump_restart (Source File: noah32_writerst.F90) 1187
76.6.9 noah32_getpeobspred_ARMobs (Source File: noah32_getpeobspred_ARMobs.F90) . . . 1188
76.6.10 noah32_getpeobspred_template (Source File: noah32_getpeobspred_template.F90) . . . 1189
76.7 Fortran: Module Interface noah32_peMod (Source File: noah32_peMod.F90) 1189
76.7.1 noah32_setup_pedecvars (Source File: noah32_peMod.F90) 1190
76.7.2 initializeDecSpace (Source File: noah32_peMod.F90) 1190
76.7.3 noah32_set_pedecvars (Source File: noah32_set_pedecvars.F90) 1190
76.7.4 noah32_setupobspred_ARMobs (Source File: noah32_setupobspred_ARMobs.F90) . . . 1191
76.7.5 noah32_setupobspred_template (Source File: noah32_setupobspred_template.F90) . . . 1191
76.7.6 noah32_getrunoffs (Source File: noah32_getrunoffs.F90) 1192
76.7.7 noah32_sfc2cmem3 (Source File: noah32_sfc2cmem3.F90) 1192
76.7.8 noah32_sfc2crtm (Source File: noah32_sfc2crtm.F90) 1193

76.7.9 noah33_setwrfexport.F90 (Source File: noah33_setwrfexport.F90) 1194
76.7.10 noah33_wrf_f2t (Source File: noah33_wrf_f2t.F90) 1194
76.8 Fortran: Module Interface noah33_dasnow_Mod (Source File: noah33_dasnow_Mod.F90) . . . 1195
76.8.1 noah33_dasnow_init (Source File: noah33_dasnow_Mod.F90) 1195
76.8.2 noah33_descale_snow (Source File: noah33_descale_snow.F90) 1195
76.8.3 noah33_getswfpred (Source File: noah33_getscfpred.F90) 1196
76.8.4 noah33_getsnowpred_PMWsnow (Source File: noah33_getsnowpred_PMWsnow.F90) . 1196
76.8.5 noah33_getsnowvars (Source File: noah33_getsnowvars.F90) 1197
76.8.6 noah33_getsnwdpred (Source File: noah33_getsnwdpred.F90) 1198
76.8.7 noah33_getswepred (Source File: noah33_getswepred.F90) 1198
76.8.8 noah33_map_snow (Source File: noah33_map_snow.F90) 1199
76.8.9 noah33_map_snow_DI (Source File: noah33_map_snow_DI.F90) 1199
76.8.10 noah33_map_swe (Source File: noah33_map_swe.F90) 1200
76.8.11 noah33_qc_PMWsnowobs (Source File: noah33_qc_PMWsnowobs.F90) 1201
76.8.12 noah33_qc_scfobs (Source File: noah33_qc_scfobs.F90) 1202
76.8.13 noah33_qc_snowobs (Source File: noah33_qc_snowobs.F90) 1202
76.8.14 noah33_qcsnow (Source File: noah33_qcsnow.F90) 1203
76.8.15 noah33_scale_snow (Source File: noah33_scale_snow.F90) 1204
76.8.16 noah33_setsnowvars (Source File: noah33_setsnowvars.F90) 1204
76.8.17 noah33_transform_snow (Source File: noah33_transform_snow.F90) 1205
76.8.18 noah33_updatesnowvars (Source File: noah33_updatesnowvars.F90) 1205
76.8.19 noah33_updatesnowvars_scfda (Source File: noah33_updatesnowvars_scfda.F90) 1206
76.8.20 noah33_wrtswe (Source File: noah33_wrtswe.F90) 1207
76.9 Fortran: Module Interface noah33_dasoilm_Mod (Source File: noah33_dasoilm_Mod.F90) . . . 1207
76.9.1 noah33_dasoilm_init (Source File: noah33_dasoilm_Mod.F90) 1208
76.9.2 noah33_descale_soilm (Source File: noah33_descale_soilm.F90) 1208
76.9.3 noah33_getLbandTbPred (Source File: noah33_getLbandTbPred.F90) 1209
76.9.4 noah33_getsmpred (Source File: noah33_getsmpred.F90) 1209
76.9.5 noah33_getsoilm (Source File: noah33_getsoilm.F90) 1209
76.9.6 noah33_qc_soilmobs (Source File: noah33_qc_soilmobs.F90) 1210
76.9.7 noah33_qcsoilm (Source File: noah33_qcsoilm.F90) 1211
76.9.8 noah33_scale_soilm (Source File: noah33_scale_soilm.F90) 1211
76.9.9 noah33_setsoilm (Source File: noah33_setsoilm.F90) 1212
76.9.10 noah33_updatesoilm (Source File: noah33_updatesoilm.F90) 1212
76.9.11 noah33_write_soilm (Source File: noah33_write_soilm.F90) 1213
76.9.12 noah33_getirrigationstates (Source File: noah33_getirrigationstates.F90) 1214
76.9.13 noah33_coldstart (Source File: noah33_coldstart.F90) 1214
76.9.14 noah33_dynsetup (Source File: noah33_dynsetup.F90) 1215
76.9.15 noah33_f2t (Source File: noah33_f2t.F90) 1215
76.9.16 noah33_finalize (Source File: noah33_finalize.F90) 1216
76.10 Fortran: Module Interface noah33_lsmMod (Source File: noah33_lsmMod.F90) 1216
76.10.1 noah33_lsm_ini (Source File: noah33_lsmMod.F90) 1217
76.10.2 noah33_main (Source File: noah33_main.F90) 1218
76.11 Fortran: Module Interface noah33_module (Source File: noah33_module.F90) 1219
76.11.1 noah33_readcrd (Source File: noah33_readcrd.F90) 1220
76.11.2 noah33_readrst (Source File: noah33_readrst.F90) 1221
76.11.3 noah33_reset (Source File: noah33_reset.F90) 1222
76.11.4 noah33_setsoils (Source File: noah33_setsoils.F90) 1222
76.11.5 noah33_settbot (Source File: noah33_settbot.F90) 1223
76.11.6 noah33_resettbot (Source File: noah33_settbot.F90) 1224
76.11.7 noah33_setup (Source File: noah33_setup.F90) 1224

76.11.8 noah33_setvegparms (Source File: noah33_setvegparms.F90) 1225
76.11.9 noah33_resetvegparms (Source File: noah33_setvegparms.F90) 1226
76.11.10 noah33_writerst (Source File: noah33_writerst.F90) 1226
76.11.11 noah33_dump_restart (Source File: noah33_writerst.F90) 1227
76.12 Fortran: Module Interface noah33_peMod (Source File: noah33_peMod.F90) 1229
76.12.1 noah33_setup_pedecvars (Source File: noah33_peMod.F90) 1229
76.12.2 noah33_set_pedecvars (Source File: noah33_set_pedecvars.F90) 1230
76.12.3 randArray (Source File: noah33_set_pedecvars.F90) 1230
76.12.4 noah33_getpeobspred_ARMobs (Source File: noah33_getpeobspred_ARMobs.F90) . . . 1231
76.12.5 noah33_setupobspred_ARMobs (Source File: noah33_setupobspred_ARMobs.F90) . . . 1231
76.12.6 noah33_getpeobspred_FLUXNETobs (Source File: noah33_getpeobspred_FLUXNETobs.F90) 1232
76.12.7 noah33_setupobspred_FLUXNETobs (Source File: noah33_setupobspred_FLUXNETobs.F90) 1232
76.12.8 noah33_getpeobspred_LPRM_AMSREsmObs (Source File: noah33_getpeobspred_LPRMsmobs.F90) 1233
76.12.9 noah33_setupobspred_LPRM_AMSREsmObs (Source File: noah33_setupobspred_LPRMsmobs.F90) 1233
76.12.10 noah33_getpeobspred_USDA_ARSsmObs (Source File: noah33_getpeobspred_USDA_ARSsmobs.F90) 1234
76.12.11 noah33_setupobspred_USDA_ARSsmObs (Source File: noah33_setupobspred_USDA_ARSsmobs.F90) 1234
76.12.12 noah33_getpeobspred_template (Source File: noah33_getpeobspred_template.F90) . . . 1235
76.12.13 noah33_setupobspred_template (Source File: noah33_setupobspred_template.F90) . . . 1235
76.12.14 noah33_getrunoffs (Source File: noah33_getrunoffs.F90) 1236
76.12.15 noah33_getrunoffs_mm (Source File: noah33_getrunoffs_mm.F90) 1236
76.12.16 noah33_sfc2cmem3 (Source File: noah33_sfc2cmem3.F90) 1237
76.12.17 noah33_sfc2crtm (Source File: noah33_sfc2crtm.F90) 1237
76.12.18 RDHM356_coldstart (Source File: RDHM356_coldstart.F90) 1238
76.12.19 RDHM356_dynsetup (Source File: RDHM356_dynsetup.F90) 1238
76.12.20 RDHM356_f2t (Source File: RDHM356_f2t.F90) 1239
76.12.21 RDHM356_finalize (Source File: RDHM356_finalize.F90) 1240
76.13 Fortran: Module Interface RDHM356_lsmMod (Source File: RDHM356_lsmMod.F90) 1240
76.13.1 RDHM356_ini (Source File: RDHM356_lsmMod.F90) 1243
76.13.2 RDHM356_main (Source File: RDHM356_main.F90) 1244
76.14 Fortran: Module Interface RDHM356_module (Source File: RDHM356_module.F90) 1245
76.14.1 RDHM356_readcrd (Source File: RDHM356_readcrd.F90) 1249
76.14.2 RDHM356_readrst (Source File: RDHM356_readrst.F90) 1250
76.14.3 RDHM356_setup (Source File: RDHM356_setup.F90) 1251
76.14.4 RDHM356_read_multilevel_param (Source File: RDHM356_setup.F90) 1252
76.14.5 RDHM356_writerst (Source File: RDHM356_writerst.F90) 1253
76.14.6 RDHM356_dump_restart (Source File: RDHM356_writerst.F90) 1254

77 Template **1255**
77.0.7 template_dynsetup (Source File: template_dynsetup.F90) 1256
77.0.8 template_f2t (Source File: template_f2t.F90) 1256
77.0.9 template_finalize (Source File: template_finalize.F90) 1257
77.0.10 template_getrunoffs (Source File: template_getrunoffs.F90) 1257
77.1 Fortran: Module Interface template_lsmMod (Source File: template_lsmMod.F90) 1257
77.1.1 template_lsm_ini (Source File: template_lsmMod.F90) 1258
77.1.2 template_main (Source File: template_main.F90) 1258
77.1.3 template_main.F90 (Source File: template_main.F90) 1258
77.2 Fortran: Module Interface template_module.F90 (Source File: template_module.F90) 1259
77.2.1 template_readcrd (Source File: template_readcrd.F90) 1260
77.2.2 template_readrst (Source File: template_readrst.F90) 1260
77.2.3 template_setup (Source File: template_setup.F90) 1260
77.2.4 template_writerst (Source File: template_writerst.F90) 1261

78 VIC land surface model version 4.1.1 **1261**
78.1 Notes regarding incorporating VIC 4.1.1 into LIS . . . 1261
78.2 Limitations of VIC 4.1.1 within LIS . . . 1262
78.2.1 vic411_diagnoseoutputvar (Source File: vic411_diag.F90) . . . 1262
78.2.2 vic411_dynsetup (Source File: vic411_dynsetup.F90) . . . 1263
78.2.3 vic411_f2t (Source File: vic411_f2t.F90) . . . 1263
78.2.4 vic411_finalize (Source File: vic411_finalize.F90) . . . 1264
78.3 Fortran: Module Interface vic411_lsmMod (Source File: vic411_lsmMod.F90) . . . 1264
78.3.1 vic411_lsm_ini (Source File: vic411_lsmMod.F90) . . . 1265
78.3.2 vic411_main (Source File: vic411_main.F90) . . . 1266
78.4 Fortran: Module Interface vic411_module.F90 (Source File: vic411_module.F90) . . . 1266
78.4.1 vic411_readcard (Source File: vic411_readcard.F90) . . . 1267
78.4.2 vic411_readrst (Source File: vic411_readrst.F90) . . . 1267
78.4.3 vic411_setup (Source File: vic411_setup.F90) . . . 1268
78.4.4 vic411_writerst (Source File: vic411_writerst.F90) . . . 1268
78.4.5 finalize_vic411 (Source File: finalize_vic411.c) . . . 1269
78.4.6 setup_vic411 (Source File: setup_vic411.c) . . . 1269
78.4.7 vic411_add_atmosdata (Source File: vic411_add_atmosdata.c) . . . 1270
78.4.8 vic411_nr_atmosdata (Source File: vic411_add_atmosdata.c) . . . 1270
78.4.9 find_min_tfactor (Source File: vic411_find_min_tfactor.c) . . . 1271
78.4.10 fsvp (Source File: vic411_fsvp.c) . . . 1271
78.4.11 get_max_snow_temp (Source File: vic411_get_max_snow_temp.c) . . . 1272
78.4.12 vic411_lis_read_soilparam_arc (Source File: vic411_lis_read_soilparam_arc.c) . . . 1272
78.4.13 lis_scan_soilparam_flat_ascii (Source File: vic411_lis_scan_soilparam_flat_ascii.c) . . . 1272
78.4.14 vic411_run (Source File: vic411_run.c) . . . 1273
78.4.15 vic411_set_vegparam (Source File: vic411_set_vegparam.c) . . . 1276
78.4.16 vic411_write_atmos (Source File: vic411_write_atmos.c) . . . 1276
78.4.17 vic411_bin_atmosdata (Source File: vic411_write_bin_atmosdata.c) . . . 1277
78.4.18 write_lakecon (Source File: vic411_write_lakecon.c) . . . 1277
78.4.19 write_prcp (Source File: vic411_write_prcp.c) . . . 1277
78.4.20 vic411_write_state (Source File: vic411_write_state.c) . . . 1278

79 VIC land surface model version 4.1.2.l **1278**
79.1 Notes regarding incorporating VIC 4.1.2.l into LIS . . . 1279
79.2 Limitations of VIC 4.1.2.l within LIS . . . 1279
79.2.1 vic412_diagnoseoutputvar (Source File: vic412_diag.F90) . . . 1279
79.2.2 vic412_dynsetup (Source File: vic412_dynsetup.F90) . . . 1280
79.2.3 vic412_f2t (Source File: vic412_f2t.F90) . . . 1280
79.2.4 vic412_finalize (Source File: vic412_finalize.F90) . . . 1281
79.3 Fortran: Module Interface vic412_lsmMod (Source File: vic412_lsmMod.F90) . . . 1281
79.3.1 vic412_lsm_ini (Source File: vic412_lsmMod.F90) . . . 1283
79.3.2 vic412_main (Source File: vic412_main.F90) . . . 1283
79.4 Fortran: Module Interface vic412_module.F90 (Source File: vic412_module.F90) . . . 1283
79.4.1 vic412_readcard (Source File: vic412_readcard.F90) . . . 1284
79.4.2 vic412_readrst (Source File: vic412_readrst.F90) . . . 1284
79.4.3 vic412_setup (Source File: vic412_setup.F90) . . . 1285
79.4.4 vic412_writerst (Source File: vic412_writerst.F90) . . . 1285
79.4.5 finalize_vic412 (Source File: finalize_vic412.c) . . . 1286
79.4.6 find_min_tfactor (Source File: find_min_tfactor.c) . . . 1286
79.4.7 fsvp (Source File: fsvp.c) . . . 1287
79.4.8 get_max_snow_temp (Source File: get_max_snow_temp.c) . . . 1287

79.4.9 lis_scan_soilparam_flat_ascii (Source File: lis_scan_soilparam_flat_ascii.c) 1287
79.4.10 set_vegparam (Source File: set_vegparam.c) 1288
79.4.11 setup_vic412 (Source File: setup_vic412.c) 1289
79.4.12 vic412_add_atmosdata (Source File: vic412_add_atmosdata.c) 1290
79.4.13 vic412_nr_atmosdata (Source File: vic412_add_atmosdata.c) 1290
79.4.14 vic412_initialize (Source File: vic412_intialize.c) 1291
79.4.15 vic412_run (Source File: vic412_run.c) 1291
79.4.16 vic412_write_atmos (Source File: vic412_write_atmos.c) 1294
79.4.17 vic412_write_state (Source File: vic412_write_state.c) 1294
79.4.18 vic412_bin_atmosdata (Source File: write_bin_atmosdata.c) 1295
79.4.19 write_lakecon (Source File: write_lakecon.c) 1296
79.4.20 write_prcp (Source File: write_prcp.c) 1296

Part I
LIS Overview

1 Overview of LIS

Land Information System (LIS) is a flexible land surface modeling and data assimilation framework developed with the goal to integrate satellite- and ground-based observational data products and advanced land surface modeling techniques to produce optimal fields of land surface states and fluxes. The LIS infrastructure provides the modeling tools to integrate these observations with model forecasts to generate improved estimates of land surface conditions such as soil moisture, evaporation, snow pack, and runoff, at 1km and finer spatial resolutions and at one-hour and finer temporal resolutions. The fine scale spatial modeling capability of LIS allows it take advantage of the EOS-era observations, such as MODIS leaf area index, snow cover, and surface temperature, at their full native resolution. LIS features a high performance and flexible design, provides infrastructure for data integration and assimilation, and operates on an ensemble of land surface models (LSM) for extension over user-specified regional or global domains. LIS is designed using advanced software engineering principles to enable reuse and community sharing of modeling tools, data resources, and assimilation algorithms. The system is designed as an object-oriented framework, with abstractions defined for customization and extension to different applications. These extensible interfaces allow the incorporation of new domains, LSMs, land surface parameters, meteorological inputs, data assimilation and optimization algorithms. The extensible nature of these interfaces and the component style specification of the system allow rapid prototyping and development of new applications. These features enable LIS to serve both as a Problem Solving Environment (PSE) for hydrologic research to enable accurate global water and energy cycle predictions, and as a Decision Support System (DSS) to generate useful information for application areas including disaster management, water resources management, agricultural management, numerical weather prediction, air quality and military mobility assessment.

Part II
LIS offline program

2 LIS offline program

2.0.1 lisdrv (Source File: lisdrv.F90)

Main program for LIS in an offline simulation
Main driver program for LIS. It performs four main functions

LIS_config_init calls the routines to read the runtime configurations

LIS_Init calls the initializations based on the runmode

LIS_run calls the run routines based on the runmode

LIS_finalize calls the cleanup routines

The routines invoked are :

LIS_config_init (3.10.2)
call to initialize configuration tool and read model independent options

lisinit (3.57.2)
call to initialize lis based on the runmode

lisrun (3.57.4)
call to run lis based on the runmode

lisfinalize (3.57.6)
call to cleanup lis structures based on the runmode

REVISION HISTORY:

```
    14Nov02    Sujay Kumar  Initial Specification
    21Oct05    Sujay Kumar  Modified to include the runmodes. Switched
                            to a init,run,finalize mode
 program lisdrv
```

USES:

```
   use LIS_coreMod, only : LIS_config_init, LIS_rc

  call LIS_config_init
  call lisinit(trim(LIS_rc%runmode)//char(0))
  call lisrun(trim(LIS_rc%runmode)//char(0))
  call lisfinalize(trim(LIS_rc%runmode)//char(0))
```

Part III
LIS core structures and methods

3 LIS core structures and methods

LIS core, the central part of the software, is the infrastructure that integrates the use of different components in LIS. The functions performed by LIS core include operations related to the overall control, runtime statistics, inter-language support, error logging and dynamic memory management functions. Routines to manage domain decomposition, load balancing, fault tolerance, etc. are also encapsulated as generic routines in the high performance computing and communications (HPCC) component. The time management tools in LIS provides useful functions for time and data calculations and higher level functions to control model timestepping and alarms. Another tool implemented in the LIS core structure is the generic I/O tool, which provides capabilities to read the input data locally, handling different data formats. also provide support for distributed data output and multiple formats. Other miscellaneous tools incorporated in the LIS core include methods to perform spatial and temporal interpolation, reprojection, domain subsetting etc. The abstractions providing representations for the behavior of LSMs, domains, and data, runmodes, and data assimilation are also incorporated in the LIS core.

3.1 Fortran: Module Interface LIS_DAobservationsMod (Source File: LIS_DAobservationsM

The code in this file controls the handling of observations to be used for data assimilation.

REVISION HISTORY:

```
 21 Jun 2006: Sujay Kumar; Initial implementation
use ESMF

implicit none

PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
 public ::  LIS_initDAobservations
 public ::  LIS_readDAobservations
 public ::  LIS_perturb_DAobservations
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
 public :: LIS_OBS_State
 public :: LIS_OBS_Pert_State

 type(ESMF_State), allocatable :: LIS_OBS_State(:,:)
 type(ESMF_State), allocatable :: LIS_OBS_Pert_State(:,:)
```

3.1.1 LIS_initDAobservations (Source File: LIS_DAobservationsMod.F90)

INTERFACE:

```
subroutine LIS_initDAobservations
```

DESCRIPTION:

This routine allocates the structures required for managing observational data used for data assimilation. The routine creates ESMF State objects used to incorporate observational data.
The methods invoked are:

LIS_DAobs_plugin (4.1.1)
sets up function table registries for implemented observation sources to be assimilated.

readObsDataConfig (3.42.16)
invokes the generic method in the registry to set up the structures to read the observation data, for assimilation.

perturbinit (3.54.2)
invokes the generic method in the registry to initialize the observation perturbation algorithm.

perturbsetup (3.54.4)
invokes the generic method in the registry to populate the observation perturbation objects.

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_logunit, LIS_verify
use LIS_DAobs_pluginMod, only : LIS_DAobs_plugin
```

3.1.2 LIS_readDAobservations (Source File: LIS_DAobservationsMod.F90)

INTERFACE:

```
subroutine LIS_readDAobservations(n)
```

USES:

```
use LIS_coreMod,   only : LIS_rc
```

ARGUMENTS:

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

DESCRIPTION:

This routines calls the method to read the specific observation source.
The methods invoked are:

readDAObs (3.42.18)
invokes the method in the registry to read the observations and incorporate into the observation state.

3.1.3 LIS_perturb_DAobservations (Source File: LIS_DAobservationsMod.F90)

INTERFACE:

```
subroutine LIS_perturb_DAobservations(n)
```

USES:

```
use LIS_coreMod,   only : LIS_rc
use LIS_logMod,    only : LIS_verify
```

ARGUMENTS:

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

DESCRIPTION:

This routines calls the method to perturb the observations based on the specified choice.
The methods invoked are:

perturbmethod (3.54.6)
invokes the specified algorithm to perturb the observations.

writeDAObs (3.42.20)
invokes the registry method to write the perturbed and processed observations to disk.

REVISION HISTORY:

```
14 Nov 2002    Sujay Kumar  Initial Specification
26 May 2011    Soni Yatheendradas    Potential ET variables added For FEWSNET
14 Mar 2014    David Mocko: Added CAPE, CH, and CM to forcing variables

use ESMF

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------

-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------

public :: LIS_FORC_Tair
public :: LIS_FORC_Qair
public :: LIS_FORC_SWdown
public :: LIS_FORC_SWdirect
public :: LIS_FORC_SWdiffuse
public :: LIS_FORC_LWdown
```

```
 public :: LIS_FORC_Wind_E
 public :: LIS_FORC_Wind_N
 public :: LIS_FORC_Psurf
 public :: LIS_FORC_Rainf
 public :: LIS_FORC_Snowf
 public :: LIS_FORC_CRainf

 public :: LIS_FORC_Forc_Hgt
 public :: LIS_FORC_Ch
 public :: LIS_FORC_Cm
 public :: LIS_FORC_Emiss
 public :: LIS_FORC_Q2sat
 public :: LIS_FORC_Cosz
 public :: LIS_FORC_Alb
 public :: LIS_FORC_XICE
 public :: LIS_FORC_QSFC

 public :: LIS_FORC_CHS2
 public :: LIS_FORC_CQS2
 public :: LIS_FORC_T2
 public :: LIS_FORC_Q2
 public :: LIS_FORC_TH2
 public :: LIS_FORC_TMN

 public :: LIS_FORC_LPressure
 public :: LIS_FORC_O3    !absorber

 public :: LIS_FORC_PET ! SY for FEWSNET
 public :: LIS_FORC_RefET ! SY for FEWSNET

 public :: LIS_FORC_CMFORC ! dmm
 public :: LIS_FORC_CHFORC ! dmm for NLDAS-2
 public :: LIS_FORC_CAPE ! dmm for NLDAS-2

 public :: LIS_FORC_PARDR
 public :: LIS_FORC_PARDF
 public :: LIS_FORC_SWNET
<for vic>
 public :: LIS_FORC_SNOWFLAG
 public :: LIS_FORC_DENSITY
 public :: LIS_FORC_VAPORPRESS
 public :: LIS_FORC_VAPORPRESSDEFICIT
 public :: LIS_FORC_WIND
</for vic>

ccc - for CABLE
 public :: LIS_FORC_CO2
```

3.2 Fortran: Module Interface LIS_LMLCMod (Source File: LIS_LMLCMod.F90)

The code in this file implements routines to read various sources of landmask and landcover data.

3.2.1 Overview

This module provides routines for reading and modifying landmask and landcover data.

REVISION HISTORY:

```
 18 Jul 2008: Sujay Kumar; Initial implementation
 3  Apr 2012: Sujay Kumar; Switched to the use of LPT based parameter file
use LIS_fileIOMod

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
 public :: LIS_LMLC_init          ! initializes data structures and memory
 public :: LIS_diagnoselandmask   ! diagnoses the LIS landmask for
                                  ! history output
 public :: LIS_diagnoselandcover  ! diagnoses the LIS landcover data for
                                  !  history output

-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
 public :: LIS_LMLC  ! derived datatype that stores the landmask and landcover data
```

3.2.2 LIS_LMLC_init (Source File: LIS_LMLCMod.F90)

INTERFACE:

```
subroutine LIS_LMLC_init()
```

USES:

```
  use ESMF
  use LIS_coreMod,  only : LIS_rc, LIS_config, LIS_domain
  use LIS_logMod,   only : LIS_verify, LIS_logunit
  use LIS_fileIOMod, only : LIS_readDomainConfigSpecs
```

DESCRIPTION:

Reads the configurable options related to the landmask and landcover datasets (filenames, geographical extent information) and allocates memory for data structures for reading landmask and landcover datasets

---

3.2.3 LIS_diagnoselandmask (Source File: LIS_LMLCMod.F90)

INTERFACE:

```
subroutine LIS_diagnoselandmask(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_histDataMod
```

ARGUMENTS:

```
implicit none
integer, intent(in)   :: n
```

DESCRIPTION:

This routine diagnoses the LIS landmask to be used later for history output
The arguments are:

n index of the nest

The routines invoked are:

LIS_diagnoseOutputVar (3.16.5)
This routine maps landmask data to the history writing routines

---

3.2.4 LIS_diagnoseLandcover (Source File: LIS_LMLCMod.F90)

INTERFACE:

```
subroutine LIS_diagnoseLandcover(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_histDataMod
```

ARGUMENTS:

```
implicit none
integer, intent(in)   :: n
```

DESCRIPTION:

This routine diagnoses the LIS landcover to be later used for history output
The arguments are:

n index of the nest

The routines invoked are:

LIS_diagnoseOutputVar (3.16.5)
This routine maps landcover data to the history writing routines

---

3.2.5 read_landcover (Source File: LIS_LMLCMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_landcover(n)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
  use LIS_coreMod,        only : LIS_rc, LIS_localPet,&
       LIS_ews_ind, LIS_ewe_ind,&
       LIS_nss_ind, LIS_nse_ind, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
       LIS_nss_halo_ind, LIS_nse_halo_ind
  use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
       LIS_releaseUnitNumber, LIS_endrun, LIS_verify

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine reads the landcover data
The arguments are:

n index of n

locallc landlc for the region of interest

3.2.6 read_surfacetype (Source File: LIS_LMLCMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_surfacetype(n)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
  use LIS_coreMod,        only : LIS_rc, LIS_localPet,&
       LIS_ews_ind, LIS_ewe_ind,&
       LIS_nss_ind, LIS_nse_ind, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
       LIS_nss_halo_ind, LIS_nse_halo_ind
  use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
       LIS_releaseUnitNumber, LIS_endrun, LIS_verify

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine reads the surfacetype data
The arguments are:

n index of n

locallc landlc for the region of interest

3.3 Fortran: Module Interface LIS_PE_HandlerMod (Source File: LIS_PE_HandlerMod.F90

The code in this file implements methods to handle the program flow related to parameter estimation (PE). The module supports parameter estimation of different model types (LSMs, RTMs, Routing, Landslide models) using any of the optimization and uncertainty estimation algorithms in LIS.

REVISION HISTORY:

```
 21 Jun 2009: Sujay Kumar; Initial implementation
 12 Jan 2012: Sujay Kumar; Implemented an updated version that includes
                           generic support for different model types.

!NOTES: TBD - There has to be one additional step where we assemble the
individual patches from the surface models and send to the optimization
algorithm. The OPT/UE algorithm is assumed to work in the tile space.
use ESMF

implicit none

PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
 public ::  LIS_PE_init                !initializes the PE handler
 public ::  LIS_readPEobs              !read observations for PE
 public ::  LIS_updatePEObjectiveFunc  !updates the objective function
 public ::  LIS_computePEObjectiveFunc !computes objective function
 public ::  LIS_setPEdecisionSpace     !sets the decision space
 public ::  LIS_resetPEObjectiveFunc   !resets objective function
 public ::  LIS_resetPEobs             !resets the PE observations
 public ::  LIS_PE_restart             !reads a PE restart file
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
 public :: LIS_PEOBS_State     !object to store observations for PE
 public :: LIS_PEOBSPred_State !object to store the model simulated values
                               !that correspond to the PE observations
```

3.3.1 LIS_PE_init (Source File: LIS_PE_HandlerMod.F90)

INTERFACE:

```
subroutine LIS_PE_init
```

DESCRIPTION:

This routine allocates the structures required for managing observational data, the corresponding "obs predictor (obspred)" from the models and the decision space (list of variables being adjusted) for parameter estimation.
The methods invoked are:

setupPEOBSSpace (3.53.18)
invokes the setup method for the specified PE observational source

setupPEOBSPredSpace (3.3.2)
invokes the setup method for the specified PE "obspred" - the simulated value of observation

setupDecSpaceVars (3.3.3)
invokes the methods for setting up the decision space variables

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod,  only : LIS_logunit, LIS_verify
use LIS_timeMgrMod, only : LIS_date2time
```

3.3.2 setupPEOBSPredSpace (Source File: LIS_PE_HandlerMod.F90)

INTERFACE:

```
subroutine setupPEOBSPredSpace()
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

DESCRIPTION:

This routine cycles through different model types to setup the OBSPred object. Depending on which model types are chosen in the optimization, the setup method for those model types are invoked.
The methods invoked are:

lsmpesetupobspredspace (3.50.54)
sets up the LSM's ObsPred object

rtmpesetupobspredspace (3.39.16)
sets up the RTM's ObsPred object

3.3.3 setupDecSpaceVars (Source File: LIS_PE_HandlerMod.F90)

INTERFACE:

```
subroutine setupDecSpaceVars()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_optUEMod, only : LIS_decisionSpace,LIS_feasibleSpace
```

DESCRIPTION:

This routine cycles through different model types to setup the decision space object. Depending on which model types are chosen in the optimization, the methods to setup decision space variables for those model types are invoked.
The methods invoked are:

lsmpesetupdecisionspace (3.50.52)
sets up the LSM's decision space object

3.3.4 LIS_readPEobs (Source File: LIS_PE_HandlerMod.F90)

INTERFACE:

```
subroutine LIS_readPEobs(n)
```

USES:

```
use LIS_coreMod,   only : LIS_rc
```

ARGUMENTS:

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

DESCRIPTION:

This routines calls the method to read the specific observation source.
The methods invoked are:

getPEObs (3.53.22)
invokes the method to read the observations and incorporate into the observation state object

writePEObs (3.53.24)
invokes the method to write the processed PE observations to disk

---

3.3.5 LIS_updatePEObjectiveFunc (Source File: LIS_PE_HandlerMod.F90)

INTERFACE:

```
subroutine LIS_updatePEObjectiveFunc(n)
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc
use LIS_logMod,    only : LIS_verify
use LIS_optUEMod,  only : LIS_objectiveFunc
```

ARGUMENTS:

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

DESCRIPTION:

Invokes the call to update the specified objective function
The methods invoked are:

getPEObsPred (3.3.6)
invokes the method to retrieve the model simulated value of the PE observations

updateObjectiveFunc (3.53.30)
invokes the method to update the objective function based on the PE observations and the corresponding obspreds.

3.3.6 getPEOBSPred (Source File: LIS_PE_HandlerMod.F90)

INTERFACE:

```
subroutine getPEOBSPred()
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

DESCRIPTION:

This routine cycles through different model types to generate the obspred object, depending on which model types are chosen in the optimization.
The methods invoked are:

lsmpegetobspred (3.50.56)
invokes the method to retrieve obsPred object from the LSM.

rtmpegetobspred (3.39.18)
invokes the method to retrieve obsPred object from the RTM.

3.3.7 LIS_computePEObjectiveFunc (Source File: LIS_PE_HandlerMod.F90)

INTERFACE:

```
subroutine LIS_computePEObjectiveFunc
```

USES:

```
use LIS_coreMod,    only : LIS_rc
```

DESCRIPTION:

This routine calls the method to compute the objective function prior to calling the optimization/uncertainty estimation algorithm
The methods invoked are:

computeObjectiveFuncType (3.53.28)
invokes the objective function computation based on the specified method

---

3.3.8 LIS_setPEDecisionSpace (Source File: LIS_PE_HandlerMod.F90)

INTERFACE:

```
subroutine LIS_setPEDecisionSpace()
```

USES:

```
use LIS_coreMod,  only : LIS_rc
```

DESCRIPTION:

This routine invokes the calls to retrieve the decision space variables from the optimization/uncertainty estimation algorithm and assigns them to the model structures.
The methods invoked are:

getOptUEAlgnparam (3.53.14)
invokes the method to return the number of parameters in the decision space from the optimization/UE algorithm

setDecisionSpace (3.3.9)
invokes the method to assign the reconciled decision space variables to the respective model types.

3.3.9 setDecisionSpace (Source File: LIS_PE_HandlerMod.F90)

INTERFACE:

```
subroutine setDecisionSpace()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_optUEMod, only : LIS_decisionSpace, LIS_feasibleSpace
```

DESCRIPTION:

This routine cycles through different model types to set the decision space variables back to the respective models.
The methods invoked are:

lsmpesetdecisionspace (3.50.48)
invokes the method to return decision space object from the LSM.

3.3.10 LIS_resetPEobjectiveFunc (Source File: LIS_PE_HandlerMod.F90)

INTERFACE:

```
subroutine LIS_resetPEobjectiveFunc
```

USES:

```
use LIS_coreMod,     only : LIS_rc
```

DESCRIPTION:

This routine calls the method to reset the objective function for the next iteration of the optimization/uncertainty estimation algorithm.
The methods invoked are:

resetObjectiveFuncType (3.53.32)
invokes the method to reset the objective function objects

3.3.11 LIS_resetPEobs (Source File: LIS_PE_HandlerMod.F90)

INTERFACE:

```
subroutine LIS_resetPEobs
```

USES:

```
use LIS_coreMod,     only : LIS_rc
```

DESCRIPTION:

This routine calls the method to reset the observation data structures for the next iteration of the optimization/uncertainty estimation algorithm.
The methods invoked are:

resetpeobsspace (3.53.20)
invokes the method to reset the obs state objects

3.3.12 LIS_PE_restart (Source File: LIS_PE_HandlerMod.F90)

INTERFACE:

```
subroutine LIS_PE_restart
```

USES:

```
use LIS_optUEMod, only : LIS_optUEAlg_readrestart
```

DESCRIPTION:

This routine invokes the method to read the restart file for the optimization algorithm. If a restart file is read, then the call to update the model's decision space variables is also invoked.
The methods invoked are:

LIS_optUEAlg_readrestart (3.27.7)
invokes the method to read the restart file of the OPT/UE algorithm.

LIS_setPEdecisionSpace (3.3.8) calls the method to set the decision space variables in the model.

3.4 Fortran: Module Interface LIS_PRIV_gridMod (Source File: LIS_PRIV_gridMod.F90)

The code in this file provides a description of the grid data structure in LIS

3.4.1 Overview

This module contains the grid data structure used in LIS. The data structure contains the variables specified for a single grid cell. It includes:

lat latitude of the grid cell

lon longitude of the grid cell

col index of the grid point along the East-West grid dimension

row index of the grid point along the North-South grid dimension

elev Topological elevation of the grid cell

slope Topological slope of the grid cell

aspect Topological aspect of the grid cell

curv Topological curvature of the grid cell

ntiles Number of sub-grid tiles contained within this grid cell

subgrid_tiles Array of tile indices corresponding to the sub-grid tiles

REVISION HISTORY:

```
14 Nov 2002: Sujay Kumar; Optimized version of grid representation
```

3.5 Fortran: Module Interface LIS_PRIV_rcMod (Source File: LIS_PRIV_rcMod.F90)

Module for specifying model independent variables in LIS. The module does not contain any variables that are specific to the extensible components in LIS. The module specifies variables for overall run control, options for the choice of parameter datasets, data assimilation choices, and the variables controlling the time management in LIS.
The variables specified in this module include:

runmode choice of running mode in LIS

nnest number of nests or running instances (1 or higher)

ntiles This array stores the size of tilespace of the running domain, for each processor, for each nest

glbntiles This array stores the size of the tilespace of the overall running domain for each nest (including the halo regions)

glbntiles_red This array stores the size of the tilespace of the overall running domain for each nest (excluding the halo regions)

ngrid This array stores the size of gridspace of the running domain, for each processor, for each nest

glbngrid This array stores the size of the gridspace of the overall running domain for each nest

glbngrid_red This array stores the size of the gridspace of the overall running domain for each nest (excluding the halo regions)

gnc Array containing the East-West grid dimension of the overall running grid for each nest

gnr Array containing the North-South grid dimension of the overall running grid for each nest

lnc Array containing the East-West grid dimension of the running grid for each processor, for each nest (including the halo regions)

lnr Array containing the North-South grid dimension of the running grid for each processor, for each nest (including the halo regions)

lnc_red Array containing the East-West grid dimension of the running grid for each processor, for each nest (excluding the halo regions)

lnr_red Array containing the North-South grid dimension of the running grid for each processor, for each nest (excluding the halo regions)

pnc Array containing the East-West grid dimension of the parameter space for each nest, from which data is subsetted.

pnr Array containing the North-South grid dimension of the parameter space for each nest, from which data is subsetted.

ncatg Total number of land tiles in the catchment-based parameter files per nest.

lis_map_proj Choice of map projection used in LIS.

lsm Choice of the land surface model in LIS.

param_proj Map projection type of parameter datasets.

texturemap boolean value to denote if soil texture map should be used (0- do not use, 1-use). If this value is set to 1, the sand, clay and silt fraction maps will not be read. Otherwise the texture data is read the fraction data will not be used.

vegsrc Choice of the source of landcover data source

soilsrc Choice of the source of soil parameter data source

colorsrc Choice of the soil color data source(0-do not use)

elevsrc Choice of the elevation data source (0-do not use)

slopesrc Choice of the slope data source (0-do not use)

aspectsrc Choice of the aspect data source (0-do not use)

curvsrc Choice of the curvature data source (0-do not use)

albedosrc Choice of the albedo data source (0-do not use)

porositysrc Choice of the porosity data source (0-do not use)

psisatsrc Choice of the saturated matric potential data source (0-do not use)

ksatsrc Choice of the saturated hydraulic conductivity data source (0-do not use)

bexpsrc Choice of the b parmeter data source (0-do not use)

quartzsrc Choice of the quartz data source (0-do not use)

laisrc Choice of the LAI data source (0-do not use)

saisrc Choice of the SAI data source (0-do not use)

tbotsrc Choice of the TBOT data source (0-do not use)

shdmaxsrc Choice of the SHDMAX data source (0-do not use)

shdminsrc Choice of the SHDMIN data source (0-do not use)

slopetypesrc Choice of the slope type data source (0-do not use)

snowsrc whether snow data sources are being stored (0 - do not use)

surface_maxt Maximum number of tiles per grid to be considered for subgrid tiling

surface_minp Minumum cutoff percentage of vegetation distribution for subgrid tiling

npesx Number of processors in the East-West dimensions for processor layout

npesy Number of processors in the North-South dimensions for processor layout

halox Halo size (in grid points) in the East-West dimensions for processor layout

haloy Halo size (in grid points) in the North-South dimensions for processor layout

udef Value to be used as undefined variable

gridDesc Array describing the running grid and the parameter grid specification, for each nest. The size of the array is 50 for a particular nest. The first 30 locations of this array are used to specify the running grid and the rest are used to specify parameter grid. Note that though run domains can be specified in several map projections, specification of the parameter domains are currently only allowed for lat/lon, gaussian, and UTM projections. For e.g., if the user specifies a running domain in lambert, the parameter domain is expected to be in one of the supported grids for parameter domain specification. The array indices for different map projections are defined as follows:

Lat/lon or geographic projection gridDesc(1) = 0 ; indicates lat/lon projection
gridDesc(2) = number of columns in the domain ;
gridDesc(3) = number of rows in the domain ;
gridDesc(4) = latitude of the lower left corner of the domain
gridDesc(5) = longitude of the lower left corner of the domain
gridDesc(6) = 128 ; not used
gridDesc(7) = latitude of the upper right corner of the domain
gridDesc(8) = longitude of the upper right corner of the domain
gridDesc(9) = spatial resolution (in degrees) along the E-W dimension
gridDesc(10) = spatial resolution (in degrees) along the N-S dimension
gridDesc(11) = 64 ; not used
gridDesc(20) = 255 ; used to specify the ordering of data (non-divisible by 32 indicates E-W ordering else N-S ordering
gridDesc(30) = 0 ; indicates lat/lon projection gridDesc(32) = number of columns in the domain ;
gridDesc(33) = number of rows in the domain ;
gridDesc(34) = latitude of the lower left corner of the domain
gridDesc(35) = longitude of the lower left corner of the domain
gridDesc(36) = 128 ; not used
gridDesc(37) = latitude of the upper right corner of the domain
gridDesc(38) = longitude of the upper right corner of the domain
gridDesc(39) = spatial resolution (in degrees) along the E-W dimension
gridDesc(40) = spatial resolution (in degrees) along the N-S dimension

Mercator projection gridDesc(1) = 1 ; indicates mercator projection
gridDesc(2) = number of columns in the domain ;
gridDesc(3) = number of rows in the domain ;
gridDesc(4) = latitude of the lower left corner of the domain
gridDesc(5) = longitude of the lower left corner of the domain
gridDesc(6) = 8 ; not used
gridDesc(7) = 0.0; not used
gridDesc(8) = spatial resolution (in km) along E-W dimension
gridDesc(9) = spatial reolution (in km) along N-S dimension
gridDesc(10) = true latitude 1
gridDesc(11) = standard longitude
gridDesc(20) = 255 ; used to specify the ordering of data (non-divisible by 32 indicates E-W ordering else N-S ordering
gridDesc(30) = TBD
gridDesc(32) = TBD
gridDesc(33) = TBD
gridDesc(34) = TBD
gridDesc(35) = TBD
gridDesc(36) = TBD
gridDesc(37) = TBD
gridDesc(38) = TBD
gridDesc(39) = TBD
gridDesc(40) = TBD

Lambert conformal projection gridDesc(1) = 3 ; indicates lambert projection
gridDesc(2) = number of columns in the domain ;
gridDesc(3) = number of rows in the domain ;
gridDesc(4) = latitude of the lower left corner of the domain
gridDesc(5) = longitude of the lower left corner of the domain
gridDesc(6) = 8 ; not used
gridDesc(7) = true latitude 2
gridDesc(8) = spatial resolution (in km) along E-W dimension
gridDesc(9) = spatial reolution (in km) along N-S dimension
gridDesc(10) = true latitude 1
gridDesc(11) = standard longitude
gridDesc(20) = 255 ; used to specify the ordering of data (non-divisible by 32 indicates E-W ordering else N-S ordering
gridDesc(30) = TBD
gridDesc(32) = TBD
gridDesc(33) = TBD
gridDesc(34) = TBD
gridDesc(35) = TBD
gridDesc(36) = TBD
gridDesc(37) = TBD
gridDesc(38) = TBD
gridDesc(39) = TBD
gridDesc(40) = TBD

Gaussian projection gridDesc(1) = 4 ; indicates gaussian projection
gridDesc(2) = number of columns in the domain ;
gridDesc(3) = number of rows in the domain ;
gridDesc(4) = latitude of the lower left corner of the domain
gridDesc(5) = longitude of the lower left corner of the domain
gridDesc(6) = 8 ; not used
gridDesc(7) = latitude of the upper right corner of the domain

gridDesc(8) = longitude of the upper right corner of the domain
gridDesc(9) = spatial resolution (in degrees) along the E-W dimension
gridDesc(10) = number of latitude circles
gridDesc(11) = 64 ; not used
gridDesc(20) = 255 ; used to specify the ordering of data (non-divisible by 32 indicates E-W ordering else N-S ordering
gridDesc(41) = 4
gridDesc(42) = number of columns in the domain
gridDesc(43) = number of rows in the domain
gridDesc(44) = latitude of the lower left corner point
gridDesc(45) = longitude of the lower left corner point
gridDesc(46) = 128; not used
gridDesc(47) = latitude of the upper right corner point
gridDesc(48) = latitude of the upper right corner point
gridDesc(49) = spatial resolution in the E-W dimension
gridDesc(50) = number of latitude circles

polar stereographic projection gridDesc(1) = 5 ; indicates polar stereographic projection
gridDesc(2) = number of columns in the domain ;
gridDesc(3) = number of rows in the domain ;
gridDesc(4) = latitude of the lower left corner of the domain
gridDesc(5) = longitude of the lower left corner of the domain
gridDesc(6) = 8 ; not used
gridDesc(7) = orientation of the grid
gridDesc(8) = spatial resolution (in km) along E-W dimension
gridDesc(9) = spatial reolution (in km) along N-S dimension
gridDesc(10) = true latitude
gridDesc(11) = standard longitude
gridDesc(20) = 255 ; used to specify the ordering of data (non-divisible by 32 indicates E-W ordering else N-S ordering
gridDesc(30) = TBD
gridDesc(32) = TBD
gridDesc(33) = TBD
gridDesc(34) = TBD
gridDesc(35) = TBD
gridDesc(36) = TBD
gridDesc(37) = TBD
gridDesc(38) = TBD
gridDesc(39) = TBD
gridDesc(40) = TBD

UTM projection gridDesc(1) = 7 ; indicates UTM projection
gridDesc(2) = number of columns in the domain ;
gridDesc(3) = number of rows in the domain ;
gridDesc(4) = northing of the lower left corner of the domain
gridDesc(5) = easting of the lower left corner of the domain
gridDesc(6) = 128 ; not used
gridDesc(7) = northing of the upper right corner of the domain
gridDesc(8) = easting of the upper right corner of the domain
gridDesc(9) = spatial resolution (in meters)
gridDesc(10) = UTM zone
gridDesc(11) = 64 ; not used
gridDesc(20) = 255 ; used to specify the ordering of data (non-divisible by 32 indicates E-W ordering else N-S ordering
gridDesc(34) = UTM zone

gridDesc(35) = northing of the lower left corner point
gridDesc(37) = easting of the lower left corner point
gridDesc(38) = number of columns in the domain
gridDesc(39) = number of rows in the domain
gridDesc(40) = spatial resolution (in meters)

soil_gridDesc, topo_gridDesc, lc_gridDesc Array describing the soils, topography, and landmask/landcover data grid, respectively, for each nest. The specification of these arrays for different map projections are as follows:

Lat/lon or geographic projection gridDesc(1) = latitude of the lower left corner of the domain
gridDesc(2) = longitude of the lower left corner of the domain
gridDesc(3) = latitude of the upper right corner of the domain
gridDesc(4) = longitude of the upper right corner of the domain
gridDesc(5) = spatial resolution (in degrees) along the E-W dimension
gridDesc(6) = spatial resolution (in degrees) along the N-S dimension

Gaussian projection gridDesc(1) = latitude of the lower left corner of the domain
gridDesc(2) = longitude of the lower left corner of the domain
gridDesc(3) = latitude of the upper right corner of the domain
gridDesc(4) = longitude of the upper right corner of the domain
gridDesc(5) = spatial resolution (in degrees) along the E-W dimension
gridDesc(6) = number of latitude circles

UTM projection gridDesc(1) = UTM zone gridDesc(2) = northing of the lower left corner of the domain
gridDesc(3) = easting of the lower left corner of the domain
gridDesc(4) = number of columns in the domain
gridDesc(5) = number of rows in the domain
gridDesc(6) = spatial resolution (in meters)

nf Number of forcing variables

nmetforc Number of meteorological forcing datasets used in a LIS simulation.

nperforc Number of ensemble members that correspond to a single forcing source.

metforc_blend_alg Blending algorithm used for merging different forcing datasets.

metforc Choice of meteorological forcings

met_ecor Choice of topographical downscaling method for met forcing

met_nf Number of forcing variables in each met forcing dataset.

met_interp Spatial interpolation option for met forcing (1-bilinear, 2-conservative, 3-neighbor)

met_tinterp Temporal interpolation option for met forcing (1-next, 2-uber-next)

rstflag boolean array denoting if the current simulation is a restart or not, for each nest

gridchange boolean array denoting if the native grid for the meteorological forcing need to be changed.

shortflag Shortwave radiation source flag (1-instantaneous, 2-time averaged)

longflag Longwave radiation source flag (1-instantaneous, 2-time averaged)

perturb_forcing Choice of forcing perturbation algorithm (0-do not use)

pertforcInterval Forcing perturbation frequency (in seconds)

perturb_obs Choice of observation perturbation algorithm (0-do not use)

pertobsInterval Observation perturbation frequency (in seconds)

perturb_state Choice of state perturbation algorithm (0-do not use)

pertstateInterval State perturbation frequency (in seconds)

pertrestart ??

pertrestartInterval ??

pertrestartfile ??

nt Number of vegetation classes in the landcover dataset

bareclass Index of bare class in the landcover dataset

urbanclass Index of urban class in the landcover dataset

snowclass Index of snow class in the landcover dataset

waterclass Index of water class in the landcover dataset

laiflag flag to control the LAI data read

saiflag flag to contro the SAI data read

mfile Name of the landmask file, for each nest

vfile Name of the landcover file, for each nest

vfile_form

safile Name of the sand fraction data file, for each nest

clfile Name of the clay fraction data file, for each nest

sifile Name of the silt fraction data file, for each nest

txtfile Name of the soil texture data file, for each nest

pofile Name of the porosity data file, for each nest

psisatfile Name of the saturated matric potential data file, for each nest

ksatfile Name of the saturated hydraulic conductivity data file, for each nest

bexpfile Name of the b parameter data file, for each nest

qzfile Name of the quartz data file, for each nest

iscfile Name of the soil color data file, for each nest

elevfile Name of the elevation data file, for each nest

slfile Name of the slope data file, for each nest

aspfile Name of the aspect data file, for each nest

curvfile Name of the curvature data file, for each nest

albfile Name of the climatology albedo data file, for each nest

mxsnal Name of the max snow albedo data file, for each nest

tbotfile Name of the bottom temperature data file, for each nest

shdmaxfile Name of the maximum greenness data file, for each nest

shdminfile Name of the minimum greenness data file, for each nest

slopetypefile Name of the slope type data file, for each nest

tile_coord_file Name of the catchment-based tile coordinate file, for each nest

tile_veg_file Name of the catchment-based vegetation type file, for each nest

outputSpecFile Name of the model output specification file

output_at_specifictime Option to specify if output is to be written only at a particular time (instead of at synoptic intervals)

albInterval Frequency of albedo climatology in months

laiInterval Frequency of LAI climatology in months

laitime Variable to keep track of LAI data interval

saitime Variable to keep track of SAI data interval

wopt Output methodology (none, 1d tilespace,2d gridspace, 1d gridspace)

wout Output data format (binary, grib1, netcdf)

wsingle Option to write each variable to a separate file

wstyle Output file naming style (3 level hierarchy, 5 level, WMO convention)

grib_table Grib Table version number

grib_center_id Grib center id

grib_process_id Grib process id

grib_grid_id Grib grid id

startcode Start mode (restart, coldstart)

plevel Logging level

odir Output directory

dfile Diagnostic output file

sdoy Starting julian day

sss Starting second

smn Starting minute

shr Starting hour

sda Starting day

smo Starting month

syr Starting year

endcode End mode (1-specific date)

edoy Ending julian day

ess Ending second

emn Ending minute

ehr Ending hour

eda Ending day

emo Ending month

eyr Ending year

endtime Flag to indicate the end of simulation (1-end of simulation)

etime Ending time

egmt Ending time in GMT

nts Array containing the model timestep for each nest

ts Timestep for the clock (minimum timestep of different nests)

tscount Timestep count for each nest

doy Current julian day

ss Current second

mn Current minute

hr Current hour

da Current day

mo Current month

yr Current year

time Current time

gmt Current time in GMT

ndas Number of data assimilation instances

nperts Number of perturbation instances

daalg Choice of data assimilation algorithm

biasalg Choice of bias estimation algorithm

incroption Option to specify whether to apply both bias and analysis increments (1- apply both, 0 - apply bias correction only)

daset Choice of Assimilation set: (variable to be assimilated + observation dataset)

nstvars Number of state variables to be assimilated

nobtypes Number of observation types to be assimilated

daoutInterval output interval for DA diagnostics

nensem Number of ensembles per grid

forcvarlistFile name of forcing variable list file

forcattribFile name of forcing attributes file

forcpertattribFile name of forcing perturbation attributes file

progpertattribFile name of prognostic variable attributes file

progpertattribFile name of prognostic variable perturbation attributes file

obsattribFile name of observation attributes file

obspertattribFile name of observation perturbation attributes file

biasOptionsFile name of bias specification attributes file

nforcepert Number of perturbation

wensems flag to specify if ensemble members are to be output

wobs flag to specify if observations are to be output

winnov flag to specify if normalized innovations are to be output

optUEAlg Choice of optimization/uncertainty estimation algorithm

optUEset Dataset to calibrate to

optUEtype Choice of optimization/uncertainty estimation type (parameter estimation, tuning DA params, etc.)

julbeg Beginning julian time of the current LIS cycle

julend Ending julian time of the current LIS cycle

security_class Security classification code for the LIS application

distribution_class Distribution classification code for the LIS application

data_category Data category code for the LIS application

area_of_data Geographical code for the LIS domain

lis_config_file Name of the run-time LIS configuration file (default = lis.config)

rtm Choice of radiative transfer model

nappmodel Total number of application models

landslidemodel Choice of landslide model.

REVISION HISTORY:

```
12 Apr 2001: Urszula Jambor; Added domain,lsm,& force namefile paramters
30 Jul 2001: Matt Rodell; Add new soil parameter variables
14 Nov 2002; Sujay Kumar; Optimized version for LIS
14 Oct 2003; Sujay Kumar; Removed LSM specific variables.
19 Jan 2007; Chuck Alonge; Added Flag to output parameters
17 Jan 2011: David Mocko, added max/min greenness & slope type
```

3.5.1 LIS_PRIV_tileMod (Source File: LIS_PRIV_tileMod.F90)

The code in this file provides a description of the tile data structure in LIS

3.5.2 Overview

This module contains the tile data structure used in LIS. The tile space in LIS is used to simulate subgrid scale variability, where each tile represents a vegetation class within a grid cell. The tile space is created from the vegetation distribution within a grid cell.
The tile data structure contains the following variables:

col column index of the corresponding grid cell

row row index of the corresponding grid cell

index index of the corresponding grid cell

vegt vegetation type of the tile

ensem ensemble index of the tile

tile_id global catchment id for the tile

fgrd fraction of grid covered by the tile

d2g local tile count to global tile count

com_lon center-of-mass longitude of the tile

com_lat center-of-mass latitude of the tile

REVISION HISTORY:

```
14 Nov 2002: Sujay Kumar; Optimized version of tile representation
```

3.6 Fortran: Module Interface LIS_RTMMod (Source File: LIS_RTMMod.F90)

REVISION HISTORY:

```
21 Mar 2009: Sujay Kumar; Initial implementation
```

3.6.1 LIS_RTM_init (Source File: LIS_RTMMod.F90)

INTERFACE:

```
subroutine LIS_RTM_init
```

DESCRIPTION:

USES:

```
use ESMF
use LIS_logMod,       only : LIS_verify
use LIS_fileIOMod,    only : LIS_create_stats_filename
use LIS_timeMgrMod,   only : LIS_registerAlarm, LIS_parseTimeString
```

3.6.2 RTM_run (Source File: LIS_RTMMod.F90)

INTERFACE:

```
subroutine RTM_run(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_logMod, only : LIS_logunit
```

DESCRIPTION:

USES:

```
implicit none
integer,   intent(in) :: n
```

3.6.3 LIS_RTM_output (Source File: LIS_RTMMod.F90)

INTERFACE:

```
subroutine LIS_RTM_output(n)
```

USES:

```
use LIS_timeMgrMod, only : LIS_isAlarmRinging
use LIS_histDataMod, only : LIS_histData
use LIS_historyMod,  only : LIS_writeModelOutput
use LIS_fileIOMod,  only : LIS_create_output_directory, &
     LIS_create_output_filename,  &
     LIS_create_stats_filename

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

3.6.4 LIS_RTM_final (Source File: LIS_RTMMod.F90)

INTERFACE:

```
subroutine LIS_RTM_finalize
```

DESCRIPTION:

USES:

3.7 Fortran: Module Interface LIS_albedoMod (Source File: LIS_albedoMod.F90)

The code in this file implements routines to read various sources of albedo parameter data.

3.7.1 Overview

This routines in this module reads two sources of albedo:

- Albedo climatology
- Static, maximum albedo expected over deep snow

This module provides routines to read the albedo climatology (monthly) data and allows the users to specify the frequency of climatology (in months). The climatological data is temporally interpolated between months to the current simulation date.

REVISION HISTORY:

```
 8 Aug 2005: Sujay Kumar; Initial implementation
use ESMF
use LIS_coreMod
use LIS_logMod
use LIS_fileIOMod
use LIS_timeMgrMod
use LIS_histDataMod

implicit none

PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
 public :: LIS_albedo_setup       !allocates memory for required variables
 public :: LIS_read_albedo        !reads climatological albedo
 public :: LIS_diagnosealbedo     !maps variables for history output
 public :: LIS_albedo_finalize    !cleanup allocated structures
 public :: LIS_albedo_reset       !resets the alarms and the data holders

-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
 public :: LIS_alb      !data structure containing albedo data
```

3.7.2 LIS_albedo_setup (Source File: LIS_albedoMod.F90)

INTERFACE:

```
subroutine LIS_albedo_setup
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
```

DESCRIPTION:

Allocates memory for data structures for reading in albedo datasets
The routines invoked are:

LIS_read_mxsnalb (3.7.3)
method to read the max snow albedo

readalbedo (3.40.4)
invokes the generic method in the registry to read the albedo climatology data

albedosetup (3.40.2)
calls the registry to invoke the albedo setup method.

3.7.3 LIS_read_mxsnalb (Source File: LIS_albedoMod.F90)

INTERFACE:

```
subroutine LIS_read_mxsnalb(n)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Reads the static albedo upper bound over deep snow for each domain.
The arguments are:

n index of n

localmask mxsnalb for the region of interest

3.7.4 LIS_read_albedo (Source File: LIS_albedoMod.F90)

INTERFACE:

```
subroutine LIS_read_albedo(n)
```

USES:

ARGUMENTS:

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

DESCRIPTION:

Reads the snow free albedo climatology data and temporally interpolates it to the current day
The arguments are:

n index of the domain or nest.

The routines invoked are:

LIS_computeTemporalweights (3.36.23)
computes the temporal interpolation weights

readalbedo (3.40.4)
invokes the generic method in the registry to read the albedo climatology data

---

3.7.5 LIS_albedo_finalize (Source File: LIS_albedoMod.F90)

INTERFACE:

```
subroutine LIS_albedo_finalize
```

USES:

```
implicit none
```

DESCRIPTION:

Deallocates objects created in this module

3.7.6 LIS_diagnosealbedo (Source File: LIS_albedoMod.F90)

INTERFACE:

```
subroutine LIS_diagnosealbedo(n)
```

USES:

ARGUMENTS:

```
implicit none
integer, intent(in)   :: n
```

DESCRIPTION:

This routine diagnoses the LIS albedo for history output.
The arguments are:

n index of the nest

The routines called are:

LIS_diagnoseOutputVar (3.16.5)
generic routine to map a single variable to the LIS history writer

---

3.7.7 LIS_albedo_reset (Source File: LIS_albedoMod.F90)

INTERFACE:

```
subroutine LIS_albedo_reset
```

USES:

DESCRIPTION:

Resets data structures for reading in albedo datasets
The routines invoked are:

LIS_read_mxsnalb (3.7.3)
method to read the max snow albedo

albedosetup (3.40.2)
calls the registry to invoke the albedo setup method.

readalbedo (3.40.4)
invokes the generic method in the registry to read the albedo climatology data

---

3.7.8 read_albedoclimo (Source File: LIS_albedoMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_albedoclimo(n,time,array)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
    use netcdf
#endif


    implicit none
```

ARGUMENTS:

```
    integer, intent(in) :: n
    integer             :: time
    real, intent(inout) :: array(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine reads the greenness data climatology
The arguments are:

n index of n

3.8 Fortran: Module Interface LIS_appMod (Source File: LIS_appMod.F90)

This module controls the operation of different application models within LIS. An application model is defined one that does not have any feedback to the land surface states.

REVISION HISTORY:

```
implicit none

PRIVATE

-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
PUBLIC :: LIS_appModel_init
PUBLIC :: LIS_runAppModel
PUBLIC :: LIS_outputAppModel
PUBLIC :: LIS_appModel_finalize ! SY
PUBLIC :: LIS_app_param_reset
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
```

3.8.1 LIS_appModel_init (Source File: LIS_appMod.F90)

INTERFACE:

```
subroutine LIS_appModel_init()
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_config
use LIS_logMod,     only : LIS_verify
use LIS_landslideMod, only : LIS_initLandSlideModel
implicit none
```

DESCRIPTION:

Reads the configuration options related to application models, invokes the initialization of specific application models,
The methods invoked are:

LIS_initLandSlideModel (3.20.1)
invokes the initialization of landslide models.

3.8.2 LIS_runAppModel (Source File: LIS_appMod.F90)

INTERFACE:

```
subroutine LIS_runAppModel(n)
```

USES:

```
use LIS_coreMod,   only : LIS_rc
use LIS_landslideMod,     only : LIS_runLandSlideModel
```

ARGUMENTS:

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

DESCRIPTION:

This routine invokes the execution of specified application models.

3.8.3 LIS_outputAppModel(n) (Source File: LIS_appMod.F90)

INTERFACE:

```
subroutine LIS_outputAppModel(n)
```

USES:

```
use LIS_coreMod,   only : LIS_rc
use LIS_landslideMod,     only : LIS_outputLandSlideModel
```

ARGUMENTS:

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

DESCRIPTION:

3.8.4 LIS_appModel_finalize() (Source File: LIS_appMod.F90)

INTERFACE:

```
subroutine LIS_appModel_finalize()
```

USES:

```
use LIS_coreMod,   only : LIS_rc
use LIS_landslideMod,     only : LIS_finalizeLandSlideModel
```

DESCRIPTION:

3.9 Fortran: Module Interface LIS_constantsMod (Source File: LIS_constantsMod.F90)

The code in this file provides values of physical constants for consistent use across different components.

REVISION HISTORY:

```
    17 Feb 2004    Sujay Kumar  Initial Specification

!-----------------------------------------------------------------------------
! physical constants (all data public)
!-----------------------------------------------------------------------------
   public
!   real,parameter :: CONST_PI     = 3.14159265358979323846  ! pi
   real,parameter :: LIS_CONST_PI     = 3.14159265   ! pi
   real,parameter :: LIS_CONST_CDAY   = 86400.0      ! sec in calendar day ~ sec
   real,parameter :: LIS_CONST_SDAY   = 86164.0      ! sec in siderial day ~ sec
   real,parameter :: LIS_CONST_OMEGA  = 2.0*LIS_CONST_PI/LIS_CONST_SDAY ! earth rot ~ rad/sec
   real,parameter :: LIS_CONST_REARTH = 6.37122e6    ! radius of earth ~ m
   real,parameter :: LIS_CONST_G      = 9.80616      ! acceleration of gravity ~ m/s^2
   real,parameter :: LIS_CONST_PSTD   = 101325.0     ! standard pressure ~ pascals

   real,parameter :: LIS_CONST_STEBOL = 5.67e-8      ! Stefan-Boltzmann constant ~ W/m^2/K^4
   real,parameter :: LIS_CONST_BOLTZ  = 1.38065e-23  ! Boltzmann's constant ~ J/K/molecule
   real,parameter :: LIS_CONST_AVOGAD = 6.02214e26   ! Avogadro's number ~ molecules/kmole
   real,parameter :: LIS_CONST_RGAS   = LIS_CONST_AVOGAD*LIS_CONST_BOLTZ ! Universal gas constant ~ J/K/
   real,parameter :: LIS_CONST_MWDAIR = 28.966       ! molecular weight dry air ~ kg/kmole
   real,parameter :: LIS_CONST_MWWV   = 18.016       ! molecular weight water vapor
   real,parameter :: LIS_CONST_RDAIR  = LIS_CONST_RGAS/LIS_CONST_MWDAIR  ! Dry air gas constant ~ J/K/kg
   real,parameter :: LIS_CONST_RWV    = LIS_CONST_RGAS/LIS_CONST_MWWV    ! Water vapor gas constant ~ J/
   real,parameter :: LIS_CONST_ZVIR   = (LIS_CONST_RWV/LIS_CONST_RDAIR)-1.0   ! RWV/RDAIR - 1.0
   real,parameter :: LIS_CONST_KARMAN = 0.4          ! Von Karman constant

   real,parameter :: LIS_CONST_TKFRZ  = 273.16       ! freezing T of fresh water ~ K (intentionally made
   real,parameter :: LIS_CONST_TKTRIP = 273.16       ! triple point of fresh water ~ K

   real,parameter :: LIS_CONST_RHODAIR=LIS_CONST_PSTD/ &
     (LIS_CONST_RDAIR*LIS_CONST_TKFRZ)         ! density of dry air at STP   ~ kg/m^3
   real,parameter :: LIS_CONST_RHOFW  = 1.000e3      ! density of fresh water ~ kg/m^3
   real,parameter :: LIS_CONST_RHOSW  = 1.026e3      ! density of sea water ~ kg/m^3
   real,parameter :: LIS_CONST_RHOICE = 0.917e3      ! density of ice   ~ kg/m^3
   real,parameter :: LIS_CONST_CPDAIR = 1.00464e3    ! specific heat of dry air ~ J/kg/K
   real,parameter :: LIS_CONST_CPFW   = 4.188e3      ! specific heat of fresh h2o ~ J/kg/K
   real,parameter :: LIS_CONST_CPSW   = 3.996e3      ! specific heat of sea h2o ~ J/kg/K
   real,parameter :: LIS_CONST_CPWV   = 1.810e3      ! specific heat of water vap ~ J/kg/K
   real,parameter :: LIS_CONST_CPICE  = 2.11727e3    ! specific heat of fresh ice ~ J/kg/K
   real,parameter :: LIS_CONST_LATICE = 3.337e5      ! latent heat of fusion ~ J/kg
   real,parameter :: LIS_CONST_LATVAP = 2.501e6      ! latent heat of evaporation ~ J/kg
   real,parameter :: LIS_CONST_LATSUB = LIS_CONST_LATICE + LIS_CONST_LATVAP ! latent heat of sublimatio

   real,parameter :: LIS_CONST_OCN_REF_SAL = 34.7    ! ocn ref salinity (psu)
   real,parameter :: LIS_CONST_ICE_REF_SAL =  4.0    ! ice ref salinity (psu)
   real,parameter :: LIS_CONST_SOLAR  =  1369.2    !Solar constant (W/m2)

   real, parameter :: LIS_MS2KMDAY = 86.4 ! m/s to km/day
   real, parameter :: LIS_CONST_ANGULAR_VELOCITY = 0.2618 ! Angular velocity of earth's rotation (radia
```

3.10 Fortran: Module Interface LIS_coreMod (Source File: LIS_coreMod.F90)

The code in this file contains the basic datastructures and control routines for the operation of LIS

3.10.1 Overview

This module contains the defintion and specification of the basic datastructures that define a LIS instance. It consists of:

LIS_rc datastructure containg generic LIS run control (rc) variables

LIS_domain datastructure containing grid, tile spaces, grid projection, domain decomposition information.

LIS_histData datastructure defining the metadata for model output.

LIS_config instance of the configuration class.

LIS_vm instance of the virtual machine controlling the LIS resources.

LIS_masterproc logical variable to specify the master processor (true = for master, false for non-zero processors)

LIS_npes number of processors used in LIS

LIS_ews_ind starting East West index of each nest (excluding halo regions)

LIS_ewe_ind ending East West index of each nest (excluding halo regions)

LIS_nss_ind starting North-South index of each nest (excluding halo regions)

LIS_nse_ind ending North-South index of each nest (excluding halo regions)

LIS_ews_ind_halo starting East West index of each nest (including halo regions)

LIS_ewe_ind_halo ending East West index of each nest (including halo regions)

LIS_nss_ind_halo starting North-South index of each nest (including halo regions)

LIS_nse_ind_halo ending North-South index of each nest (including halo regions)

LIS_deltas size of unmasked grid space buffers

LIS_offsets offsets for unmasked grid space buffers

LIS_tdeltas size of tile space buffers

LIS_toffsets offsets of tile space buffers

LIS_gdeltas size of grid space buffers

LIS_goffsets offsets of grid space buffers

LIS_odeltas size of observation space buffers

LIS_ooffsets offsets of observation space buffers

LIS_ntiless size of tile spaces

LIS_ngrids size of grid spaces

LIS_vecTile grid object for the tile space-vector of tiles

LIS_vecGrid grid object for the grid space-vector of grid pts

LIS_ensOnGrid grid object for the ensemble space on grid

REVISION HISTORY:

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

USES:

```
use ESMF
use LIS_PRIV_rcMod
use LIS_PRIV_gridMod
use LIS_PRIV_tileMod
use LIS_timeMgrMod
use LIS_logMod
use LIS_mpiMod
use map_utils

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: LIS_config_init
public :: LIS_core_init
public :: LIS_ticktime
public :: LIS_endofrun
public :: LIS_endOfTimeWindow
public :: LIS_TimeToRunNest
public :: LIS_resetTimeMgr
public :: LIS_finalize
```

PUBLIC TYPES:

```
public :: LIS_rc
public :: LIS_domain
public :: LIS_surface
public :: LIS_config
public :: LIS_initialized
public :: LIS_vm
public :: LIS_masterproc
public :: LIS_localPet
public :: LIS_npes
public :: LIS_ews_ind
public :: LIS_ewe_ind
public :: LIS_nss_ind
public :: LIS_nse_ind
public :: LIS_ews_halo_ind
public :: LIS_ewe_halo_ind
public :: LIS_nss_halo_ind
public :: LIS_nse_halo_ind
```

```
public :: LIS_deltas
public :: LIS_offsets
public :: LIS_tdeltas
public :: LIS_toffsets
public :: LIS_gdeltas
public :: LIS_goffsets
public :: LIS_odeltas
public :: LIS_ooffsets
public :: LIS_patch_deltas
public :: LIS_patch_offsets
public :: LIS_ntiless
public :: LIS_ngrids
public :: LIS_npatches
Three different types of grid objects are defined because ESMF3.1.0 does
not support defining tiles on a single grid.
public :: LIS_vecTile
public :: LIS_vecPatch
public :: LIS_vecGrid
public :: LIS_ensOnGrid
```

3.10.2 LIS_config_init (Source File: LIS_coreMod.F90)

INTERFACE:

```
interface LIS_config_init
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure lisconfig_offline
module procedure lisconfig_coupled1
module procedure lisconfig_coupled2
```

DESCRIPTION:

This interface provides routines for the setup of LIS configuration management. It initializes the LIS Config utility, reads the model independent specifications, and intializes the SPMD parallel processing mode. This routine also initializes the LIS time manager, and finally allocates memory for the LIS generic datastructures.

3.10.3 lisconfig_offline (Source File: LIS_coreMod.F90)

INTERFACE:

```
Private name: call using LIS_config_init()
subroutine lisconfig_offline()
```

DESCRIPTION:

Performs the initialization of LIS runtime configuration for an offline (uncoupled simulation).
The Calling sequence is :

spmd_init_offline (3.10.13)
performs SPMD initializations

LIS_log_init (3.21.2)
initializes the LIS log handler

LIS_readConfig (3.30.2)
reads the model independent options

spmd_setup (3.10.15)
allocates memory for SPMD variables

LIS_timemgr_init (3.36.2)
initializes the time manager

3.10.4 lisconfig_coupled1 (Source File: LIS_coreMod.F90)

INTERFACE:

```
Private name: call using LIS_config_init
subroutine lisconfig_coupled1(nx,ny)

  implicit none
```

ARGUMENTS:

```
integer,intent(in) :: nx, ny
```

DESCRIPTION:

Performs the initialization of LIS runtime configuration for an coupled simulation. The processor layout is obtained from a parent component.
The arguments are:

nx Number of processors in the East-West direction in a processor layout

ny Number of processors in the North-South direction in a processor layout

The Calling sequence is :

spmd_init_offline (3.10.13)
performs SPMD initializations

LIS_log_init (3.21.2)
initializes the LIS log handler

LIS_readConfig (3.30.2)
reads the model independent options

spmd_setup (3.10.15)
allocates memory for SPMD variables

LIS_timemgr_init (3.36.2)
initializes the time manager

3.10.5 lisconfig_coupled2 (Source File: LIS_coreMod.F90)

INTERFACE:

```
Private name: call using LIS_config_init()
subroutine lisconfig_coupled2(vm, clock)

  implicit none
```

ARGUMENTS:

```
  type(ESMF_VM)       :: vm
  type(ESMF_Clock)    :: clock
```

DESCRIPTION:

Performs the initialization of LIS runtime configuration for an coupled simulation. The virtual machine is obtained from a different component or an application driver.
The Calling sequence is :

spmd_init_offline (3.10.13)
performs SPMD initializations

LIS_log_init (3.21.2)
initializes the LIS log handler

LIS_readConfig (3.30.2)
reads the model independent options

spmd_setup (3.10.15)
allocates memory for SPMD variables

LIS_timemgr_init (3.36.2)
initializes the time manager

3.10.6 LIS_core_init (Source File: LIS_coreMod.F90)

INTERFACE:

```
subroutine LIS_core_init
```

DESCRIPTION:

Completes the initialization of the LIS' clock and alarms.

3.10.7 LIS_ticktime (Source File: LIS_coreMod.F90)

INTERFACE:

```
subroutine LIS_ticktime()
```

DESCRIPTION:

This routine calls the time manager to increment the runtime clock by the model timestep.
The Calling sequence is :

LIS_advance_timestep (3.36.7)
advances the clock

3.10.8 LIS_endofrun (Source File: LIS_coreMod.F90)

INTERFACE:

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

ARGUMENTS:

```
logical :: finish
```

DESCRIPTION:

This function checks to see if the runtime clock has reached the specified stop time of the simulation. The arguments are:

finish boolean value indicating if the end of simulation is reached.

The calling sequence is:

LIS_is_last_step (3.36.16)
check if the clock has reached the stop time

3.10.9 LIS_endofTimeWindow (Source File: LIS_coreMod.F90)

INTERFACE:

```
function LIS_endofTimeWindow() result(finish)
```

USES:

```
use LIS_logMod, only : LIS_logunit
```

ARGUMENTS:

```
logical :: finish
```

DESCRIPTION:

This function checks to see if the runtime clock has reached the specified stop time of the simulation. The arguments are:

finish boolean value indicating if the end of simulation is reached.

The calling sequence is:

LIS_is_last_step (3.36.16)
check if the clock has reached the stop time

3.10.10 LIS_timeToRunNest (Source File: LIS_coreMod.F90)

INTERFACE:

```
function LIS_timeToRunNest(n) result(check)
```

ARGUMENTS:

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

DESCRIPTION:

Check to see if it is time to run the nest based on the model timestep for that particular domain

n index of the nest or domain

check boolean value indicating if the time to run the nest has reached or not

---

3.10.11 LIS_resetTimeMgr (Source File: LIS_coreMod.F90)

INTERFACE:

```
subroutine LIS_resetTimeMgr
```

DESCRIPTION:

This routine resets the time manager clock back to the starting time.
The calling sequence is:

LIS_resetclock (3.36.4)

3.10.12 LIS_finalize (Source File: LIS_coreMod.F90)

INTERFACE:

```
subroutine LIS_finalize
```

DESCRIPTION:

This routine issues the invocation to deallocate and cleanup any allocated data structures.
The calling sequence is:

spmd_finalize (3.10.16)
cleanup SPMD structures

3.10.13 spmd_init_offline (Source File: LIS_coreMod.F90)

INTERFACE:

```
subroutine spmd_init_offline(lisvm)
```

DESCRIPTION:

Initializes MPI and retrieves the number of CPUs and the processors IDs. The MPI initialization is done in an offline mode. In a coupled mode, the initialized environment is passed to LIS from a parent component.

3.10.14 spmd_init_coupled (Source File: LIS_coreMod.F90)

INTERFACE:

```
subroutine spmd_init_coupled()
```

DESCRIPTION:

Obtains the number of processors and the processors IDs from an already initialized MPI state. The initialization is assumed to be performed in a parent component.

3.10.15 spmd_setup (Source File: LIS_coreMod.F90)

INTERFACE:

```
subroutine spmd_setup(nnest, ndas, nmodels)

  implicit none
```

ARGUMENTS:

```
  integer, intent(in):: nnest
  integer, intent(in):: ndas
  integer, intent(in):: nmodels
```

DESCRIPTION:

Allocates memory for the variables describing domain decomposition.
The arguments are:

nnest Number of nests or domains

3.10.16 spmd_finalize (Source File: LIS_coreMod.F90)

INTERFACE:

```
subroutine spmd_finalize
```

DESCRIPTION:

This routine issues the invocation to deallocate and cleanup any allocated data structures.

3.11 Fortran: Module Interface LIS_dataAssimMod (Source File: LIS_dataAssimMod.F90)

The module controls the management of different data assimilation implementations.

3.11.1 Overview

This module defines some of the interface plugins for the incorporation of different data assimilation implementations. There are three main levels of abstractions identified in a data assimilation implementation:

- Data Assimilation Algorithm: various methods such as direct insertion, extended kalman filter, ensemble kalman filter, etc.
- Observations/Variable being assimilated: this abstraction identifies the source of variables, and the translation of observations to model state variables.
- Land surface model: the model used in the assimilation implementation.
- Pertubation techniques: methods to perturb the forcing and model prognostic variables

A complete implementation would require the interaction between the specific instances of the components listed above. This module includes methods to invoke some of the above.

REVISION HISTORY:

```
27 Feb 2005    Sujay Kumar; Initial Specification
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_logMod
use LIS_historyMod

implicit none

PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
public :: LIS_dataassim_init      !initialize data assimilation routines
public :: LIS_dataassim_run       !invoke data assimilation algorithms
public :: LIS_dataassim_output   !invoke data assimilation output
public :: LIS_dataassim_finalize  !cleanup allocated structures
public :: LIS_rescale_with_CDF_matching !rescale with CDF matching
public :: LIS_readCDFdata !read the CDF files generated by LDT
public :: LIS_readMeanSigmaData
```

3.11.2 LIS_dataassim_init (Source File: LIS_dataAssimMod.F90)

INTERFACE:

```
subroutine LIS_dataassim_init
```

USES:

DESCRIPTION:

The routine performs the initializations required for data assimilation. The registries that define the instances of data assimilation algorithms, observations, and perturbation methods are invoked, followed by the creation of required data structures.
The calling sequence is:

LIS_dataassim_plugin (4.7.1)
sets up function table registries for implemented data assimilation algorithms

LIS_biasEstimation_plugin (4.6.1)
sets up function table registries for implemented bias estimation algorithms.

dataassiminit (3.42.2)
invokes the generic method in the registry to initialize the data assimilation algorithm- structures

dataassimsetup (3.42.4)
invokes the generic method in the registry to set up the data assimilation algorithm- structures

biasestimationsetup (3.41.4)
invokes the generic method in the registry to set up the bias estimation algorithm structures.

3.11.3 LIS_dataassim_run (Source File: LIS_dataAssimMod.F90)

INTERFACE:

```
subroutine LIS_dataassim_run(n)
```

USES:

DESCRIPTION:

The routine initiates the call to the specific assimilation algorithm.
The calling sequence is:

computeIncrements (3.42.8)
computes the increments from the analysis

applybiascorrection (3.41.8)
applies the bias correction increments

applyIncrements (3.42.6)
applies the increments from the analysis

3.11.4 LIS_dataassim_output (Source File: LIS_dataAssimMod.F90)

INTERFACE:

```
subroutine LIS_dataassim_output(n)
```

USES:

DESCRIPTION:

This routine issues the invocation to write data assimilation specific output
The calling sequence is:

daoutput (3.42.12)
invokes the generic method in the registry to write data assimilation specific output

---

3.11.5 LIS_dataassim_finalize (Source File: LIS_dataAssimMod.F90)

INTERFACE:

```
subroutine LIS_dataassim_finalize
```

USES:

DESCRIPTION:

This routine issues the invocation to deallocate and cleanup any allocated data structures in the specific instance of the data assimilation implementations.
The calling sequence is:

dafinalize (3.42.10)
invokes the generic method in the registry to cleanup the allocated structures for the data assimilation algorithm

---

3.11.6 LIS_rescale_with_CDF_matching (Source File: LIS_dataAssimMod.F90)

INTERFACE:

```
subroutine LIS_rescale_with_CDF_matching(&
     n,             &
     nbins,         &
     max_obs_value, &
     min_obs_value, &
     model_xrange,  &
     obs_xrange,    &
     model_cdf,     &
     obs_cdf,       &
     obs_value)


  implicit none
```

ARGUMENTS:

```
integer             :: n
integer             :: nbins
real                :: max_obs_value
real                :: min_obs_value
real                :: model_xrange(LIS_rc%ngrid(n),nbins)
real                :: obs_xrange(LIS_rc%ngrid(n),nbins)
real                :: model_cdf(LIS_rc%ngrid(n),nbins)
real                :: obs_cdf(LIS_rc%ngrid(n),nbins)
real                :: obs_value(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

This routine rescales the input observation data to the model's climatology so that the cumulative distribution functions (CDFs) of the observations and the model match (for each grid point).
Ref: Reichle and Koster, 2004, Bias reduction in short records of satellite soil moisture, Geophys. Res. Lett. 31, L19501, doi:10.1029/2004GL020938.
The arguments are:

n index of the nest

nbins number of bins used to compute the model and obs CDFs

max_obs_value maximum allowable value of observation

min_obs_value minimum allowable value of observation

model_xrange x-axis values corresponding to the model CDF

obs_xrange x-axis values corresponding to the obs CDF

model_cdf y-axis (CDF) values corresponding to the model CDF

obs_cdf y-axis (CDF) values corresponding to the obs CDF

obs_value observation value to be rescaled.

3.11.7 LIS_readCDFdata (Source File: LIS_dataAssimMod.F90)

INTERFACE:

```
   subroutine LIS_readCDFdata(n, nbins, filename, varname, xrange, cdf)

#if(defined USE_NETCDF3 || defined USE_NETCDF4)
     use netcdf
#endif
     implicit none
```

ARGUMENTS:

```
     integer,   intent(in)    :: n
     integer,   intent(in)    :: nbins
     character(len=*)         :: filename
     character(len=*)         :: varname
     real                     :: xrange(LIS_rc%ngrid(n),nbins)
     real                     :: cdf(LIS_rc%ngrid(n),nbins)
```

DESCRIPTION:

This routine reads the input CDF file (generated by LDT in NETCDF format) The xrange values and the corresponding CDFs are read for each grid point. Both these fields are expected to be in the 1-d grid vector dimension.
The arguments are:

n index of the nest

nbins number of bins used to compute the model and obs CDFs

filename name of the CDF file

varname name of the variable being extracted.

xrange x-axis values corresponding to the CDF

cdf y-axis (CDF) values corresponding to the CDF

3.11.8 LIS_readMeanSigmaData (Source File: LIS_dataAssimMod.F90)

INTERFACE:

```
    subroutine LIS_readMeanSigmaData(n, filename, varname, mu, sigma)

 #if(defined USE_NETCDF3 || defined USE_NETCDF4)
      use netcdf
 #endif
      implicit none
```

ARGUMENTS:

```
      integer,   intent(in)    :: n
      character(len=*)         :: filename
      character(len=*)         :: varname
      real                     :: mu(LIS_rc%ngrid(n))
      real                     :: sigma(LIS_rc%ngrid(n))
```

DESCRIPTION:

This routine reads the input CDF file (generated by LDT in NETCDF format) and reads the mean (mu) and standard deviation (sigma) values for each grid point. Both these fields are expected to be in the 1-d grid vector dimension.
The arguments are:

n index of the nest

filename name of the CDF file

varname name of the variable being extracted.

mu mean values

sigma standard deviation values

3.12 Fortran: Module Interface LIS_domainMod (Source File: LIS_domainMod.F90)

The code in this file provides interfaces to manage the creation of the domain (grid, tile, patch and ensemble spaces) used in the LIS run

3.12.1 Overview

The 'domain' is defined in LIS as a cartesian grid in two dimensions with an ordering from the lower left corner to the upper right corner. This rule is followed for the running domain of LIS that may use different map projections.
When the running domain is created, LIS generates a 1-d vector representation of the 2-d input grid based on the input landmask (where open water points are typically excluded). The basic computational unit in LIS is called a "tile". If options for subgrid variability are employed, LIS will create multiple number of tiles per each grid cell. The subgrid tiles can be defined based on the distribution of vegetation types, soil characteristics or topography. Each tile can further define a specified number of ensembles.
LIS also supports the use of multiple surface types. For example, the domain could consist of 'land' surfaces and 'lake' surfaces and LSMs will be executed over land points and Lake models will be executed over lake points. Within LIS, these surface models are defined to run over "patches". The patches across different surface models sum to the tile dimension.

REVISION HISTORY:

```
 17 Feb 2004    Sujay Kumar  Initial Specification
 24 Aug 2008    Sujay Kumar  Implemented halo support
  3 Aug 2012    Sujay Kumar  Added support for flexible tiling

use ESMF
use LIS_coreMod
use LIS_logMod
use LIS_LMLCMod
use LIS_topoMod
use LIS_soilsMod
use LIS_domain_pluginMod
use LIS_histDataMod
use LIS_mpiMod
use map_utils

implicit none
PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: LIS_domain_init          !initialize specified domains
public :: LIS_domain_setup        !setup domain related structures
public :: LIS_quilt_domain         !generate quilted domains
public :: LIS_setParameterDomainSpecs !sets map projection information for parameters
public :: LIS_domain_finalize      !cleanup allocated structures
```

3.12.2 LIS_domain_init (Source File: LIS_domainMod.F90)

INTERFACE:

```
subroutine LIS_domain_init
```

USES:

```
use ESMF
```

DESCRIPTION:

This routine invokes the registry that defines the domain implementations, first. This is followed by calling the routines to read the runtime specific parameters for each domain instance. A call to create the domain is issued next, which is expected to generate the grid, tile, patch and ensemble spaces for each domain. This routine is also expected to perform any domain decomposition needed for parallel processing. Finally, the domain decomposition information is disseminated to individual processors.
The calling sequence is:

LIS_domain_plugin (4.8.1)
sets up function table registries for implemented domains

readinput (3.43.4)
invokes the generic method in the registry to read domain specific input options from the configuration file

LIS_LMLC_init (3.2.2)
initialize landmask and landcover data structures

LIS_topo_init (3.37.2)
initialize topography data structures

LIS_soils_init (3.33.2)
initialize soils data structures

make_domain (3.12.7)
issues the call to generate the tile, patch, and grid dimensions, and associated datastructures.

3.12.3 LIS_domain_setup (Source File: LIS_domainMod.F90)

INTERFACE:

```
subroutine LIS_domain_setup(n)
```

USES:

ARGUMENTS:

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

DESCRIPTION:

This routines computes the variables used to map between the tile, grid, and patch spaces in LIS. The global indices including the halo regions are also computed.

3.12.4 LIS_quilt_domain (Source File: LIS_domainMod.F90)

INTERFACE:

```
subroutine LIS_quilt_domain(n, nc, nr )
```

USES:

ARGUMENTS:

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

DESCRIPTION:

This routine generates the quilted domain extents and sizes based on the processor layout and the size of the specified halos.

3.12.5 LIS_setParameterDomainSpecs (Source File: LIS_domainMod.F90)

INTERFACE:

```
subroutine LIS_setParameterDomainSpecs(param_dd, gridDesc)
```

USES:

ARGUMENTS:

```
real,   intent(in)    :: param_dd(:)
real,   intent(inout) :: gridDesc(50)
```

DESCRIPTION:

This subroutine sets the grid descriptor array section that defines the parameter domain and projection information, based on the map projection used for defining surface parameters

---

3.12.6 LIS_domain_finalize (Source File: LIS_domainMod.F90)

INTERFACE:

```
subroutine LIS_domain_finalize
```

DESCRIPTION:

This routine issues the invocation to deallocate and cleanup any allocated data structures in the specific instance of the domain implementation.

3.12.7 make_domain (Source File: LIS_domainMod.F90)

INTERFACE:

```
subroutine make_domain()
```

DESCRIPTION:

This subroutine generates the tile, patch and grid data structures based on the vegetation, soils, and topography datasets. The routine first computes the dominant distributions for vegetation and soils and then generates the multi-dimensional tile structure.

3.12.8 calculate_domdistribution (Source File: LIS_domainMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine calculate_domdistribution(n, ntypes, minp, maxt, fgrd)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)  :: n
integer              :: ntypes
real                 :: minp
integer              :: maxt
real                 :: fgrd(LIS_rc%lnc(n), LIS_rc%lnr(n), ntypes)
```

DESCRIPTION:

This routine determines the percentages of dominant types for subgrid tiling. The routine uses the distribution of data within a grid cell and imposes the specified cutoff percentage and the maximum number of data types allowed to create the number of tiles.
The arguments are:

n index of the nest

ntypes number of categories or bins in the input data.

minp minimum cutoff percentage to be used in the renormalization

maxt maximum number of tiles per grid cell to be used in the renormalization

fgrd fraction of grid covered by a data type.

3.12.9 create_tilespace (Source File: LIS_domainMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine create_tilespace(n,soilt_selected, soilf_selected, elev_selected,&
    slope_selected, aspect_selected)

  implicit none
```

ARGUMENTS:

```
  integer, intent(in)   :: n
  logical, intent(in)   :: soilt_selected
  logical, intent(in)   :: soilf_selected
  logical, intent(in)   :: elev_selected
  logical, intent(in)   :: slope_selected
  logical, intent(in)   :: aspect_selected
```

DESCRIPTION:

This routine creates the tilespace based on the specified dimensions of tiling. Vegetation is used by default. Soil texture, soil fractions, elevation, slope, aspect can be specified as additional dimensions for tiling. The routine also generates the patch data structures which are the tile data structures in each surface model class.
The arguments are:

n index of the nest

soilt_selected flag to indicate if soil texture based tiling is used

soilf_selected flag to indicate if soil fraction based tiling is used

soilf_selected flag to indicate if soil fraction based tiling is used

elev_selected flag to indicate if elevation based tiling is used

slope_selected flag to indicate if slope based tiling is used

aspect_selected flag to indicate if aspect based tiling is used

---

3.12.10 isSurfaceTypeSelected (Source File: LIS_domainMod.F90)

INTERFACE:

```
function isSurfaceTypeSelected(surface_type)
```

ARGUMENTS:

```
  integer       :: surface_type
```

DESCRIPTION:

This function determines if the input surface type is among the surface model types that are currently chosen in the LIS run

---

3.12.11 ifContainsSelectedSurfaceType (Source File: LIS_domainMod.F90)

INTERFACE:

```
function ifContainsSelectedSurfaceType(surface_types)
```

ARGUMENTS:

```
integer       :: surface_types(LIS_rc%nsurfacetypes)
```

DESCRIPTION:

This function determines if any of the input list of surface types is among the surface model types that are currently chosen in the LIS run

---

3.12.12 compute_ntiles_surface (Source File: LIS_domainMod.F90)

INTERFACE:

```
function compute_ntiles_surface(n,c,r)
```

ARGUMENTS:

```
integer :: n
integer :: c
integer :: r
```

DESCRIPTION:

This function computes the number of tiles based on the distribution of landcover types in a given grid cell

---

3.12.13 compute_npatches_surface (Source File: LIS_domainMod.F90)

INTERFACE:

```
function compute_npatches_surface(n,c,r,t,sf_index)
```

ARGUMENTS:

```
integer :: n
integer :: c
integer :: r
integer :: t
integer :: sf_index
```

DESCRIPTION:

This function computes the number of patches (for the given surface type) based on the distribution of landcover types in a given grid cell

---

3.12.14 compute_ntiles_soilt (Source File: LIS_domainMod.F90)

INTERFACE:

```
function compute_ntiles_soilt(n,c,r,flag)
```

ARGUMENTS:

```
integer :: n
integer :: c
integer :: r
logical :: flag
```

DESCRIPTION:

This function computes the number of tiles based on the distribution of soil texture types in a given grid cell

---

3.12.15 compute_ntiles_soilf (Source File: LIS_domainMod.F90)

INTERFACE:

```
function compute_ntiles_soilf(n,c,r,flag)
```

ARGUMENTS:

```
integer :: n
integer :: c
integer :: r
logical :: flag
```

DESCRIPTION:

This function computes the number of tiles based on the distribution of soil fraction data in a given grid cell

---

3.12.16 compute_ntiles_elev (Source File: LIS_domainMod.F90)

INTERFACE:

```
function compute_ntiles_elev(n,c,r,flag)
```

ARGUMENTS:

```
integer :: n
integer :: c
integer :: r
logical :: flag
```

DESCRIPTION:

This function computes the number of tiles based on the distribution of elevation bands in a given grid cell

---

3.12.17 compute_ntiles_slope (Source File: LIS_domainMod.F90)

INTERFACE:

```
function compute_ntiles_slope(n,c,r,flag)
```

ARGUMENTS:

```
integer :: n
integer :: c
integer :: r
logical :: flag
```

DESCRIPTION:

This function computes the number of tiles based on the distribution of slope bands in a given grid cell

---

3.12.18 compute_ntiles_aspect (Source File: LIS_domainMod.F90)

INTERFACE:

```
function compute_ntiles_aspect(n,c,r,flag)
```

ARGUMENTS:

```
integer :: n
integer :: c
integer :: r
logical :: flag
```

DESCRIPTION:

This function computes the number of tiles based on the distribution of aspect bands in a given grid cell

---

3.12.19 get_vegt_value (Source File: LIS_domainMod.F90)

INTERFACE:

```
subroutine get_vegt_value(n,c,r,i,vegt)
```

ARGUMENTS:

```
integer  :: n
integer  :: c
integer  :: r
integer  :: i
integer  :: vegt
```

DESCRIPTION:

This routine computes the vegetation type value for given tile number and grid cell.

3.12.20 get_surface_value (Source File: LIS_domainMod.F90)

INTERFACE:

```
subroutine get_surface_value(n,c,r,i,sf_index)
  integer  :: n
  integer  :: c
  integer  :: r
  integer  :: i
  integer  :: sf_index
```

DESCRIPTION:

This routine computes the surface type value for given tile number and grid cell.

3.12.21 get_soilt_value (Source File: LIS_domainMod.F90)

INTERFACE:

```
subroutine get_soilt_value(n,c,r,i,soilt_selected,soilt)
  integer  :: n
  integer  :: c
  integer  :: r
  integer  :: i
  logical  :: soilt_selected
  integer  :: soilt
```

DESCRIPTION:

This routine computes the soil texture value for given tile number and grid cell.

3.12.22 get_soilf_value (Source File: LIS_domainMod.F90)

INTERFACE:

```
subroutine get_soilf_value(n,c,r,i,soilf_selected,sand,clay,silt,soilf_index)
  integer  :: n
  integer  :: c
  integer  :: r
  integer  :: i
  logical  :: soilf_selected
  real     :: sand
  real     :: clay
  real     :: silt
  integer  :: soilf_index
```

DESCRIPTION:

This routine computes the elevation value for given tile number and grid cell.

3.12.23 get_elev_value (Source File: LIS_domainMod.F90)

INTERFACE:

```
subroutine get_elev_value(n,c,r,i,elev_selected,elev,elev_index)
  integer  :: n
  integer  :: c
  integer  :: r
  integer  :: i
  logical  :: elev_selected
  real     :: elev
  integer  :: elev_index
```

DESCRIPTION:

This routine computes the elevation value for given tile number and grid cell.

3.12.24 get_slope_value (Source File: LIS_domainMod.F90)

INTERFACE:

```
subroutine get_slope_value(n,c,r,i,slope_selected,slope,slope_index)
  integer  :: n
  integer  :: c
  integer  :: r
  integer  :: i
  logical  :: slope_selected
  real     :: slope
  integer  :: slope_index
```

DESCRIPTION:

This routine computes the slope value for given tile number and grid cell.

3.12.25 get_aspect_value (Source File: LIS_domainMod.F90)

INTERFACE:

```
subroutine get_aspect_value(n,c,r,i,aspect_selected,aspect,aspect_index)
```

ARGUMENTS:

```
  integer  :: n
  integer  :: c
  integer  :: r
  integer  :: i
  logical  :: aspect_selected
  real     :: aspect
  integer  :: aspect_index
```

DESCRIPTION:

This routine computes the aspect value for given tile number and grid cell.

3.13 Fortran: Module Interface LIS_emissMod (Source File: LIS_emissMod.F90)

The code in this file implements routines to read emissivity data.

3.13.1 Overview

This routines in this module provides routines to read the emissivity climatology data and allows the users to specify the frequency of climatology (in months). The climatological data is temporally interpolated between months to the current simulation date.

REVISION HISTORY:

```
 08 Aug 2005: Sujay Kumar; Initial implementation
use ESMF
use LIS_coreMod
use LIS_logMod
use LIS_timeMgrMod
use LIS_histDataMod
use LIS_fileIOMod

implicit none

PRIVATE

-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
 public :: LIS_emiss_setup    !allocates memory for required structures
 public :: LIS_read_emiss     !reads the emiss data
 public :: LIS_diagnoseemiss      !diagnoses emiss data for history output
 public :: LIS_emiss_finalize !cleanup allocated structures
 public :: LIS_emiss_reset    !resets alarms and data structures

-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
 public :: LIS_emiss !data structure containing emiss fraction data.
```

3.13.2 LIS_emiss_setup (Source File: LIS_emissMod.F90)

INTERFACE:

```
subroutine LIS_emiss_setup
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
    use netcdf
#endif
```

DESCRIPTION:

Allocates memory for data structures for reading the emiss fraction datasets
The routines invoked are:

emissivitysetup (3.44.2)
calls the registry to invoke the emiss setup methods.

3.13.3 LIS_read_emiss (Source File: LIS_emissMod.F90)

INTERFACE:

```
subroutine LIS_read_emiss(n)
```

USES:

```
implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Reads the emiss fraction climalotogy and temporally interpolates it to the current day.
The arguments are:

n index of the domain or nest.

The routines invoked are:

LIS_computeTemporalWeights (3.36.23)
computes the interpolation weights

rademissivity (3.44.4)
invokes the generic method in the registry to read the emiss climatology data

---

3.13.4 LIS_emiss_finalize (Source File: LIS_emissMod.F90)

INTERFACE:

```
subroutine LIS_emiss_finalize
```

USES:

DESCRIPTION:

Deallocates objects created in this module

3.13.5 LIS_diagnoseemiss (Source File: LIS_emissMod.F90)

INTERFACE:

```
subroutine LIS_diagnoseemiss(n)
```

USES:

ARGUMENTS:

```
implicit none
integer, intent(in)   :: n
```

DESCRIPTION:

This routine maps the emiss data to the LIS history writer.
The arguments are:

n index of the nest

The routines called are:

LIS_diagnoseOutputVar (3.16.5)
generic routine to map a single variable to the LIS history writer

---

3.13.6 LIS_emiss_reset (Source File: LIS_emissMod.F90)

INTERFACE:

```
subroutine LIS_emiss_reset
```

USES:

DESCRIPTION:

Resets the data structures for reading the emiss fraction datasets
The routines invoked are:

emissivitysetup (3.44.2)
calls the registry to invoke the emiss data reading methods.

3.13.7 read_emissclimo (Source File: LIS_emissMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_emissclimo(n,time,array)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
    use netcdf
#endif

    implicit none
```

ARGUMENTS:

```
    integer, intent(in)         :: n
    integer, intent(in)         :: time
    real, intent(inout)         :: array(LIS_rc%ntiles(n))
```

DESCRIPTION:

This subroutine reads the emiss data climatology
The arguments are:

n index of n

3.14 Fortran: Module Interface LIS_fileIOMod (Source File: LIS_fileIOMod.F90)

This module contains a number of routines useful for various file I/O operations in LIS. The module provides routines to create output directories, filenames, that can be used in the model output routines.

REVISION HISTORY:

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

USES:

```
implicit none
PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
public :: LIS_create_output_filename   ! create an output filename
public :: LIS_create_output_directory  ! create the output directory
public :: LIS_create_restart_filename  ! create a restart filename
public :: LIS_create_stats_filename    ! create a stats filename
public :: LIS_create_dapert_filename   ! create a perturbations file
public :: LIS_create_innov_filename    ! create an innovations filename
public :: LIS_create_daspread_filename ! create an innovations filename
public :: LIS_create_obs_filename      ! create an observations filename
public :: LIS_create_gain_filename
public :: LIS_putget                   !a generic read/write method.
public :: LIS_readData                 !generic method for reading surface parameters
public :: LIS_readDomainConfigSpecs    !generic method to read configurable options
                                       !for a particular surface dataset
public :: LIS_checkDomainExtents       !checks if the data domain extents are
                                       !contained in the LIS running domain
public :: LIS_read_param
public :: LIS_convertParamDataToLocalDomain
```

3.14.1 LIS_readData (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
interface LIS_readData
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure read2Ddata
```

DESCRIPTION:

Routine to read 2d data from a binary file, with direct access format. A special routine is required to read the landcover data since it includes a distribution of vegetation types at each grid point.

3.14.2 LIS_create_restart_filename (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
interface LIS_create_restart_filename
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure create_restart_filename
module procedure create_restart_filename_withtime
```

DESCRIPTION:

This routine puts together a restart filename, either based on the LIS current time or based on the time that is provided.

3.14.3 LIS_putget (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
interface LIS_putget
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure putget_int
module procedure putget_real
```

DESCRIPTION:

This is a generic method to read from or write to direct access files.
method:
- open file
- if iofunc = r, read buffer from file, abort on error
- if iofunc = w, write buffer to file, abort on error
- if iofunc = anything else, abort

REVISION HISTORY:

```
04 aug 1999 initial version ..........................mr gayno/dnxm
02 nov 2005 incorporated into LIS                      sujay kumar
```

3.14.4 LIS_read_param (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
interface LIS_read_param
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure readparam_real_2d
module procedure readparam_int_2d
```

DESCRIPTION:

Routine to read parameter data from a netcdf file.

3.14.5 LIS_convertParamDataToLocalDomain (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
interface LIS_convertParamDataToLocalDomain
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure convertParam_int
module procedure convertParam_real
```

DESCRIPTION:

Routine to subset the data read from the parameter attributes file supplied by LDT to the LIS domain being run. It is assumed that the parameter data domain is at the same spatial resolution and map projection as that of the LIS domain.

3.14.6 LIS_create_output_directory (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine LIS_create_output_directory(mname,dir_name)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_log_msg
implicit none
```

ARGUMENTS:

```
character(len=*)  :: mname
character(len=*), optional   :: dir_name
```

DESCRIPTION:

Create the output directory for the output data files. The call creates a hierarchy of directories in the following format, if the directory name is not specified.
2 level hierarchy

```
<output directory>/<model name>
```

3 level hierarchy

```
<output directory>/<model name>/<yrmo>
```

4 level hierarchy

```
<output directory>/<model name>/<yr>/<yrmoda>
```

Once the directory name is created, the subroutine issues a system call to create the structure.
The arguments are:

mname a string describing the name of the model

dir_name name of the directory to override the above format

3.14.7 LIS_create_output_filename (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine LIS_create_output_filename(n, fname, model_name, odir, writeint)
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use LIS_logMod,   only : LIS_log_msg, LIS_endrun

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
character(len=*), intent(out)          :: fname
character(len=*), intent(in), optional :: model_name ! needed for gswp run
character(len=*), intent(in), optional :: odir ! needed for gswp run
real, intent(in), optional             :: writeint ! output writing interval
```

DESCRIPTION:

Create the file name for the output data files. It creates both the GSWP style of output filenames and the standard LIS style. The convention used in LIS creates a filename in a hierarchical style (output directory, model name, date, file extention)
2 level hierarchy

```
<output directory>/<model name>/LIS_HIST_<yyyymmddhhmnss>.<extension>
```

3 level hierarchy

```
<output directory>/<model name>/<yyyymm>/LIS_HIST_<yyyymmddhhmnss>.<extension>
```

4 level hierarchy

```
<output directory>/<model name>/<yyyy>/<yyyymm>/LIS_HIST_<yyyymmddhhmnss>.<extension>
```

WMO convention

```
<output directory>/<AFWA Weather product style>
```

A filename in the convention of weather products (such as):
PS.AFWA_SC.U_DI.C_DC.ANLYS_GP.LIS_GR.C0P25DEG_AR.GLOBAL_PA.03-HR-SUM_DD.YYYYMMDD_DT.HH00_DF.GR1
where
PS = Product source
SC = security classification
DI = distribution classification
DC = data category
GP = generating process
GR = grid
AR = area of data
PA = parameter
DD = date
DT = data time
DF = data format

The arguments are:

n index of the domain or nest

fname the created file name.

model_name string describing the name of the model

writeint output writing interval of the model

style style option as described above

3.14.8 LIS_create_dapert_filename (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine LIS_create_dapert_filename(n, fname)
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use LIS_logMod,   only : LIS_log_msg, LIS_endrun

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
character(len=*), intent(out)          :: fname
```

DESCRIPTION:

Create the file name for the output data files. It creates both the GSWP style of output filenames and the standard LIS style. The convention used in LIS creates a filename in a hierarchical style (output directory, model name, date, file extention)
2 level hierarchy

```
<output directory>/<model name>/LIS_HIST_<yyyymmddhhmnss>.<extension>
```

3 level hierarchy

```
<output directory>/<model name>/<yyyymm>/LIS_HIST_<yyyymmddhhmnss>.<extension>
```

4 level hierarchy

```
<output directory>/<model name>/<yyyy>/<yyyymm>/LIS_HIST_<yyyymmddhhmnss>.<extension>
```

WMO convention

```
<output directory>/<AFWA Weather product style>
```

A filename in the convention of weather products (such as):
PS.AFWA_SC.U_DI.C_DC.ANLYS_GP.LIS_GR.C0P25DEG_AR.GLOBAL_PA.03-HR-SUM_DD.YYYYMMDD_DT.HH00_DF.GR1
where
PS = Product source
SC = security classification
DI = distribution classification
DC = data category
GP = generating process
GR = grid
AR = area of data
PA = parameter
DD = date
DT = data time
DF = data format

The arguments are:

n index of the domain or nest

fname the created file name.

model_name string describing the name of the model

writeint output writing interval of the model

style style option as described above

3.14.9 create_restart_filename (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine create_restart_filename(n, fname,dir_name,model_name,wformat)
```

USES:

```
use LIS_coreMod,    only : LIS_rc

implicit none
```

ARGUMENTS:

```
integer, intent(in)           :: n
character(len=*), intent(out) :: fname
character(len=*), intent(in)  :: dir_name
character(len=*), intent(in)  :: model_name
character(len=*), optional    :: wformat
```

DESCRIPTION:

Create the file name for the restart data files. The convention used in LIS creates a restart filename in the following format.

```
<output directory>/EXP<expno>/<dir name>/<yr>/<yrmoda>/<yrmodahrmn>.<extn>
```

The arguments are:

n index of the domain or nest

fname the created file name.

model_name string describing the name of the model

wformat optional argument specifying the restart file format.

3.14.10 create_restart_filename_withtime (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine create_restart_filename_withtime(n, fname,dir_name,model_name,&
     yr,mo,da,hr,mn,ss,wformat)
```

USES:

```
use LIS_coreMod,    only : LIS_rc

implicit none
```

ARGUMENTS:

```
integer, intent(in)           :: n
character(len=*), intent(out) :: fname
character(len=*), intent(in)  :: dir_name
character(len=*), intent(in)  :: model_name
integer         , intent(in)  :: yr, mo, da, hr, mn, ss
character(len=*), optional    :: wformat
```

DESCRIPTION:

Create the file name for the restart data files. The convention used in LIS creates a restart filename in the following format.

```
<output directory>/EXP<expno>/<dir name>/<yr>/<yrmoda>/<yrmodahrmn>.<extn>
```

The arguments are:

n index of the domain or nest

fname the created file name.

model_name string describing the name of the model

wformat optional argument specifying the restart file format.

3.14.11 LIS_create_stats_filename (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine LIS_create_stats_filename(n, fname, mname)
```

USES:

```
use LIS_coreMod,  only : LIS_rc

implicit none
```

ARGUMENTS:

```
integer, intent(in)           :: n
character(len=*), intent(out) :: fname
character(len=*), intent(in)  :: mname
```

DESCRIPTION:

Create the file name for the stats files. The convention used in LIS creates a restart filename in the following format.

```
<output directory>/EXP<expno>/<model name>stats.dat
```

The arguments are:

n index of the domain or nest

fname the created file name.

mname string describing the name of the model

style output directory style

3.14.12 LIS_create_innov_filename (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine LIS_create_innov_filename(n, fname, mname)
   use LIS_coreMod,  only : LIS_rc
   use LIS_logMod,   only : LIS_log_msg, LIS_endrun

   implicit none
```

ARGUMENTS:

```
   integer, intent(in) :: n
   character(len=*), intent(out)          :: fname
   character(len=*), intent(in)  :: mname
```

DESCRIPTION:

Create the file name for the output data files. It creates both the GSWP style of output filenames and the standard LIS style. The convention used in LIS creates a filename in a hierarchical style (output directory, model name, date, file extention)
2 level hierarchy

```
<output directory>/<model name>/LIS_HIST_<yyyymmddhhmnss>.<extension>
```

3 level hierarchy

```
<output directory>/<model name>/<yyyymm>/LIS_HIST_<yyyymmddhhmnss>.<extension>
```

4 level hierarchy

```
<output directory>/<model name>/<yyyy>/<yyyymm>/LIS_HIST_<yyyymmddhhmnss>.<extension>
```

WMO convention

```
<output directory>/<AFWA Weather product style>
```

A filename in the convention of weather products (such as):
PS.AFWA_SC.U_DI.C_DC.ANLYS_GP.LIS_GR.C0P25DEG_AR.GLOBAL_PA.03-HR-SUM_DD.YYYYMMDD_DT.HH00_DF.GR1
where
PS = Product source
SC = security classification
DI = distribution classification
DC = data category
GP = generating process
GR = grid
AR = area of data
PA = parameter
DD = date
DT = data time
DF = data format

The arguments are:

n index of the domain or nest

fname the created file name.

model_name string describing the name of the model

writeint output writing interval of the model

style style option as described above

3.14.13 LIS_create_daspread_filename (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine LIS_create_daspread_filename(n, fname, mname)
   use LIS_coreMod,  only : LIS_rc
   use LIS_logMod,   only : LIS_log_msg, LIS_endrun

   implicit none
```

ARGUMENTS:

```
   integer, intent(in) :: n
   character(len=*), intent(out)          :: fname
   character(len=*), intent(in)  :: mname
```

DESCRIPTION:

Create the file name for the output data files. It creates both the GSWP style of output filenames and the standard LIS style. The convention used in LIS creates a filename in a hierarchical style (output directory, model name, date, file extention)
2 level hierarchy

```
<output directory>/<model name>/LIS_HIST_<yyyymmddhhmnss>.<extension>
```

3 level hierarchy

```
<output directory>/<model name>/<yyyymm>/LIS_HIST_<yyyymmddhhmnss>.<extension>
```

4 level hierarchy

```
<output directory>/<model name>/<yyyy>/<yyyymm>/LIS_HIST_<yyyymmddhhmnss>.<extension>
```

WMO convention

```
<output directory>/<AFWA Weather product style>
```

A filename in the convention of weather products (such as):
PS.AFWA_SC.U_DI.C_DC.ANLYS_GP.LIS_GR.C0P25DEG_AR.GLOBAL_PA.03-HR-SUM_DD.YYYYMMDD_DT.HH00_DF.GR1
where
PS = Product source
SC = security classification
DI = distribution classification
DC = data category
GP = generating process

GR = grid
AR = area of data
PA = parameter
DD = date
DT = data time
DF = data format

The arguments are:

n index of the domain or nest

fname the created file name.

model_name string describing the name of the model

writeint output writing interval of the model

style style option as described above

3.14.14 LIS_create_obs_filename (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine LIS_create_obs_filename(n, fname, mname)
```

USES:

```
use LIS_coreMod,   only : LIS_rc

implicit none
```

ARGUMENTS:

```
integer, intent(in)           :: n
character(len=*), intent(out) :: fname
character(len=*), intent(in)  :: mname
```

DESCRIPTION:

Create the file name for the observation files. The convention used in LIS creates a restart filename in the following format.

```
<output directory>/EXP<expno>/<model name><timestamp>.obs
```

The arguments are:

n index of the domain or nest

fname the created file name.

mname string describing the name of the model

3.14.15 putget_int (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
!Private name: call using putget()
  subroutine putget_int ( buffer, iofunc, file_name, &
       routine_name, imax, jmax )
```

USES:

```
use LIS_logMod,    only : LIS_abort, LIS_endrun, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber

implicit none
```

ARGUMENTS:

```
integer,           intent(in)     :: imax
integer,           intent(in)     :: jmax
integer,           intent(inout)  :: buffer (imax , jmax)
character*1,       intent(in)     :: iofunc
character(len=*),  intent(in)     :: file_name
character(len=*),  intent(in)     :: routine_name
```

DESCRIPTION:

put or get any size integer array routine
The arguments are:

buffer integer buffer array

iofunc I/O function (r=read, w=write)

file_name input file name

routine_name name of the calling routine

imax grid dimension of the buffer array - East/West direction

jmax grid dimension of the buffer array - North/South direction

The methods invoked are:

LIS_abort (3.21.4)
abort the program in case of a fatal error

3.14.16 putget_real (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
!Private name: call using putget()
  subroutine putget_real ( buffer, iofunc, file_name, &
       routine_name, imax, jmax )
```

USES:

```
use LIS_logMod,    only : LIS_abort, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber
implicit none

integer,       intent(in)     :: imax
integer,       intent(in)     :: jmax
real,         intent(inout)   :: buffer      (imax , jmax)
character(len=*), intent(in)  :: file_name
character*1,   intent(in)     :: iofunc
character(len=*),  intent(in) :: routine_name
```

DESCRIPTION:

put or get any size real array routine
The arguments are:

buffer integer buffer array

iofunc I/O function (r=read, w=write)

file_name input file name

routine_name name of the calling routine

imax grid dimension of the buffer array - East/West direction

jmax grid dimension of the buffer array - North/South direction

The methods invoked are:

LIS_abort (3.21.4)
abort the program in case of a fatal error

3.14.17 LIS_create_gain_filename (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine LIS_create_gain_filename(n, fname, mname)
   use LIS_coreMod,  only : LIS_rc
   use LIS_logMod,   only : LIS_log_msg, LIS_endrun

   implicit none
```

ARGUMENTS:

```
   integer, intent(in) :: n
   character(len=*), intent(out)          :: fname
   character(len=*), intent(in)  :: mname
```

DESCRIPTION:

Create the file name for the output data files. It creates both the GSWP style of output filenames and the standard LIS style. The convention used in LIS creates a filename in a hierarchical style (output directory, model name, date, file extention)
2 level hierarchy

```
<output directory>/<model name>/LIS_HIST_<yyyymmddhhmnss>.<extension>
```

3 level hierarchy

```
<output directory>/<model name>/<yyyymm>/LIS_HIST_<yyyymmddhhmnss>.<extension>
```

4 level hierarchy

```
<output directory>/<model name>/<yyyy>/<yyyymm>/LIS_HIST_<yyyymmddhhmnss>.<extension>
```

WMO convention

```
<output directory>/<AFWA Weather product style>
```

A filename in the convention of weather products (such as):
PS.AFWA_SC.U_DI.C_DC.ANLYS_GP.LIS_GR.C0P25DEG_AR.GLOBAL_PA.03-HR-SUM_DD.YYYYMMDD_DT.HH00_DF.GR1
where
PS = Product source
SC = security classification
DI = distribution classification
DC = data category
GP = generating process
GR = grid
AR = area of data
PA = parameter
DD = date
DT = data time
DF = data format

The arguments are:

n index of the domain or nest

fname the created file name.

model_name string describing the name of the model

writeint output writing interval of the model

style style option as described above

3.14.18 read2Ddata (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine read2Ddata(n, ftn, gridDesc, array)
```

USES:

```
use LIS_coreMod
use LIS_logMod

implicit none
```

ARGUMENTS:

```
integer,  INTENT(IN)           :: n
integer,  INTENT(IN)           :: ftn
real,     INTENT(INOUT)        :: array(LIS_rc%lnc(n),LIS_rc%lnr(n))
real,     INTENT(IN)           :: gridDesc(6)
```

DESCRIPTION:

This routine retrieves a 2-d data from a binary direct access file.

3.14.19 LIS_readDomainConfigSpecs (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine LIS_readDomainConfigSpecs(segment_name, domain_info)
```

USES:

```
use ESMF
use LIS_coreMod,  only : LIS_rc, LIS_config
use LIS_logMod,   only : LIS_verify
```

ARGUMENTS:

```
character(len=*),  intent(in)    :: segment_name
real,              intent(inout) :: domain_info(:,:)
 real,              intent(inout) :: domain_info(LIS_rc%nnest,6)
```

DESCRIPTION:

This subroutine reads the relevant section of the lis.config file that defines the domain specifications for a particular surface parameter dataset, based on the map projection used to define the surface parameter dataset.

3.14.20 LIS_checkDomainExtents (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine LIS_checkDomainExtents(n, data_gridDesc)
```

USES:

```
use LIS_coreMod,  only : LIS_rc, LIS_domain
use LIS_logMod,   only : LIS_logunit, LIS_endrun
use map_utils,    only : ij_to_latlon
```

ARGUMENTS:

```
integer              :: n
real                 :: data_gridDesc(6)
```

DESCRIPTION:

This subroutine checks to see if the domain extents of the data is within the LIS running domain.

3.14.21 readparam_real_2d (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine readparam_real_2d(n,pname,array)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
  use LIS_coreMod,        only : LIS_rc, LIS_localPet,&
       LIS_domain, LIS_ews_ind, LIS_ewe_ind,&
       LIS_nss_ind, LIS_nse_ind, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
       LIS_nss_halo_ind, LIS_nse_halo_ind
  use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
       LIS_releaseUnitNumber, LIS_endrun, LIS_verify

  implicit none
```

ARGUMENTS:

```
  integer, intent(in)    :: n
  character(len=*)       :: pname
  real,    intent(inout) :: array(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

Reads a parameter field from the input LIS parameter file
The arguments are:

n index of n

pname name of the parameter field

array retrieved parameter value

3.14.22 readparam_int_2d (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine readparam_int_2d(n,pname,array)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
  use LIS_coreMod,        only : LIS_rc, LIS_localPet,&
       LIS_domain, LIS_ews_ind, LIS_ewe_ind,&
       LIS_nss_ind, LIS_nse_ind, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
       LIS_nss_halo_ind, LIS_nse_halo_ind
  use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
       LIS_releaseUnitNumber, LIS_endrun, LIS_verify

  implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
character(len=*)       :: pname
integer,    intent(inout) :: array(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

Reads a parameter field from the input LIS parameter file
The arguments are:

n index of n

pname name of the parameter field

array retrieved parameter value

3.14.23 convertParam_real (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine convertParam_real(n, pdata, ldata)
```

USES:

```
use LIS_coreMod
use map_utils
```

ARGUMENTS:

```
integer,     intent(in)    :: n
real,        intent(in)    :: pdata(LIS_rc%pnc(n),LIS_rc%pnr(n))
real,        intent(inout) :: ldata(LIS_rc%gnc(n),LIS_rc%gnr(n))
```

DESCRIPTION:

This routine subsets the floating point data read from the parameter attributes file supplied by LDT to the LIS domain being run. It is assumed that the parameter data domain is at the same spatial resolution and map projection as that of the LIS domain.

3.14.24 convertParam_int (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine convertParam_int(n, pdata, ldata)
```

USES:

```
use LIS_coreMod
use map_utils
```

ARGUMENTS:

```
integer,     intent(in)    :: n
integer,     intent(in)    :: pdata(LIS_rc%pnc(n),LIS_rc%pnr(n))
integer,     intent(inout) :: ldata(LIS_rc%gnc(n),LIS_rc%gnr(n))
```

DESCRIPTION:

This routine subsets the floating point data read from the parameter attributes file supplied by LDT to the LIS domain being run. It is assumed that the parameter data domain is at the same spatial resolution and map projection as that of the LIS domain.

3.15 Fortran: Module Interface LIS_gswpMod (Source File: LIS_gswpMod.F90)

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

REVISION HISTORY:

```
 24 Feb 2004    Sujay Kumar  Initial Specification

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: getgswp_monindex
public :: getgswp_timeindex
```

3.15.1 getgswp_monindex (Source File: LIS_gswpMod.F90)

INTERFACE:

```
subroutine getgswp_monindex(yr,mo,index)
```

ARGUMENTS:

```
  integer, intent(out) :: index
  integer, intent(in) :: yr, mo
```

DESCRIPTION:

This routine computes the GSWP monthly index (base time being 1982), to be used to read monthly data.

3.15.2 getgswp_timeindex (Source File: LIS_gswpMod.F90)

INTERFACE:

```
subroutine getgswp_timeindex(yr,mo,da,hr,mn,ss,index)

  implicit none
```

ARGUMENTS:

```
integer, intent(in)  :: yr, mo, da, hr, mn, ss
integer, intent(out) :: index
```

DESCRIPTION:

This routine computes the GSWP timestep index (base time being 1982), to be used to read the forcing files. Note: GSWP forcing data are written into monthly files with a frequency of 3 hours. The first entry corresponds to 03:00:00 of the first day of the given month. The last entry corresponds to 00:00:00 of the first day of the next month.
E.g.; for Tair_cru198207.nc, the data run from 1982-07-01T03:00:00 through 1982-08-01T00:00:00, inclusive. So, when you are at hour 0 on the first day of the month, reset the day to the last day of the previous month, and reset the hour from 0 to 24. This will compute the correct index for the last entry in the forcing file. E.g.; 1982-08-01T00:00:00 should be re-written as 1982-07-31T24:00:00.

3.16 Fortran: Module Interface LIS_histDataMod (Source File: LIS_histDataMod.F90)

This module is used by the user to define the metadata associated with model output. The module lists a superset of the land surface model variables. The user can choose a subset from this list through the lis configuration file for model output. Currently this list includes the variable definitions from ALMA specification.
http://www.lmd.jussieu.fr/ALMA/

REVISION HISTORY:

```
21 Oct 2005    Sujay Kumar  Initial Specification
19 Jan 2007    Chuck Alonge Added Snow Depth Option (Future use in WPS)
 4 Jul 2008    Sujay Kumar Redesigned the I/O to enable generic I/O for
                   all LSMs.
11 May 2011    Sujay Kumar, Updated to be generic across all LIS I/O and
                   not just land surface model output
26 May 2011    Soni Yatheendradas: Added Potential ET forcings for FEWSNET
11 Jan 2012    Jiarui Dong, Added the Sac-HT/Snow17 implementation
28 Jan 2014    David Mocko; Updates for AFWA GRIB files using GRIBAPI library
21 Feb 2014    David Mocko; More updates for matching AFWA GRIB files
                   to the LIS-6 GRIB output
14 Mar 2014    David Mocko: Added CAPE, CRAINFORC, CMFORC, and CHFORC outputs
28 Mar 2014    David Mocko: More refinements/fixes to AFWA GRIB CONFIGS
```

```
real*8  :: time         ! time to begin writing output
integer :: syear        ! year to begin writing output
integer :: smonth       ! month to begin writing output
integer :: sday         ! day to begin writing output
integer :: shour        ! hour to begin writing output
integer :: smin         ! minutes to begin writing output
integer :: ssec         ! seconds to begin writing output

integer :: month       ! specific output writing time (month)
integer :: day         ! specific output writing time (day)
integer :: hour        ! specific output writing time (hour)
integer :: min         ! specific output writing time (min)
integer :: sec         ! specific output writing time (sec)

type(LIS_metadataEntry), pointer :: head_lsm_list
type(LIS_metadataEntry), pointer :: head_routing_list
```

```
type(LIS_metadataEntry), pointer :: head_rtm_list
type(LIS_metadataEntry), pointer :: head_irrig_list

type(dep), allocatable, dimension(:) :: ptr_into_lsm_list
type(dep), allocatable, dimension(:) :: ptr_into_routing_list
type(dep), allocatable, dimension(:) :: ptr_into_rtm_list
type(dep), allocatable, dimension(:) :: ptr_into_irrig_list
```

3.16.1 LIS_histDataInit (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine LIS_histDataInit(n, ntiles)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer,  intent(IN)   :: n
integer,  intent(IN)   :: ntiles
```

DESCRIPTION:

This routine initializes the required linked lists to hold the selected list of LSM variables
The arguments are:

n index of the nest

ntiles size of the tilespace

The routines invoked are:

get_moc_attributes (3.16.2)
adds history output objects to the linked lists and sets the user-definable elements

register_dataEntry (3.16.14)
completes the setting of the elements in the history output objects and allocates the data structures related to the variable being output

LIS_resetOutputVars (3.16.11)
resets the arrays storing the variable values.

3.16.2 get_moc_attributes (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine get_moc_attributes(modelSpecConfig, head_dataEntry, short_name, &
                              standard_name, long_name, status)
```

DESCRIPTION:

This routine reads the model output configuration attributes for the given history output variable.
For each user-selected history output variable, this routine allocates the history output object for the given history output linked list, and it then assigns the user-definable attributes.

3.16.3 allocate_dataEntry (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine allocate_dataEntry(dataEntry,nunits,ntiles,unittypes,ndirs,&
     dirtypes, model_patch)

  implicit none
```

ARGUMENTS:

```
  type(LIS_metadataEntry), pointer :: dataEntry
  integer                 :: nunits
  integer                 :: ntiles
  character(len=*)        :: unittypes(nunits)
  integer                 :: ndirs
  character(len=*)        :: dirtypes(ndirs)
  logical                 :: model_patch
```

DESCRIPTION:

This routine initializes the datastructures required for the specified output variable.

3.16.4 convertToCFunits (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine convertToCFunits(unit,cfunit)
```

DESCRIPTION:

This routine converts the LIS specified units to CF compliant unit specifications.

3.16.5 LIS_diagnoseSurfaceOutputVar (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine LIS_diagnoseSurfaceOutputVar(n, t, index, vlevel, value, unit,&
                                        direction, valid_min, valid_max, &
                                        surface_type)
  use  LIS_coreMod, only : LIS_domain
  implicit none
```

ARGUMENTS:

```
integer, intent(in)           :: n
integer, intent(in)           :: t
integer, intent(in)           :: index
integer, intent(in)           :: vlevel
real   , intent(in)           :: value
character(len=*), intent(in)  :: unit
character(len=*)              :: direction
real,    intent(in), optional :: valid_min
real,    intent(in), optional :: valid_max
integer, intent(in), optional :: surface_type
```

DESCRIPTION:

This routine is the user callable interface to the LIS_diagnoseOutputVar routine.
See LIS_diagnoseOutputVar for more details.

3.16.6 LIS_diagnoseRoutingOutputVar (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine LIS_diagnoseRoutingOutputVar(n, t, index, vlevel, value, unit,&
                                        direction,valid_min,valid_max)
  implicit none
```

ARGUMENTS:

```
integer, intent(in)           :: n
integer, intent(in)           :: t
integer, intent(in)           :: index
integer, intent(in)           :: vlevel
real   , intent(in)           :: value
character(len=*), intent(in)  :: unit
character(len=*)              :: direction
real,    intent(in), optional :: valid_min
real,    intent(in), optional :: valid_max
```

DESCRIPTION:

This routine is the user callable interface to the LIS_diagnoseOutputVar routine.
See LIS_diagnoseOutputVar for more details.

3.16.7 LIS_diagnoseRTMOutputVar (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine LIS_diagnoseRTMOutputVar(n, t, index, vlevel, value, unit,&
                                    direction,valid_min,valid_max)
  use  LIS_coreMod, only : LIS_domain
  implicit none
```

ARGUMENTS:

```
integer, intent(in)           :: n
integer, intent(in)           :: t
integer, intent(in)           :: index
integer, intent(in)           :: vlevel
real   , intent(in)           :: value
character(len=*), intent(in)  :: unit
character(len=*)              :: direction
real,    intent(in), optional :: valid_min
real,    intent(in), optional :: valid_max
```

DESCRIPTION:

This routine is the user callable interface to the LIS_diagnoseOutputVar routine.
See LIS_diagnoseOutputVar for more details.

3.16.8 LIS_diagnoseIrrigationOutputVar (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine LIS_diagnoseIrrigationOutputVar(n, t, index, vlevel, value, unit,&
                                           direction,valid_min,valid_max)
  use  LIS_coreMod, only : LIS_domain
  implicit none
```

ARGUMENTS:

```
integer, intent(in)           :: n
integer, intent(in)           :: t
integer, intent(in)           :: index
integer, intent(in)           :: vlevel
real   , intent(in)           :: value
character(len=*), intent(in)  :: unit
character(len=*)              :: direction
real,    intent(in), optional :: valid_min
real,    intent(in), optional :: valid_max
```

DESCRIPTION:

This routine is the user callable interface to the LIS_diagnoseOutputVar routine.
See LIS_diagnoseOutputVar for more details.

3.16.9 LIS_diagnoseOutputVar (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine LIS_diagnoseOutputVar(head_dataEntry,&
                                 count, ptr_into_list,&
                                 n, t, index, vlevel, value, unit,&
                                 direction,valid_min,valid_max,&
                                 model_patch)
  use  LIS_coreMod, only : LIS_domain
  use  LIS_logMod, only : LIS_logunit, LIS_endrun
  implicit none
```

ARGUMENTS:

```
type(LIS_metadataEntry), pointer, intent(in) :: head_dataEntry
integer, intent(in)           :: count
type(dep), dimension(count), intent(in) :: ptr_into_list
integer, intent(in)           :: n
integer, intent(in)           :: t
integer, intent(in)           :: index
integer, intent(in)           :: vlevel
real   , intent(in)           :: value
character(len=*), intent(in)  :: unit
character(len=*)              :: direction
real,    intent(in), optional :: valid_min
real,    intent(in), optional :: valid_max
logical, intent(in), optional :: model_patch
```

DESCRIPTION:

This routine maps the LSM, routing, or RTM specific variable to the appropriate variable in the given history output linked list, and it also enables time averaging of the variable, if specified.
The arguments are:

head_dataEntry head of the given history linked list

n index of the nest

t index of the specified variable in the tile space

index the index of the specified variable in the generic (ALMA) variable list

vlevel vertical dimension of the variable, if any

value value of the specified variable

3.16.10 diagnoseDataEntry (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine diagnoseDataEntry(n,dataentry, t, vlevel,in_value, unit, &
                             direction, vmin, vmax, nsiblings, &
                             siblings,model_patch)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer                 :: n
type(LIS_metadataEntry) :: dataEntry
integer                 :: t
integer                 :: vlevel
real, intent(in)        :: in_value
character(len=*)        :: unit
character(len=*)        :: direction
real                    :: vmin
real                    :: vmax
integer, intent(in)     :: nsiblings
integer, intent(in), dimension(nsiblings) :: siblings
logical                 :: model_patch
```

DESCRIPTION:

This routine maps a single output variable to the appropriate variable in the generic list of the LIS history writer.

3.16.11 LIS_resetOutputVars (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine LIS_resetOutputVars(n,group)
```

ARGUMENTS:

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

DESCRIPTION:

This routine resets the specified variables to enable time averaging for the next history output step.
It also resets the minimum and maximum fields.
The arguments are:

n index of the nest

group output group (1- LSM, 2-ROUTING, 3-RTM)

3.16.12 resetOutputVar (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine resetOutputVar(dataEntry)

implicit none
```

ARGUMENTS:

```
type(LIS_metadataEntry), pointer :: dataEntry
```

DESCRIPTION:

This routine resets the specified variable to enable time averaging for the next history output step.
It also resets the minimum and maximum fields.
The arguments are:

dataEntry data structure to reset

3.16.13 LIS_rescaleCount (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine LIS_rescaleCount(n,group)

  use LIS_mpiMod

   implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine rescales the count. dataEntryhow many time the corresponding variable has been diagnosed. This count is incremented each time the diagnose routine is called in terms of both time and space. Say variable var has 10 tiles, and it is diagnosed for 3 model time-steps, then its count will be 30. Thus count must be rescaled by the number of tiles to get the desired number of model time-step calls.
count is used to time-average output. count is not used when writing accumulated output. count is 1 when diagnosing instantaneous output.
The arguments are:

n index of the nest

group output group (1 = LSM; 2 = ROUTING; 3 = RTM)

3.16.14 register_dataEntry (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine register_dataEntry(count,lis_moc_index,head_dataEntry,&
                              n,nunits,ntiles,&
                              unittypes,ndirs,dirtypes,form,gribsfc,griblvl,&
                              model_patch)
implicit none
```

ARGUMENTS:

```
integer                 :: count
integer                 :: lis_moc_index
type(LIS_metadataEntry), pointer :: head_dataEntry
integer                 :: n
integer                 :: nunits
integer                 :: ntiles
character(len=*)        :: unittypes(nunits)
integer                 :: ndirs
character(len=*)        :: dirtypes(ndirs)
integer                 :: form
integer                 :: gribsfc
integer                 :: griblvl
logical, optional       :: model_patch
```

DESCRIPTION:

This routine completes the assignment of elements of the history output linked list object corresponding to the variable given by lis_moc_index.
The arguments are:

lis_moc_index variable to process

head_dataEntry head of the given history output linked list

n nest

nunits number of units associated with this variable

ntiles number of tiles

unittypes types of units associated with this variable

ndirs number of direction types associated with this variable

dirtypes types of direction types associated with this variable

form form for stats file

gribsfc type of GRIB surface

griblvl type of GRIB level

The routines invoked are:

allocate_dataEntry (3.16.3)
allocates the data structures related to the variable being output

---

3.16.15 set_ptr_into_list (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine set_ptr_into_list(count, head, array)
   implicit none
```

ARGUMENTS:

```
integer                          :: count
type(LIS_metadataEntry), pointer :: head
type(dep), dimension(count)      :: array
```

DESCRIPTION:

This routine takes an array of pointers and sets each element of the array to point directly to its corresponding element in the given history output linked list. This allows for direct access to the elements in the history output linked list.
The arguments are:

count number of elements in the given history output linked list

head head of the given history output linked list

array array of pointers to point directly to the elements of the given history output linked list

---

3.17 Fortran: Module Interface LIS_historyMod (Source File: LIS_historyMod.F90)

The code in this file provides interfaces to manage LIS model output in different file formats.

3.17.1 Overview

The module provides generic interfaces to manage output from different land surface models. Currently LIS supports three data formats.

- binary: in binary, big-endian.
- grib1: WMO grib1 format
- netcdf: NCAR unidata netcdf-4 format

This module also provides generic interfaces for writing and reading model restart files. LIS uses the following convention on the filename extensions:

- .1gd4r: binary output, 1 correspondes to one variable, g represents gridded data, d represnts direct access, 4r represents 4 byte real
- .grb: files in grib1 format
- .nc: files in netcdf-4 format
- *rst: restart files. Currently all restart files are written in netcdf-4 format

This module also manages any data aggregatiion from computing nodes when parallel processing is employed.
!NOTES: LIS history writer employs grib-api library for writing grib output and NETCDF-4 for writing NETCDF output. If the support for these data formats are expected, these libraries must be downloaded externally and built before LIS compilation.
GRIB-API: http://www.ecmwf.int/products/data/software/grib_api.html
NETCDF-4: http://www.unidata.ucar.edu/software/netcdf/

REVISION HISTORY:

```
    10 Feb 2004: Sujay Kumar; Initial Specification
     4 Jul 2008: Sujay Kumar; Redesigned with generic routines that can handle
                     model output in different formats for all LSMs
    25 Jan 2012: Sujay Kumar; The grib writers were modified to use grib-api
    28 Feb 2012: Sujay Kumar; Updated NETCDF interfaces to use NETCDF-4 library
    28 Jan 2014: David Mocko; Updates for AFWA GRIB files using GRIBAPI library
    21 Feb 2014: David Mocko; More updates for matching AFWA GRIB files
                     to the LIS-6 GRIB output
    28 Mar 2014: David Mocko; More refinements/fixes to AFWA GRIB CONFIGS
```

USES:

```
  use LIS_coreMod
  use LIS_histDataMod
  use LIS_timeMgrMod
  use LIS_logMod

#if ( defined USE_GRIBAPI)
  use grib_api
#endif

#if ( defined USE_NETCDF3 || defined USE_NETCDF4 )
  use netcdf
#endif


#if (defined SPMD)
#if (defined USE_INCLUDE_MPI)
  implicit none
  include 'mpif.h'
#else
  use mpi
  implicit none
#endif
#endif

  PRIVATE
 ------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
 ------------------------------------------------------------------------------
  public :: LIS_writevar_bin    ! write a variable into a binary file
  public :: LIS_writevar_grib1   ! write a variable into a grib1 file
  public :: LIS_writevar_netcdf ! write a variable into a netcdf file
  public :: LIS_writevar_restart ! writes a variable into a restart file
  public :: LIS_readvar_restart ! reads a variable from a restart file
  public :: LIS_writevar_reduced_tilespace!writes a variable in reduced tilespace (ngrid)
  public :: LIS_readvar_reduced_tilespace !reads a reduced tilespace variable
  public :: LIS_readvar_gridded ! read a variable from a gridded file
  public :: LIS_writevar_gridded ! write a variable in grid space to a file
  public :: LIS_tile2grid       ! convert a tilespace variable to gridspace
  public :: LIS_patch2tile       ! convert a patchspace variable to tilespace
  public :: LIS_grid2patch       ! convert a gridspace variable to patchspace
  public :: LIS_write_innov     ! writes innovations to a gridded file
  public :: LIS_write_dastats   ! writes DA-related diagnostics to an ASCII file
  public :: LIS_writeModelOutput ! writes model output based on the selected
```

```
                                        ! format and list of variables
public :: LIS_gather_gridded_output  !gather the 1d tiled output variable into a 2d gridded array
public :: LIS_gather_tiled_vector_output !gather the 1d tiled output variables in tile space
public :: LIS_gather_gridded_vector_output !gather the 1d tiled output variable into a 1d gridded ar
public :: LIS_gather_tiled_vector_withhalo_output
public :: LIS_writeGlobalHeader_restart
public :: LIS_writeHeader_restart
public :: LIS_closeHeader_restart
public :: LIS_convertVarToLocalSpace
```

3.17.2 LIS_writevar_bin (Source File: LIS_historyMod.F90)

INTERFACE:

```
interface LIS_writevar_bin
```

PRIVATE MEMBER FUNCTIONS:

```
   module procedure writevar_bin_real
   module procedure writevar_bin_real_direct
   module procedure writevar_bin_withstats_real
```

DESCRIPTION:

This interface provides routines for writing variables (real) in a binary file. The aggregation from decomposed tile spaces in each processor to their respective grid spaces and the aggregations of these grid spaces from each processor are also performed by this routine. The interface also provides options to write some diagnostic statistics about the variable being written.

3.17.3 LIS_writevar_restart (Source File: LIS_historyMod.F90)

INTERFACE:

```
interface LIS_writevar_restart
```

PRIVATE MEMBER FUNCTIONS:

```
   module procedure writevar_restart_tile_int
   module procedure writevar_restart_tile_real
   module procedure writevar_restart_patch_int
   module procedure writevar_restart_patch_real
    module procedure writevar_restart_char
```

DESCRIPTION:

This interface provides routines for writing variables (integer or real) in a restart file. By definition, the restart files are written in the model tile space. The aggregations from tile spaces of each processors are also performed by this routine.

3.17.4 LIS_writevar_reduced_tilespace (Source File: LIS_historyMod.F90)

INTERFACE:

```
interface LIS_writevar_reduced_tilespace
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure writevar_tile_noensemble_real
```

DESCRIPTION:

This interface provides routines for writing variables in a reduced tilespace (tilespace without ensembles).

3.17.5 LIS_readvar_reduced_tilespace (Source File: LIS_historyMod.F90)

INTERFACE:

```
interface LIS_readvar_reduced_tilespace
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure readvar_tile_noensemble_real
```

DESCRIPTION:

This interface provides routines for writing variables in a reduced tilespace (tilespace without ensembles).

3.17.6 LIS_readvar_restart (Source File: LIS_historyMod.F90)

INTERFACE:

```
interface LIS_readvar_restart
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure readvar_restart_tile_int
module procedure readvar_restart_tile_real
module procedure readvar_restart_patch_real
module procedure readvar_restart_patch_int
 module procedure readvar_restart_char
```

DESCRIPTION:

This interface provides routines for reading variables (integer or real) from a restart file. The restart files are read by the master processor. The domain decomposition of the "global" tile space on the master processor to individual processors are also performed by this routine.

3.17.7 LIS_readvar_gridded (Source File: LIS_historyMod.F90)

INTERFACE:

```
interface LIS_readvar_gridded
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure readvar_1dgridded_real
module procedure readvar_2dgridded_real
module procedure readvar_1dgridded_fromvector_real
```

DESCRIPTION:

This interface provides routines for reading variables (real) from a gridded (output) file into a 1d or 2d local gridded space. The gridded files are read by the master processor. The domain decomposition of the "global" grid space on the master processor to individual processors are also performed by this routine.

3.17.8 LIS_writevar_gridded (Source File: LIS_historyMod.F90)

INTERFACE:

```
interface LIS_writevar_gridded
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure writevar_gridded_real
module procedure writevar_gridded_real_withstats
```

DESCRIPTION:

This interface provides routines for writing variables (real) in grid space to a file.

3.17.9 LIS_writevar_grib1 (Source File: LIS_historyMod.F90)

INTERFACE:

```
interface LIS_writevar_grib1
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure writevar_grib1_withstats_real
```

DESCRIPTION:

This interface provides routines for writing variables (real) in a grib file. The aggregation from decomposed tile spaces in each processor to their respective grid spaces and the aggregations of these grid spaces from each processor are also performed by this routine. The interface also provides options to write some diagnostic statistics about the variable being written.

3.17.10 LIS_grid2patch (Source File: LIS_historyMod.F90)

INTERFACE:

```
interface LIS_grid2patch
```

PRIVATE MEMBER FUNCTIONS:

```
     module procedure grid2patch
    module procedure grid2patch_global
    module procedure grid2patch_global_ens
!DESCRIPTON:
```

3.17.11 LIS_writevar_netcdf (Source File: LIS_historyMod.F90)

INTERFACE:

```
interface LIS_writevar_netcdf
```

PRIVATE MEMBER FUNCTIONS:

```
    module procedure writevar_netcdf_withstats_real
```

DESCRIPTION:

This interface provides routines for writing variables (2d or 3d) in a netcdf file. The aggregation from decomposed tile spaces in each processor to their respective grid spaces and the aggregations of these grid spaces from each processor are also performed by this routine.

3.17.12 LIS_gather_gridded_output (Source File: LIS_historyMod.F90)

INTERFACE:

```
interface LIS_gather_gridded_output
```

PRIVATE MEMBER FUNCTIONS:

```
    module procedure gather_gridded_output_tile
    module procedure gather_gridded_output_patch
```

DESCRIPTION:

This interface provides routines for writing variables (real) in a binary file. The aggregation from decomposed tile spaces in each processor to their respective grid spaces and the aggregations of these grid spaces from each processor are also performed by this routine. The interface also provides options to write some diagnostic statistics about the variable being written.

3.17.13 LIS_tile2grid (Source File: LIS_historyMod.F90)

INTERFACE:

```
interface LIS_tile2grid
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure tile2grid_local
module procedure tile2grid_global_ens
module procedure tile2grid_global_noens
```

DESCRIPTION:

This interface provides routines for converting a tile space variable to the grid space

---

3.17.14 LIS_writeModelOutput (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine LIS_writeModelOutput(n, lsmoutfile, lsmstatsfile, &
     sopen, nsoillayers, outInterval, lyrthk, &
     group, model_name)
```

USES:

ARGUMENTS:

```
integer,   intent(in)   :: n
character(len=*),   intent(in)   :: lsmoutfile
character(len=*),   intent(in)   :: lsmstatsfile
logical,   intent(in)   :: sopen
real,      intent(in)   :: outInterval
integer,   intent(in)   :: nsoillayers
real,      intent(in)   :: lyrthk(nsoillayers)
integer,   intent(in),optional :: group
character(len=*), intent(in),optional :: model_name
```

DESCRIPTION:

This subroutine invokes the routine to write LSM output in the selected data format (binary/grib1/netcdf) and using the selected list of variables. Further, the variables are also written as instantaneous, time averaged, or accumulated based on the user specifications.
The arguments are:

n index of the nest

lsmoutfile name of the LSM history file

lsmstatsfile name of the LSM history stats file

sopen flag determining whether to open the LIS history stats file

outInterval history output frequency

nsoillayers Number of soil layers

lyrthk Thickness of soil layers

The routines invoked are:

writeBinaryOutput (3.17.15)
writes the history files in binary format

writeGrib1Output (3.17.17)
writes the history files in GRIB1 format

writeNetcdfOutput (3.17.19)
writes the history files in NETCDF format

LIS_resetOutputVars (3.16.11)
resets the time averaged varibles for the next output.

3.17.15 writeBinaryOutput (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine writeBinaryOutput(n, group, ftn, ftn_stats)
```

USES:

ARGUMENTS:

```
implicit none
integer,   intent(in)   :: n
integer,   intent(in)   :: group
integer,   intent(in)   :: ftn
integer,   intent(in)   :: ftn_stats
```

DESCRIPTION:

This routine writes a binary output file based on the list of selected output variables. The arguments are:

n index of the nest

group output group (LSM, routing, RTM)

ftn file unit for the output file

ftn_stats file unit for the output statistics file

The routines invoked are:

writeSingleBinaryVar (3.17.16)
writes a single variable into a flat binary file.

3.17.16 writeSingleBinaryVar (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine writeSingleBinaryVar(ftn, ftn_stats, n, dataEntry)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer :: ftn
integer :: ftn_stats
integer :: n
type(LIS_metadataEntry), pointer :: dataEntry
```

DESCRIPTION:

This routine writes a single variable to a binary file The arguments are:

n index of the nest

ftn file unit for the output file

ftn_stats file unit for the output statistics file

dataEntry object containing the specifications for the output variable

The routines invoked are:

LIS_writevar_bin (3.17.2) writes a variable into a flat binary file.

LIS_endrun (3.21.12) call to abort program when a fatal error is detected.

3.17.17 writeGrib1Output (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine writeGrib1Output(n, group, ftn,ftn_stats, outInterval, &
                            nsoillayers, lyrthk)
```

USES:

ARGUMENTS:

```
implicit none
integer,   intent(in)   :: n
integer,   intent(in)   :: group
integer,   intent(in)   :: ftn
integer,   intent(in)   :: ftn_stats
real,      intent(in)   :: outInterval
integer,   intent(in)   :: nsoillayers
real,      intent(in)   :: lyrthk(nsoillayers)
```

DESCRIPTION:

This routine writes an output file in the GRIB1 format based on the list of selected output variables. The arguments are:

n index of the nest

ftn file unit for the output file

ftn_stats file unit for the output statistics file

outInterval history output frequency

nsoillayers Number of soil layers

lyrthk Thickness of soil layers The depths for layers below the land surface (surface = 112) are expected in centimeters, per GRIB specifications. Convert lyrthk to cm in the call to LIS_writeModelOutput.

The routines invoked are:

writeSingleGrib1Var (3.17.18)
writes a single variable into a grib1 formatted file.

3.17.18 writeSingleGrib1Var (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine writeSingleGrib1Var(ftn,ftn_stats,n,time_unit, time_past, &
    time_curr, nsoillayers,toplev, botlev,var_index,dataEntry)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer,   intent(in)    :: n
integer,   intent(in)    :: ftn
integer,   intent(in)    :: ftn_stats
integer,   intent(in)    :: var_index
integer,   intent(in)    :: time_unit
integer,   intent(in)    :: time_past
integer,   intent(in)    :: time_curr
integer                  :: nsoillayers
real, dimension(nsoillayers) :: toplev, botlev
real, dimension(nsoillayers) :: toplevtemp, botlevtemp
type(LIS_metadataEntry),pointer :: dataEntry
```

DESCRIPTION:

This routine writes a single variable to a GRIB1 file The arguments are:

n index of the nest

ftn file unit for the output file

ftn_stats file unit for the output statistics file

var_index index of the variable

dataEntry

The routines invoked are:

LIS_writevar_grib1 (3.17.9) writes a variable into a grib1 formatted file.

LIS_endrun (3.21.12) call to abort program when a fatal error is detected.

3.17.19 writeNetcdfOutput (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine writeNetcdfOutput(n, group, ftn, ftn_stats, outInterval, &
     nsoillayers, lyrthk, model_name)
```

USES:

ARGUMENTS:

```
integer,   intent(in)   :: n
integer,   intent(in)   :: group
integer,   intent(in)   :: ftn
integer,   intent(in)   :: ftn_stats
real,      intent(in)   :: outInterval
integer,   intent(in)   :: nsoillayers
real,      intent(in)   :: lyrthk(nsoillayers)
character*100, intent(in) :: model_name
```

DESCRIPTION:

This routine writes an output file in the NETCDF format based on the list of selected output variables. The arguments are:

n index of the nest

ftn file unit for the output file

ftn_stats file unit for the output statistics file

outInterval history output frequency

nsoillayers Number of soil layers

lyrthk Thickness of soil layers

model_name Name of the model that generates the output

The routines invoked are:

defineNETCDFheadervar (3.17.20) writes the required headers for a single variable

writeSingleNETCDFvar (3.17.21) writes a single variable into a netcdf formatted file.

LIS_verify (3.21.1) call to check if the return value is valid or not.

3.17.20 defineNETCDFheaderVar (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine defineNETCDFheaderVar(n,ftn,dimID, dataEntry, non_model_fields)
```

USES:

ARGUMENTS:

```
integer                 :: n
integer                 :: ftn
type(LIS_metadataEntry), pointer :: dataEntry
logical,   optional     :: non_model_fields
integer      :: dimID(3)
```

DESCRIPTION:

This routine writes the required NETCDF header for a single variable
The arguments are:

n index of the nest

ftn NETCDF file unit handle

dimID NETCDF dimension ID corresponding to the variable

dataEntry object containing the values and attributes of the variable to be written

The routines invoked are:

LIS_endrun (3.21.12) call to abort program when a fatal error is detected.

LIS_verify (3.21.1) call to check if the return value is valid or not.

3.17.21 writeSingleNETCDFvar (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine writeSingleNETCDFvar(ftn,ftn_stats,n,dataEntry,&
     non_model_fields)
```

USES:

```
use LIS_coreMod,   only : LIS_rc

implicit none

integer,   intent(in)   :: n
integer,   intent(in)   :: ftn
integer,   intent(in)   :: ftn_stats
type(LIS_metadataEntry), pointer :: dataEntry
logical,   optional     :: non_model_fields
```

DESCRIPTION:

This routine writes a single variable to a NETCDF file The arguments are:

ftn file unit for the output file

ftn_stats file unit for the output statistics file

n index of the nest

dataEntry object containing the values and attributes of the variable to be written

The routines invoked are:

LIS_writevar_netcdf (3.17.11) writes a variable into a netcdf formatted file.

3.17.22 LIS_writeGlobalHeader_restart (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine LIS_writeGlobalHeader_restart(ftn,n,m, &
     model_name, dimID, dim1,dim2,dim3,dim4,dim5,&
     output_format)
```

USES:

ARGUMENTS:

```
integer,   intent(in)     :: n
integer,   intent(in)     :: m
integer,   intent(in)     :: ftn
character(len=*), intent(in) :: model_name
integer                   :: dimID(6)
integer,    optional      :: dim1
integer,    optional      :: dim2
integer,    optional      :: dim3
integer,    optional      :: dim4
integer,    optional      :: dim5
character(len=*), optional :: output_format
```

DESCRIPTION:

This routine writes an output file in the NETCDF format based on the list of selected output variables. The arguments are:

n index of the nest

ftn file unit for the output file

model_name Name of the model that generates the output

The routines invoked are:

defineNETCDFheadervar (3.17.20) writes the required headers for a single variable

writeSingleNETCDFvar (3.17.21) writes a single variable into a netcdf formatted file.

LIS_verify (3.21.1) call to check if the return value is valid or not.

3.17.23 LIS_writeHeader_restart (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine LIS_writeHeader_restart(ftn,n,dimID, vid, standard_name, &
     long_name, units, vlevels, valid_min, valid_max, var_flag)
```

USES:

ARGUMENTS:

```
integer                    :: ftn
integer                    :: n
integer                    :: dimID(6)
integer                    :: vid
character(len=*)           :: standard_name
character(len=*)           :: long_name
character(len=*)           :: units
integer                    :: vlevels
real                       :: valid_min
real                       :: valid_max
character(len=*), optional :: var_flag
```

DESCRIPTION:

This routine writes the required NETCDF header for a single variable
The arguments are:

n index of the nest

ftn NETCDF file unit handle

dimID NETCDF dimension ID corresponding to the variable

dataEntry object containing the values and attributes of the variable to be written

The routines invoked are:

LIS_endrun (3.21.12) call to abort program when a fatal error is detected.

LIS_verify (3.21.1) call to check if the return value is valid or not.

3.17.24 LIS_closeHeader_restart (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine LIS_closeHeader_restart(ftn,n,m,dimID, rstInterval)

  implicit none
```

ARGUMENTS:

```
  integer            :: ftn
  integer            :: n
  integer            :: m
  real               :: rstInterval
  integer            :: dimID(4)
```

DESCRIPTION:

This routine closes the required NETCDF header.
The arguments are:

ftn NETCDF file unit handle

n index of the nest

m index of the surface type

dimID NETCDF dimension ID corresponding to the variable

rstInterval the restart interval

3.17.25 writevar_bin_withstats_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_writevar_bin
subroutine writevar_bin_withstats_real(ftn,ftn_stats,n,var,mvar,form)
```

USES:

```
  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: n
  integer, intent(in) :: ftn
  integer, intent(in) :: ftn_stats
  real, intent(in)    :: var(LIS_rc%ntiles(n))
  character (len=*)   :: mvar
  integer, intent(in) :: form
```

DESCRIPTION:

Write a real variable to a binary output file with a number of diagnostic statistics (mean, standardar deviation, min/max during the temporal output interval) written to a text file.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

ftn_stats unit number of the ASCII text statistics file

var variables being written, dimensioned in the tile space

mvar short name of the variable being written (will be used in the stats file)

form format to be used in the stats file (1-decimal format, 2-scientific format)

The routines invoked are:

LIS_gather_tiled_vector_output (3.17.53) call to gather the 1d tiled output variables in tile space

LIS_gather_gridded_output (3.17.12) call to gather the 1d tiled output variable into a 2d gridded array

LIS_gather_gridded_vector_output (3.17.58) call to gather the 1d tiled output variable into a 1d gridded array

write_stats (3.17.52) call to compute the diagnostic statistics

---

3.17.26 writevar_bin_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_writevar_bin
subroutine writevar_bin_real(ftn, n, var)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: ftn
real, intent(in)    :: var(LIS_rc%ntiles(n))
```

DESCRIPTION:

Write a real variable to a binary output file.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

The routines invoked are:

LIS_gather_tiled_vector_output (3.17.53) call to gather the 1d tiled output variables in tile space

LIS_gather_gridded_output (3.17.12) call to gather the 1d tiled output variable into a 2d gridded array

LIS_gather_gridded_vector_output (3.17.58) call to gather the 1d tiled output variable into a 1d gridded array

---

3.17.27 writevar_bin_real_direct (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_writevar_bin
subroutine writevar_bin_real_direct(ftn, n, var, direct)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: ftn
real,    intent(in) :: var(LIS_rc%lnc(n), LIS_rc%lnr(n))
integer, intent(in) :: direct
```

DESCRIPTION:

Write a real variable to a binary output file in a direct access format
The arguments are:

ftn unit number of the binary output file

n index of the domain or nest.

var variables being written, dimensioned in the tile space

direct dummy argument indicating direct access output

3.17.28 writevar_restart_tile_int (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_writevar_restart
subroutine writevar_restart_tile_int(ftn, n, var)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in) :: ftn
integer, intent(in) :: n
integer, intent(in) :: var(LIS_rc%ntiles(n))
```

DESCRIPTION:

Writes an integer variable to a binary restart file.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

3.17.29 writevar_restart_tile_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_writevar_restart
subroutine writevar_restart_tile_real(ftn, n, var, varid, dim, wformat)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in) :: ftn
integer, intent(in) :: n
real                :: var(LIS_rc%ntiles(n))
integer,          optional :: varid
integer,          optional :: dim
character(len=*), optional :: wformat
```

DESCRIPTION:

Writes a real variable to a binary restart file.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

3.17.30 writevar_restart_patch_int (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_writevar_restart
subroutine writevar_restart_patch_int(ftn, n, m, var, varid, dim, wformat)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in) :: ftn
integer, intent(in) :: n
integer, intent(in) :: m
integer             :: var(LIS_rc%npatch(n,m))
integer,          optional :: varid
integer,          optional :: dim
character(len=*), optional :: wformat
```

DESCRIPTION:

Writes a real variable to a binary restart file.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

3.17.31 writevar_restart_patch_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_writevar_restart
subroutine writevar_restart_patch_real(ftn, n, m, var, varid, dim, wformat)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in) :: ftn
integer, intent(in) :: n
integer, intent(in) :: m
real                :: var(LIS_rc%npatch(n,m))
integer,          optional :: varid
integer,          optional :: dim
character(len=*), optional :: wformat
```

DESCRIPTION:

Writes a real variable to a binary restart file.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

3.17.32 writevar_tile_noensemble_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_writevar_reduced_tilespace
subroutine writevar_tile_noensemble_real(ftn, n, var)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in) :: ftn
integer, intent(in) :: n
real, intent(in)    :: var(LIS_rc%ntiles(n)/LIS_rc%nensem(n))
```

DESCRIPTION:

Writes a real variable to a binary restart file.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

3.17.33 readvar_tile_noensemble_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call LIS_readvar_reduced_tilespace
subroutine readvar_tile_noensemble_real(ftn, n, var)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: ftn
integer, intent(in)   :: n
real, intent(inout)   :: var(LIS_rc%ntiles(n)/LIS_rc%nensem(n))
```

DESCRIPTION:

Reads a real variable from a binary restart file.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

3.17.34 readvar_restart_tile_int (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_readvar_restart
subroutine readvar_restart_tile_int(ftn, n, var)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: ftn
integer, intent(in)    :: n
integer, intent(inout) :: var(LIS_rc%ntiles(n))
```

DESCRIPTION:

Reads an integer variable from a binary restart file.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

3.17.35 readvar_restart_patch_int (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call LIS_readvar_restart
subroutine readvar_restart_patch_int(ftn, n, m,var, varname,dim,vlevels, wformat)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: ftn
integer, intent(in)    :: n
integer, intent(in)    :: m
integer, intent(inout) :: var(LIS_rc%npatch(n,m))
character(len=*), optional :: varname
integer,          optional :: dim
integer,          optional :: vlevels
character(len=*), optional :: wformat
```

DESCRIPTION:

Reads a real variable from a binary restart file.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

3.17.36 readvar_restart_patch_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call LIS_readvar_restart
subroutine readvar_restart_patch_real(ftn, n, m,var, varname,dim,vlevels, wformat)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: ftn
integer, intent(in)   :: n
integer, intent(in)   :: m
real, intent(inout)   :: var(LIS_rc%npatch(n,m))
character(len=*), optional :: varname
integer,          optional :: dim
integer,          optional :: vlevels
character(len=*), optional :: wformat
```

DESCRIPTION:

Reads a real variable from a binary restart file.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

3.17.37 readvar_restart_tile_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call LIS_readvar_restart
subroutine readvar_restart_tile_real(ftn, n, var, varname,dim,vlevels, wformat)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: ftn
integer, intent(in)   :: n
real, intent(inout)   :: var(LIS_rc%ntiles(n))
character(len=*), optional :: varname
integer,          optional :: dim
integer,          optional :: vlevels
character(len=*), optional :: wformat
```

DESCRIPTION:

Reads a real variable from a binary restart file.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

3.17.38 readvar_1dgridded_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call LIS_readvar_gridded
subroutine readvar_1dgridded_real(ftn, n, var)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: ftn
integer, intent(in)   :: n
real, intent(inout)   :: var(LIS_rc%ngrid(n))
```

DESCRIPTION:

Reads a real variable from a binary sequential access, gridded file. After reading the global data, the routine subroutine subsets the data for each processor's domain.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

3.17.39 readvar_1dgridded_fromvector_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call LIS_readvar_gridded
subroutine readvar_1dgridded_fromvector_real(ftn, n, var, oned)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: ftn
integer, intent(in)   :: n
real, intent(inout)   :: var(LIS_rc%ngrid(n))
integer, intent(in)   :: oned
```

DESCRIPTION:

Reads a real variable from a binary sequential access, 1d gridded file. After reading the global data, the routine subroutine subsets the data for each processor's domain.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

oned dummy variable to distinguish the interface.

3.17.40 readvar_2dgridded_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call LIS_readvar_gridded
subroutine readvar_2dgridded_real(ftn, n, var)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: ftn
integer, intent(in)   :: n
real, intent(inout)   :: var(LIS_rc%lnc(n), LIS_rc%lnr(n))
```

DESCRIPTION:

Reads a real variable from a binary sequential access, gridded file. After reading the global data, the routine subroutine subsets the data for each processor's domain.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

3.17.41 writevar_gridded_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call LIS_writevar_gridded
subroutine writevar_gridded_real(ftn, n, var, wopt)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: ftn
integer, intent(in)   :: n
real, intent(inout)   :: var(LIS_rc%ngrid(n))
character(len=*), intent(in), optional :: wopt
```

DESCRIPTION:

Writes a real variable to a binary sequential access, gridded file as either 2-dimensional/1-dimensional gridded field. (The 1-d gridded field consists of a vector of valid land points) After reading the global data, the routine subroutine subsets the data for each processor's domain.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

3.17.42 writevar_gridded_real_withstats (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call LIS_writevar_gridded
subroutine writevar_gridded_real_withstats(ftn,ftn_stats,n,var,mvar,form)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: ftn
integer, intent(in)   :: ftn_stats
integer, intent(in)   :: n
real, intent(inout)   :: var(LIS_rc%ngrid(n))
character(len=*)      :: mvar
integer, intent(in)   :: form
```

DESCRIPTION:

Writes a real variable to a binary sequential access, gridded file as either 2-dimensional/1-dimensional gridded field. (The 1-d gridded field consists of a vector of valid land points) After reading the global data, the routine subroutine subsets the data for each processor's domain.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

3.17.43 writevar_netcdf_withstats_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_writevar_netcdf
subroutine writevar_netcdf_withstats_real(ftn,ftn_stats, n, var,varid, mvar, form, dim1)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: ftn
integer, intent(in) :: ftn_stats
integer             :: varid
real, intent(in)    :: var(LIS_rc%ntiles(n))
character (len=*)   :: mvar
integer, intent(in) :: form
integer, intent(in), optional :: dim1
```

DESCRIPTION:

Write a real variable to a netcdf output file with some diagnostic statistics written to a text file.
The arguments are:

n index of the domain or nest.

ftn unit number of the netcdf output file

ftn_stats unit number of the ASCII text statistics file

var variables being written, dimensioned in the tile space

mvar name of the variable being written (will be used in the stats file)

form format to be used in the stats file (1-decimal format, 2-scientific format)

flag option to determine if the variable needs to be written (1-write, 0-do not write)

The routines invoked are:

stats (3.17.50)
call to compute the diagnostic statistics

3.17.44 writevar_grib1_withstats_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
   Private name: call using LIS_writevar_grib1
 subroutine writevar_grib1_withstats_real(ftn, ftn_stats, n,   &
                                          var, mvar, &
                                          gribid, gribSF, gribSfc, &
                                          gribLvl, sType, &
                                          time_unit, time_p1, time_p2, &
                                          timeRange, &
                                          form,     &
                                          toplev, botlev, kdim)
```

USES:

```
   use map_utils
   use LIS_pluginIndices

   implicit none
```

ARGUMENTS:

```
   integer, intent(in)         :: ftn
   integer, intent(in)         :: ftn_stats
   integer, intent(in)         :: n
   real, intent(in)            :: var(LIS_rc%ntiles(n))
   character(len=*),intent(in) :: mvar
   integer, intent(in)         :: gribid
   integer, intent(in)         :: gribSF
   integer, intent(in)         :: gribSfc
   integer, intent(in)         :: gribLvl
   character(len=*), intent(in)  :: sType
   integer, intent(in)         :: timeRange
   integer, intent(in)         :: form
   integer, intent(in)         :: time_unit
   integer, intent(in)         :: time_p1
   integer, intent(in)         :: time_p2
   integer                     :: kdim
   real,    intent(in)         :: toplev(kdim), botlev(kdim)
```

DESCRIPTION:

Write a real variable to a grib output file with some diagnostic statistics written to a text file.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

ftn_stats unit number of the ASCII text statistics file

var variables being written, dimensioned in the tile space

mvar name of the variable being written (will be used in the stats file)

flag option to determine if the variable needs to be written (1-write, 0-do not write)

form format to be used in the stats file (1-decimal format, 2-scientific format)

info Array containing the sizes of the various grib sections

sect1 Array for storing the contents of Section 1.

s1opt Array for storing optional Section 1 information (not used)

sect2 Array for storing the contents of Section 2.

sect3 Array for storing the contents of Section 3.

toplev Array of the heights of the top of each layer

botlev Array of the heights of the bottom of each layer

kdim Grid dimension k-direction (number of levels/layers)

size_3 Size of the sect3 array

The routines invoked are:

LIS_gather_gridded_output (3.17.12) call to gather the 1d tiled output variable into a 2d gridded array

write_stats (3.17.52) call to compute diagnostic statistics of a variable

---

3.17.45 tile2grid_local (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine tile2grid_local(n,gvar,tvar)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
real              :: gvar(LIS_rc%lnc(n),LIS_rc%lnr(n))
real, intent(in)  :: tvar(LIS_rc%ntiles(n))
```

DESCRIPTION:

This routine converts a tile space variable to the corresponding grid space. The aggregation involves weighted average of each tile in a grid cell based on the vegetation distribution.
The arguments are:

n index of the domain or nest.

tvar variable dimensioned in the tile space.

gvar variable after converstion to the grid space

3.17.46 LIS_patch2tile (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine LIS_patch2tile(n,m,tvar,pvar)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: m
real                :: pvar(LIS_rc%npatch(n,m))
real                :: tvar(LIS_rc%ntiles(n))
```

DESCRIPTION:

This routine converts a tile space variable to the corresponding grid space. The aggregation involves weighted average of each tile in a grid cell based on the vegetation distribution.
The arguments are:

n index of the domain or nest.

tvar variable dimensioned in the tile space.

gvar variable after converstion to the grid space

3.17.47 LIS_write_innov (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine LIS_write_innov(ftn, n, k, varid, var)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in) :: ftn
integer, intent(in) :: n
integer, intent(in) :: k
integer, intent(in) :: varid
real, intent(in)    :: var(LIS_rc%ngrid(n)*LIS_rc%nobtypes(k))
```

DESCRIPTION:

Writes the innovations to a binary file in a gridded format.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the observation space

---

3.17.48 LIS_write_dastats (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine LIS_write_dastats(ftn,n,var)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: ftn
real, intent(in)    :: var(LIS_rc%ntiles(n))
```

DESCRIPTION:

Write a real variable to a ASCII text output file with DA-related diagnostic statistics written to a text file.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

ftn_stats unit number of the ASCII text statistics file

var variables being written, dimensioned in the tile space

mvar name of the variable being written (will be used in the stats file)

form format to be used in the stats file (1-decimal format, 2-scientific format)

range range of variables to be written

The routines invoked are:

stats_da (3.17.51)
call to compute the diagnostic statistics

3.17.49 makepdsn (Source File: LIS_historyMod.F90)

REVISION HISTORY:

```
19 Mar 2002  Cosgrove; Initial specification
```

INTERFACE:

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

  implicit none
```

ARGUMENTS:

```
  character*8 :: yesterday, beforeyester
  integer     :: kpds(25), hour
  real        :: writeint
```

DESCRIPTION:

This routine computes the coefficients and terms required for grib output
The arguments are:

yesterday character representation of previous time

beforeyester character representation of uber previous time

kpds grib description array

hour current hour

writeint output writing interval in seconds

3.17.50 stats (Source File: LIS_historyMod.F90)

Calculates some diagnostic statistics for a given variable for model output. The routine provides statistics such as mean, standard deviation, minimum and maximum values of a given variable.

REVISION HISTORY:

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

INTERFACE:

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

ARGUMENTS:

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

3.17.51 stats_da (Source File: LIS_historyMod.F90)

Calculates some diagnostic statistics for a given variable for data assimilation output. The routine provides statistics such as mean, standard deviation, minimum and maximum values and spread of a given variable.

REVISION HISTORY:

```
aug 22 2006:  Sujay Kumar; Initial code
```

INTERFACE:

```
subroutine stats_da(var,udef,ntiles,mean,spread, stdev,min,max)
```

ARGUMENTS:

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

3.17.52 write_stats (Source File: LIS_historyMod.F90)

REVISION HISTORY:

```
29 Oct 2008:  James Geiger; Initial code
```

INTERFACE:

```
subroutine write_stats(var, size, mvar, ftn_stats, form)
```

USES:

ARGUMENTS:

```
implicit none

real, dimension(*), intent(in) :: var
integer, intent(in) :: size
character (len=*)   :: mvar
integer, intent(in) :: ftn_stats
integer, intent(in) :: form
```

DESCRIPTION:

Top level call to write statistics to the STATS file. The arguments are:

var output data to process

size size of array var

mvar name of the variable being written

ftn_stats unit number of the ASCII text statistics file

form format to be used in the stats file (1-decimal format, 2-scientific format)

3.17.53 LIS_gather_tiled_vector_output (Source File: LIS_historyMod.F90)

REVISION HISTORY:

```
30 Jan 2009:  Sujay Kumar; Initial code
```

INTERFACE:

```
subroutine LIS_gather_tiled_vector_output(n, gtmp, var)
```

USES:

ARGUMENTS:

```
implicit none

integer                        :: n
real, allocatable                 :: gtmp(:)
real, intent(in)              :: var(LIS_rc%ntiles(n))
```

DESCRIPTION:

This routine gathers the output data into a tiled 1d array.
This process aggregates the variable into a tile space.
This process accounts for the halo.
The arguments are:

n index of the current nest

gtmp return array for the tiled output data

var output data to process

3.17.54 LIS_gather_tiled_vector_withhalo_output (Source File: LIS_historyMod.F90)

REVISION HISTORY:

```
30 Jan 2009:  Sujay Kumar; Initial code
```

INTERFACE:

```
subroutine LIS_gather_tiled_vector_withhalo_output(n, gtmp, var)
```

USES:

ARGUMENTS:

```
implicit none

integer                        :: n
real, allocatable                 :: gtmp(:)
real, intent(in)              :: var(LIS_rc%ntiles(n))
```

DESCRIPTION:

This routine gathers the output data into a tiled 1d array.
This process aggregates the variable into a tile space.
This process accounts for the halo.
The arguments are:

n index of the current nest

gtmp return array for the tiled output data

var output data to process

3.17.55 LIS_gather_patch_vector_output (Source File: LIS_historyMod.F90)

REVISION HISTORY:

```
30 Jan 2009:  Sujay Kumar; Initial code
```

INTERFACE:

```
subroutine LIS_gather_patch_vector_output(n, m, gtmp, var)
```

USES:

ARGUMENTS:

```
implicit none

integer                        :: n
integer                        :: m
real, allocatable                 :: gtmp(:)
real, intent(in)              :: var(LIS_rc%npatch(n,m))
```

DESCRIPTION:

This routine gathers the output data into a patch 1d array.
This process aggregates the variable into a tile space.
This process accounts for the halo.
The arguments are:

n index of the current nest

gtmp return array for the patch output data

var output data to process

3.17.56 gather_gridded_output_tile (Source File: LIS_historyMod.F90)

REVISION HISTORY:

```
29 Oct 2008:  James Geiger; Initial code
```

INTERFACE:

```
subroutine gather_gridded_output_tile(n, gtmp, var)
```

USES:

ARGUMENTS:

```
implicit none

integer, intent(in)          :: n
real, allocatable                :: gtmp(:,:)
real, intent(in)             :: var(LIS_rc%ntiles(n))
```

DESCRIPTION:

This routine gathers the output data into a gridded array.
This process aggregates tiles to their grid, accounting for ensemble runs.
This process accounts for the halo.
The arguments are:

n index of the current nest

gtmp return array for the gridded output data

var output data to process

3.17.57 gather_gridded_output_patch (Source File: LIS_historyMod.F90)

REVISION HISTORY:

```
29 Oct 2008:  James Geiger; Initial code
```

INTERFACE:

```
subroutine gather_gridded_output_patch(n, mtype, gtmp, var)
```

USES:

ARGUMENTS:

```
implicit none

integer, intent(in)          :: n
integer, intent(in)          :: mtype
real, allocatable                :: gtmp(:,:)
real, intent(in)             :: var(LIS_rc%npatch(n,mtype))
```

DESCRIPTION:

This routine gathers the output data into a gridded array.
This process aggregates tiles to their grid, accounting for ensemble runs.
This process accounts for the halo.
The arguments are:

n index of the current nest

mtype index of the surface model type

gtmp return array for the gridded output data

var output data to process

3.17.58 LIS_gather_gridded_vector_output (Source File: LIS_historyMod.F90)

REVISION HISTORY:

```
30 Jan 2009: Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine LIS_gather_gridded_vector_output(n, gtmp, var)
```

USES:

ARGUMENTS:

```
implicit none

integer, intent(in)          :: n
real, allocatable                :: gtmp(:)
real, intent(in)             :: var(LIS_rc%ntiles(n))
```

DESCRIPTION:

This routine gathers the output data into a gridded 1d array.
This process aggregates tiles to their grid, accounting for ensemble runs.
This process accounts for the halo.
The arguments are:

n index of the current nest

gtmp return array for the gridded output data

var output data to process

3.17.59 LIS_convertVarToLocalSpace (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine LIS_convertVarToLocalSpace(n,gvar,lvar)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: n
real                  :: gvar(LIS_rc%glbngrid_red(n))
real                  :: lvar(LIS_rc%ngrid(n))
```

DESCRIPTION:

Reads a real variable from a binary sequential access, 1d gridded file. After reading the global data, the routine subroutine subsets the data for each processor's domain.
The arguments are:

n index of the domain or nest.

ftn unit number of the binary output file

var variables being written, dimensioned in the tile space

oned dummy variable to distinguish the interface.

3.17.60 LIS_initialize_registries (Source File: LIS_initialize_registries.F90)

REVISION HISTORY:

```
19 Aug 2010: Sujay Kumar; Initial specification
17 Jan 2011: David Mocko, added max/min greenness & slope type
```

INTERFACE:

```
subroutine LIS_initialize_registries()
```

USES:

```
use ESMF
use LIS_param_pluginMod, only : LIS_laisai_plugin,&
     LIS_alb_plugin, LIS_gfrac_plugin , LIS_roughness_plugin,&
     LIS_emissivity_plugin
use LIS_runmode_pluginMod, only : LIS_runmode_plugin
use LIS_dataassim_pluginMod,      only : LIS_dataassim_plugin
use LIS_biasEstimation_pluginMod, only : LIS_biasEstimation_plugin
use LIS_optUEAlgorithm_pluginMod, only : LIS_optUEAlgorithm_plugin
 use LIS_optUEType_pluginMod,      only : LIS_optUEType_plugin
use LIS_lsmoptue_pluginMod,     only : LIS_lsmoptue_plugin
use LIS_PEobs_pluginMod,      only : LIS_PEobs_plugin
use LIS_ObjFunc_pluginMod,    only : LIS_ObjFunc_plugin
use LIS_RTM_pluginMod,        only : LIS_RTM_plugin
use LIS_rtmoptue_pluginMod,        only : LIS_rtmoptue_plugin
```

```
use LIS_lsmrtm_pluginMod,     only : LIS_lsmrtm_plugin
use LIS_landslidemodel_pluginMod, only : LIS_landslidemodel_plugin
use LIS_routing_pluginMod, only : LIS_routing_plugin
use LIS_lsmrouting_pluginMod, only : LIS_lsmrouting_plugin
use LIS_irrigationmodel_pluginMod, only : LIS_irrigationmodel_plugin
use LIS_lsmirrigation_pluginMod, only : LIS_lsmirrigation_plugin
```

DESCRIPTION:

The code in this file initializes registries that set up the component plugin definitions.
The routines invoked are:

LIS_runmode_plugin (4.26.1)
sets up function table registries for implemented runmodes

LIS_laisai_plugin (4.22.1)
sets up function table registries for implemented LAI/SAI data sources

LIS_alb_plugin (4.22.5)
sets up function table registries for implemented albedo data sources

LIS_gfrac_plugin (4.22.2)
sets up function table registries for implemented greenness fraction data sources

LIS_optUEAlgorithm_plugin (4.21.1)
sets up function table registries for implemented parameter estimation algorithms

LIS_lsmoptue_plugin (4.16.1)
sets up function table registries for implemented lsm plugins related to parameter estimation

LIS_PEobs_plugin (4.3.1)
sets up function table registries for implemented plugins for objective space handling related to parameter estimation

---

3.18 Fortran: Module Interface LIS_irrigationMod (Source File: LIS_irrigationMod.F90)

REVISION HISTORY:

```
11 Nov 2012: Sujay Kumar; Initial implementation
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_logMod

implicit none

PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
public :: LIS_irrigation_init
public :: LIS_irrigation_run
public :: LIS_irrigation_output
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
 ------------------------------------------------------------------------------
  public :: LIS_irrig_state      !data structure containing irrigation variables
```

3.18.1 LIS_irrigation_init (Source File: LIS_irrigationMod.F90)

Allocates memory for data structures used for reading irrigation datasets. The irrigationdepth field is updated by the external files. The irrigation water equivalent fields are expected to be set by the model.

INTERFACE:

```
  subroutine LIS_irrigation_init
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
   use netcdf
#endif
   use LIS_timeMgrMod,   only : LIS_registerAlarm, LIS_parseTimeString
```

3.18.2 LIS_irrigation_run (Source File: LIS_irrigationMod.F90)

INTERFACE:

```
  subroutine LIS_irrigation_run(n)
```

USES:

```
    implicit none
```

ARGUMENTS:

```
    integer  :: n
```

DESCRIPTION:

This routine runs the specified irrigation model.
The arguments are:

n index of the nest

3.18.3 LIS_irrigation_output (Source File: LIS_irrigationMod.F90)

INTERFACE:

```
  subroutine LIS_irrigation_output(n)
```

USES:

```
use LIS_timeMgrMod, only : LIS_isAlarmRinging
use LIS_historyMod,  only : LIS_writeModelOutput
use LIS_fileIOMod,  only : LIS_create_output_directory, &
     LIS_create_output_filename,  &
     LIS_create_stats_filename
```

ARGUMENTS:

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

DESCRIPTION:

This routine writes the irrigation model output.
The arguments are:

n index of the nest

3.19 Fortran: Module Interface LIS_lakemodelMod (Source File: LIS_lakemodelMod.F90)

The code in this file provides interfaces to manage the operation of different lake models

REVISION HISTORY:

```
 16 Jul 2012    Sujay Kumar  Initial Specification

use ESMF

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
 public :: LIS_lakemodel_init       ! initialize lakemodel variables, memory
 public :: LIS_setuplakemodel       ! set land surface parameters
 public :: LIS_lakemodel_readrestart    ! read the restart file
 public :: LIS_lakemodel_f2t     ! transfer forcing to model tiles
 public :: LIS_lakemodel_run       ! execute the land model
 public :: LIS_lakemodel_output     ! write model output
 public :: LIS_lakemodel_setdynparams   ! set the time dependent parameters
 public :: LIS_lakemodel_writerestart   ! write the restart file
 public :: LIS_lakemodel_finalize   ! cleanup allocated structures
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
```

3.19.1 LIS_lakemodel_init (Source File: LIS_lakemodelMod.F90)

INTERFACE:

```
subroutine LIS_lakemodel_init()
```

USES:

```
use ESMF
use LIS_surfaceModelDataMod
use LIS_coreMod, only : LIS_rc, LIS_config, LIS_vecTile
use LIS_lakemodel_pluginMod,    only : LIS_lakemodel_plugin
use LIS_logMod,       only : LIS_logunit, LIS_verify, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber
```

DESCRIPTION:

Interface for initializing the land model. The intialization includes the allocation of memory for LAKEMODEL specific variables and datastructures and specification of any runtime specific options.
The calling sequence is:

LIS_lakemodel_plugin (4.10.1)
sets up function table registries for implemented land surface models

lakemodelinit (3.48.2)
invokes the generic method in the registry to initialize the land surface model

---

3.19.2 LIS_setuplakemodel (Source File: LIS_lakemodelMod.F90)

INTERFACE:

```
subroutine  LIS_setuplakemodel
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

DESCRIPTION:

The setup interfaces are used for the LAKEMODEL specification of LAKEMODEL bparameters. If a parameter falls outside the LIS-specified generic parameter list, the LAKEMODEL is expected to provide routines for the handling of any input data, specific to those parameters.
The calling sequence is:

lakemodelsetup (3.48.8)
invokes the generic method in the registry to set up the land surface model

---

3.19.3 LIS_lakemodel_run (Source File: LIS_lakemodelMod.F90)

INTERFACE:

```
subroutine LIS_lakemodel_run(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

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

DESCRIPTION:

This interface provides the entry point to the LAKEMODEL routines that invokes the land surface model physics.
The arguments are:

n index of the nest or domain

The calling sequence is:

lakemodelrun (3.48.4)
invokes the generic method in the registry to run the land surface model

---

3.19.4 LIS_lakemodel_readrestart (Source File: LIS_lakemodelMod.F90)

INTERFACE:

```
subroutine LIS_lakemodel_readrestart
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

DESCRIPTION:

This interface provides the entry point to read LAKEMODEL specific model restart files.
The calling sequence is:

lakemodelrestart (3.48.10)
invokes the generic method in the registry to read restart files for the land surface model

---

3.19.5 LIS_lakemodel_output (Source File: LIS_lakemodelMod.F90)

INTERFACE:

```
subroutine LIS_lakemodel_output(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

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

DESCRIPTION:

This interface provides the entry point to write LAKEMODEL specific model output files.
The arguments are:

n index of the nest or domain

The calling sequence is:

lakemodeloutput (3.48.14)
invokes the generic method in the registry to write the land surface model output

---

3.19.6 LIS_lakemodel_setdynparams (Source File: LIS_lakemodelMod.F90)

INTERFACE:

```
subroutine LIS_lakemodel_setdynparams(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

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

DESCRIPTION:

This interface provides an entry point to update any time dependent land surface model parameters in an LAKEMODEL.
The arguments are:

n index of the nest or domain

The calling sequence is:

lakemodeldynsetup (3.48.12)
invokes the generic method in the registry to set up time dependent parameters for the land surface model

---

3.19.7 LIS_lakemodel_f2t (Source File: LIS_lakemodelMod.F90)

INTERFACE:

```
subroutine LIS_lakemodel_f2t(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

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

DESCRIPTION:

This interface is used to transfer the forcing variables to the actual model tile space. Any forcing perturbations that need to be applied to the input forcing is applied at this stage as well.
The arguments are:

n index of the nest or domain

The calling sequence is:

lakemodelf2t (3.48.16)
invokes the generic method in the registry to transfer the forcing to the land surface model tiles

---

3.19.8 LIS_lakemodel_writerestart (Source File: LIS_lakemodelMod.F90)

INTERFACE:

```
subroutine LIS_lakemodel_writerestart(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

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

DESCRIPTION:

This interface provides the entry point to read the LAKEMODEL specific model restart files.
The arguments are:

n index of the nest or domain

The calling sequence is:

lakemodelwrst (3.48.18)
invokes the generic method in the registry to write restart files for the land surface model

---

3.19.9 LIS_lakemodel_finalize (Source File: LIS_lakemodelMod.F90)

INTERFACE:

```
subroutine LIS_lakemodel_finalize()
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

DESCRIPTION:

This routine issues the invocation to deallocate and cleanup any allocated data structures in the specific instance of a land surface model
The calling sequence is:

lakemodelfinalize (3.48.6)
invokes the generic method in the registry to cleanup the LAKEMODEL related datastructures

---

3.20 Fortran: Module Interface LIS_landslideMod (Source File: LIS_landslideMod.F90)

REVISION HISTORY:

```
use LIS_landslidemodel_pluginMod ! SY

implicit none

PRIVATE

-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
 ------------------------------------------------------------------------------
  PUBLIC :: LIS_initLandSlideModel
  PUBLIC :: LIS_runLandSlideModel
  PUBLIC :: LIS_outputLandSlideModel
  PUBLIC :: LIS_finalizeLandSlideModel ! SY
  PUBLIC :: LIS_landslide_param_reset
 ------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
 ------------------------------------------------------------------------------
```

3.20.1 LIS_initLandSlideModel (Source File: LIS_landslideMod.F90)

INTERFACE:

```
subroutine LIS_initLandSlideModel()
```

USES:

```
  use ESMF
  use LIS_coreMod,    only : LIS_rc, LIS_config
  use LIS_logMod,     only : LIS_verify

  implicit none
```

DESCRIPTION:

3.20.2 LIS_runLandSlideModel (Source File: LIS_landslideMod.F90)

INTERFACE:

```
subroutine LIS_runLandSlideModel(n)
```

USES:

```
  use LIS_coreMod,   only : LIS_rc
```

ARGUMENTS:

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

DESCRIPTION:

3.20.3 LIS_outputLandSlideModel (Source File: LIS_landslideMod.F90)

INTERFACE:

```
subroutine LIS_outputLandSlideModel(n)
```

USES:

```
use LIS_coreMod,   only : LIS_rc
```

ARGUMENTS:

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

DESCRIPTION:

3.20.4 LIS_finalizeLandSlideModel (Source File: LIS_landslideMod.F90)

INTERFACE:

```
subroutine LIS_finalizeLandSlideModel()
```

USES:

```
use LIS_coreMod,   only : LIS_rc
```

DESCRIPTION:

3.21 Fortran: Module Interface LIS_logMod (Source File: LIS_logMod.F90)

The code in this file provides routines for diagnostic and error logging in LIS. A file unit number is reserved for the diagnostic log file. When a multiprocessor run is conducted, a diagnostic file from each processor is written to a separate file. An entry to the diagnostic log can be made by using:

```
write(LIS_logunit,*) msg
```

REVISION HISTORY:

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

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
 public :: LIS_log_init ! initializes LIS logging.
 public :: LIS_log_msg ! writes a time-processor stamped message
                       ! to the diagnostic log
 public :: LIS_abort  ! generates a standard abort message
 public :: LIS_alert  ! generates a standard alert message
 public :: LIS_flush ! flushes any unwrittend writes to the log
```

```
public :: LIS_getNextUnitNumber   !get the next available unit number
public :: LIS_releaseUnitNumber   !release the unit number!
public :: LIS_endrun  ! Ends the program
public :: LIS_verify  ! checks a return a code for success
public :: LIS_warning ! checks a return code and issue a warning
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
public :: LIS_logunit ! file unit number used for diagnostic logging
```

3.21.1 LIS_verify (Source File: LIS_logMod.F90)

INTERFACE:

```
interface LIS_verify
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure check_error
module procedure verify
```

DESCRIPTION:

checks the return code for success. The two private methods can be used to enable user-specified log messages.

3.21.2 LIS_log_init (Source File: LIS_logMod.F90)

INTERFACE:

```
subroutine LIS_log_init(iunit)
  integer, intent(IN) :: iunit
```

DESCRIPTION:

Initializes the lis log management. Reserves the assigned file unit number for lis diagnostic logging.

3.21.3 LIS_log_msg (Source File: LIS_logMod.F90)

INTERFACE:

```
subroutine LIS_log_msg(msg)
  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

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.
The arguments are:

msg message to be written to the logfile

3.21.4 LIS_abort (Source File: LIS_logMod.F90)

REVISION HISTORY:

```
03 mar 1999 initial version .........................mr moore/dnxm
09 aug 1999 ported to IBM SP-2. increase size of abort message
            to 100 characters. removed stop 255 with a call
            to IBM utility "abort"...................mr gayno/dnxm
29 oct 2005  Sujay Kumar, Adopted in LIS
```

INTERFACE:

```
subroutine LIS_abort( abort_message )

  implicit none

  character*100              :: abort_message(20)
```

DESCRIPTION:

to generate standard abort message and abort calling program.

- open abort file, write abort message and then call the system abort routine. this will cause program termination and cause script calling this routine to abort.
- if there is an error opening or writing abort file, send message to unit 6 and call the system abort routine

The arguments are:

abort_message abort message

3.21.5 LIS_alert (Source File: LIS_logMod.F90)

REVISION HISTORY:

```
03 mar 1999 initial version .........................mr moore/dnxm
09 aug 1999 ported to IBM SP-2.  increase size of message to
            100 characters.  added intent attributes to
            arguments...............................mr gayno/dnxm
29 oct 2005  Sujay Kumar, Adopted in LIS
```

INTERFACE:

```
subroutine LIS_alert( program_name, alert_number, message )
```

3.21.6 LIS_flush (Source File: LIS_logMod.F90)

REVISION HISTORY:

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

```
subroutine LIS_flush(unit)

  implicit none

  integer :: unit
```

DESCRIPTION:

This routine is a generic interface to the system flush routine.
The arguments are:

unit unit to be flushed out

3.21.7 check_error (Source File: LIS_logMod.F90)

INTERFACE:

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

ARGUMENTS:

```
  implicit none
  integer, intent(in)          :: ierr
  character(len=*), intent(in) :: msg
```

DESCRIPTION:

This is an error check routine. Program exits in case of error with an associated error message written to the 'abort message' file.

3.21.8 verify (Source File: LIS_logMod.F90)

INTERFACE:

```
subroutine verify(ierr)
```

ARGUMENTS:

```
  implicit none
  integer, intent(in)          :: ierr
```

DESCRIPTION:

This is an error check routine. Program exits in case of error with an associated error message written to the 'abort message' file.

3.21.9 LIS_warning (Source File: LIS_logMod.F90)

INTERFACE:

```
subroutine LIS_warning(ierr,msg)
```

ARGUMENTS:

```
implicit none
integer, intent(in)          :: ierr
character(len=*), intent(in) :: msg
```

DESCRIPTION:

This is an error check routine. Program issues a warning in case of error with an associated error message.

3.21.10 LIS_getNextUnitNumber (Source File: LIS_logMod.F90)

INTERFACE:

```
integer function LIS_getNextUnitNumber()
```

DESCRIPTION:

get next available Fortran unit number
Method: Get next available Fortran unit number itst. This routine is modified from the CCSM codes

---

3.21.11 LIS_releaseUnitNumber (Source File: LIS_logMod.F90)

INTERFACE:

```
subroutine LIS_releaseUnitNumber(iunit)
```

DESCRIPTION:

close and release Fortran unit no longer in use

3.21.12 LIS_endrun (Source File: LIS_logMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine LIS_endrun
```

USES:

```
use LIS_mpiMod

implicit none

integer :: ierr
```

DESCRIPTION:

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

LIS_flush (3.21.6)
flushes the output streams

3.22 Fortran: Module Interface LIS_lsmMod (Source File: LIS_lsmMod.F90)

The code in this file provides interfaces to manage the operation of different land surface models

3.22.1 Overview

This module defines the interface plugins for the incorporation of different land surface models. These interfaces provide entry points for introducing a new land surface scheme in LIS. The following implementations for each LSM are expected to specify methods to initialize and set the LSM specific variables and parameters, provide methods for model simulation, model output, and restart operations. A number of other optional interfaces need to specified depending on the mode of operation of the LSM. For example, if the LSM is used for a coupled simulation with an atmospheric component, the `LIS_lsm_setexport` interface need to be implemented. Similar implementations need to be specified for the use of the LSM in data assimilation and optimization applications.

REVISION HISTORY:

```
 14 Nov 2002    Sujay Kumar  Initial Specification

use ESMF
use LIS_coreMod
use LIS_lsm_pluginMod
use LIS_lsmcpl_pluginMod
use LIS_lsmda_pluginMod
use LIS_perturbMod
use LIS_logMod

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: LIS_lsm_init       ! initialize lsm variables, memory
public :: LIS_setuplsm       ! set land surface parameters
public :: LIS_lsm_readrestart    ! read the restart file
public :: LIS_lsm_f2t     ! transfer forcing to model tiles
```

```
  public :: LIS_lsm_run       ! execute the land model
  public :: LIS_setLSMDynParams    ! set the time dependent parameters
  public :: LIS_lsm_writerestart   ! write the restart file
  public :: LIS_lsm_setexport  ! set the variables to be exported to the
                               ! atmospheric component
  public :: LIS_lsm_perturb_states ! perturbs the prognostic variables
  public :: LIS_lsm_finalize   ! cleanup allocated structures
  public :: LIS_lsm_reset      ! reset structures
  public :: LIS_lsm_diagnoseVarsForDA  ! DA related updates to variables LSM
 ------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
 ------------------------------------------------------------------------------
  public :: LIS_LSM_State
  public :: LIS_LSM_Pert_State
  public :: LIS_LSM_Incr_State
```

3.22.2 LIS_lsm_setexport (Source File: LIS_lsmMod.F90)

INTERFACE:

```
interface LIS_lsm_setexport
```

PRIVATE MEMBER FUNCTIONS:

```
    module procedure lsm_setexport_noesmf
```

DESCRIPTION:

This interface provides the entry point for specifying an export state (a list of model specific variables) from a land surface model. The routine is used in a coupled simulation to provide feedback to a different model component such as an atmospheric model.

3.22.3 LIS_lsm_init (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_lsm_init()
```

USES:

```
   use LIS_surfaceModelDataMod
```

DESCRIPTION:

Interface for initializing the land model. The intialization includes the allocation of memory for LSM specific variables and datastructures and specification of any runtime specific options.
The calling sequence is:

LIS_lsm_plugin (4.12.1)
sets up function table registries for implemented land surface models

LIS_lsmcpl_plugin (4.13.1)
sets up function table registries for implemented land surface models used in coupled modes

LIS_lsmda_plugin (4.14.1)
sets up function table registries for land surface models used in data assimilation

lsminit (3.50.2)
invokes the generic method in the registry to initialize the land surface model

perturbinit (3.54.2)
invokes the generic method in the registry to perturb the prognostic variables using the specified algorithm.

3.22.4 LIS_setuplsm (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine  LIS_setuplsm
```

USES:

DESCRIPTION:

The setup interfaces are used for the LSM specification of LSM bparameters. If a parameter falls outside the LIS-specified generic parameter list, the LSM is expected to provide routines for the handling of any input data, specific to those parameters.
The calling sequence is:

lsmsetup (3.50.10)
invokes the generic method in the registry to set up the land surface model

---

3.22.5 LIS_lsm_run (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_lsm_run(n)
```

USES:

ARGUMENTS:

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

DESCRIPTION:

This interface provides the entry point to the LSM routines that invokes the land surface model physics. The arguments are:

n index of the nest or domain

The calling sequence is:

lsmrun (3.50.4)
invokes the generic method in the registry to run the land surface model

---

3.22.6 LIS_lsm_perturb_states (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_lsm_perturb_states(n)
```

USES:

ARGUMENTS:

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

DESCRIPTION:

This interface provides the entry point to the LSM routines that computes the perturbations on the LSM prognostic state variables
The arguments are:

n index of the nest or domain

The calling sequence is:

perturbmethod (3.54.6)
invokes the abstract method to invoke the perturbation algorithm to perturb LSM prognostic variables

lsmdagetstatevar (3.50.22)
obtains the list of prognostic variables

applyLSMPert (3.22.14)
applies the specified perturbations to the LSM state

lsmdaqcstate (3.50.34)
performs the QC of the perturbed LSM state

lsmdasetstatevar (3.50.24)
assigns the prognostic variables back to the model states

3.22.7 LIS_lsm_readrestart (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_lsm_readrestart
```

USES:

DESCRIPTION:

This interface provides the entry point to read LSM specific model restart files.
The calling sequence is:

lsmrestart (3.50.12)
invokes the generic method in the registry to read restart files for the land surface model

---

3.22.8 LIS_setLSMDynparams (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_setLSMDynparams(n)
```

USES:

ARGUMENTS:

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

DESCRIPTION:

This interface provides an entry point to update any time dependent land surface model parameters in an LSM.
The arguments are:

n index of the nest or domain

The calling sequence is:

lsmdynsetup (3.50.14)
invokes the generic method in the registry to set up time dependent parameters for the land surface model

---

3.22.9 LIS_lsm_f2t (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_lsm_f2t(n)
```

USES:

ARGUMENTS:

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

DESCRIPTION:

This interface is used to transfer the forcing variables to the actual model tile space. Any forcing perturbations that need to be applied to the input forcing is applied at this stage as well.
The arguments are:

n index of the nest or domain

The calling sequence is:

lsmf2t (3.50.16)
invokes the generic method in the registry to transfer the forcing to the land surface model tiles

---

3.22.10 LIS_lsm_writerestart (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_lsm_writerestart(n)
```

USES:

ARGUMENTS:

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

DESCRIPTION:

This interface provides the entry point to read the LSM specific model restart files.
The arguments are:

n index of the nest or domain

The calling sequence is:

lsmwrst (3.50.18)
invokes the generic method in the registry to write restart files for the land surface model

---

3.22.11 lsm_setexport_noesmf (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine lsm_setexport_noesmf(n)
```

USES:

ARGUMENTS:

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

DESCRIPTION:

This interface provides the entry point for specifying an export state (a list of model specific variables) from a land surface model. The routine is used in a coupled simulation to provide feedback to a different model component such as an atmospheric model.
The arguments are:

n index of the nest or domain

The calling sequence is:

lsmcplsetexport (3.50.46)
invokes the generic method in the registry to set the export state from the land surface model

---

3.22.12 LIS_lsm_finalize (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_lsm_finalize()
```

USES:

DESCRIPTION:

This routine issues the invocation to deallocate and cleanup any allocated data structures in the specific instance of a land surface model
The calling sequence is:

lsmfinalize (3.50.6)
invokes the generic method in the registry to cleanup the LSM related datastructures

---

3.22.13 LIS_lsm_reset (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_lsm_reset()
```

USES:

DESCRIPTION:

This routine issues the invocation to deallocate and cleanup any allocated data structures in the specific instance of a land surface model
The calling sequence is:

lsmreset (3.50.8)
invokes the generic method in the registry to cleanup the LSM related datastructures

---

3.22.14 applyLSMPert (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine applyLSMPert(n, k, LIS_LSM_State, LIS_LSM_Pert_State)
```

USES:

ARGUMENTS:

```
integer, intent(IN) :: n
integer, intent(IN) :: k
type(ESMF_State)    :: LIS_LSM_State
type(ESMF_State)    :: LIS_LSM_Pert_State
```

DESCRIPTION:

This routine applies the specified perturbations to the LSM prognostic state variables.
The arguments are:

n index of the nest

LSM_State ESMF State with prognostic variables

LSM_Pert_State ESMF State with prognostic variable perturbations

---

3.22.15 LIS_mapSoilType (Source File: LIS_mapSoilType.F90)

3 Jul 2001: Matt Rodell Initial specifiation 13 Dec 2004: Sujay Kumar Updated with support for STATSGO classification 16 Jan 2014: Greg Fall Adjusted STATSGO classification logic to remove discontinuities and fix missing clay/silt loam regions of the USDA soil triangle. The logic is stated explicitly in the USDA training module "Soil Mechanics Level I, Module 3: USDA Textural Soil Classification," pp. 15-16, retrieved from ftp://ftp.wcc.nrcs.usda.gov/wntsc/H&H/training/soilsOther/soil-USDA-textural-class.pdf

INTERFACE:

```
subroutine LIS_mapSoilType(class,sand,clay,silt,soiltyp)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: class
real, intent(in)       :: sand, clay, silt
integer, intent(inout) :: soiltyp
```

DESCRIPTION:

This subroutine uses the percentages of sand, silt, and clay to convert to soil texture data. The transformation is based on the type of classification used. This routine supports the transformation to a Zobler (9 classes) or the STATSGO (19 classes) scheme.
The arguments are:

class soil classification scheme (1-zobler, 2-statsgo)

sand array containing the sand fraction data

clay array containing the clay fraction data

silt array containing the silt fraction data

soiltyp array containing the derived soil texture

3.22.16 LIS_mapSurfaceModelType (Source File: LIS_mapSurfaceModelType.F90)

REVISION HISTORY:

```
1 July 2012: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine LIS_mapSurfaceModelType(surface_model, sf_index)

  implicit none
```

ARGUMENTS:

```
  character(len=*)        :: surface_model
  integer                 :: sf_index
```

DESCRIPTION:

This subroutine maps the surface model type name to the index used in LIS. The following convention is adopted:
Land Surface Model (LSM) - 1
Lake Model - 2
Glacier Model - 3
Wetland Model - 4
Openwater Model - 5

3.23 Fortran: Module Interface LIS_metforcingMod (Source File: LIS_metforcingMod.F90)

The code in this file provides interfaces to manage different meteorological forcing analyses

3.23.1 Overview

This module defines interface plugins for the incorporation of various meteorological forcing analyses. The analyses implemented by extending these interfaces should contain all (and possibly more) of the basic meteorological forcing variables. Analyses containing one or more of the basic variables, but not the entire set, should be implmented as a supplemental forcing. Further, any analysis that does not cover the entire globe should be implemented as a supplemental forcing. The following is a list of the basic meterological forcing variables.

- `T 2m: Temperature interpolated to 2m` (K)
- `Q 2m: Instantaneous specific humidity interpolated to 2m` (kg/kg)
- `SWdown: Downward shortwave flux at the ground` (W/m^2)
- `LWdown: Downward shortwave flux at the ground` (W/m^2)
- `U 10m: Instantaneous zonal wind interpolated to 10m` (m/s)
- `V 10m: Instantaneous meridional wind interpolated to 10m` (m/s)
- `Psurf: Instantaneous surface pressure` (Pa)
- `rainf: Total precipitation` (mm/s)
- `rainf_c: Convective precipitation` (mm/s)
- `snowf: Total snowfall` (mm/s)
- `PET: Potential ET for FEWSNET` (mm/s)
- `RefET: Reference ET for FEWSNET` (mm/s)

- CAPE: Convective Available Potential Energy from NLDAS-2 (J/kg)

REVISION HISTORY:

```
 14 Nov 2002    Sujay Kumar  Initial Specification
 10 Oct 2006:   Sujay Kumar: Switched to using ESMF_State for storing
                             forcing data.
 26 May 2011:   Soni Yatheendradas: Added Potential ET forcings for FEWSNET
 14 Mar 2014:   David Mocko: Added CAPE as a forcing variable from NLDAS-2.
                             Moved CRainf (convective rainfall forcing)
                                from LIS_MOC_RAINFCONV to LIS_MOC_CRAINFFORC.
                             Added units of [kg/m^2] for PET and CRainf.

use ESMF
use LIS_FORC_AttributesMod
use LIS_spatialDownscalingMod
use LIS_logMod
use LIS_coreMod

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------

public :: LIS_metforcing_init      !initialize met forcing setup
public :: LIS_get_met_forcing    !retrieve data and interpolate
                                            !spatially and temporally
public :: LIS_perturb_forcing       !perturbs the met forcing variables
public :: LIS_metforcing_reset     !resets required forcing variables
public :: LIS_metforcing_finalize  !cleanup allocated structures
 public :: diagnoseForcingOutput ! prepares forcing variables for output

-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public   :: LIS_forc
public   :: LIS_FORC_State
public   :: LIS_FORC_Base_State
public   :: LIS_FORC_Pert_State
```

3.23.2 LIS_metforcing_init (Source File: LIS_metforcingMod.F90)

INTERFACE:

```
interface LIS_metforcing_init
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure metforcing_init_offline
module procedure metforcing_init_coupled
```

DESCRIPTION:

Initialization routine for metforcing structures. There are two private methods metd on whether LIS is run in an offline mode or coupled mode.

3.23.3 metforcing_init_offline (Source File: LIS_metforcingMod.F90)

INTERFACE:

```
!Private name: call using LIS_metforcing_init
subroutine metforcing_init_offline
```

USES:

```
use LIS_metforcing_pluginMod, only : LIS_metforcing_plugin
```

DESCRIPTION:

This routine sets up the structures to include meteorological forcing analyses for an offline run. The registry that defines the implemented forcing schemes are invoked followed by the routines to initialize the specific instance of the forcing scheme.
The methods invoked are:

LIS_metforcing_plugin (4.19.1)
sets up function table registries for implemented met forcing analyses

create_forcing_structures (3.23.5)
create and allocate memory for required structures

initmetforc (3.51.2)
invokes the generic method in the registry to define the native domain of the met forcing scheme

forcingPerturbSetup (3.23.6)
setup and allocate structures required for forcing perturbations.

3.23.4 metforcing_init_coupled (Source File: LIS_metforcingMod.F90)

INTERFACE:

```
subroutine metforcing_init_coupled(flag)
```

USES:

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

DESCRIPTION:

This routine sets up the structures to include meteorological forcing analyses in a coupled runmode. The routine simply allocates memory for the forcing variables.
The methods invoked are:

create_forcing_structures (3.23.5)
create and allocate memory for required structures

forcingPerturbSetup (3.23.6)
setup and allocate structures required for forcing perturbations.

3.23.5 create_forcing_structures (Source File: LIS_metforcingMod.F90)

INTERFACE:

```
subroutine create_forcing_structures()
```

USES:

DESCRIPTION:

This subroutine creates and allocates the memory for the forcing variables for an offline run. The ESMF object for forcing state is created with placeholders for all meteorological forcing variables

---

3.23.6 forcingPerturbSetup (Source File: LIS_metforcingMod.F90)

INTERFACE:

```
subroutine forcingPerturbSetup
```

USES:

```
use LIS_perturbMod
```

DESCRIPTION:

This routine sets up and allocates structures required for forcing pertubations. The ESMF state objects for perturbed forcing fields and the perturbations themselves are created. The perturbation related attributed are also assigned in this routine.

3.23.7 LIS_get_met_forcing (Source File: LIS_metforcingMod.F90)

INTERFACE:

```
subroutine LIS_get_met_forcing(n)
```

USES:

ARGUMENTS:

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

DESCRIPTION:

This routine issues the calls to retrieve the forcing variables from the specific forcing scheme. The retrieval call is expected to read the data, perform any spatial interpolation and other transformations needed to grid the data to the running domain and resolution. This invocation is followed by the call to temporally interpolate the data to the current model timestep. The temporal interpolation is performed based on the two consecutive forcing analyses. A zenith angle based interpolation is performed for the temporal disaggregation of downward shortwave radiation. Finally any topographic corrections and downscaling to the variables are applied.
The arguments are:

n index of the domain or nest.

The methods invoked are:

retrievemetforc (3.51.4)
invokes the generic method in the registry to retrieve the met forcing data

timeinterpmetforc (3.51.6)
invokes the generic method in the registry to perform temporal interpolation

LIS_lapseRateCorrection (3.34.1)
method to apply lapse-rate based topographical corrections

LIS_slopeAspectCorrection (3.34.2)
method to apply slope aspect based topographical corrections

LIS_microMetCorrection (3.23.16)
method to apply topographical corrections (not currently supported)

3.23.8 overlayForcings (Source File: LIS_metforcingMod.F90)

INTERFACE:

```
subroutine overlayForcings(n)

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine blends different forcing inputs by sequentially overlaying them on top of each other in the order in which they are specified.

3.23.9 ensembleForcings (Source File: LIS_metforcingMod.F90)

INTERFACE:

```
subroutine ensembleForcings(n)

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine blends different forcing inputs by assigning them to different ensemble members.

3.23.10 LIS_perturb_forcing (Source File: LIS_metforcingMod.F90)

INTERFACE:

```
subroutine LIS_perturb_forcing(n)
```

USES:

ARGUMENTS:

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

DESCRIPTION:

This routine computes the forcing perturbations and then applies them to the forcing fields. This routine also invokes the diagnose routine to map the forcing fields to the history writer.

diagnoseForcingOutput (3.23.15)
routine to map a forcing fields to the LIS history writer

3.23.11 LIS_metforcing_reset (Source File: LIS_metforcingMod.F90)

INTERFACE:

```
subroutine LIS_metforcing_reset
```

USES:

DESCRIPTION:

This subroutine issues the invocation to reset variables for specific forcing scheme.

3.23.12 LIS_metforcing_finalize (Source File: LIS_metforcingMod.F90)

INTERFACE:

```
subroutine LIS_metforcing_finalize
```

USES:

DESCRIPTION:

This routine issues the invocation to deallocate and cleanup any allocated data structures in the specific instance of the forcing scheme.
The methods invoked are:

finalmetforc (3.51.8)
invokes the generic method in the registry to cleanup the allocated structures for the met forcing scheme.

3.23.13 get_forcingvar_attributes (Source File: LIS_metforcingMod.F90)

INTERFACE:

```
subroutine get_forcingvar_attributes(forcConfig, forc_attrib, &
     forcname, varcount, &
     checkentry)
```

USES:

```
implicit none
```

ARGUMENTS:

```
type(ESMF_Config)      :: forcConfig
type(forc_attrib_type) :: forc_attrib
character(len=*)       :: forcname
integer                :: varcount
integer                :: checkentry
```

DESCRIPTION:

This subroutine reads the configurable options for each forcing variable. If the entry for the variable is not present, then the variable is set to be not chosen

---

3.23.14 add_forcing_fields (Source File: LIS_metforcingMod.F90)

INTERFACE:

```
subroutine add_forcing_fields(n, FORC_State, FORC_Base_State,forc_attrib)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer                   :: n
type(ESMF_State)          :: FORC_State
type(ESMF_State)          :: FORC_Base_State(LIS_rc%nmetforc)
type(forc_attrib_type)    :: forc_attrib
```

DESCRIPTION:

This subroutine creates fields for the specified forcing variables and initializes them to be undefined values.

3.23.15 diagnoseForcingOutput (Source File: LIS_metforcingMod.F90)

INTERFACE:

```
subroutine diagnoseForcingOutput(n)
```

USES:

```
use LIS_histDataMod
use LIS_constantsMod,   only : LIS_MS2KMDAY
```

ARGUMENTS:

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

DESCRIPTION:

This routines issues the diagnose calls to map the forcing variables to the history output writer.
The routines invoked are:

LIS_diagnoseOutputVar (3.16.5)
generic routine to map a single variable to the LIS history writer

---

3.23.16 LIS_microMetCorrection (Source File: LIS_microMetCorrection.F90)

REVISION HISTORY:

```
21 Dec 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine LIS_microMetCorrection(nest)
```

USES:

```
use LIS_logMod, only : LIS_logunit

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Corrects Temperature, Pressure, Humidity and Longwave Radiation values for topography (based on Liston et al; 2005)
WARNING: This routine is not currently implemented
The arguments are:

nest index of the nest

3.24 Fortran: Module Interface LIS_mpiMod (Source File: LIS_mpiMod.F90)

REVISION HISTORY:

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

USES:

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

DESCRIPTION:

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

3.25 Fortran: Module Interface LIS_numerRecipesMod (Source File: LIS_numerRecipesMod

The code in this file contains a few borrowed routines from the Fortran Numerical Recipes book

REVISION HISTORY:

```
implicit none

PRIVATE

public :: LIS_gasdev
public :: LIS_rand_func
```

3.25.1 LIS_gasdev (Source File: LIS_numerRecipesMod.F90)

Returns a normally distributed deviate with zero mean and unit variance, using ran2(idum) as the source of uniform deviates. This routine is adapted from the Numerical Recipies for Fortran

REVISION HISTORY:

```
27 Feb 2005: Sujay Kumar : Specification in LIS.
```

INTERFACE:

```
FUNCTION LIS_gasdev(idum)
```

3.25.2 ran2 (Source File: LIS_numerRecipesMod.F90)

Long period (¿2!1e18) random number generator of L'Ecuyer with Bays-Durham shuffle and added safeguards. Returns a uniform random deviate between 0.0 and 1.0 (exclusive of the endpoint values). Call with "idum" a negative integer to initialize; thereafter, do not alter "idum" between successive deviates in a sequence. RNMX should approximate the largest floating value that is less than 1. This function is adapted from Numerical Recipies for Fortran.

REVISION HISTORY:

```
27 Feb 2005: Sujay Kumar : Specification in LIS.
```

INTERFACE:

```
FUNCTION ran2(idum)
```

3.26 Fortran: Module Interface LIS_openwatermodelMod (Source File: LIS_openwatermode

The code in this file provides interfaces to manage the operation of different ocean models

REVISION HISTORY:

```
 16 Jul 2012    Sujay Kumar  Initial Specification

use ESMF

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: LIS_openwatermodel_init       ! initialize openwater variables, memory
public :: LIS_setupopenwatermodel       ! set land surface parameters
public :: LIS_openwatermodel_readrestart    ! read the restart file
public :: LIS_openwatermodel_f2t     ! transfer forcing to model tiles
public :: LIS_openwatermodel_run       ! execute the land model
public :: LIS_openwatermodel_output     ! write model output
public :: LIS_openwatermodel_setdynparams   ! set the time dependent parameters
public :: LIS_openwatermodel_writerestart   ! write the restart file
public :: LIS_openwatermodel_finalize   ! cleanup allocated structures
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
```

3.26.1 LIS_openwatermodel_init (Source File: LIS_openwatermodelMod.F90)

INTERFACE:

```
subroutine LIS_openwatermodel_init()
```

USES:

```
use ESMF
use LIS_surfaceModelDataMod
use LIS_coreMod, only : LIS_rc, LIS_config, LIS_vecTile
use LIS_openwatermodel_pluginMod,    only : LIS_openwatermodel_plugin
use LIS_logMod,       only : LIS_logunit, LIS_verify, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber
```

DESCRIPTION:

Interface for initializing the land model. The intialization includes the allocation of memory for OPENWATER specific variables and datastructures and specification of any runtime specific options.
The calling sequence is:

LIS_openwatermodel_plugin (4.20.1)
sets up function table registries for implemented land surface models

openwaterinit (3.52.2)
invokes the generic method in the registry to initialize the land surface model

---

3.26.2 LIS_setupopenwatermodel (Source File: LIS_openwatermodelMod.F90)

INTERFACE:

```
subroutine  LIS_setupopenwatermodel
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

DESCRIPTION:

The setup interfaces are used for the OPENWATER specification of OPENWATER bparameters. If a parameter falls outside the LIS-specified generic parameter list, the OPENWATER is expected to provide routines for the handling of any input data, specific to those parameters.
The calling sequence is:

openwatersetup (3.52.8)
invokes the generic method in the registry to set up the land surface model

---

3.26.3 LIS_openwatermodel_run (Source File: LIS_openwatermodelMod.F90)

INTERFACE:

```
subroutine LIS_openwatermodel_run(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

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

DESCRIPTION:

This interface provides the entry point to the OPENWATER routines that invokes the land surface model physics.
The arguments are:

n index of the nest or domain

The calling sequence is:

openwaterrun (3.52.4)
invokes the generic method in the registry to run the land surface model

---

3.26.4 LIS_openwatermodel_readrestart (Source File: LIS_openwatermodelMod.F90)

INTERFACE:

```
subroutine LIS_openwatermodel_readrestart
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

DESCRIPTION:

This interface provides the entry point to read OPENWATER specific model restart files.
The calling sequence is:

openwaterrestart (3.52.10)
invokes the generic method in the registry to read restart files for the land surface model

---

3.26.5 LIS_openwatermodel_output (Source File: LIS_openwatermodelMod.F90)

INTERFACE:

```
subroutine LIS_openwatermodel_output(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

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

DESCRIPTION:

This interface provides the entry point to write OPENWATER specific model output files.
The arguments are:

n index of the nest or domain

The calling sequence is:

openwateroutput (3.52.14)
invokes the generic method in the registry to write the land surface model output

---

3.26.6 LIS_openwatermodel_setdynparams (Source File: LIS_openwatermodelMod.F90)

INTERFACE:

```
subroutine LIS_openwatermodel_setdynparams(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

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

DESCRIPTION:

This interface provides an entry point to update any time dependent land surface model parameters in an OPENWATER.
The arguments are:

n index of the nest or domain

The calling sequence is:

openwaterdynsetup (3.52.12)
invokes the generic method in the registry to set up time dependent parameters for the land surface model

---

3.26.7 LIS_openwatermodel_f2t (Source File: LIS_openwatermodelMod.F90)

INTERFACE:

```
subroutine LIS_openwatermodel_f2t(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

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

DESCRIPTION:

This interface is used to transfer the forcing variables to the actual model tile space. Any forcing perturbations that need to be applied to the input forcing is applied at this stage as well.
The arguments are:

n index of the nest or domain

The calling sequence is:

openwaterf2t (3.52.16)
invokes the generic method in the registry to transfer the forcing to the land surface model tiles

---

3.26.8 LIS_openwatermodel_writerestart (Source File: LIS_openwatermodelMod.F90)

INTERFACE:

```
subroutine LIS_openwatermodel_writerestart(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

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

DESCRIPTION:

This interface provides the entry point to read the OPENWATER specific model restart files.
The arguments are:

n index of the nest or domain

The calling sequence is:

openwaterwrst (3.52.18)
invokes the generic method in the registry to write restart files for the land surface model

---

3.26.9 LIS_openwatermodel_finalize (Source File: LIS_openwatermodelMod.F90)

INTERFACE:

```
subroutine LIS_openwatermodel_finalize()
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

DESCRIPTION:

This routine issues the invocation to deallocate and cleanup any allocated data structures in the specific instance of a land surface model
The calling sequence is:

openwaterfinalize (3.52.6)
invokes the generic method in the registry to cleanup the OPENWATER related datastructures

---

3.27 Fortran: Module Interface LIS_optUEMod (Source File: LIS_optUEMod.F90)

The code in this file provides interfaces to manage optimization and uncertainty modeling implementations in LIS

3.27.1 Overview

The module provides interfaces for incorporating optimization and uncertainty modeling implementations into LIS. The abstract representation is to have the algorithm work between a certain objective function and a certain decision space using a specified opt/UE algorithm. The algorithm attempts to find potential solutions by optimizating the objective function with respect to the decision space constraints.
An example of an optimization algorithm is the suite of Genetic Algorithm-based heuristic optimization techniques

REVISION HISTORY:

```
15 May 2007: Sujay Kumar; Initial implementation
```

3.27.2 LIS_optUE_init (Source File: LIS_optUEMod.F90)

INTERFACE:

```
subroutine LIS_optUE_init
```

USES:

```
use ESMF
use LIS_coreMod,  only : LIS_rc, LIS_config, LIS_vecTile
use LIS_logMod,   only : LIS_verify
```

DESCRIPTION:

This routine intializes the optimization/uncertainty estimation mode, the objective function and decision space objects.

3.27.3 LIS_objectiveFunc_init (Source File: LIS_optUEMod.F90)

INTERFACE:

```
subroutine LIS_objectiveFunc_init
```

USES:

```
use ESMF
use LIS_coreMod,  only : LIS_rc
```

DESCRIPTION:

This routine intializes the optimization/uncertainty estimation algorithm, the objective function
The methods invoked are:

objectivefunctypeinit (3.53.26)
invokes the initialization routine for the specified objection function metric/method

---

3.27.4 LIS_optUEAlg_init (Source File: LIS_optUEMod.F90)

INTERFACE:

```
subroutine LIS_optUEAlg_init
```

USES:

```
use ESMF
use LIS_coreMod,  only : LIS_rc
```

DESCRIPTION:

This routine intializes the optimization/uncertainty estimation algorithm.
The methods invoked are:

optUEalginit (3.53.2)
invokes the init routine for the specified optimization or uncertainty estimation algorithm.

3.27.5 LIS_isOptStopCriterionTrue (Source File: LIS_optUEMod.F90)

INTERFACE:

```
function LIS_isOptStopCriterionTrue() result(finish)
```

USES:

```
use LIS_coreMod,   only : LIS_rc
```

ARGUMENTS:

```
logical :: finish
```

DESCRIPTION:

Invokes the appropriate method from the registry to check if the convergence criteria for the specified optimization /uncertainty estimation algorithm is met.
The methods invoked are:

checkconvergence (3.53.6)
invokes the method from the registry that specifies the stopping criteria of the algorithm

---

3.27.6 LIS_runoptUE (Source File: LIS_optUEMod.F90)

INTERFACE:

```
subroutine LIS_runOptUEAlg()
```

USES:

```
use LIS_coreMod,   only : LIS_rc
```

DESCRIPTION:

invokes the run method of the selected optimization/uncertainty estimation algorithm from the registry
The methods invoked are:

runoptue (3.53.8)
invokes the specific optue algorithm related operations to solve for potential solutions

resetobjectivefunctype (3.53.32)
invokes the call to reset the objective function objects based on the specific method

---

3.27.7 LIS_optUEAlg_readrestart (Source File: LIS_optUEMod.F90)

INTERFACE:

```
subroutine LIS_optUEAlg_readrestart()
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

DESCRIPTION:

invokes the read restart method for the selected optimization/uncertainty estimation algorithm from the registry
The methods invoked are:

optuereadrestart (3.53.16) invokes the call to read the restart file from the OPT/UE algorithm.

3.28 Fortran: Module Interface LIS_paramsMod (Source File: LIS_paramsMod.F90)

The code in this file provides interfaces to manage different sources of parameter datasets.

3.28.1 Overview

This module contains interface plugins for the incorporating I/O associated with various sources of land surface parameter maps. These interfaces act as the entry points for handing issues such as format, ordering,and projection associated with a dataset. The parameter reading routines are expected to perform all these transformations and convert the data to same grid, domain, and data ordering as the ones used in the running domain of LIS. The `param_module` provides interfaces to incorporate the following sources of parameter maps.

soils sand, silt, clay fractions, soil color, soil texture

albedo climatology and max albedo over deep snow

greenness climatology

Leaf Area Index climatology

Stem Area Index climatology

Topography static elevation, slope, aspect, curvature data

Bottom Temperature static

REVISION HISTORY:

```
 26 Oct 2005    Sujay Kumar  Initial Specification

use LIS_LMLCMod
use LIS_topoMod
use LIS_vegDataMod
use LIS_albedoMod
use LIS_emissMod
use LIS_soilsMod

implicit none

PRIVATE

-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
 public :: LIS_param_init  !initializes structures, read static data
 public :: LIS_setDynParams !read time dependent data
 public :: LIS_param_finalize !cleanup allocated structures
 public :: LIS_param_reset
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
```

3.28.2 LIS_param_init (Source File: LIS_paramsMod.F90)

INTERFACE:

```
subroutine LIS_param_init()
```

USES:

```
use LIS_coreMod,    only : LIS_rc
```

DESCRIPTION:

This interface provides the entry point for the initialization of data structures required for reading in parameter datasets. It also invokes routines to read static datasets such as soils.
The calling sequence is:

LIS_albedo_setup (3.7.2)
call to setup albedo parameter options

LIS_greenness_setup (3.38.2)
call to setup greenness parameter options

LIS_lai_setup (3.38.20)
call to setup LAI parameter options

LIS_sai_setup (3.38.25)
call to setup SAI parameter options

3.28.3 LIS_setDynparams (Source File: LIS_paramsMod.F90)

INTERFACE:

```
subroutine LIS_setDynparams(n)
```

USES:

```
use LIS_coreMod,   only : LIS_rc
use LIS_lsmMod,    only : LIS_setLSMDynparams
 use prism_module, only : prism_dataops
```

ARGUMENTS:

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

DESCRIPTION:

This interface provides the entry point for routines to update any time dependent land surface parameters. Current implementation includes datasets such as albedo, greenness, LAI, and SAI.
The arguments are:

n index of the domain or nest.

The calling sequence is:

LIS_read_greenness (3.38.3)
call to read the greenness data source

LIS_read_albedo (3.7.4)
call to read the albedo data source

LIS_read_lai (3.38.21)
call to read the LAI data source

LIS_read_sai (3.38.26)
call to read the SAI data source

3.28.4 diagnoseOutputparams (Source File: LIS_paramsMod.F90)

INTERFACE:

```
subroutine diagnoseOutputparams(n)
```

USES:

```
use LIS_coreMod,      only : LIS_rc
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: n
```

DESCRIPTION:

This routine is used to enable the output of land surface parameters to a file, once a month.
The arguments are:

n index of the nest

The routines called are:

LIS_diagnoselandmask (3.2.3)
maps the landmask data to the LIS history writer

LIS_diagnoseLandcover (3.2.4)
maps the landcover data to the LIS history writer

LIS_diagnoseTopography (3.37.3)
maps the topography data to the LIS history writer

LIS_diagnosesoils (3.33.3)
maps the soils data to the LIS history writer

LIS_diagnosegfrac (3.38.5)
maps the greenness data to the LIS history writer

LIS_diagnoseLAI (3.38.23)
maps the LAI data to the LIS history writer

LIS_diagnoseSAI (3.38.28)
maps the SAI data to the LIS history writer

LIS_diagnosealbedo (3.7.6)
maps the albedo data to the LIS history writer

3.28.5 LIS_param_finalize (Source File: LIS_paramsMod.F90)

INTERFACE:

```
subroutine LIS_param_finalize()
```

USES:

```
use LIS_coreMod, only  : LIS_rc
```

DESCRIPTION:

This routine issues the invocation to deallocate and cleanup any allocated data structures used in the parameter dataset implementations.
The calling sequence is:

LIS_albedo_finalize (3.7.5)
call to cleanup albedo related structures

LIS_greenness_finalize (3.38.4)
call to cleanup greenness related structures

LIS_lai_finalize (3.38.22)
call to LAI related structures

LIS_sai_finalize (3.38.27)
call to SAI related structures

3.28.6 LIS_param_reset (Source File: LIS_paramsMod.F90)

INTERFACE:

```
subroutine LIS_param_reset()
```

USES:

```
use LIS_coreMod,   only : LIS_rc
```

ARGUMENTS:

DESCRIPTION:

This interface provides the entry point for routines to update any time dependent land surface parameters. Current implementation includes datasets such as albedo, greenness, LAI, and SAI.
The arguments are:

n index of the domain or nest.

The calling sequence is:

LIS_greenness_reset (3.38.6)
resets the greenness data structures

LIS_albedo_reset (3.7.7)
resets the albedo data structures

LIS_lai_reset (3.38.24)
resets the lai data structures

LIS_sai_reset (3.38.29)
resets the sai data structures

3.29 Fortran: Module Interface LIS_perturbMod (Source File: LIS_perturbMod.F90)

The code in this file provides abstractions to manage different perturbation implementations. This includes methods to perturb forcing, observations, and land surface states.

REVISION HISTORY:

```
25 Jun 2006: Sujay Kumar; Initial implementation
24Nov10    Sujay Kumar; Added support for time varying
                        perturbations
```

3.29.1 LIS_perturb_init (Source File: LIS_perturbMod.F90)

INTERFACE:

```
subroutine LIS_perturb_init
```

DESCRIPTION:

The routines invoked are:

LIS_perturb_plugin (4.23.1)
sets up the function table registries for implemented perturbation algorithms

USES:

```
use LIS_logMod,             only : LIS_verify
use LIS_timeMgrMod,         only : LIS_registerAlarm
use LIS_perturb_pluginMod
```

3.29.2 LIS_perturb_readrestart (Source File: LIS_perturbMod.F90)

INTERFACE:

```
subroutine LIS_perturb_readrestart
```

USES:

ARGUMENTS:

DESCRIPTION:

3.29.3 LIS_perturb_writerestart (Source File: LIS_perturbMod.F90)

INTERFACE:

```
subroutine LIS_perturb_writerestart(n)
```

USES:

ARGUMENTS:

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

DESCRIPTION:

3.29.4 LIS_readPertAttributes (Source File: LIS_perturbMod.F90)

REVISION HISTORY:

```
14 Oct 2006: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine LIS_readPertAttributes(nvars, fname,pert_struc)
```

USES:

```
use LIS_logMod,    only : LIS_logunit, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber

implicit none
```

ARGUMENTS:

```
integer             :: nvars
character(len=*)    :: fname
type(pert_dec_type) :: pert_struc
```

DESCRIPTION:

This routine reads the observation perturbation attributes for each observation type. The format of the perturbation attributes file is:

```
<Variable name>
<perturbation type> <perturbation standard deviation type>
<standard deviation> <coeff standard deviation (optional, only if
perturbation standard deviation type is > 0) <standard normal max> (contd.)
<ensure zeromean?> <temporal correlation scale> (contd.)
<x correlation scale>  <y correlation scale> <cross correlations>
```

The arguments are:

n index of nest

vname name of the observation variables

perttype type of perturbation (0-additive, 1-multiplicative)

ssdev standard deviation of perturbation

stdmax standard normal max for the perturbation distribution

zeromean flag to specify if zero mean is to be ensured

tcorr temporal correlation scale (seconds)

xcorr horizontal x-correlation scale

ycorr horizontal y-correlation scale

ccorr cross correlations

3.30 Fortran: Module Interface LIS_precisionMod (Source File: LIS_precisionMod.F90)

REVISION HISTORY:

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

ARGUMENTS:

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

#if (defined DOUBLE_PRECISION)
  real(r8), parameter :: inf = O'77760000000000000000'
  real(r8), parameter :: nan = O'77767777777777777777'
#else

#if (defined SYSLINUX)
  real(r8), parameter :: inf = Z'7F800000'
  real(r8), parameter :: nan = Z'7FC00000'
#else
   real(r8), parameter :: inf = O'17740000000'
   real(r8), parameter :: nan = O'17757777777'
  real(r8), parameter :: inf = 0
  real(r8), parameter :: nan = 0
#endif


#endif
   integer, parameter  :: bigint = 100000000
  integer, parameter  :: bigint = 100
```

DESCRIPTION:

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

3.30.1 LIS_process_cmd_args (Source File: LIS_process_cmd_args.F90)

INTERFACE:

```
subroutine LIS_process_cmd_args
```

USES:

```
  use LIS_coreMod,   only : LIS_masterproc, LIS_rc
  use LIS_logMod,    only : LIS_endrun
#if ( defined AIX )
   use xlfutility, only : iargc
#endif
```

ARGUMENTS:

```
None
```

DESCRIPTION:

Processes command line arguments supplied to LIS at runtime.

Option	Description
`-V` or `--version`	cause LIS to dump information about itself and then end.
`-f <name>` or `--file <name>`	specify the name of the LIS runtime configuration file. If this option is not specified, then LIS will default to reading "lis.config".

3.30.2 LIS_readConfig (Source File: LIS_readConfig.F90)

REVISION HISTORY:

```
23 Feb 2001: Urszula Jambor; Added GEOS or GDAS forcing option
27 Mar 2001: Jon Gottschalck; Revision of subroutine by implementing namelists
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
02 Feb 2006: Sujay Kumar; Switched to the Inpak format
19 Jan 2007: Chuck Alonge; Added read for Snow Depth (snowh)
             and added parameter output option (wparm)
29 Dec 2007: Marv Freimund; Used trim on filenames
17 Jan 2011: David Mocko, added max/min greenness & slope type
```

INTERFACE:

```
subroutine LIS_readConfig()
```

USES:

```
  use ESMF
  use LIS_coreMod,    only : LIS_rc, LIS_config, &
       LIS_localPet, LIS_npes, LIS_masterproc
  use LIS_histDataMod, only : LIS_histData
  use LIS_timeMgrMod, only : LIS_date2time, LIS_parseTimeString
  use LIS_logMod
```

DESCRIPTION:

The code in this file initializes the LIS configuration management utility. The generic, model-independent runtime specifications of a LIS simulation are read by this routine. The routine also initializes the LIS log buffers
The routines invoked are:

LIS_date2time (3.36.24)
convert the date format to a floating point time format

3.30.3 LIS_readDAObsAttributes (Source File: LIS_readDAObsAttributes.F90)

REVISION HISTORY:

```
14 Oct 2006: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine LIS_readDAObsAttributes(k, vname,varmin,varmax)
```

USES:

```
use LIS_coreMod,   only : LIS_rc
use LIS_logMod,    only : LIS_logunit, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber

implicit none
```

ARGUMENTS:

```
integer                     :: k
character(len=*), allocatable  :: vname(:)
real        , allocatable      :: varmin(:)
real        , allocatable      :: varmax(:)
integer                     :: ftn
```

DESCRIPTION:

This routine reads the observation attributes for each observation type. The format of the attributes file is:
Variable name Min Value Max Value
The arguments are:

n index of nest

vname name of the observation variables

varmin minimum value of the variable

varmax maximum value of the variable

3.30.4 LIS_readPEDecSpaceAttributes (Source File: LIS_readPEDecSpaceAttributes.F90)

REVISION HISTORY:

```
12 Jan 2012: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine LIS_readPEDecSpaceAttributes(decattribfile, &
     nparam, vname,selectOpt,varmin,varmax)
```

USES:

```
use LIS_coreMod,   only : LIS_rc
use LIS_logMod,    only : LIS_logunit, LIS_endrun, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber

implicit none
```

ARGUMENTS:

```
character(len=*)   :: decattribfile
integer            :: nparam
character(len=*)   :: vname(nparam)
integer            :: selectOpt(nparam)
real               :: varmin(nparam)
real               :: varmax(nparam)
```

DESCRIPTION:

This routine reads the observation attributes for each observation type. The format of the attributes file is:
Variable name standard deviation Max Value Min Value
The arguments are:

n index of nest

vname name of the observation variables

ssdev error rate in observation

varmin minimum value of the variable

varmax maximum value of the variable

3.31 Fortran: Module Interface LIS_routingMod (Source File: LIS_routingMod.F90)

The code in this file provides the top level calls to manage the operation of different runoff-routing algorithms and models.

REVISION HISTORY:

```
6 May 2011: Sujay Kumar, Initial implementation
```

USES:

```
use ESMF

implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: LIS_routing_init         ! initialize the routing model
public :: LIS_routing_readrestart  ! read the routing model restart file
public :: LIS_routing_run           ! execute the routing model
public :: LIS_routing_writeoutput   ! write the output file
public :: LIS_routing_writerestart   ! write the restart file
```

PUBLIC TYPES:

```
public :: LIS_runoff_state
```

3.31.1 LIS_routing_init (Source File: LIS_routingMod.F90)

INTERFACE:

```
subroutine LIS_routing_init
```

USES:

```
use LIS_coreMod,  only : LIS_rc, LIS_Config
use LIS_logMod,   only : LIS_verify
```

DESCRIPTION:

3.31.2 LIS_routing_readrestart (Source File: LIS_routingMod.F90)

INTERFACE:

```
subroutine LIS_routing_readrestart
```

USES:

```
use LIS_coreMod,  only : LIS_rc
```

DESCRIPTION:

3.31.3 LIS_routing_run (Source File: LIS_routingMod.F90)

INTERFACE:

```
subroutine LIS_routing_run(n)
```

USES:

```
use LIS_coreMod,  only : LIS_rc
```

DESCRIPTION:

3.31.4 LIS_routing_writeoutput (Source File: LIS_routingMod.F90)

INTERFACE:

```
subroutine LIS_routing_writeoutput(n)
```

USES:

```
use LIS_coreMod,  only : LIS_rc
```

DESCRIPTION:

3.31.5 LIS_routing_writerestart (Source File: LIS_routingMod.F90)

INTERFACE:

```
subroutine LIS_routing_writerestart(n)
```

USES:

```
use LIS_coreMod,  only : LIS_rc
```

DESCRIPTION:

3.32 Fortran: Module Interface LIS_snowMod (Source File: LIS_snowMod.F90)

The code in this file implementts routines to read various sources of snow depth data. Currently this module is used to ingest the SNODEP products from AFWA, which is obtained at 12z daily.

REVISION HISTORY:

```
08 Aug 2005: Sujay Kumar; Initial implementation
```

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: LIS_snow_setup       !allocates memory for required variables
public :: LIS_snow_finalize    !cleanup allocated structures
```

PUBLIC TYPES:

```
public :: LIS_snow_struc      !data structure containing snow variables
```

3.32.1 LIS_snow_setup (Source File: LIS_snowMod.F90)

Allocates memory for data structures used for reading snow datasets. The snowdepth field is updated by the external files. The snow water equivalent fields are expected to be set by the model.

INTERFACE:

```
subroutine LIS_snow_setup
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc
```

3.32.2 LIS_snow_finalize (Source File: LIS_snowMod.F90)

This routine cleans up snow-related structures

INTERFACE:

```
subroutine LIS_snow_finalize
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

3.33 Fortran: Module Interface LIS_soilsMod (Source File: LIS_soilsMod.F90)

The code in this file implements routines to read various sources of soil parameter data.

3.33.1 Overview

This module provides routines for reading and manipulating various parameters related to soil properties. The following list of soil parameters are currently supported.

sand, silt, clay fractions

soil color data

soil texture data

soil porosity data

hydraulic conductivity data

thermal conductivity data

b parameter data

quartz data

REVISION HISTORY:

```
    21 Oct 2005: Sujay Kumar; Initial implementation
    3  Apr 2012: Sujay Kumar; Switched to the use of LPT based parameter file
   use LIS_fileIOMod

   implicit none

   PRIVATE
  ------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
  ------------------------------------------------------------------------------
   public :: LIS_soils_init  ! initializes data structures and read soil data
   public :: LIS_diagnosesoils ! routine that maps soils data to the history writer
   public :: LIS_soils_finalize !cleanup allocated structures
  ------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
  ------------------------------------------------------------------------------
   public :: LIS_soils      !data structure containing soils data
```

3.33.2 LIS_soils_init (Source File: LIS_soilsMod.F90)

INTERFACE:

```
   subroutine LIS_soils_init()
```

USES:

```
     use ESMF
     use LIS_coreMod,  only : LIS_rc
     use LIS_logMod,   only : LIS_logunit, LIS_endrun
```

DESCRIPTION:

Allocates memory for data structures for reading soils datasets
Reads the soils data based on the choice of options specified in the lis configuration.
The routines invoked are:

read_soiltexture (3.33.6)
invokes the generic method in the registry to read the soil texture data

read_soilfraction (3.33.7)
invokes the generic method in the registry to read the soil fraction data

read_porosity (3.33.5)
invokes the generic method in the registry to read the soil porosity data

LIS_read_param (3.14.4)
reads the specified soil property from the LIS input parameters file (e.g.; lis_input.d01.nc)

3.33.3 LIS_diagnosesoils (Source File: LIS_soilsMod.F90)

INTERFACE:

```
subroutine LIS_diagnosesoils(n)
```

USES:

```
use LIS_coreMod,     only : LIS_rc, LIS_domain
use LIS_histDataMod, only : LIS_diagnoseSurfaceOutputVar, &
                            LIS_MOC_SOILTYPE,             &
                            LIS_MOC_SANDFRAC,             &
                            LIS_MOC_CLAYFRAC,             &
                            LIS_MOC_SILTFRAC,             &
                            LIS_MOC_SOILCOLOR,            &
                            LIS_MOC_POROSITY
```

ARGUMENTS:

```
implicit none
integer, intent(in)   :: n
```

DESCRIPTION:

This routine writes the LIS soils to the LIS history writer
The arguments are:

n index of the nest

ftn file unit number to be used

The routines called are:

LIS_diagnoseOutputVar (3.16.5)
This routine maps a variable to the history writing routines

3.33.4 LIS_soils_finalize (Source File: LIS_soilsMod.F90)

INTERFACE:

```
subroutine LIS_soils_finalize()
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

DESCRIPTION:

deallocates memory for all datastructures used for reading soils datasets. This method is typically called after the information is translated to the LSM model tiles.

3.33.5 read_porosity (Source File: LIS_soilsMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_porosity(n)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
  use LIS_coreMod,        only : LIS_rc, LIS_localPet,&
       LIS_ews_ind, LIS_ewe_ind,&
       LIS_nss_ind, LIS_nse_ind, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
       LIS_nss_halo_ind, LIS_nse_halo_ind
  use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
       LIS_releaseUnitNumber, LIS_endrun, LIS_verify

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine reads the porosity data
The arguments are:

n index of n

localmask porosity for the region of interest

3.33.6 read_soiltexture (Source File: LIS_soilsMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_soiltexture(n)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
  use LIS_coreMod,        only : LIS_rc, LIS_localPet,&
       LIS_ews_ind, LIS_ewe_ind,&
       LIS_nss_ind, LIS_nse_ind, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
       LIS_nss_halo_ind, LIS_nse_halo_ind
  use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
       LIS_releaseUnitNumber, LIS_endrun, LIS_verify

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine reads the soiltexture data
The arguments are:

n index of n

locallc landlc for the region of interest

3.33.7 read_soilfraction (Source File: LIS_soilsMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_soilfraction(n)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
  use LIS_coreMod,        only : LIS_rc, LIS_localPet,&
       LIS_ews_ind, LIS_ewe_ind,&
       LIS_nss_ind, LIS_nse_ind, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
       LIS_nss_halo_ind, LIS_nse_halo_ind
  use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
       LIS_releaseUnitNumber, LIS_endrun, LIS_verify

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine reads the soilfraction data
The arguments are:

n index of n

locallc landlc for the region of interest

3.34 Fortran: Module Interface LIS_spatialDownscalingMod (Source File: LIS_spatialDowns

REVISION HISTORY:

```
 6 Nov 2012: Sujay Kumar; initial specification

use ESMF
use LIS_logMod
use LIS_constantsMod
use LIS_coreMod
use LIS_FORC_AttributesMod
use LIS_fileIOMod

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
PUBLIC :: LIS_lapseRateCorrection
PUBLIC :: LIS_slopeAspectCorrection
PUBLIC :: LIS_init_pcpclimo
PUBLIC :: LIS_init_pcpclimo_native
PUBLIC :: LIS_generatePcpClimoRatioField
PUBLIC :: LIS_pcpClimoDownscaling
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
PUBLIC :: LIS_pcpclimo

type, public :: pcpclimo_dec_type
   real, allocatable :: pcpdata1_native(:)
   real, allocatable :: pcpdata2_native(:)
   real, allocatable :: pcpdata1(:)
   real, allocatable :: pcpdata2(:)
   integer       :: month1
   integer       :: month2
end type pcpclimo_dec_type

type(pcpclimo_dec_type), allocatable :: LIS_pcpclimo(:,:)
```

3.34.1 LIS_lapseRateCorrection (Source File: LIS_spatialDownscalingMod.F90)

REVISION HISTORY:

```
 11 Apr 2000: Brian Cosgrove; Initial Code
 12 May 2000: Brian Cosgrove; Corrected for zero humidities
 25 Jan 2001: Matt Rodell; Compute number of input and output
grid points, use to allocate local arrays
 15 Mar 2001: Jon Gottschalck; if-then to handle negative vapor
pressures in long wave correction
 14 Nov 2003: Sujay Kumar; Adopted in LIS
```

INTERFACE:

```
subroutine LIS_lapseRateCorrection(nest, modelelev, LIS_FORC_Base_State)
```

USES:

```
  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: nest
  real                :: modelelev(LIS_rc%ngrid(nest))
  type(ESMF_State)    :: LIS_FORC_Base_State
```

DESCRIPTION:

Corrects the lowest model level Temperature, Pressure, Humidity and Longwave Radiation forcing values for differences in elevation between the LIS running grid and the native forcing grid. The corrections are based on the lapse-rate and hypsometric adjustments to these variables described in Cosgrove et. al (2003). Cosgrove, B.A. et.al, Real-time and retrospective forcing in the North American Land Data Assimilation (NLDAS) project, Journal of Geophysical Research, 108(D22), 8842, DOI: 10.1029/2002JD003118, 2003. The arguments are:

nest index of the domain or nest.

findex index of the forcing dataset

3.34.2 LIS_slopeAspectCorrection (Source File: LIS_spatialDownscalingMod.F90)

REVISION HISTORY:

```
1 Feb 2011: Sujay Kumar; Initial implementation
```

INTERFACE:

```
subroutine LIS_slopeAspectCorrection(nest,LIS_FORC_Base_State)
```

USES:

```
  implicit none
```

ARGUMENTS:

```
integer, intent(in) :: nest
type(ESMF_State)    :: LIS_FORC_Base_State
```

DESCRIPTION:

This subroutine adjusts the downward shortwave radiation to correct for the influence of topographic slope and aspect. The method separates the downward shortwave values into direct and diffuse components first. The topographic corrections are then applied to the direct component.
Adapted from Dingman, S.L., Physical Hydrology - 2nd ed.- Appendix E, Prentice Hall, 2002. The separation of the downward shortwave forcing into direct and diffuse components is adopted from HySSIB LSM. The slope and aspect based correction of the direct shortwave term is adopted from FASST LSM.
The arguments are:

nest index of the domain or nest.

findex index of the forcing dataset

3.34.3 LIS_generatePcpClimoRatioField (Source File: LIS_spatialDownscalingMod.F90)

REVISION HISTORY:

INTERFACE:

```
subroutine LIS_generatePcpClimoRatioField(n, m, forcing_name, month, &
     input_size, input_data, input_bitmap)
```

USES:

```
#if ( defined USE_NETCDF3 || defined USE_NETCDF4 )
  use netcdf
#endif

    implicit none
```

ARGUMENTS:

```
    integer                 :: n
    integer                 :: m
    character(len=*)        :: forcing_name
    integer                 :: month
    integer                 :: input_size
    real                    :: input_data(input_size)
    logical*1               :: input_bitmap(Input_size)
```

DESCRIPTION:

3.34.4 LIS_surfaceModel_setexport (Source File: LIS_surfaceModelMod.F90)

INTERFACE:

```
interface LIS_surfaceModel_setexport
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure surfaceModel_setexport_noesmf
```

DESCRIPTION:

This interface provides the entry point for specifying an export state (a list of model specific variables) from a land surface model. The routine is used in a coupled simulation to provide feedback to a different model component such as an atmospheric model.

3.34.5 LIS_surfaceModel_init (Source File: LIS_surfaceModelMod.F90)

INTERFACE:

```
subroutine LIS_surfaceModel_init
```

DESCRIPTION:

This routine initialized the surface models.

3.34.6 LIS_surfaceModel_setup (Source File: LIS_surfaceModelMod.F90)

INTERFACE:

```
subroutine LIS_surfaceModel_setup
```

DESCRIPTION:

This routine sets up the surface models.

3.34.7 LIS_surfaceModel_readrestart (Source File: LIS_surfaceModelMod.F90)

INTERFACE:

```
subroutine LIS_surfaceModel_readrestart
```

DESCRIPTION:

This routine reads the restart data for each surface model.

3.34.8 LIS_surfaceModel_run (Source File: LIS_surfaceModelMod.F90)

INTERFACE:

```
subroutine LIS_surfaceModel_run(n)
```

ARGUMENTS:

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

DESCRIPTION:

This routine runs the surface models.
The arguments are:

n index of the nest

3.34.9 LIS_surfaceModel_f2t (Source File: LIS_surfaceModelMod.F90)

INTERFACE:

```
subroutine LIS_surfaceModel_f2t(n)
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers met forcing to the surface model tiles.
The arguments are:

n index of the nest

3.34.10 LIS_surfaceModel_output (Source File: LIS_surfaceModelMod.F90)

INTERFACE:

```
subroutine LIS_surfaceModel_output(n)
```

USES:

```
use LIS_timeMgrMod, only : LIS_isAlarmRinging
use LIS_histDataMod, only : LIS_histData
use LIS_historyMod,  only : LIS_writeModelOutput
use LIS_fileIOMod,  only : LIS_create_output_directory, &
     LIS_create_output_filename,  &
     LIS_create_stats_filename
```

ARGUMENTS:

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

DESCRIPTION:

This routine writes the surface model output.
The arguments are:

n index of the nest

3.34.11 LIS_surfaceModel_writerestart (Source File: LIS_surfaceModelMod.F90)

INTERFACE:

```
subroutine LIS_surfaceModel_writerestart(n)
```

ARGUMENTS:

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

DESCRIPTION:

This routine writes the surface model restart data.
The arguments are:

n index of the nest

3.34.12 LIS_surfaceModel_perturb_states (Source File: LIS_surfaceModelMod.F90)

INTERFACE:

```
subroutine LIS_surfaceModel_perturb_states(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

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

DESCRIPTION:

This routine perturbs the surface model states.
The arguments are:

n index of the nest

3.34.13 LIS_surfaceModel_diagnoseVarsforDA (Source File: LIS_surfaceModelMod.F90)

INTERFACE:

```
subroutine LIS_surfaceModel_diagnoseVarsforDA(n)
```

ARGUMENTS:

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

DESCRIPTION:

This routine diagnoses output for data assimilation.
The arguments are:

n index of the nest

3.34.14 LIS_surfaceModel_finalize (Source File: LIS_surfaceModelMod.F90)

INTERFACE:

```
subroutine LIS_surfaceModel_finalize
```

DESCRIPTION:

This routine cleans up the surface models.

3.34.15 LIS_surfaceModel_reset (Source File: LIS_surfaceModelMod.F90)

INTERFACE:

```
subroutine LIS_surfaceModel_reset
```

DESCRIPTION:

This routine resets the surface models.

3.34.16 surfaceModel_setexport_noesmf (Source File: LIS_surfaceModelMod.F90)

INTERFACE:

```
subroutine surfaceModel_setexport_noesmf(n)
```

USES:

```
use LIS_LMLCMod
use LISWRFGridCompMod, only : LISWRF_export
use LIS_historyMod, only : LIS_tile2grid
```

ARGUMENTS:

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

DESCRIPTION:

3.35 Fortran: Module Interface LIS_tbotAdjustMod (Source File: LIS_tbotAdjustMod.F90)

The code in this file adjusts the deep soil temperature based on options included in WRF.

3.35.1 Overview

Options exist to adjust the initial deep soil temperature to account for terrain influences, and to update the deep soil temperature as a function of lagged skin temperature. Also includes routines to write and read time lag information to and from a restart file.

LIS_tmnUpdateTileDec User defined type containing time lag information used for dynamically adjusting deep soil temperature as function of skin temperature for a single tile.

LIS_tmnUpdateDec User defined type containing arrays of type LIS_tmnUpdateTileDec, storing data for all tiles.

LIS_tmnUpdate Array of type LIS_tmnUpdate, storing data for all domains.

REVISION HISTORY:

```
3 June 2013: Eric Kemp; Initial implementation
```

3.35.2 LIS_createTmnUpdate (Source File: LIS_tbotAdjustMod.F90)

INTERFACE:

```
subroutine LIS_createTmnUpdate()
!USES
    use LIS_coreMod,   only : LIS_rc
    use LIS_logMod,    only : LIS_logunit, LIS_endrun
```

DESCRIPTION:

This routine allocates the LIS_tmnUpdate array and component data structures. Initial values of all member variables is zero.

3.35.3 LIS_initTmnUpdateTile (Source File: LIS_tbotAdjustMod.F90)

INTERFACE:

```
subroutine LIS_initTmnUpdateTile(n,t,tmn)
```

USES:

```
    use LIS_coreMod,   only : LIS_rc
```

ARGUMENTS:

```
    integer,intent(in) :: n ! Domain
    integer,intent(in) :: t ! Tile
    real,intent(in) :: tmn  ! Original deep soil temperature
```

DESCRIPTION:

Sets annual mean skin temperature for previous year and time series of daily mean skin temperatures to climatological values on tile by tile basis.

3.35.4 LIS_updateTbot (Source File: LIS_tbotAdjustMod.F90)

INTERFACE:

```
subroutine LIS_updateTbot(n,i,julian_in,yr,dt,tsk,tmn)
```

USES:

```
use LIS_coreMod,   only : LIS_rc
use LIS_logMod,    only : LIS_logunit
```

ARGUMENTS:

```
integer,intent(in) :: n      ! Domain
integer,intent(in) :: i      ! Tile in domain
real,intent(in) :: julian_in ! Day of year
integer,intent(in) :: yr     ! Current year
real,intent(in) :: dt        ! Time step
real,intent(in) :: tsk       ! Current skin temperature
real,intent(inout) :: tmn    ! Modified deep soil temperature
```

DESCRIPTION:

Dynamically adjusts deep soil temperature as weighted average of previous year's annual mean skin temperature and mean of time series of recent daily mean skin temperatures. The length of the time series is set by lagday. Based on WRF.

3.35.5 LIS_writeTmnUpdateRestart (Source File: LIS_tbotAdjustMod.F90)

INTERFACE:

```
subroutine LIS_writeTmnUpdateRestart(n,ftn)
```

USES:

```
use LIS_coreMod, only: LIS_rc, LIS_masterproc
use LIS_historyMod, only : LIS_writevar_restart
use LIS_logMod, only: LIS_logunit
```

ARGUMENTS:

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

DESCRIPTION:

Writes dynamic deep soil temperature time lag information to an already opened restart file. Assumes LSM data has already been read in.

3.35.6 LIS_readTmnUpdateRestart (Source File: LIS_tbotAdjustMod.F90)

INTERFACE:

```
subroutine LIS_readTmnUpdateRestart(n,ftn)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_historyMod, only : LIS_readvar_restart
use LIS_logMod,     only : LIS_logunit, LIS_endrun
```

ARGUMENTS:

```
integer,intent(in) :: n ! Domain
integer,intent(in) :: ftn
```

DESCRIPTION:

Reads in dynamic deep soil temperature time lag information from already opened restart file. Assumes LSM data has already been read in.

3.35.7 LIS_tbotTerrainAdjustment (Source File: LIS_tbotAdjustMod.F90)

REVISION HISTORY:

```
30 May 2013: Eric Kemp; Initial Code
```

INTERFACE:

```
subroutine LIS_tbotTerrainAdjustment(nx,ny,placetbot1,n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_logMod,     only : LIS_logunit, LIS_endrun
use LIS_topoMod,    only : LIS_topo
```

ARGUMENTS:

```
integer,intent(in) :: nx ! Global x-dimension
integer,intent(in) :: ny ! Global y-dimension
real,intent(inout) :: placetbot1(nx,ny) ! 2-D climatological deep soil
                                        ! temperature (K)
integer,intent(in) :: n ! Domain
```

DESCRIPTION:

This subroutine adjusts the input deep soil temperature for the Noah LSM. This original source for this is a time-averaged skin temperature climatology from ECMWF, which was then extrapolated to mean sea level using the standard atmosphere lapse rate (6.5 K / km). This process is reversed to adjust the temperatures to the LIS terrain. Based on WRF.

3.36 Fortran: Module Interface LIS_timeMgrMod (Source File: LIS_timeMgrMod.F90)

This module contains routines for time managment.The module provides routines for clock initialization, model timestepping, some basic alarm functions, and other useful time managment utilities.

USES:

```
use ESMF
use LIS_PRIV_rcMod
use LIS_logMod
```

3.36.1 LIS_isAlarmRinging (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
interface LIS_isAlarmRinging
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure isMonthlyAlarmRinging
module procedure isAlarmRinging
```

DESCRIPTION:

checks if the monthly alarm is ringing. The private functions have different arguments based on the input options specified.

3.36.2 LIS_timemgr_init (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_timemgr_init(LIS_rc)
```

USES:

```
use LIS_logMod,   only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(lisrcdec) :: LIS_rc
```

DESCRIPTION:

Initialize the LIS time manager.
NOTE - This assumes that the LIS time specific variables pertaining to start and end times have been set before this routine is called.

LIS_rc instance of the `lis_module`

The calling sequence is:

LIS_date2time (3.36.24)
to convert current date to a floating point format

LIS_timemgr_print (3.36.6)
display the contents of the time manager

3.36.3 LIS_resetClockForTimeWindow (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_resetClockForTimeWindow(LIS_rc)
```

USES:

```
implicit none
```

ARGUMENTS:

```
type(lisrcdec) :: LIS_rc
```

DESCRIPTION:

resets the time manager clock based on the time window (currently assumed to be a month). This routine is intended for the smoother DA run mode where at the end of each month the code sets the start time (of the time window) to be beginning of the previous month, the middle time to be the beginning of the (current) month and the stop time to be the the beginning of the next month. For e.g. if the routine is invoked on July 1st, then the twStartTime will be set to Jun 1, twMiddleTime would be July 1 and twStopTime would be Aug 1. The clock is also reset to Jun 1.

LIS_rc instance of the `lis_module`

The calling sequence is:

LIS_date2time (3.36.24)
convert date to a flotating point format

3.36.4 LIS_resetclock (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_resetclock(LIS_rc)
```

USES:

```
use LIS_logMod,  only : LIS_verify
implicit none
```

ARGUMENTS:

```
type(lisrcdec) :: LIS_rc
```

DESCRIPTION:

Resets the LIS time manager.
NOTE - This assumes that the LIS time specific variables pertaining to start and end times have been set before this routine is called.

LIS_rc instance of the `lis_module`

The calling sequence is:

LIS_date2time (3.36.24)
to convert current date to a floating point format

LIS_timemgr_print (3.36.6)
display the contents of the time manager

3.36.5 LIS_timemgr_set (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_timemgr_set(LIS_rc,yr,mo,da,hr,mn,ss,ms,ts)
```

USES:

```
implicit none
```

ARGUMENTS:

```
type(lisrcdec) :: LIS_rc
integer :: yr,mo,da,hr,mn,ss,ms
real    :: ts
```

DESCRIPTION:

sets the time manager clock based on the input time specification. This method is used to initialize the time mangager when the clock is passed down to LIS from a parent component

LIS_rc instance of the `lis_module`

yr year

mn month

da day of the month

hr hour of the day

mn minute of the hour

ss second

The calling sequence is:

LIS_date2time (3.36.24)
convert date to a flotating point format

3.36.6 LIS_timemgr_print (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_timemgr_print(LIS_rc)
```

3.36.7 LIS_advance_timestep (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_advance_timestep(LIS_rc)
```

USES:

```
implicit none
```

ARGUMENTS:

```
type(lisrcdec) :: LIS_rc
```

DESCRIPTION:

Increments time manager clock by the model timestep. In case of LIS running multiple nests, each running at different timesteps, the smallest timestep is chosen to advance the clock.

LIS_rc instance of the `lis_module`

The calling sequence is:

LIS_date2time (3.36.24)
convert date to a flotating point format

3.36.8 LIS_update_timestep (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_update_timestep(LIS_rc, n, ts)

  implicit none
```

ARGUMENTS:

```
  type(lisrcdec) :: LIS_rc
  integer        :: n
  real           :: ts
```

3.36.9 LIS_update_clock (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_update_clock(ts)
```

USES:

```
  use LIS_logMod,   only : LIS_verify

  implicit none
```

ARGUMENTS:

```
  real       :: ts
```

3.36.10 LIS_get_step_size (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
function LIS_get_step_size(LIS_rc)

  implicit none
```

ARGUMENTS:

```
  type(lisrcdec) :: LIS_rc
  integer :: LIS_get_step_size
```

DESCRIPTION:

Return the timestep used by the clock in the time manager.

LIS_rc instance of the `lis_module`

LIS_get_step_size timestep value

3.36.11 LIS_get_nstep (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
function LIS_get_nstep(LIS_rc,n)

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: n
  type(lisrcdec) :: LIS_rc
  integer :: LIS_get_nstep
```

DESCRIPTION:

Return the timestep number for each nest.

LIS_rc instance of the `lis_module`

n index of the nest

LIS_get_nstep timestep number

3.36.12 LIS_get_curr_day (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_get_curr_date(LIS_rc, yr, mon, day, tod, offset)

  implicit none
```

ARGUMENTS:

```
  type(lisrcdec) :: LIS_rc
  integer, intent(out) ::yr,mon,day,tod
  integer, optional, intent(in) :: offset
```

DESCRIPTION:

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

LIS_rc instance of the `lis_module`

yr year

mon month

day day of the month

tod time of day (seconds past 0Z)

offset offset from current time in seconds positive for future times, negative for previous times.

3.36.13 LIS_get_julhr (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_get_julhr(yr,mo,da,hr,mn,ss,julhr)

  implicit none
```

ARGUMENTS:

```
  integer, intent(in)     :: yr
  integer, intent(in)     :: mo
  integer, intent(in)     :: da
  integer, intent(in)     :: hr
  integer, intent(in)     :: mn
  integer, intent(in)     :: ss
  integer                 :: julhr
```

DESCRIPTION:

Returns the julian hour. In this convention, julian day began at midnight at the beginning of May 24, 1968. Interestingly, this convention was introduced by NASA for the space program.
The arguments are:

yr year

mo month

da day of the month

hr hour of day

mn minute

ss second

julhr julian hour

3.36.14 LIS_get_julss (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_get_julss(yr,mo,da,hr,mn,ss,julss)

  implicit none
```

ARGUMENTS:

```
  integer, intent(in)     :: yr
  integer, intent(in)     :: mo
  integer, intent(in)     :: da
  integer, intent(in)     :: hr
  integer, intent(in)     :: mn
  integer, intent(in)     :: ss
  integer                 :: julss
```

DESCRIPTION:

Returns the julian hour. In this convention, julian day began at midnight at the beginning of May 24, 1968. Interestingly, this convention was introduced by NASA for the space program.
The arguments are:

yr year

mo month

da day of the month

hr hour of day

mn minute

ss second

julss julian hour

3.36.15 LIS_get_curr_calday (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
function LIS_get_curr_calday(LIS_rc,offset)

 implicit none
```

ARGUMENTS:

```
type(lisrcdec) :: LIS_rc
integer, optional, intent(in) :: offset
real :: LIS_get_curr_calday
```

DESCRIPTION:

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

LIS_rc instance of the `lis_module`

offset offset from current time in seconds, positive for future times, negative for previous times.

LIS_get_curr_calday calendar day value

3.36.16 LIS_is_last_step (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
function LIS_is_last_step(LIS_rc)
   implicit none
```

ARGUMENTS:

```
type(lisrcdec) :: LIS_rc
logical :: LIS_is_last_step
```

DESCRIPTION:

Function returns true on last timestep.

LIS_rc instance of the `lis_module`

LIS_is_last_step result of the function

3.36.17 isMonthlyAlarmRinging (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
!Private name: call using isAlarmRinging
 function isMonthlyAlarmRinging(LIS_rc, alarmTime, intervalType, &
     midmonth)

  implicit none
```

ARGUMENTS:

```
  type(lisrcdec), intent(in)   :: LIS_rc
  real*8, intent(inout)        :: alarmTime
  character(len=*), intent(in) :: intervalType
  logical, intent(in)          :: midmonth
  logical                      :: isMonthlyAlarmRinging
```

DESCRIPTION:

checks if the monthly alarm is ringing. The function returns true when the elapsed alarm time is greater than the number of months specified in the alarm's interval. If the midmonth flag is enabled, the elapsed time is counted from the middle of the month to middle of another month, rather than from the beginning of the month to the end of another month.
The arguments are:

LIS_rc instance of the `lis_module`

alarmTime the elapsed alarm time

intervalType the alarm's frequency

midmonth flag to indicate if the elapsed time is to be counted from the beginning or the middle of the month

ringflag flag indicating the status of the call

The calling sequence is:

LIS_date2time (3.36.24)
convert date to a flotating point format

3.36.18 LIS_getDekad (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
function LIS_getDekad(LIS_rc, offset, which)

   implicit none
```

ARGUMENTS:

```
   type(lisrcdec), intent(in)             :: LIS_rc
   integer, intent(in), optional          :: offset
   character(len=*), intent(in), optional :: which
```

DESCRIPTION:

This function returns the dekad of the month

```
   dekad 1 : 1  <= day <  11
   dekad 2 : 11 <= day <  21
   dekad 3 :       day >= 21
```

The arguments are:

LIS_rc instance of the `lis_module`

offset offset in seconds to add to beginning and ending of dekad ranges

which specifies which to dekad to report. A value of "next" will report
the dekad corresponding to the next LIS time-step (LIS_rcAll other values will report the dekad corresponding
to the current time-step.

3.36.19 LIS_isDekadalAlarmRinging (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
function LIS_isDekadalAlarmRinging(LIS_rc, current)

   implicit none
```

ARGUMENTS:

```
   type(lisrcdec), intent(in)               :: LIS_rc
   type(lisalarmEntry), pointer, intent(in) :: current
```

DESCRIPTION:

This routine checks whether the given dekadal alarm, represented by current, is ringing.
The routine returns True when ref_dekad stored in current differs from the dekad returned by LIS_getDekad.
The arguments are:

LIS_rc instance of the `lis_module`

current representation of the current LIS dekadal alarm

3.36.20 LIS_getSecsInDekad (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
function LIS_getSecsInDekad(LIS_rc, dekad)

    implicit none
```

ARGUMENTS:

```
    type(lisrcdec), intent(in) :: LIS_rc
    integer, intent(in) :: dekad
```

DESCRIPTION:

This function returns the number of seconds in the given dekad.
The arguments are:

LIS_rc instance of the `lis_module`

dekad dekad number: 1, 2, or 3

3.36.21 LIS_finishDekadalAlarms (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
   subroutine LIS_finishDekadalAlarms(LIS_rc)
```

USES:

```
     use LIS_logMod,  only : LIS_verify
     implicit none
```

ARGUMENTS:

```
     type(lisrcdec) :: LIS_rc
```

DESCRIPTION:

Completes the initialization of dekadal-based alarms.

LIS_rc instance of the `lis_module`

3.36.22 LIS_computeTimeBookEnds (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
   subroutine LIS_computeTimeBookEnds(LIS_rc,interval,time1,time2)
     implicit none
```

ARGUMENTS:

```
type(lisrcdec), intent(in) :: LIS_rc
integer, intent(in) :: interval
real*8, intent(inout) :: time1,time2
```

DESCRIPTION:

compute the time end points based on the interval. If the interval is 3 hours, and the current time is 1:30Z, time1 will be set to 1Z and time2 will be set to 4Z.
The arguments are:

LIS_rc instance of the `lis_module`

interval time interval in hours

time1 previous time

time2 next time

The calling sequence is:

LIS_tick (3.36.27)
advance the time by the specified increment

3.36.23 LIS_computeTemporalWeights (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_computeTemporalWeights(LIS_rc,intervalType,time1,time2,&
     wt1,wt2, alarmName)

  implicit none
```

ARGUMENTS:

```
type(lisrcdec)   :: LIS_rc
character(len=*) :: intervalType
integer          :: time1, time2
real             :: wt1,wt2
character(len=*), optional :: alarmName
```

DESCRIPTION:

This routine computes the time interpolation weights based on the climatology interval, current time, and the previous and next month. For example, a monthly climatology data to the current date can be interpolated as value = value1 *wt1+valu2 *wt2 where value1 is the value from the previous month's climatology, and value2 is the value from the next month's climatology and value is the interpolated value.
The arguments are:

LIS_rc instance of the `lis_module`

interval interval type for the climatology

mo1 previous month

mo2 next month

wt1, wt2 interpolation weights to be used on the climatology data to interpolate to the current day.

The calling sequence is:

LIS_date2time (3.36.24)
converts date to a floating point format

3.36.24 LIS_date2time (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_date2time(time,doy,gmt,yr,mo,da,hr,mn,ss)

  implicit none
```

ARGUMENTS:

```
  integer :: yr,mo,da,hr,mn,ss, doy
  real*8  :: time
  real    :: gmt
```

DESCRIPTION:

determines the time, time in GMT, and the day of the year based on the value of year, month, day of month, hour of the day, minute and second. This method is the inverse of time2date
NOTE: This routine has been known to give round off error problems when attempting to retrieving minutes and seconds from the given number. Use at your own risk!
The arguments are:

yr year

mo month

da day of the month

hr hour of day

mn minute

ss second

time lis time

gmt time in GMT

doy day of the year

3.36.25 LIS_time2date (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_time2date(time,doy,gmt,yr,mo,da,hr,mn)

  implicit none
```

ARGUMENTS:

```
integer :: yr,mo,da,hr,mn,ss,doy
real*8  :: time
real    :: gmt
```

DESCRIPTION:

determines the value of the year, month, day of month, hour of the day, minute and second based on the specified time. This method is the inverse of date2time
The arguments are:

yr year

mo month

da day of the month

hr hour of day

mn minute

ss second

time lis time

gmt time in GMT

doy day of the year

3.36.26 LIS_doy2date (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_doy2date(yr, doy, mo, da)

  implicit none
```

DESCRIPTION:

This subroutine converts a given julian day to month and days

---

3.36.27 LIS_tick (Source File: LIS_timeMgrMod.F90)

INTERFACE:

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

ARGUMENTS:

```
real*8  :: time
integer :: yr,mo,da,hr,mn,ss,doy
real    :: ts,gmt
```

DESCRIPTION:

Method to advance or retract the time by the specified time increment
The arguments are:

yr year

mo month

da day of the month

hr hour of day

mn minute

ss second

ts time increment in seconds

time lis time

gmt time in GMT

doy day of the year

3.36.28 LIS_julhr_date (Source File: LIS_timeMgrMod.F90)

REVISION HISTORY:

```
15 oct 1998  initial version.......................mr moore/dnxm
10 aug 1999  ported to ibm sp2.  added intent attributes to
             arguments.............................mr gayno/dnxm
29 oct 2005  Sujay Kumar, Adopted in LIS
```

INTERFACE:

```
subroutine LIS_julhr_date( julhr, yyyy,mm,dd,hh)
```

USES:

```
use LIS_logMod, only : lis_abort

implicit none
```

ARGUMENTS:

```
integer,          intent(in)   :: julhr
integer                        :: yyyy
integer                        :: mm
integer                        :: dd
integer                        :: hh
```

DESCRIPTION:

to convert from a julian hour to a 10 digit date/time group (yyyymmddhh)

3.36.29 Method

: call utility routine LIS_tmjul4 to convert from julian hours to year, month, day, hour.
perform several checks to ensure date information passed back from LIS_tmjul4 is valid.
if date is good, convert from integer data to a 10 digit date/time group and pass back to calling routine.

The arguments are:

julhr input julian hour

yyyy four digit year

mm month of the year

dd day of the month

hh time of the day in hours

The calling sequence is:

LIS_tmjul4 (3.36.30)
convert julian hour to hour, day, month, and year

LIS_abort (3.21.4)
abort the code in case of error

3.36.30 LIS_tmjul4 (Source File: LIS_timeMgrMod.F90)

REVISION HISTORY:

```
18 mar 98 initial version.........................sra milburn/dnxm
08 jul 99 fixed error which initialzed done flag in a data
          data statement.  variables must be initialized
          using an assignment statement.  ported to ibm sp2.......
          ..........................................mr gayno/dnxm
29 oct 2005  Sujay Kumar, Adopted in LIS
```

INTERFACE:

```
subroutine LIS_tmjul4( hour, day, month, year, julhr )

  implicit none
```

ARGUMENTS:

```
  integer,  intent(out)       :: day
  integer,  intent(out)       :: hour
  integer,  intent(in)        :: julhr
  integer,  intent(out)       :: month
  integer,  intent(out)       :: year
```

DESCRIPTION:

uses the julian hour to determine the 2-digit zulu time, day, month, and 4-digit year.

3.36.31 Method

- determine the current zulu hour
- determine the total number of elapsed days
- count forward to the current day/month/year

The arguments are:

hour the zulu time of the julian hour

day day of the month (1..31)

month month of the year (1..12)

year four digit year

julhr the julian hour being processed

3.36.32 LIS_localtime2gmt (Source File: LIS_timeMgrMod.F90)

Calculates GMT based on the local time

INTERFACE:

```
subroutine LIS_localtime2gmt (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)
```

3.36.33 LIS_registerAlarm (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_registerAlarm(name, ts, interval, intervalType, alarm_offset, &
                             dek_offset, when)
```

USES:

```
use LIS_logMod, only : LIS_logunit

implicit none
```

ARGUMENTS:

```
character(len=*)           :: name
real                       :: ts
real                       :: interval
character(len=*), optional :: intervalType
integer, optional          :: alarm_offset
integer, optional          :: dek_offset
character(len=*), optional :: when

type(lisalarmEntry), pointer :: alrmEntry, current
integer                      :: yr,mo,da,hr,mn,ss,rc
type(ESMF_Time)              :: currTime
type(ESMF_TimeInterval)      :: deltaT
integer                      :: refjuld, sjuld
```

DESCRIPTION:

name name of the alarm

interval frequency of the alarm in seconds

intervalType type of the alarm for non-interval-based alarms. Acceptable values are: "monthly", "quarterly", or "dekad"

alarm_offset offset in seconds to add to beginning of alarm

dek_offset (for dekadal alarms only) offset in seconds to add to beginning and ending of dekad ranges

when (for dekadal alarms only) specifies when the alarm should ring; a value of "begin" specifies that the alarm should ring at the beginning of the dekad; a value of "end" specifies that the alarm should ring at the end of the dekad.

3.36.34 isAlarmRinging (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
function isAlarmRinging(LIS_rc, model_TS, alarmInterval)
function isAlarmRinging(LIS_rc, alarmName, intervaltype)
```

USES:

```
implicit none
```

ARGUMENTS:

```
type(lisrcdec)             :: LIS_rc
character(len=*)           :: alarmName
character(len=*), optional :: intervaltype
logical                    :: isAlarmRinging
```

DESCRIPTION:

checks if the alarm is ringing. Currently hourly, daily, and monthly alarms are supported.
The arguments are:

LIS_rc instance of the `lis_module`

alarmInterval alarmInterval in seconds

model_TS timestep of the model

isAlarmRinging flag indicating the status of the call

3.36.35 LIS_parseTimeString (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_parseTimeString(inputstr,time_value)

  implicit none
```

ARGUMENTS:

```
  character(len=*)          :: inputstr
  real                      :: time_value
```

DESCRIPTION:

This routine parses the input time string and converts it to an integer value, which represents the time in seconds. The input time string should have 2-character suffixes of ss (seconds), mn (minutes), hr (hours), da (days), mo (month) or yr (years).

3.37 Fortran: Module Interface LIS_topoMod (Source File: LIS_topoMod.F90)

The code in this file implements routines to read various sources of topography data

3.37.1 Overview

This module provides routines for reading and modifying topography data.

REVISION HISTORY:

```
 18 Jul 2008: Sujay Kumar; Initial implementation
 3  Apr 2012: Sujay Kumar; Switched to the use of LPT based parameter file
use LIS_fileIOMod
implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
 public :: LIS_topo_init  ! initializes data structures and memory
 public :: LIS_diagnosetopography
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
 public :: LIS_topo
```

3.37.2 LIS_topo_init (Source File: LIS_topoMod.F90)

INTERFACE:

```
subroutine LIS_topo_init()
```

USES:

```
use ESMF
use LIS_coreMod,  only : LIS_rc, LIS_config
use LIS_fileIOMod, only : LIS_readDomainConfigSpecs
```

DESCRIPTION:

Allocates memory for data structures for reading topo datasets

---

3.37.3 LIS_diagnoseTopography (Source File: LIS_topoMod.F90)

INTERFACE:

```
subroutine LIS_diagnoseTopography(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_histDataMod
```

ARGUMENTS:

```
implicit none
integer, intent(in)   :: n
```

DESCRIPTION:

This routine writes the LIS topography to history writer
The arguments are:

n index of the nest

The routines called are:

LIS_diagnoseOutputVar 3.16.5

3.37.4 read_elevation (Source File: LIS_topoMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_elevation(n)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
    use netcdf
#endif
    use LIS_coreMod,        only : LIS_rc, LIS_localPet,&
         LIS_ews_ind, LIS_ewe_ind,&
         LIS_nss_ind, LIS_nse_ind, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
         LIS_nss_halo_ind, LIS_nse_halo_ind
    use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
         LIS_releaseUnitNumber, LIS_endrun, LIS_verify

    implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine reads the elevation data
The arguments are:

n index of n

locallc landlc for the region of interest

3.37.5 read_slope (Source File: LIS_topoMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
   subroutine read_slope(n)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
    use netcdf
#endif
    use LIS_coreMod,        only : LIS_rc, LIS_localPet,&
         LIS_ews_ind, LIS_ewe_ind,&
         LIS_nss_ind, LIS_nse_ind, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
         LIS_nss_halo_ind, LIS_nse_halo_ind
    use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
         LIS_releaseUnitNumber, LIS_endrun, LIS_verify

    implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine reads the slope data
The arguments are:

n index of n

locallc landlc for the region of interest

3.37.6 read_aspect (Source File: LIS_topoMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_aspect(n)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
    use netcdf
#endif
    use LIS_coreMod,        only : LIS_rc, LIS_localPet,&
         LIS_ews_ind, LIS_ewe_ind,&
         LIS_nss_ind, LIS_nse_ind, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
         LIS_nss_halo_ind, LIS_nse_halo_ind
    use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
         LIS_releaseUnitNumber, LIS_endrun, LIS_verify

    implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine reads the aspect data
The arguments are:

n index of n

locallc landlc for the region of interest

3.38 Fortran: Module Interface LIS_vegDataMod (Source File: LIS_vegDataMod.F90)

The code in this file implements routines to read various vegetation datasets.

3.38.1 Overview

This routines in this module provides routines to read the greenness fraction, LAI, SAI and roughness data. Both real-time and climatological datasets are supported. The climatological data is expected to be provided in the parameter input file (from LPT)

REVISION HISTORY:

```
    08 Aug 2005: Sujay Kumar; Initial implementation
    10 Aug 2010: Jonathan Case: Modified for SPORT daily GVF data.
   use ESMF
   use LIS_precisionMod
   use LIS_coreMod
   use LIS_logMod
   use LIS_timeMgrMod
   use LIS_histDataMod
   use LIS_fileIOMod
 #if(defined USE_NETCDF3 || defined USE_NETCDF4)
   use netcdf
 #endif

   implicit none

   PRIVATE

  ------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
  ------------------------------------------------------------------------------
   public :: LIS_greenness_setup    !allocates memory for required structures
   public :: LIS_read_greenness     !reads the greenness data
   public :: LIS_diagnosegfrac      !diagnoses greenness data for history output
   public :: LIS_greenness_finalize !cleanup allocated structures
   public :: LIS_greenness_reset    !resets alarms and data structures
   public :: LIS_read_shdmax        !read the shdmax map
   public :: LIS_read_shdmin        !read the shdmin map

   public :: LIS_roughness_setup   !allocates memory for required structures
   public :: LIS_read_roughness    !reads the roughness data
   public :: LIS_diagnoseroughness !maps the roughness data to the history writer
   public :: LIS_roughness_finalize !cleanup allocated structures
   public :: LIS_roughness_reset    !resets datastructures

   public :: LIS_lai_setup ! allocates memory for required structures
   public :: LIS_read_lai  ! reads the LAI data
   public :: LIS_lai_finalize ! cleanup allocated structures
   public :: LIS_diagnoseLAI !maps LAI data to the history writer
   public :: LIS_lai_reset  !resets datastructures
   public :: LIS_read_laimax        !read the laimax map
   public :: LIS_read_laimin        !read the laimin map

   public :: LIS_sai_setup ! allocates memory for required structures
   public :: LIS_read_sai  ! reads the SAI data
   public :: LIS_diagnoseSAI !maps the SAI data to the LIS history writer
   public :: LIS_sai_finalize ! cleanup allocated structures
   public :: LIS_sai_reset  !resets the sai datastructures and alarams
```

```
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
 public :: LIS_gfrac !data structure containing greenness fraction data.
 public :: LIS_roughness !data structure containing roughness data.
 public :: LIS_lai ! data structure containing LAI data.
 public :: LIS_sai ! data structure containing SAI data.
```

3.38.2 LIS_greenness_setup (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_greenness_setup
```

USES:

DESCRIPTION:

Allocates memory for data structures for reading the greenness fraction datasets. This routine also reads the greenness datasets as initial values.
The routines invoked are:

gfracsetup (3.45.2)
calls the registry to invoke the gfrac setup methods.

LIS_registerAlarm (3.36.33)
registers the alarm for reading greenness datasets

LIS_computeTemporalWeights (3.36.23)
computes the interpolation weights

read_gfracclimo (3.38.7)
reads the greenness climatology data from the LIS parameter data file.

readgfrac (3.45.4)
invokes the method from the registry to read the real-time (time-varying) greenness data.

3.38.3 LIS_read_greenness (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_read_greenness(n)
```

USES:

```
implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Reads the greenness fraction climalotogy and temporally interpolates it to the current day.
The arguments are:

n index of the domain or nest.

The routines invoked are:

LIS_computeTemporalWeights (3.36.23)
computes the interpolation weights

read_gfracclimo (3.38.7)
reads the greenness climatology data from the LIS parameter data file.

readgfrac (3.45.4)
invokes the method from the registry to read the real-time (time-varying) greenness data.

3.38.4 LIS_greenness_finalize (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_greenness_finalize
```

USES:

DESCRIPTION:

Deallocates objects created in this module

3.38.5 LIS_diagnosegfrac (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_diagnosegfrac(n)
```

USES:

ARGUMENTS:

```
implicit none
integer, intent(in)   :: n
```

DESCRIPTION:

This routine maps the greenness data to the LIS history writer.
The arguments are:

n index of the nest

The routines called are:

LIS_diagnoseOutputVar (3.16.5)
maps the greenness data to the LIS history writer

3.38.6 LIS_greenness_reset (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_greenness_reset
```

USES:

DESCRIPTION:

Resets the data structures for reading the greenness fraction datasets
The routines invoked are:

gfracsetup (3.45.2)
calls the registry to invoke the gfrac data reading methods.

LIS_computeTemporalWeights (3.36.23)
computes the interpolation weights

read_gfracclimo (3.38.7)
reads the greenness climatology data from the LIS parameter data file.

readgfrac (3.45.4)
invokes the method from the registry to read the real-time (time-varying) greenness data.

3.38.7 read_gfracclimo (Source File: LIS_vegDataMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_gfracclimo(n,time,array)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)         :: n
integer, intent(in)         :: time
real, intent(inout)         :: array(LIS_rc%ntiles(n))
```

DESCRIPTION:

This subroutine reads the greenness data climatology from the LIS parameter data file
The arguments are:

n index of n

time time index (in months or quarters) of the data to be read

array array containing the greenness values

3.38.8 LIS_read_shdmin (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_read_shdmin(n,array)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real,    intent(inout) :: array(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

Reads the static albedo upper bound over deep snow for each domain.
The arguments are:

n index of n

array shdmin for the region of interest

3.38.9 LIS_read_shdmax (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_read_shdmax(n,array)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real,    intent(inout) :: array(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

Reads the static albedo upper bound over deep snow for each domain.
The arguments are:

n index of n

array shdmax for the region of interest

3.38.10 LIS_read_laimin (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_read_laimin(n,array)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real,    intent(inout) :: array(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

Reads the static albedo upper bound over deep snow for each domain.
The arguments are:

n index of n

array laimin for the region of interest

3.38.11 LIS_read_laimax (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_read_laimax(n,array)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real,    intent(inout) :: array(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

Reads the static albedo upper bound over deep snow for each domain.
The arguments are:

n index of n

array laimax for the region of interest

3.38.12 LIS_roughness_setup (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_roughness_setup
```

USES:

DESCRIPTION:

Allocates memory for data structures for reading the roughness fraction datasets
The routines invoked are:

gfracsetup (3.45.2)
calls the registry to invoke the gfrac setup methods.

3.38.13 LIS_read_roughness (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_read_roughness(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_timeMgrMod, only : LIS_isAlarmRinging, LIS_computeTemporalWeights

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Reads the roughness fraction climalotogy and temporally interpolates it to the current day.
The arguments are:

n index of the domain or nest.

The routines invoked are:

LIS_computeTemporalWeights (3.36.23)
computes the interpolation weights

readroughness (3.55.4)
invokes the generic method in the registry to read the roughness climatology data

---

3.38.14 LIS_roughness_finalize (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_roughness_finalize
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

DESCRIPTION:

Deallocates objects created in this module

3.38.15 LIS_diagnoseroughness (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_diagnoseroughness(n)
```

USES:

```
use LIS_coreMod,     only : LIS_rc, LIS_domain
use LIS_histDataMod, only : LIS_diagnoseSurfaceOutputVar, LIS_MOC_ROUGHNESS
```

ARGUMENTS:

```
implicit none
integer, intent(in)   :: n
```

DESCRIPTION:

This routine maps the roughness data to the LIS history writer.
The arguments are:

n index of the nest

The routines called are:

LIS_diagnoseOutputVar (3.16.5)
generic routine to map a single variable to the LIS history writer

---

3.38.16 LIS_roughness_reset (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_roughness_reset
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_config
use LIS_logMod,     only : LIS_verify
use LIS_timeMgrMod, only : LIS_computeTemporalWeights
```

DESCRIPTION:

Resets the data structures for reading the roughness fraction datasets
The routines invoked are:

roughnesssetup (3.55.2)
calls the registry to invoke the roughness data reading methods.

3.38.17 read_roughnessclimo (Source File: LIS_vegDataMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_roughnessclimo(n,time,array)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain, LIS_localPet,&
     LIS_ews_ind, LIS_ewe_ind,&
     LIS_nss_ind, LIS_nse_ind, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
     LIS_nss_halo_ind, LIS_nse_halo_ind
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in)         :: n
integer, intent(in)         :: time
real, intent(inout)         :: array(LIS_rc%ntiles(n))
```

DESCRIPTION:

This subroutine reads the greenness data climatology
The arguments are:

n index of n

3.38.18 LIS_lai_setup (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_lai_setup
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_Config, LIS_domain
use LIS_timeMgrMod, only : LIS_computeTemporalWeights, &
     LIS_registerAlarm, LIS_calendar
use LIS_logMod,     only : LIS_verify, LIS_logunit
```

DESCRIPTION:

Allocates memory and other structures for reading LAI datasets
The routines invoked are:

LIS_registerAlarm (3.36.33)
registers the alarm for reading LAI datasets

LIS_computeTemporalWeights (3.36.23)
computes the interpolation weights

read_laiclimo (3.38.30)
reads the climatological lai data

laisetup (3.47.2)
initializes the realtime lai reader

readlai (3.47.6)
reads the realtime lai data

3.38.19 LIS_read_lai (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_read_lai(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_timeMgrMod, only : LIS_isAlarmRinging,LIS_computeTemporalWeights, LIS_calendar
use LIS_logMod,     only : LIS_verify, LIS_logunit

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Reads the LAI climatology and temporally interpolates it to the current day.
The arguments are:

n index of the domain or nest.

The routines invoked are:

readlai (3.47.6)
invokes the generic method in the registry to read the LAI climatology data

---

3.38.20 LIS_lai_setup (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_lai_setup
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_Config, LIS_domain
use LIS_timeMgrMod, only : LIS_computeTemporalWeights, &
     LIS_registerAlarm
use LIS_logMod,     only : LIS_verify, LIS_logunit
```

DESCRIPTION:

Allocates memory and other structures for reading LAI datasets
The routines invoked are:

laisetup (3.47.2)
initializes the realtime lai reader

LIS_registerAlarm (3.36.33)
registers the alarm for reading LAI datasets

LIS_computeTemporalWeights (3.36.23)
computes the interpolation weights

read_LAIclimo (3.38.30)
reads the climatological lai data

readlai (3.47.6)
reads the realtime lai data

3.38.21 LIS_read_lai (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_read_lai(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_timeMgrMod, only : LIS_isAlarmRinging,LIS_computeTemporalWeights
use LIS_logMod,     only : LIS_verify, LIS_logunit

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Reads the LAI climatology and temporally interpolates it to the current day.
The arguments are:

n index of the domain or nest.

The routines invoked are:

readlai (3.47.6)
invokes the generic method in the registry to read the LAI climatology data

---

3.38.22 LIS_lai_finalize (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_lai_finalize
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

DESCRIPTION:

Deallocates objects created in this module

3.38.23 LIS_diagnoseLAI (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_diagnoseLAI(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_histDataMod, only : LIS_diagnoseSurfaceOutputVar, LIS_MOC_LAI
```

ARGUMENTS:

```
implicit none
integer, intent(in)   :: n
```

DESCRIPTION:

This routine writes the LIS LAI to the parameter output file.
The arguments are:

n index of the nest

ftn file unit number to be used

The routines called are:

LIS_diagnoseOutputVar (3.16.5)
This routine maps a variable to the history writing routines

3.38.24 LIS_lai_reset (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_lai_reset
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_Config, LIS_domain
use LIS_timeMgrMod, only : LIS_computeTemporalWeights, &
     LIS_registerAlarm, LIS_calendar
use LIS_logMod,     only : LIS_verify, LIS_logunit
```

DESCRIPTION:

Resets data structures for reading LAI datasets
The routines invoked are:

laisetup (3.47.2)
initializes the realtime lai reader

LIS_registerAlarm (3.36.33)
registers the alarm for reading LAI datasets

LIS_computeTemporalWeights (3.36.23)
computes the interpolation weights

read_LAIclimo (3.38.30)
reads the climatological lai data

readlai (3.47.6)
reads the realtime lai data

3.38.25 LIS_sai_setup (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_sai_setup
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_config
use LIS_timeMgrMod, only : LIS_computeTemporalWeights, &
     LIS_registerAlarm
use LIS_logMod,     only : LIS_verify
```

DESCRIPTION:

Allocates memory and other structures for reading SAI datasets
The routines invoked are:

saisetup (3.47.4)
initializes the realtime sai reader

LIS_registerAlarm (3.36.33)
registers the alarm for reading SAI datasets

LIS_computeTemporalWeights (3.36.23)
computes the interpolation weights

read_saiclimo (3.38.31)
reads the climatological sai data

readsai (3.47.8)
reads the realtime sai data

3.38.26 LIS_read_sai (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_read_sai(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_timeMgrMod, only : LIS_computeTemporalWeights, &
     LIS_isAlarmRinging

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Reads the SAI climatology and temporally interpolates it to the current day.
The arguments are:

n index of the domain or nest.

The routines invoked are:

readsai (3.47.8)
invokes the generic method in the registry to read the SAI climatology data

---

3.38.27 LIS_sai_finalize (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_sai_finalize
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

DESCRIPTION:

Deallocates objects created in this module

3.38.28 LIS_diagnoseSAI (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_diagnoseSAI(n)
```

USES:

```
use LIS_coreMod,     only : LIS_rc, LIS_domain
use LIS_histDataMod, only : LIS_diagnoseSurfaceOutputVar, LIS_MOC_SAI
```

ARGUMENTS:

```
implicit none
integer, intent(in)   :: n
```

DESCRIPTION:

This routine diagnoses the LIS SAI to the parameter output file.
The arguments are:

n index of the nest

ftn file unit number to be used

The routines called are:

LIS_diagnoseOutputVar (3.16.5)
This routine maps a variable to the history writing routines

3.38.29 LIS_sai_reset (Source File: LIS_vegDataMod.F90)

INTERFACE:

```
subroutine LIS_sai_reset
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_computeTemporalWeights, &
     LIS_Clock
use LIS_logMod,     only : LIS_verify
```

DESCRIPTION:

resets the structures that hold SAI datasets
The routines invoked are:

saisetup (3.47.4)
initializes the realtime sai reader

LIS_computeTemporalWeights (3.36.23)
computes the interpolation weights

read_saiclimo (3.38.31)
reads the climatological sai data

readsai (3.47.8)
reads the realtime sai data

3.38.30 read_laiclimo (Source File: LIS_vegDataMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_laiclimo(n,time,array)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain, LIS_localPet,&
     LIS_ews_ind, LIS_ewe_ind,&
     LIS_nss_ind, LIS_nse_ind, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
     LIS_nss_halo_ind, LIS_nse_halo_ind
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in)         :: n
integer, intent(in)         :: time
real, intent(inout)         :: array(LIS_rc%ntiles(n))
```

DESCRIPTION:

This subroutine reads the lai data climatology
The arguments are:

n index of n

3.38.31 read_saiclimo (Source File: LIS_vegDataMod.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_saiclimo(n,time,array)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain, LIS_localPet,&
     LIS_ews_ind, LIS_ewe_ind,&
     LIS_nss_ind, LIS_nse_ind, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
     LIS_nss_halo_ind, LIS_nse_halo_ind
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in)         :: n
integer, intent(in)         :: time
real, intent(inout)         :: array(LIS_rc%ntiles(n))
```

DESCRIPTION:

This subroutine reads the greenness data climatology
The arguments are:

n index of n

3.38.32 ft_check_index (Source File: FTable.c)

INTERFACE:

```
void ft_check_index(int index, int max, char * regrtn)
```

DESCRIPTION:

When registering a plugin into a function pointer table, one must specify where (the index) into the table to place this plugin. This routine checks the index into the function pointer table against the table's maximum value.
The purpose of this check is to avoid overwriting a function pointer table and causing a memory error.
This routines exits with a -1 if the index fails the test.
The arguments are:

index Position into the function pointer table

max Maximum value of index

regrtn String containing the name of the register routine. Used for printing an error message.

3.39 Fortran: Module Interface LIS_RTM_FTable (Source File: LIS_RTM_FTable.c)

Function table registries for storing the interface implementations for managing the operations of different radiative transfer models

---

3.39.1 registerrtminit (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(registerrtminit)(char *j, void (*func)(int*), int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to initialize the radiative transfer model
The arguments are:

j name of the RTM .

3.39.2 rtminit (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(rtminitialize)(char *j, int *index, int len)
```

DESCRIPTION:

Invokes the routine from the registry to initialize the RTM
The arguments are:

j name of the RTM

3.39.3 registerrtmf2t (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(registerrtmf2t)(char *j, void (*func)(int*), int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to transfer atmospheric profiles to the RTM data structure
The arguments are:

j name of the RTM .

3.39.4 rtmf2t (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(rtmf2t)(char *j, int *index, int len)
```

DESCRIPTION:

Invokes the routine from the registry to transfer atmospheric profiles to the RTM data structure
The arguments are:

j name of the RTM

3.39.5 registergeometry2rtm (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(registergeometry2rtm)(char *j, void (*func)(int*), int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to specify the sensor geometries in the RTM
The arguments are:

j name of the RTM .

3.39.6 geometry2rtm (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(geometry2rtm)(char *j, int *index, int len)
```

DESCRIPTION:

Invokes the routine from the registry to map sensor geometry information to the RTM data structure
The arguments are:

j name of the RTM

3.39.7 registerrtmrun (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(registerrtmrun)(char *j, void (*func)(int*), int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to specify the forward model integration step of the RTM
The arguments are:

j name of the RTM .

3.39.8 rtmrun (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(rtmrun)(char *j, int *index, int len)
```

DESCRIPTION:

Invokes the routine from the registry to call the forward model integration step of the RTM
The arguments are:

j name of the RTM

3.39.9 registerrtmpesetdecisionspace (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(registerrtmpesetdecisionspace)(char *j, void (*func)(void*,void*),int len)
   Makes an entry in the registry for the routine
   to set the decision space for parameter estimation
```

DESCRIPTION:

j name of the RTM + instance of the runmode

3.39.10 rtmpesetdecisionspace (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(rtmpesetdecisionspace)(char *j, void *dec, void *feas, int len)
```

DESCRIPTION:

Invokes the routine from the registry to set the decision space for parameter estimation

j name of the RTM

dec decision space object

feas feasible space object

3.39.11 registerrtmpegetdecisionspace (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(registerrtmpegetdecisionspace)(char *j, void (*func)(void*),int len)
   Makes an entry in the registry for the routine
   to get the decision space for parameter estimation
```

DESCRIPTION:

j name of the RTM

3.39.12 rtmpegetdecisionspace (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(rtmpegetdecisionspace)(char *j, void *state, int len)
```

DESCRIPTION:

Invokes the routine from the registry to get the decision space for parameter estimation

j name of the RTM

statevars pointer to the prognostic variable array

3.39.13 registerrtmpesetupdecisionspace (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(registerrtmpesetupdecisionspace)(char *j, void (*func)(void*, void*),int len)
```

DESCRIPTION:

Method to registry an interface implementation for setting up the RTM decision space.

j name of the RTM

3.39.14 rtmpesetupdecisionspace (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(rtmpesetupdecisionspace)(char *j, void *dec, void *feas, int len)
```

DESCRIPTION:

Method to setup the RTM decision space

j name of the RTM

dec decision space object

feas feasible space object

3.39.15 registerrtmpesetupobspred (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(registerrtmpesetupobspred)(char *j, void (*func)(void*),int len)
```

DESCRIPTION:

registers the method to initialize the obs pred for the PE observation (model simulated value of the observation)

j name of the RTM + PE instance

3.39.16 rtmpesetupobspredspace (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(rtmpesetupobspredspace)(char *j, void *pepred, int len)
```

DESCRIPTION:

invokes the method to compute the obs pred for the PE observation (model simulated value of the observation)

j name of the RTM + PE instance

pepred object containing the PE obspred

3.39.17 registerrtmpegetobspred (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(registerrtmpegetobspred)(char *j, void (*func)(void*),int len)
```

DESCRIPTION:

registers the method to compute the obs pred for the PE observation (model simulated value of the observation)

j name of the RTM

3.39.18 rtmpegetobspred (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(rtmpegetobspred)(char *j, void *pepred, int len)
```

DESCRIPTION:

invokes the method to compute the obs pred for the PE observation (model simulated value of the observation)

j name of the RTM + PE instance

pepred object containing the PE obspred

3.39.19 registerrtmreset (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(registerrtmreset)(char *j, void (*func)(), int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine to perform reset of RTM variables.

3.39.20 resetrtm (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(resetrtm)(char *j, int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform rtm model reset

j name of the RTM

3.40 Fortran: Module Interface LIS_alb_FTable (Source File: LIS_alb_FTable.c)

Function table registries for storing the interface implementations for managing different sources of albedo parameter data

---

3.40.1 registeralbedosetup (Source File: LIS_alb_FTable.c)

INTERFACE:

```
void FTN(registeralbedosetup)(char *j,void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to setup albedo data reading routines
The arguments are:

i index of the domain

j index of the albedo data source

3.40.2 albedosetup (Source File: LIS_alb_FTable.c)

INTERFACE:

```
void FTN(albedosetup)(char *j, int *n, int len)
```

DESCRIPTION:

Invokes the routine from the registry to setup albedo data reading
The arguments are:

n index of the nest

j index of the albedo data source

3.40.3 registerreadalbedo (Source File: LIS_alb_FTable.c)

INTERFACE:

```
void FTN(registerreadalbedo)(char *j,void (*func)(int*, void*, void*, float*, float*), int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine to read albedo climatology data
The arguments are:

j index of the albedo source

3.40.4 readalbedo (Source File: LIS_alb_FTable.c)

INTERFACE:

```
void FTN(readalbedo)(char *j, int *n, void *time1, void *time2, float *array1, float *array2,int len)
```

DESCRIPTION:

Invokes the routines from the registry for reading climatology albedo files
The arguments are:

n index of the nest

j index of the albedo source

time time index value

array pointer to the array containing the albedo data.

3.41 Fortran: Module Interface LIS_biasestim_FTable (Source File: LIS_biasestim_FTable.c)

Function table registries for storing the interface implementations for the operation of various onlie bias estimation routines to be used in data assimilation

---

3.41.1 registerbiasestimationinit (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(registerbiasestimationinit)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for initializing bias estimation algorithm for use in data assimilation
The arguments are:

j name of the bias estimation algorithm.

3.41.2 biasestimationinit (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(biasestimationinit)(char *j,int len)
```

DESCRIPTION:

Invokes the the selected bias estimation algorithm initialization for use in data assimilation
The arguments are:

j name of the bias estimation scheme

3.41.3 registerbiasestimationsetup (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(registerbiasestimationsetup)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for setting up bias estimation algorithm for use in data assimilation
The arguments are:

j name of the bias estimation algorithm.

3.41.4 biasestimationsetup (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(biasestimationsetup)(char *j, int *k,int len)
```

DESCRIPTION:

Invokes the the selected bias estimation algorithm setup for use in data assimilation
The arguments are:

j name of the bias estimation scheme

k index of the DA instance

3.41.5 registerbiasestimationcompute (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(registerbiasestimationcompute)(char *j, void (*func)(int*, int*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for computing bias correction for use in data assimilation
The arguments are:

j name of the bias estimation algorithm.

3.41.6 computebiascorrection (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(computebiascorrection)(char *j, int *n, int *k,int len)
```

DESCRIPTION:

Invokes the the selected bias estimation algorithm for computing bias correction prior to model propagation
The arguments are:

j name of the bias estimation scheme

n index of the nest

k index of the assimilation instance

3.41.7 registerbiasestimationupdate (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(registerbiasestimationupdate)(char *j, void (*func)(int*, int*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for updating bias correction estimation
The arguments are:

j name of the bias estimation algorithm.

3.41.8 applybiascorrection (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(applybiascorrection)(char *j, int *n, int *k,int len)
```

DESCRIPTION:

Invokes the the selected bias estimation algorithm for updating bias correction estimate
The arguments are:

j name of the bias estimation scheme

n index of the nest

k index of the assimilation instance

3.41.9 registerbiasestimationrestart (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(registerbiasestimationrestart)(char *j, void (*func)(int*, int*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for writing bias restart
The arguments are:

j name of the bias estimation algorithm.

3.41.10 writebiasrestart (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(writebiasrestart)(char *j, int *n, int *k,int len)
```

DESCRIPTION:

Invokes the the selected bias estimation algorithm for writing bias restart
The arguments are:

j name of the bias estimation scheme

n index of the nest

k index of the assimilation instance

3.41.11 registerbiasestimationfinalize (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(registerbiasestimationfinalize)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that finalizes structures related to bias correction algorithm.
The arguments are:

j name of the bias estimation algorithm.

3.41.12 biasestimationfinalize (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(biasestimationfinalize)(char *j,int len)
```

DESCRIPTION:

Finalizes the the selected bias estimation algorithm related structures
The arguments are:

j name of the bias estimation algorithm

3.42 Fortran: Module Interface LIS_dataassim_FTable (Source File: LIS_dataassim_FTable.c

Function table registries for storing the interface implementations for the operation of data assimilation algorithms and observations.

3.42.1 registerdainit (Source File: LIS_dataassim_FTable.c)

Creates an entry in the registry for the routine to intialize algorithm specific structures for the data assimilation method used.
The arguments are:

j name of the assimilation algorithm

INTERFACE:

```
void FTN(registerdainit)(char *j, void (*func)(),int len)
```

3.42.2 dataassiminit (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(dataassiminit)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to initialize the specific structures for the data assimilation algorithm used.
The arguments are:

j name of the assimilation algorithm

3.42.3 registerdasetup (Source File: LIS_dataassim_FTable.c)

Creates an entry in the registry for the routine to set up algorithm specific structures for the data assimilation method used.
The arguments are:

j name of the assimilation algorithm

INTERFACE:

```
void FTN(registerdasetup)(char *j, void (*func)(int*),int len)
```

3.42.4 dataassimsetup (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(dataassimsetup)(char *j, int *k,int len)
```

DESCRIPTION:

Invokes the routine from the registry to setup the specific structures for the data assimilation algorithm used.
The arguments are:

j name of the assimilation algorithm

k index of the DA instance

3.42.5 registerapplyincrements (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(registerapplyincrements)(char *j, void (*func)(int*, int*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that applies the analysis increments
The arguments are:

j name of the assimilation algorithm

3.42.6 applyincrements (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(applyincrements)(char *j, int *n, int *k,int len)
```

DESCRIPTION:

Invoke the routine from the registry that applies analysis increments
The arguments are:

j name of the assimilation algorithm

n index of the nest

k index of the assimilation instance

3.42.7 registercomputeincrements (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(registercomputeincrements)(char *j, void (*func)(int*, int*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that computes the analysis increments
The arguments are:

j name of the assimilation algorithm

3.42.8 computeincrements (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(computeincrements)(char *j, int *n, int *k,int len)
```

DESCRIPTION:

Invoke the routine from the registry that computes analysis increments
The arguments are:

j name of the assimilation algorithm

n index of the nest

k index of assimilation instance

3.42.9 registerdafinalize (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(registerdafinalize)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Cretes an entry in the registry for the routine to cleanup allocated structures
The arguments are:

j name of the assimilation algorithm

3.42.10 dafinalize (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(dafinalize)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry that cleans up the data assimilation specific structures
The arguments are:

j name of the assimilation algorithm

3.42.11 registerdaoutput (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(registerdaoutput)(char *j, void (*func)(int*, int*),int len)
```

DESCRIPTION:

Cretes an entry in the registry for the writing diagnostic output from the data assimilation routines
The arguments are:

j name of the assimilation algorithm

3.42.12 daoutput (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(daoutput)(char *j,int *n, int *k,int len)
```

DESCRIPTION:

Invokes the routine from the registry to write data assimilation specific output
The arguments are:

j name of the assimilation algorithm

n index of the nest

k index of the assimilation instance

3.42.13 registergetnso (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(registergetnso)(char *j, void (*func)(int*, int*, int*, int*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that computes the number of selected observations for the specified grid point.
The arguments are:

j index of the observations variable

3.42.14 getselectedobsnumber (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(getselectedobsnumber)(char *j, int *n, int *gid, int *sid, int *eid,int len)
```

DESCRIPTION:

Invokes the routine from the registry that computes the number of selected observations for a particular modeling point
The arguments are:

j index of the observations variable

n index of the nest

gid model grid point index

sid starting index of the selected observations

eid ending index of the selected observations

3.42.15 registerreaddaobssetup (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(registerdaobssetup)(char *j, void (*func)(int*, void*, void*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for handling observation data for data assimilation
The arguments are:

j index of the assimilation set

3.42.16 readobsdataconfig (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(readobsdataconfig)(char *j, int *k, void *obs, void *pert,int len)
```

DESCRIPTION:

Invokes the routine from the registry to set up structures for handling observation data for data assimilation
The arguments are:

j index of the variable being assimilated

k index of the DA instance

obs ESMF state that contain the observations

pert ESMF state that contain the observation perturbations

3.42.17 registerreaddaobs (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(registerreaddaobs)(char *j, void (*func)(int*, void*, void*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that reads observations for data assimilation.
The arguments are:

j index of the observations variable

3.42.18 readdaobs (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(readdaobs)(char *j,int *n, void *obsstate,void *obspertstate, int len)
```

DESCRIPTION:

Invokes the routine from the registry to read observations for data assimilation. This routine also packages the observations into an ESMF state
The arguments are:

j index of the observations variable

n index of the nest

obsstate pointer to the ESMF observation state

obspertstate pointer to the ESMF observation perturbation state

---

3.42.19 registerwritedaobs (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(registerwritedaobs)(char *j, void (*func)(int*, void*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that writes observations used in data assimilation to disk.
The arguments are:

j index of the observations variable

3.42.20 writedaobs (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(writedaobs)(char *j,int *n, void *obsstate,int len)
```

DESCRIPTION:

Invokes the routine from the registry to write observations used in data assimilation to disk.
The arguments are:

j index of the observations variable

n index of the nest

obsstate pointer to the ESMF observation state

3.43 Fortran: Module Interface LIS_domain_FTable (Source File: LIS_domain_FTable.c)

Function table registries for storing the interface implementations for managing the operation of different domain specifications

---

3.43.1 registerdomain (Source File: LIS_domain_FTable.c)

INTERFACE:

```
void FTN(registerdomain)(char *j,void (*func)(),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine to create grid and tile spaces for a particular domain definition
The arguments are:

j name of the domain/map projection

3.43.2 makedomain (Source File: LIS_domain_FTable.c)

INTERFACE:

```
void FTN(makedomain)(char *j, int len)
```

DESCRIPTION:

Invokes the routine from the registry to create grid and tile spaces for a particular domain definition
The arguments are:

j name of the domain/map projection

3.43.3 registerinput (Source File: LIS_domain_FTable.c)

INTERFACE:

```
void FTN(registerinput)(char *j,void (*func)(),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to read the runtime domain specifics
The arguments are:

j name of the domain/map projection

3.43.4 readinput (Source File: LIS_domain_FTable.c)

INTERFACE:

```
void FTN(readinput)(char *j, int len)
```

DESCRIPTION:

Calls the routine from the registry to read the the runtime domain specifics
The arguments are:

j name of the domain/map projection

3.44 Fortran: Module Interface LIS_emiss_FTable (Source File: LIS_emiss_FTable.c)

Function table registries for storing the interface implementations for managing different sources of greenness fraction data

---

3.44.1 registeremissivitysetup (Source File: LIS_emiss_FTable.c)

INTERFACE:

```
void FTN(registeremissivitysetup)(char *j,void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to setup emiss data reading routines
The arguments are:

i index of the domain

j index of the greenness data source

3.44.2 emissivitysetup (Source File: LIS_emiss_FTable.c)

INTERFACE:

```
void FTN(emissivitysetup)(char *j,int *n, int len)
```

DESCRIPTION:

Invokes the routine from the registry to setup emiss data reading
The arguments are:

n index of the nest

j index of the greenness data source

3.44.3 registerrademissivity (Source File: LIS_emiss_FTable.c)

INTERFACE:

```
void FTN(registerrademissivity)(char *j,void (*func)(int*, void*, void*, float*, float*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to read emiss data
The arguments are:

i index of the domain

j index of the greenness data source

3.44.4 rademissivity (Source File: LIS_emiss_FTable.c)

INTERFACE:

```
void FTN(rademissivity)(char *j,int *n, void *time1, void *time2, float *array1, float *array2, int le
```

DESCRIPTION:

Invokes the routine from the registry to reading emiss data
The arguments are:

n index of the nest

j index of the greenness data source

time month

array pointer to the greenness data

3.45 Fortran: Module Interface LIS_gfrac_FTable (Source File: LIS_gfrac_FTable.c)

Function table registries for storing the interface implementations for managing different sources of greenness fraction data

---

3.45.1 registergfracsetup (Source File: LIS_gfrac_FTable.c)

INTERFACE:

```
void FTN(registergfracsetup)(char *j,void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to setup gfrac data reading routines
The arguments are:

i index of the domain

j index of the greenness data source

3.45.2 gfracsetup (Source File: LIS_gfrac_FTable.c)

INTERFACE:

```
void FTN(gfracsetup)(char *j,int *n, int len)
```

DESCRIPTION:

Invokes the routine from the registry to setup gfrac data reading
The arguments are:

n index of the nest

j index of the greenness data source

3.45.3 registerreadgfrac (Source File: LIS_gfrac_FTable.c)

INTERFACE:

```
void FTN(registerreadgfrac)(char *j,void (*func)(int*, void*, void*, float*, float*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to read gfrac data
The arguments are:

i index of the domain

j index of the greenness data source

3.45.4 readgfrac (Source File: LIS_gfrac_FTable.c)

INTERFACE:

```
void FTN(readgfrac)(char *j,int *n, void *time1, void *time2, float *array1, float *array2, int len)
```

DESCRIPTION:

Invokes the routine from the registry to reading gfrac data
The arguments are:

n index of the nest

j index of the greenness data source

time month

array pointer to the greenness data

3.46 Fortran: Module Interface LIS_irrigation_FTable (Source File: LIS_irrigation_FTable.c)

Function table registries for storing the interface implementations for managing the operations of different land slide models.

3.46.1 registerirrigationschemeinit (Source File: LIS_irrigation_FTable.c)

Creates an entry in the registry for the routine to perform land surface model initialization

INTERFACE:

```
void FTN(registerirrigationschemeinit)(char *j, void (*func)(void*),int len)
```

3.46.2 irrigationschemeinit (Source File: LIS_irrigation_FTable.c)

INTERFACE:

```
void FTN(irrigationschemeinit)(char *j,void *state,int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform land surface scheme initialization

i index of the LSM

3.46.3 registerirrigationupdate (Source File: LIS_irrigation_FTable.c)

INTERFACE:

```
void FTN(registerirrigationupdate)(char *j, void (*func)(int*, void*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine to run the land surface model

i index of the LSM

3.46.4 applyirrigationupdates (Source File: LIS_irrigation_FTable.c)

INTERFACE:

```
void FTN(applyirrigationupdates)(char *j,int *n, void *state, int len)
```

DESCRIPTION:

Invokes the routine from the registry to run the land surface model

i index of the LSM

n index of the nest

3.47 Fortran: Module Interface LIS_laisai_FTable (Source File: LIS_laisai_FTable.c)

Function table registries for storing the interface implementations for managing different sources of LAI and SAI data

---

3.47.1 registerlaisetup (Source File: LIS_laisai_FTable.c)

INTERFACE:

```
void FTN(registerlaisetup)(char *j,void (*func)(int*), int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to setup lai data reading routines
The arguments are:

i index of the domain

j index of the lai data source

3.47.2 laisetup (Source File: LIS_laisai_FTable.c)

INTERFACE:

```
void FTN(laisetup)(char *j, int *n, int len)
```

DESCRIPTION:

Invokes the routine from the registry to setup lai data reading
The arguments are:

n index of the nest

j index of the lai data source

3.47.3 registersaisetup (Source File: LIS_laisai_FTable.c)

INTERFACE:

```
void FTN(registersaisetup)(char *j,void (*func)(int*), int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to setup sai data reading routines
The arguments are:

i index of the domain

j index of the sai data source

3.47.4 saisetup (Source File: LIS_laisai_FTable.c)

INTERFACE:

```
void FTN(saisetup)(char *j, int *n, int len)
```

DESCRIPTION:

Invokes the routine from the registry to setup sai data reading
The arguments are:

n index of the nest

j index of the sai data source

3.47.5 registerreadlai (Source File: LIS_laisai_FTable.c)

INTERFACE:

```
void FTN(registerreadlai)(char *j, void (*func)(int*, void*, void*, float*, float*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routines to read LAI data
The arguments are:

j index of the LAI source

3.47.6 readlai (Source File: LIS_laisai_FTable.c)

INTERFACE:

```
void FTN(readlai)(char *j, int *n, void *time1, void *time2, float *array1, float *array2, int len)
```

DESCRIPTION:

Invokes the routine from the registry to read LAI data.
The arguments are:

n index of the nest

j index of the LAI source

time time file search begins

array pointer to the LAI data

time time of found file

3.47.7 registerreadsai (Source File: LIS_laisai_FTable.c)

INTERFACE:

```
void FTN(registerreadsai)(char *j, void (*func)(int*, void*, float*), int len)
```

DESCRIPTION:

Creates an entry in the registry for the routines to read SAI data
The arguments are:

j index of the SAI source

3.47.8 readsai (Source File: LIS_laisai_FTable.c)

INTERFACE:

```
void FTN(readsai)(char *j, int *n, void *time, float* array, int len)
```

DESCRIPTION:

Invokes the routine from the registry to read SAI data.
The arguments are:

n index of nest

j index of the SAI source

time time

array pointer to the SAI data

3.48 Fortran: Module Interface LIS_lakemodel_FTable (Source File: LIS_lakemodel_FTable.c

Function table registries for storing the interface implementations for managing the operations of different land surface models. The registries also contain related interface implementations for data assimilation, WRF/GCE/GFS coupling and parameter estimation

---

3.48.1 registerlakemodelini (Source File: LIS_lakemodel_FTable.c)

Creates an entry in the registry for the routine to perform land surface model initialization

INTERFACE:

```
void FTN(registerlakemodelinit)(char *j, void (*func)(int*),int len)
```

3.48.2 lakemodelinit (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(lakemodelinit)(char *j,int *mtype, int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform land surface model initialization

j name of the LAKEMODEL

3.48.3 registerlakemodelrun (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(registerlakemodelrun)(char *j, void (*func)(int*, int*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine to run the land surface model

j name of the LAKEMODEL

3.48.4 lakemodelrun (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(lakemodelrun)(char *j,int *n,int *mtype,int len)
```

DESCRIPTION:

Invokes the routine from the registry to run the land surface model

j name of the LAKEMODEL

n index of the nest

3.48.5 registerlakemodelfinalize (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(registerlakemodelfinalize)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine to cleanup allocated structures specific to the land surface model

j name of the LAKEMODEL

3.48.6 lakemodelfinalize (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(lakemodelfinalize)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry for cleaning up allocated structures specific to the land surface model

j name of the LAKEMODEL

3.48.7 registerlakemodelsetup (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(registerlakemodelsetup)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to set up land surface model parameters

j name of the LAKEMODEL

3.48.8 lakemodelsetup (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(lakemodelsetup)(char *j,int *mtype, int len)
```

DESCRIPTION:

Invokes the routine in the registry to set up land surface model parameters

j name of the LAKEMODEL

3.48.9 registerlakemodelrestart (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(registerlakemodelrestart)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to restart the land surface model from a previously saved state

j name of the LAKEMODEL

3.48.10 lakemodelrestart (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(lakemodelrestart)(char *j,int *mtype, int len)
```

DESCRIPTION:

Invokes the routine from the registry to restart the land surface model from a previously saved state

j name of the LAKEMODEL

3.48.11 registerlakemodeldynsetup (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(registerlakemodeldynsetup)(char *j, void (*func)(int*, int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to set the time dependent land surface parameters

j name of the LAKEMODEL

3.48.12 lakemodeldynsetup (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(lakemodeldynsetup)(char *j, int *n, int *mtype,int len)
```

DESCRIPTION:

Invokes the routine from the registry to set the time dependent land surface parameters

j name of the LAKEMODEL

n index of the nest

3.48.13 registerlakemodeloutput (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(registerlakemodeloutput)(char *j, void (*func)(int*, int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to perform land surface model output

j name of the LAKEMODEL

3.48.14 lakemodeloutput (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(lakemodeloutput)(char *j, int *n, int *mtype, int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform land surface model output

j name of the LAKEMODEL

n index of the nest

3.48.15 registerlakemodelf2t (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(registerlakemodelf2t)(char *j, void (*func)(int*, int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to transfer forcing to model tiles

j name of the LAKEMODEL + name of the running mode

j index of the runmode

3.48.16 lakemodelf2t (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(lakemodelf2t)(char *j, int *n, int *mtype, int len)
```

DESCRIPTION:

Invokes the routine from the registry to transfer forcing to model tiles

j name of the LAKEMODEL

j index of the runmode

n index of the nest

3.48.17 registerlakemodelwrst (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(registerlakemodelwrst)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to write restart files for a land surface model

j name of the LAKEMODEL

3.48.18 lakemodelwrst (Source File: LIS_lakemodel_FTable.c)

INTERFACE:

```
void FTN(lakemodelwrst)(char *j, int *n, int len)
```

DESCRIPTION:

Invokes the routine from the registry to write restart files for a land surface model

j name of the LAKEMODEL

n index of the nest

3.49 Fortran: Module Interface LIS_landslide_FTable (Source File: LIS_landslide_FTable.c)

Function table registries for storing the interface implementations for managing the operations of different land slide models.

3.49.1 registerlandslidemodelinit (Source File: LIS_landslide_FTable.c)

Creates an entry in the registry for the routine to perform land surface model initialization

INTERFACE:

```
void FTN(registerlandslidemodelinit)(char *j, void (*func)(),int len)
```

3.49.2 initializelandslidemodel (Source File: LIS_landslide_FTable.c)

INTERFACE:

```
void FTN(initializelandslidemodel)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform land surface model initialization

i index of the LSM

3.49.3 registerlandslidemodelrun (Source File: LIS_landslide_FTable.c)

INTERFACE:

```
void FTN(registerlandslidemodelrun)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine to run the land surface model

i index of the LSM

3.49.4 runlandslidemodel (Source File: LIS_landslide_FTable.c)

INTERFACE:

```
void FTN(runlandslidemodel)(char *j,int *n,int len)
```

DESCRIPTION:

Invokes the routine from the registry to run the land surface model

i index of the LSM

n index of the nest

3.49.5 registerlandslidemodeloutput (Source File: LIS_landslide_FTable.c)

INTERFACE:

```
void FTN(registerlandslidemodeloutput)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine to run the land surface model

i index of the LSM

3.49.6 outputlandslidemodel (Source File: LIS_landslide_FTable.c)

INTERFACE:

```
void FTN(outputlandslidemodel)(char *j,int *n,int len)
```

DESCRIPTION:

Invokes the routine from the registry to run the land surface model

i index of the LSM

n index of the nest

3.49.7 registerlandslidemodelfinalize (Source File: LIS_landslide_FTable.c)

INTERFACE:

```
void FTN(registerlandslidemodelfinalize)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine to cleanup allocated structures specific to the land surface model

i index of the LANDSLIDEMODEL

3.49.8 landslidemodelfinalize (Source File: LIS_landslide_FTable.c)

INTERFACE:

```
void FTN(landslidemodelfinalize)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry for cleaning up allocated structures specific to the land surface model

i index of the LANDSLIDEMODEL

3.49.9 registersetuplandslidepeobspred (Source File: LIS_landslide_FTable.c)

INTERFACE:

```
void FTN(registersetuplandslidepeobspred)(char *j, void (*func)(void*),int len)
```

DESCRIPTION:

registers the method to initialize the obs pred for the PE observation (model simulated value of the observation)

i index of the LANDSLIDE

j index of the optimization set

3.49.10 setuplandslidepeobspredspace (Source File: LIS_landslide_FTable.c)

INTERFACE:

```
void FTN(setuplandslidepeobspredspace)(char *j, void *pepred,int len)
```

DESCRIPTION:

invokes the method to compute the obs pred for the PE observation (model simulated value of the observation)

i index of the LANDSLIDE

j index of the variable

pepred object containing the PE obspred

3.49.11 registerqclandslidedecisionspace (Source File: LIS_landslide_FTable.c)

INTERFACE:

```
void FTN(registerqclandslidedecisionspace)(char *j, void (*func)(void*, void*),int len)
```

DESCRIPTION:

Method to registry an interface implementation for Qc'ing a specified decision space.

i index of the LSM

j index of the optimization set

3.49.12 qclandslidedecisionspace (Source File: LIS_landslide_FTable.c)

INTERFACE:

```
void FTN(qclandslidedecisionspace)(char *j, void *dec, void *feas,int len)
```

DESCRIPTION:

Method to QC a specified decision space.

i index of the LSM

j index of the optimization set

dec decision space object

feas feasible space object

3.49.13 registersetlandslidedecisionspace (Source File: LIS_landslide_FTable.c)

INTERFACE:

```
void FTN(registersetlandslidedecisionspace)(char *j, void (*func)(void*),int len)
   Makes an entry in the registry for the routine
   to set the decision space for optimization
```

DESCRIPTION:

i index of the LANDSLIDE

j index of the variable

3.49.14 setlandslidedecisionspace (Source File: LIS_landslide_FTable.c)

INTERFACE:

```
void FTN(setlandslidedecisionspace)(char *j, void *state,int len)
```

DESCRIPTION:

Invokes the routine from the registry to set the decision space for optimization

i index of the landslide model

n index of the nest

statevars pointer to the prognostic variable array

3.49.15 registerlandslidemodelreset (Source File: LIS_landslide_FTable.c)

Creates an entry in the registry for the routine to perform land surface model resetialization

INTERFACE:

```
void FTN(registerlandslidemodelreset)(char *j, void (*func)(),int len)
```

3.49.16 resetlandslidemodel (Source File: LIS_landslide_FTable.c)

INTERFACE:

```
void FTN(resetlandslidemodel)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform landslide model reset

i index of the LSM

3.49.17 lis_log_msgC.c (Source File: LIS_log_msgC.c)

This file provides methods for diagnostic and error logging in LIS for codes written in the C language. The diagnostic messages are appended with the appropriate time-date and processor stamps.

3.49.18 lis_log_msgC (Source File: LIS_log_msgC.c)

This routine is a C wrapper to the lis_log_msg routine.

REVISION HISTORY:

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

INTERFACE:

```
void lis_log_msgC(char * string)
```

3.50 Fortran: Module Interface LIS_lsm_FTable (Source File: LIS_lsm_FTable.c)

Function table registries for storing the interface implementations for managing the operations of different land surface models. The registries also contain related interface implementations for data assimilation, WRF/GCE/GFS coupling and parameter estimation

---

3.50.1 registerlsmini (Source File: LIS_lsm_FTable.c)

Creates an entry in the registry for the routine to perform land surface model initialization

INTERFACE:

```
void FTN(registerlsminit)(char *j, void (*func)(),int len)
```

3.50.2 lsminit (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsminit)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform land surface model initialization

j name of the LSM

3.50.3 registerlsmrun (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmrun)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine to run the land surface model

j name of the LSM

3.50.4 lsmrun (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmrun)(char *j,int *n,int len)
```

DESCRIPTION:

Invokes the routine from the registry to run the land surface model

j name of the LSM

n index of the nest

3.50.5 registerlsmfinalize (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmfinalize)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine to cleanup allocated structures specific to the land surface model

j name of the LSM

3.50.6 lsmfinalize (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmfinalize)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry for cleaning up allocated structures specific to the land surface model

j name of the LSM

3.50.7 registerlsmreset (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmreset)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine to cleanup allocated structures specific to the land surface model

j name of the LSM

3.50.8 lsmreset (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmreset)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry for cleaning up allocated structures specific to the land surface model

j name of the LSM

3.50.9 registerlsmsetup (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmsetup)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to set up land surface model parameters

j name of the LSM

3.50.10 lsmsetup (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmsetup)(char *j, int len)
```

DESCRIPTION:

Invokes the routine in the registry to set up land surface model parameters

j name of the LSM

3.50.11 registerlsmrestart (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmrestart)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to restart the land surface model from a previously saved state

j name of the LSM

3.50.12 lsmrestart (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmrestart)(char *j, int len)
```

DESCRIPTION:

Invokes the routine from the registry to restart the land surface model from a previously saved state

j name of the LSM

3.50.13 registerlsmdynsetup (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmdynsetup)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to set the time dependent land surface parameters

j name of the LSM

3.50.14 lsmdynsetup (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmdynsetup)(char *j, int *n, int len)
```

DESCRIPTION:

Invokes the routine from the registry to set the time dependent land surface parameters

j name of the LSM

n index of the nest

3.50.15 registerlsmf2t (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmf2t)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to transfer forcing to model tiles

j name of the LSM + name of the running mode

j index of the runmode

3.50.16 lsmf2t (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmf2t)(char *j, int *n, int len)
```

DESCRIPTION:

Invokes the routine from the registry to transfer forcing to model tiles

j name of the LSM

j index of the runmode

n index of the nest

3.50.17 registerlsmwrst (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmwrst)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to write restart files for a land surface model

j name of the LSM

3.50.18 lsmwrst (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmwrst)(char *j, int *n, int len)
```

DESCRIPTION:

Invokes the routine from the registry to write restart files for a land surface model

j name of the LSM

n index of the nest

3.50.19 registerlsmdainit (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmdainit)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine for initializing DA related LSM settings

j name of the LSM + DA instance

3.50.20 lsmdainit (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmdainit)(char *j, int len)
```

DESCRIPTION:

Invokes the routine from the registry for initializing DA related LSM settings.

j name of the LSM + DA instance

3.50.21 registerlsmdagetstatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmdagetstatevar)(char *j, void (*func)(int*, void*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine for obtaining the specified prognostic variables from the land surface model (for data assimilation)

j name of the LSM + DA instance

3.50.22 lsmdagetstatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmdagetstatevar)(char *j, int *n, void *state, int len)
```

DESCRIPTION:

Invokes the routine from the registry for obtaining the specified prognostic variables from the land surface model (for data assimilation)

j name of the LSM + DA instance

n index of the nest

state pointer to the prognostic variable state

3.50.23 registerlsmdasetstatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmdasetstatevar)(char *j, void (*func)(int*, void*),int len)
```

DESCRIPTION:

Makes an entry in the registry for updating the specified state variable in a land surface model (for data assimilation)

j name of the LSM + DA instance

3.50.24 lsmdasetstatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmdasetstatevar)(char *j,int *n, void *statevar, int len)
```

DESCRIPTION:

Invokes the routine from the registry for updating the specified state variable in a land surface model (for data assimilation)

j name of the LSM + DA instance

n index of the nest

statevars pointer to the prognostic variable state

3.50.25 registerlsmdaobstransform (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmdaobstransform)(char *j, void (*func)(int*, void*),int len)
```

DESCRIPTION:

Makes an entry in the registry to perform the translation of observations to state variables (for data assimilation)

j name of the LSM + DA instance

k index of the observation data

3.50.26 lsmdaobstransform (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmdaobstransform)(char *j, int *n, void *obs, int len)
```

DESCRIPTION:

Invokes the routine from the registry to translate the observations to the prognostic variable space (for data assimilation)

j name of the LSM + DA instance

n index of the nest

obs transformed variable

3.50.27 registerlsmdagetobspred (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmdagetobspred)(char *j, void (*func)(int*,float*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine that provides an LSM's estimate of the observations. (for data assimilation)

j name of the LSM + DA instance

3.50.28 lsmdagetobspred (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmdagetobspred)(char *j, int *n, float *pred, int len)
```

DESCRIPTION:

Invokes the routine from the registry to translate the observations to state variables (for data assimilation)

j name of the LSM

n index of the nest

pred model's estimated observation prediction

3.50.29 registerlsmdadiagnosevars (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmdadiagnosevars)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to perform land surface model output

j name of the LSM

3.50.30 lsmdadiagnosevars (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmdadiagnosevars)(char *j, int *n, int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform land surface model output

j name of the LSM

n index of the nest

3.50.31 registerlsmdamapobstolsm (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmdamapobstolsm)(char *j, void (*func)(int*, void*, void*),int len)
```

DESCRIPTION:

Makes an entry in the registry to perform the translation of observations to state variables (for data assimilation)

j name of the LSM + DA instance

3.50.32 lsmdamapobstolsm (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmdamapobstolsm)(char *j, int *n, void *obs, void *lsm, int len)
```

DESCRIPTION:

Invokes the routine from the registry to translate the observations to state variables (for data assimilation)

j name of the LSM + DA instance

n index of the nest

obs observations to be mapped

lsm updated lsm states

3.50.33 registerlsmdaqcstate (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmdaqcstate)(char *j, void (*func)(int*, void*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to QC the updated LSM state (for data assimilation)

j name of the LSM + DA instance

3.50.34 lsmdaqcstate (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmdaqcstate)(char *j,int *n, void *LSM_State, int len)
```

DESCRIPTION:

Invokes the routine from the registry to set the QC the updated LSM variables (for data assimilation)

j name of the LSM

n index of the nest

LSM_State The LSM state being qc'd

3.50.35 registerlsmdaqcobsstate (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmdaqcobsstate)(char *j, void (*func)(int*, void*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to QC the OBS state based on LSM variables and states (for data assimilation).

j name of the LSM + DA instance

3.50.36 lsmdaqcobsstate (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmdaqcobsstate)(char *j, int *n, void *LSM_State, int len)
```

DESCRIPTION:

Invokes the routine from the registry to set the QC the observation state based on LSM variables and states (for data assimilation).

j name of the LSM

n index of the nest

LSM_State The LSM state being qc'd

3.50.37 registerlsmdascalestatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmdascalestatevar)(char *j, void (*func)(int*, void*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to scale the LSM state variables (for data assimilation)

j name of the LSM + DA instance

j index of the assimilated variable

3.50.38 lsmdascalestatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmdascalestatevar)(char *j, int *n, void *LSM_State, int len)
```

DESCRIPTION:

Invokes the routine from the registry to scale the LSM state variables (for data assimilation)

j name of the LSM + DA instance

n index of the nest

LSM_State The LSM state being scaled

3.50.39 registerlsmdadescalestatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmdadescalestatevar)(char *j, void (*func)(int*, void*, void*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to descale the LSM state variables (for data assimilation)

j name of the LSM

j index of the assimilated variable

3.50.40 lsmdadescalestatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmdadescalestatevar)(char *j,int *n, void *LSM_State, void *LSM_Incr_State, int len)
```

DESCRIPTION:

Invokes the routine from the registry descale the LSM state variables (for data assimilation)

j name of the LSM

n index of the nest

LSM_State The LSM state being descaled

3.50.41 registerlsmdaupdatestate (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmdaupdatestate)(char *j, void (*func)(int*, void*, void*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to apply the LSM state increments to LSM state (for data assimilation)

j name of the LSM + DA instance

3.50.42 lsmdaupdatestate (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmdaupdatestate)(char *j, int *n, void *LSM_State, void *LSM_Incr_State, int len)
```

DESCRIPTION:

Invokes the routine from the registry to apply the LSM state increments to LSM State (for data assimilation)

j name of the LSM + DA instance

n index of the nest

LSM_State The LSM state being updated

LSM_Incr_State The LSM incr state

3.50.43 registerwritestatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerwritestatevar)(char *j, int *j, void (*func)(int *, int*, void*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine for obtaining the specified prognostic variables from the land surface model

j name of the LSM

j index of the variable

3.50.44 writelsmstatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(writelsmstatevar)(char *j, int *j, int *ftn, int *n, void *state)
```

DESCRIPTION:

Invokes the routine from the registry for obtaining the specified prognostic variables from the land surface model

j name of the LSM

j index of the variable

n index of the nest

statevars pointer to the prognostic variable array

3.50.45 registerlsmcplsetexport (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmcplsetexport)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to set export states for a LSM

j name of the LSM + instance of run mode

3.50.46 lsmcplsetexport (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmcplsetexport)(char *j, int *n, int len)
```

DESCRIPTION:

Invokes the routine in the registry to set the export states for a LSM

j name of the LSM + instance of run mode

n index of the nest

3.50.47 registerlsmpesetdecisionspace (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmpesetdecisionspace)(char *j, void (*func)(void*,void*),int len)
   Makes an entry in the registry for the routine
   to set the decision space for parameter estimation
```

DESCRIPTION:

j name of the LSM + instance of the runmode

3.50.48 lsmpesetdecisionspace (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmpesetdecisionspace)(char *j, void *dec, void *feas, int len)
```

DESCRIPTION:

Invokes the routine from the registry to set the decision space for parameter estimation

j name of the LSM

dec decision space object

feas feasible space object

3.50.49 registerlsmpegetdecisionspace (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmpegetdecisionspace)(char *j, void (*func)(void*),int len)
   Makes an entry in the registry for the routine
   to get the decision space for parameter estimation
```

DESCRIPTION:

j name of the LSM

3.50.50 lsmpegetdecisionspace (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmpegetdecisionspace)(char *j, void *state, int len)
```

DESCRIPTION:

Invokes the routine from the registry to get the decision space for parameter estimation

j name of the LSM

statevars pointer to the prognostic variable array

3.50.51 registerlsmpesetupdecisionspace (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmpesetupdecisionspace)(char *j, void (*func)(void*, void*),int len)
```

DESCRIPTION:

Method to registry an interface implementation for setting up the LSM decision space.

j name of the LSM

3.50.52 lsmpesetupdecisionspace (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmpesetupdecisionspace)(char *j, void *dec, void *feas, int len)
```

DESCRIPTION:

Method to setup the LSM decision space

j name of the LSM

dec decision space object

feas feasible space object

3.50.53 registerlsmpesetupobspred (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmpesetupobspred)(char *j, void (*func)(void*),int len)
```

DESCRIPTION:

registers the method to initialize the obs pred for the PE observation (model simulated value of the observation)

j name of the LSM + PE instance

3.50.54 lsmpesetupobspredspace (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmpesetupobspredspace)(char *j, void *pepred, int len)
```

DESCRIPTION:

invokes the method to compute the obs pred for the PE observation (model simulated value of the observation)

j name of the LSM + PE instance

pepred object containing the PE obspred

3.50.55 registerlsmpegetobspred (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmpegetobspred)(char *j, void (*func)(void*),int len)
```

DESCRIPTION:

registers the method to compute the obs pred for the PE observation (model simulated value of the observation)

j name of the LSM

3.50.56 lsmpegetobspred (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmpegetobspred)(char *j, void *pepred, int len)
```

DESCRIPTION:

invokes the method to compute the obs pred for the PE observation (model simulated value of the observation)

j name of the LSM + PE instance

pepred object containing the PE obspred

3.50.57 registerlsmroutinggetrunoff (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmroutinggetrunoff)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

creates an entry in the registry for the routine to return runoff fields from a LSM.

j name of the LSM + routing instance

3.50.58 lsmroutinggetrunoff (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmroutinggetrunoff)(char *j, int *n, int len)
```

DESCRIPTION:

Invokes the registered routine that returns the runoff fields from a LSM.

j name of the LSM + routing instance

n index of the nest

3.50.59 registerlsm2rtm (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsm2rtm)(char *j,  void (*func)(int*, void*), int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to transfer surface properties to the RTM data structure
The arguments are:

j name of the RTM .

3.50.60 lsm2rtm (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsm2rtm)(char *j, int *index, void *sfc, int len)
```

DESCRIPTION:

Invokes the routine from the registry to map surface properties to the RTM data structure
The arguments are:

j name of the RTM

3.50.61 registerlsmirrigationgetstates (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmirrigationgetstates)(char *j,  void (*func)(int*, void*), int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to initialize the LSM irrigation states
The arguments are:

j name of the irrigation scheme .

3.50.62 getirrigationlsmstates (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(getirrigationlsmstates)(char *j, int *n, void *irrigstate, int len)
```

DESCRIPTION:

Invokes the routine from the registry to getialize the LSM irrigation states
The arguments are:

j name of the irrigation model

3.50.63 LIS_memory_managementC (Source File: LIS_memory_managementC.c)

This file provides methods for dynamic memory management in LIS for codes written in the C language.

3.50.64 lis_calloc (Source File: LIS_memory_managementC.c)

Generic call to the C calloc function used in LIS.

REVISION HISTORY:

```
Mar 2004, James Geiger; Initial Specification
```

INTERFACE:

```
void * lis_calloc(size_t n, size_t size, char * caller)
```

3.50.65 lis_malloc (Source File: LIS_memory_managementC.c)

Generic call to the C malloc function used in LIS.

REVISION HISTORY:

```
Mar 2004, James Geiger; Initial Specification
```

INTERFACE:

```
void * lis_malloc(size_t size, char * caller)
```

3.51 Fortran: Module Interface LIS_metforcing_FTable (Source File: LIS_metforcing_FTable

Function table registries for storing the interface implementations for managing different meteorological analyses implmented as "metforcings"

3.51.1 registerinitmetforc (Source File: LIS_metforcing_FTable.c)

INTERFACE:

```
void FTN(registerinitmetforc)(char *j, void (*func)(int*), int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that define the native domain of the met forcing scheme.
The arguments are:

j name of the met forcing scheme

3.51.2 initmetforc (Source File: LIS_metforcing_FTable.c)

INTERFACE:

```
void FTN(initmetforc)(char *j,int *findex, int len)
```

DESCRIPTION:

Invokes the routine from the registry that defines the native domain of the met forcine scheme.
The arguments are:

j name of the met forcing scheme

findex order of the met forcing scheme among all selected forcings in a LIS instance

---

3.51.3 registerretrievemetforc (Source File: LIS_metforcing_FTable.c)

INTERFACE:

```
void FTN(registerretrievemetforc)(char *j,void (*func)(int*, int*), int len)
```

DESCRIPTION:

Creates an entry in the registry for the routines to open and read met forcing
The arguments are:

j name of the met forcing scheme

3.51.4 retrievemetforc (Source File: LIS_metforcing_FTable.c)

INTERFACE:

```
void FTN(retrievemetforc)(char *j, int *n, int *findex, int len)
```

DESCRIPTION:

Invokes the routine from the registry to open and read the met forcing
The arguments are:

j name of the met forcing scheme

n index of the nest

findex order of the met forcing scheme among all selected forcings in a LIS instance

---

3.51.5 registertimeinterpmetforc (Source File: LIS_metforcing_FTable.c)

INTERFACE:

```
void FTN(registertimeinterpmetforc)(char *j,void (*func)(int*, int*), int len)
```

DESCRIPTION:

Creats an entry in the registry for the routine to perform temporal interpolation
The arguments are:

j name of the met forcing scheme

3.51.6 timeinterpmetforc (Source File: LIS_metforcing_FTable.c)

INTERFACE:

```
void FTN(timeinterpmetforc)(char *j, int *n, int *findex, int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform temporal interpolation
The arguments are:

j name of the met forcing scheme

n index of the nest

findex order of the met forcing scheme among all selected forcings in a LIS instance

---

3.51.7 registerfinalmetforc (Source File: LIS_metforcing_FTable.c)

INTERFACE:

```
void FTN(registerfinalmetforc)(char *j,void (*func)(int*), int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine to cleanup allocated structures
The arguments are:

j name of the met forcing scheme

3.51.8 finalmetforc (Source File: LIS_metforcing_FTable.c)

INTERFACE:

```
void FTN(finalmetforc)(char *j, int *findex, int len)
```

DESCRIPTION:

Invokes the routine from the registry to cleanup the allocated structures associated with the met forcing
The arguments are:

j name of the met forcing scheme

findex order of the met forcing scheme among all selected forcings in a LIS instance

---

3.51.9 registerresetmetforc (Source File: LIS_metforcing_FTable.c)

INTERFACE:

```
void FTN(registerresetmetforc)(char *j,void (*func)(int*), int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine to reset the data structures
The arguments are:

j name of the met forcing scheme

3.51.10 resetmetforc (Source File: LIS_metforcing_FTable.c)

INTERFACE:

```
void FTN(resetmetforc)(char *j, int *findex, int len)
```

DESCRIPTION:

Invokes the routine from the registry to reset the data structures associated with the met forcing
The arguments are:

j name of the met forcing scheme

findex order of the met forcing scheme among all selected forcings in a LIS instance

---

3.51.11 LIS_mpDecomp (Source File: LIS_mpDecomp.c)

This file provides an algorithm for the decomposition of a given domain. The method is adopted from the Weather Research and Forecasting (WRF) source code.

REVISION HISTORY:

```
Feb 2006  Sujay Kumar Adopted in LIS
```

3.52 Fortran: Module Interface LIS_openwater_FTable (Source File: LIS_openwater_FTable.

Function table registries for storing the interface implementations for managing the operations of different land surface models. The registries also contain related interface implementations for data assimilation, WRF/GCE/GFS coupling and parameter estimation

---

3.52.1 registeropenwaterini (Source File: LIS_openwater_FTable.c)

Creates an entry in the registry for the routine to perform land surface model initialization

INTERFACE:

```
void FTN(registeropenwaterinit)(char *j, void (*func)(int*),int len)
```

3.52.2 openwaterinit (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(openwaterinit)(char *j,int *mtype, int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform land surface model initialization

j name of the OPENWATER

3.52.3 registeropenwaterrun (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(registeropenwaterrun)(char *j, void (*func)(int*, int*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine to run the land surface model

j name of the OPENWATER

3.52.4 openwaterrun (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(openwaterrun)(char *j,int *n,int *mtype,int len)
```

DESCRIPTION:

Invokes the routine from the registry to run the land surface model

j name of the OPENWATER

n index of the nest

3.52.5 registeropenwaterfinalize (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(registeropenwaterfinalize)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine to cleanup allocated structures specific to the land surface model

j name of the OPENWATER

3.52.6 openwaterfinalize (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(openwaterfinalize)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry for cleaning up allocated structures specific to the land surface model

j name of the OPENWATER

3.52.7 registeropenwatersetup (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(registeropenwatersetup)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to set up land surface model parameters

j name of the OPENWATER

3.52.8 openwatersetup (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(openwatersetup)(char *j,int *mtype, int len)
```

DESCRIPTION:

Invokes the routine in the registry to set up land surface model parameters

j name of the OPENWATER

3.52.9 registeropenwaterrestart (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(registeropenwaterrestart)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to restart the land surface model from a previously saved state

j name of the OPENWATER

3.52.10 openwaterrestart (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(openwaterrestart)(char *j,int *mtype, int len)
```

DESCRIPTION:

Invokes the routine from the registry to restart the land surface model from a previously saved state

j name of the OPENWATER

3.52.11 registeropenwaterdynsetup (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(registeropenwaterdynsetup)(char *j, void (*func)(int*, int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to set the time dependent land surface parameters

j name of the OPENWATER

3.52.12 openwaterdynsetup (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(openwaterdynsetup)(char *j, int *n, int *mtype,int len)
```

DESCRIPTION:

Invokes the routine from the registry to set the time dependent land surface parameters

j name of the OPENWATER

n index of the nest

3.52.13 registeropenwateroutput (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(registeropenwateroutput)(char *j, void (*func)(int*, int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to perform land surface model output

j name of the OPENWATER

3.52.14 openwateroutput (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(openwateroutput)(char *j, int *n, int *mtype, int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform land surface model output

j name of the OPENWATER

n index of the nest

3.52.15 registeropenwaterf2t (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(registeropenwaterf2t)(char *j, void (*func)(int*, int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to transfer forcing to model tiles

j name of the OPENWATER + name of the running mode

j index of the runmode

3.52.16 openwaterf2t (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(openwaterf2t)(char *j, int *n, int *mtype, int len)
```

DESCRIPTION:

Invokes the routine from the registry to transfer forcing to model tiles

j name of the OPENWATER

j index of the runmode

n index of the nest

3.52.17 registeropenwaterwrst (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(registeropenwaterwrst)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to write restart files for a land surface model

j name of the OPENWATER

3.52.18 openwaterwrst (Source File: LIS_openwater_FTable.c)

INTERFACE:

```
void FTN(openwaterwrst)(char *j, int *n, int len)
```

DESCRIPTION:

Invokes the routine from the registry to write restart files for a land surface model

j name of the OPENWATER

n index of the nest

3.53 Fortran: Module Interface LIS_optUE_FTable (Source File: LIS_optUE_FTable.c)

Function table registries for storing the interface implementations for the operation of optimization uncertainty estimation algorithms and observations.

3.53.1 registeroptuealginit (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuealginit)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for initializing optimization / uncertainty estimation algorithms
The arguments are:

j index of the optmization/UE algorithm

3.53.2 optuealginit (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(optuealginit)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to set up structures for initializing optimization or uncertainty estimation algorithm
The arguments are:

j name of the optimization/UE algorithm

3.53.3 registeroptuealgsetup (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuealgsetup)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for optimization/ uncertainty estimation algorithms
The arguments are:

j index of the optmization/UE algorithm

3.53.4 optuealgsetup (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(optuealgsetup)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to set up structures for optimization/uncertainty estimation algorithm
The arguments are:

j name of the optimization/uncertainty estimation algorithm

3.53.5 registeroptueconvergencecheck (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptueconvergencecheck)(char *j, void (*func)(void*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that checks for convergence for an optimization algorithm
The arguments are:

j name of the optimization algorithm

3.53.6 checkconvergence (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(checkconvergence)(char *j, void* check,int len)
```

DESCRIPTION:

Invokes the routine from the registry to check for convergence of an optimization algorithm.
The arguments are:

j name of the optimization algorithm

3.53.7 registeroptuealgrun (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuealgrun)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that executes the optimization algorithm run method
The arguments are:

j name of the optimization algorithm.

3.53.8 runoptue (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(runoptue)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to execute the optimization algorithm run method
The arguments are:

j name of the optimization algorithm

3.53.9 registergetdecisionspace (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuegetdecisionspace)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that retrieves decision space values from the optimization algorithm datastructures
The arguments are:

j name of the optimization algorithm

3.53.10 (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(getoptuealgdecspace)(char *j, int *n,int len)
```

DESCRIPTION:

Invokes the routine from the registry to invoke an objective function method
The arguments are:

j name of the optimization algorithm

n index of the nest

3.53.11 registeroptuesetdecisionspace (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuesetdecisionspace)(char *j, void (*func)(int*, void*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that assigns decision space values to the optimization algorithm datastructures
The arguments are:

j name of the optimization algorithm

3.53.12 (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(setoptuealgdecspace)(char *j, int *n, void *decspace,int len)
```

DESCRIPTION:

Invokes the routine from the registry to set a given decision space to the optimization algorithm
The arguments are:

j name of the optimization algorithm

n index of the nest

decspace decision space object

3.53.13 registeroptuegetnparam (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuegetnparam)(char *j, void (*func)(void*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that retrieves the size of the decision space
The arguments are:

j name of the optimization algorithm

3.53.14 (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(getoptuealgnparam)(char *j, void *nparam,int len)
```

DESCRIPTION:

Invokes the routine from the registry to retrieve the size of the decision space
The arguments are:

j name of the optimization algorithm

nparam size of the decision space

3.53.15 registeroptuereadrestart (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuereadrestart)(char *j, void (*func)(void*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that reads a restart file for the OPT/UE algorithm.
The arguments are:

j name of the optimization algorithm

3.53.16 (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(optuereadrestart)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to retrieve the size of the decision space
The arguments are:

j name of the optimization algorithm

rstflag flag indicating the type of optue start (cold/restart)

3.53.17 registerpeobssetup (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registerpeobssetup)(char *j, void (*func)(void*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for handling observation data for optimization.
The arguments are:

j name of the PE obs source

3.53.18 setuppeobsspace (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(setuppeobsspace)(char *j, void *obs,int len)
```

DESCRIPTION:

Invokes the routine from the registry to set up structures for handling observation data for optimization
The arguments are:

j name of the PE obs source

obs ESMF state that contain the observations

3.53.19 registerpeobsreset (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registerpeobsreset)(char *j, void (*func)(void*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for handling observation data for optimization.
The arguments are:

j name of the PE obs source

3.53.20 resetpeobsspace (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(resetpeobsspace)(char *j, void *obs,int len)
```

DESCRIPTION:

Invokes the routine from the registry to reset structures for each observation data for optimization
The arguments are:

j name of the PE obs source

obs ESMF state that contain the observations

3.53.21 registergetpeobs (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registergetpeobs)(char *j, void (*func)(void*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for reading observation data for optimization
The arguments are:

j name of the PE obs source

3.53.22 getpeobs (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(getpeobs)(char *j, void *peobs,int len)
```

DESCRIPTION:

Invokes the routine from the registry to set up structures for reading observation data for optimization
The arguments are:

j name of the PE observation source

peobs ESMF state that contain the observations

3.53.23 registerwritepeobs (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registerwritepeobs)(char *j, void (*func)(void*),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that writes the processed observations (objective space) to disk
The arguments are:

j name of the PE obs source

3.53.24 writepeobs (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(writepeobs)(char *j, void *peobs,int len)
```

DESCRIPTION:

Invokes the routine from the registry to set up structures for writing observation data for optimization
The arguments are:

j name of the PE obs source

peobs ESMF state that contain the observations

3.53.25 registerinitobjfunctype (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registerinitobjfunctype)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for initializing the method of objective function evaluation
The arguments are:

j name of the objective function method

3.53.26 objectivefunctypeinit (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(objectivefunctypeinit)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to set up structures for initializing the method of objective function evaluation
The arguments are:

j name of the objective function method

3.53.27 registercomputeobjfunctype (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registercomputeobjfunctype)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that computes objective function based on the desired method
The arguments are:

j name of the objective function method

3.53.28 computeobjectivefunctype (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(computeobjectivefunctype)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to invoke an objective function method
The arguments are:

j name of the objective function method

3.53.29 registerupdateobjfunctype (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registerupdateobjfunctype)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that updates objective function based on the desired method
The arguments are:

j name of the objective function method

3.53.30 updateobjectivefunctype (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(updateobjectivefunc)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to invoke an objective function method
The arguments are:

j name of the objective function method

3.53.31 registerresetobjfunctype (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registerresetobjfunctype)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that resets objective function
The arguments are:

j name of the objective function method

3.53.32 resetobjectivefunctype (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(resetobjectivefunctype)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to invoke an objective function method
The arguments are:

j name of the objective function method

3.53.33 registeroptuetypeobjfunceval (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuetypeobjfunceval)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Creates an entry in the registry for the routine that evaluates objective function
The arguments are:

j index of the runmode (param estimation, for e.g)

3.53.34 evaluateobjfunction (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(evaluateobjfunction)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to invoke an objective function evaluation
The arguments are:

j name of the objective function evaluation

3.54 Fortran: Module Interface LIS_perturb_FTable (Source File: LIS_perturb_FTable.c)

Function table registries for storing the interface implementations for managing the operations of different perturbation algorithms

---

3.54.1 registerperturbinit (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(registerperturbinit)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to initialize the pertubation scheme.
The arguments are:

j name of the perturbation algorithm .

3.54.2 perturbinit (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(perturbinit)(char *j, int *index,int len)
```

DESCRIPTION:

Invokes the routine from the registry to initialize the pertubation scheme
The arguments are:

j name of the perturbation algorithm

3.54.3 registerperturbsetup (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(registerperturbsetup)(char *j, void (*func)(int*, int*, void*, void*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to initialize the pertubation scheme.
The arguments are:

j name of the perturbation algorithm .

3.54.4 perturbsetup (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(perturbsetup)(char *j, int *index, int *k,  void *base, void *pert,int len)
```

DESCRIPTION:

Invokes the routine from the registry to initialize the pertubation scheme
The arguments are:

j name of the perturbation algorithm

index object of perturbation (forcing, state, obs)

k index of the DA instance

base base state for perturbation

pert output perturbation state

3.54.5 registerperturbmethod (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(registerperturbmethod)(char *j, void (*func)(int*, int*, int*, void*, void*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to invoke the pertubation scheme.
The arguments are:

j name of the perturbation algorithm .

3.54.6 perturbmethod (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(perturbmethod)(char *j, int *index, int *n, int *k, void *base, void *pert,int len)
```

DESCRIPTION:

Invokes the routine from the registry to call the pertubation scheme
The arguments are:

j name of the perturbation algorithm

n index of the nest

index object of perturbation (forcing, state, obs)

k index of the DA instance

base base state for perturbation

pert output perturbation state

3.54.7 registerperturbwriterst (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(registerperturbwriterst)(char *j, void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to write restart files for the perturbations scheme.
The arguments are:

j name of the perturbation algorithm .

3.54.8 writepertrestart (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(writepertrestart)(char *j, int *nest,int len)
```

DESCRIPTION:

Invokes the routine from the registry to write the restart file for the pertubation scheme
The arguments are:

j name of the perturbation algorithm

nest nest index

3.54.9 registerperturbreadrst (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(registerperturbreadrst)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to read restart files for the perturbations scheme.
The arguments are:

j name of the perturbation algorithm .

3.54.10 readpertrestart (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(readpertrestart)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to read the restart file for the pertubation scheme
The arguments are:

j name of the perturbation algorithm

nest nest index

3.55 Fortran: Module Interface LIS_roughness_FTable (Source File: LIS_roughness_FTable.c

Function table registries for storing the interface implementations for managing different sources of greenness fraction data

---

3.55.1 registerroughnesssetup (Source File: LIS_roughness_FTable.c)

INTERFACE:

```
void FTN(registerroughnesssetup)(char *j,void (*func)(int*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to setup roughness data reading routines
The arguments are:

i index of the domain

j index of the greenness data source

3.55.2 roughnesssetup (Source File: LIS_roughness_FTable.c)

INTERFACE:

```
void FTN(roughnesssetup)(char *j,int *n, int len)
```

DESCRIPTION:

Invokes the routine from the registry to setup roughness data reading
The arguments are:

n index of the nest

j index of the greenness data source

3.55.3 registerreadroughness (Source File: LIS_roughness_FTable.c)

INTERFACE:

```
void FTN(registerreadroughness)(char *j,void (*func)(int*, void*, void*, float*, float*),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to read roughness data
The arguments are:

i index of the domain

j index of the greenness data source

3.55.4 readroughness (Source File: LIS_roughness_FTable.c)

INTERFACE:

```
void FTN(readroughness)(char *j,int *n, void *time1, void *time2, float *array1, float *array2, int len
```

DESCRIPTION:

Invokes the routine from the registry to reading roughness data
The arguments are:

n index of the nest

j index of the greenness data source

time month

array pointer to the greenness data

3.56 Fortran: Module Interface LIS_routing_FTable (Source File: LIS_routing_FTable.c)

Function table registries for storing the interface implementations for managing the operations of different routing models.

3.56.1 registerroutinginit (Source File: LIS_routing_FTable.c)

Creates an entry in the registry for the routine to perform routing model initialization

INTERFACE:

```
void FTN(registerroutinginit)(char *j, void (*func)(),int len)
```

3.56.2 routinginit (Source File: LIS_routing_FTable.c)

INTERFACE:

```
void FTN(routinginit)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform routing model initialization

i index of the routing model

3.56.3 registerroutingreadrestart (Source File: LIS_routing_FTable.c)

Creates an entry in the registry for the routine to perform routing model read restart.

INTERFACE:

```
void FTN(registerroutingreadrestart)(char *j, void (*func)(),int len)
```

3.56.4 routingreadrestart (Source File: LIS_routing_FTable.c)

INTERFACE:

```
void FTN(routingreadrestart)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform routing model initialization

i index of the routing model

3.56.5 registerroutingrun (Source File: LIS_routing_FTable.c)

Creates an entry in the registry for the routine to perform routing model run

INTERFACE:

```
void FTN(registerroutingrun)(char *j, void (*func)(int*),int len)
```

3.56.6 routingrun (Source File: LIS_routing_FTable.c)

INTERFACE:

```
void FTN(routingrun)(char *j, int *n,int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform routing model initialization

i index of the routing model

3.56.7 registerroutingoutput (Source File: LIS_routing_FTable.c)

Creates an entry in the registry for the routine to perform routing model output

INTERFACE:

```
void FTN(registerroutingoutput)(char *j, void (*func)(int*),int len)
```

3.56.8 routingoutput (Source File: LIS_routing_FTable.c)

INTERFACE:

```
void FTN(routingoutput)(char *j, int *n,int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform routing model initialization

i index of the routing model

3.56.9 registerroutingwriterestart (Source File: LIS_routing_FTable.c)

Creates an entry in the registry for the routine to perform routing model writerestart

INTERFACE:

```
void FTN(registerroutingwriterestart)(char *j, void (*func)(int*),int len)
```

3.56.10 routingwriterestart (Source File: LIS_routing_FTable.c)

INTERFACE:

```
void FTN(routingwriterestart)(char *j, int *n,int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform routing model initialization

i index of the routing model

3.57 Fortran: Module Interface LIS_runmode_FTable (Source File: LIS_runmode_FTable.c)

Function table registries for storing the interface implementations of different running modes in LIS

3.57.1 registerlisinit (Source File: LIS_runmode_FTable.c)

INTERFACE:

```
void FTN(registerlisinit)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Makes an entry in the registry for the LIS initialization method for a certain running mode
The arguments are:

j name of the running mode

3.57.2 lisinit (Source File: LIS_runmode_FTable.c)

INTERFACE:

```
void FTN(lisinit)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform the LIS initialization for the specified running mode
The arguments are:

j name of the running mode

3.57.3 registerlisrun (Source File: LIS_runmode_FTable.c)

INTERFACE:

```
void FTN(registerlisrun)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Makes an entry in the registry for the LIS run method for a certain running mode.
The arguments are:

j name of the running mode

3.57.4 lisrun (Source File: LIS_runmode_FTable.c)

INTERFACE:

```
void FTN(lisrun)(char *j,int len)
```

DESCRIPTION:

Invokes the LIS run method from the registry for the specified running mode
The arguments are:

j name of the running mode

3.57.5 registerlisfinalize (Source File: LIS_runmode_FTable.c)

INTERFACE:

```
void FTN(registerlisfinalize)(char *j, void (*func)(),int len)
```

DESCRIPTION:

Makes an entry in the registry for the routine to perform LIS finalization for the specified running mode
The arguments are:

j name of the running mode

3.57.6 lisfinalize (Source File: LIS_runmode_FTable.c)

INTERFACE:

```
void FTN(lisfinalize)(char *j,int len)
```

DESCRIPTION:

Invokes the routine from the registry to perform LIS finalization call for the specified running mode
The arguments are:

j name of the running mode

Part IV
Plugin Interfaces in LIS

4 Plugin interfaces in LIS

The plugin interfaces in LIS are the functional abstractions to represent various user-defined components in LIS. The plugins contain the hot spots or extensible interfaces for incorporating new domains, running modes, LSMS, meteorological analyses, data assimilation tools, and parameters. These adaptable interfaces in LIS enable the reuse of the broad set of data, high performance computing, data management, visualization tools, and the land modeling infrastructure in LIS.

4.1 Fortran: Module Interface LIS_DAobs_pluginMod (Source File: LIS_DAobs_pluginMod.

This module contains the definition of the functions that are used to read observation data for data assimiliation. The user defined functions are incorporated into the appropriate registry to be later invoked through generic calls.

REVISION HISTORY:

```
27 Feb 2005;   Sujay Kumar  Initial Specification
```

4.1.1 LIS_DAobs_plugin (Source File: LIS_DAobs_pluginMod.F90)

This is a plugin point for introducing routines to handle the observation data for assimilation.As explaind in the `dataassim_module`, there are three different abstractions associated with the data assimilation implementation. The implementations defined in this subroutine complete the "wirings" required to complete the observation data-related abstractions. Other required interfaces defined in `lsmda_pluginMod` and `dataassim_pluginMod` should be completed for a successful implementation.
The following interfaces should be implemented in this routine.

Setup Initialization of data and memory structures (to be registered using `registerreaddaobssetup` and later called through `daobssetup`)

read observations Routines to read the observation data and perform any spatial transformation. (to be registered using `registerreaddaobs` and later called through `readobservations`)

get number of selected observations routines to retrieve the number of selected observations for the selected modeling point. (to be registered using `registergetnso` and later called through `getselctedobsnumber`

The user-defined functions are included in the registry using a single index. For example, consider an instance where soil moisture assimilation using TMI soil moisture data is conducted. The methods should be defined in the registry as follows, if the index of the 'assimilation set' (assimilating TMI to update soil moisture variables) is defined to be 1

```
call registerdaobssetup(1,TMIsmobs_setup)
call registerreaddaobs(1,read_TMIsm)
call registergetnso(1,getNSO_TMIsm)
```

The functions registered above are invoked using generic calls as follows:

```
call daobssetup(1)       -  calls TMIsmobs_setup
call readobservations(1) -  calls read_TMIsm
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call daobssetup(lis%daset, )
call readobservations(lis%daset)
```

where *lis%daset* is set through the configuration utility, enabling the user to make selections at runtime.

INTERFACE:

```
subroutine LIS_DAobs_plugin
```

4.2 Fortran: Module Interface LIS_ObjFunc_pluginMod (Source File: LIS_ObjFunc_pluginM

This module contains the definition of the functions that specify different methods (metrics) to compute objective function for use in optimization.
The user defined functions are incorporated into the appropriate registry to be later invoked through generic calls.

REVISION HISTORY:

```
16 Jul 2009;   Sujay Kumar  Initial Specification
```

4.2.1 LIS_ObjFunc_plugin (Source File: LIS_ObjFunc_pluginMod.F90)

This is a plugin point for introducing different methods for objective function computations (in parameter estimation)

INTERFACE:

```
subroutine LIS_ObjFunc_plugin
```

4.3 Fortran: Module Interface LIS_PEobs_pluginMod (Source File: LIS_PEobs_pluginMod.F

This module contains the definition of the functions that are used to read observation data that constitute the observation space for parameter estimation. The user defined functions are incorporated into the appropriate registry to be later invoked through generic calls.

REVISION HISTORY:

```
16 Jul 2009;   Sujay Kumar  Initial Specification
```

4.3.1 LIS_PEobs_plugin (Source File: LIS_PEobs_pluginMod.F90)

This is a plugin point for introducing routines to handle the data that constitute the observation space for parameter estimation.

INTERFACE:

```
subroutine LIS_PEobs_plugin
```

4.4 Fortran: Module Interface LIS_RTM_pluginMod (Source File: LIS_RTM_pluginMod.F90

This module contains the definition of the functions used for introducing new RTMs (Radiative Transfer Models) for use in LIS

REVISION HISTORY:

```
21 Mar 09    Sujay Kumar  Initial Specification
20 Oct 10    Yudong Tian  Added CRTM2 and CRTM2 EMonly
08 Feb 11    Yudong Tian  Added CMEM3
```

4.4.1 LIS_RTM_plugin (Source File: LIS_RTM_pluginMod.F90)

INTERFACE:

```
subroutine LIS_RTM_plugin
```

4.5 Fortran: Module Interface LIS_RTMoptue_pluginMod (Source File: LIS_RTMoptue_plu

This module contains the definition of the functions that need to be defined to enable the use of a land surface model in parameter estimation setup.

REVISION HISTORY:

```
 16 Jul 09    Sujay Kumar  Initial Specification

implicit none

PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
PUBLIC :: LIS_RTMoptue_plugin
```

4.5.1 LIS_RTMoptue_plugin (Source File: LIS_RTMoptue_pluginMod.F90)

This is a custom-defined plugin point for introducing a new RTM in a parameter estimation mode. The interface mandates that a number of routines be implemented and registered for each of the RTM that is used in a parameter estimation mode.

INTERFACE:

```
subroutine LIS_RTMoptue_plugin
```

4.6 Fortran: Module Interface LIS_biasEstimation_pluginMod (Source File: LIS_biasEstimat

This module contains the definition of the functions used for defining routines that performs bias correction in data assimilation The user defined functions are incorporated into the appropriate registry to be later invoked through generic calls.

REVISION HISTORY:

```
30 Nov 2007;   Sujay Kumar  Initial Specification
```

4.6.1 LIS_biasEstimation_plugin (Source File: LIS_biasEstimation_pluginMod.F90)

This is a plugin point for introducing a new bias estimation scheme. This is an optional extention to the overall data assimilation implementation.
The following interfaces should be implemented in this routine.

init Initialization of global datastructures (to be registered using `registerbiasestimationinit` and later called through `biasestimationinit`)

setup setup and initialization of data structures specific to each instance of bias estimation algorithm (to be registered using `registerbiasestimationsetup` and later called through `biasestimationsetup`)

bias estimation computation Routines that compute bias correction estimates prior to model propagation (to be registered using `registerbiasestimationcompute` and later called through `computebiascorrection`)

bias estimation update Routines that updates and applies the bias correction estimates. (to be registered using `registerbiasestimationupdate` and later called through `applybiascorrection`)

finalize Routines that cleans up the allocated memory structures

A single index is used to register a routine, corresponding to the type of bias estimation algorithm used. For example, consider an instance where method 'FOO' is employed, and is denoted by a numerical index y. The methods should be defined in the registry as follows.

```
call registerbiasestimationinit(y,FOO_init)
call registerbiasestimationsetup(y,FOO_setup)
call registerbiasestimationcompute(y,FOO_comp)
call registerbiasestimationupdate(y,FOO_update)
call registerbiasestimationfinalize(y,FOO_final)
```

The functions registered above are invoked using generic calls as follows:

```
call biasestimationinit(y) -  calls FOO_init
call biasestimationsetup(y) - calls FOO_run
call computebiascorrection(y) - calls FOO_comp
call applybiascorrection(y) - calls FOO_update
call biasestimationfinalize(y) - calls FOO_final
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call biasestimationinit(LIS_rc%biasalg(j))
call biasestimationsetup(LIS_rc%biasalg(j))
```

where $LIS_rc\%biasalg$ is set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_biasEstimation_plugin
```

4.7 Fortran: Module Interface LIS_dataassim_pluginMod.F90 (Source File: LIS_dataassim_p

This module contains the definition of the functions used for defining routines that performs data assimilation. The user defined functions are incorporated into the appropriate registry to be later invoked through generic calls.

REVISION HISTORY:

```
27 Feb 2005;   Sujay Kumar  Initial Specification
```

4.7.1 LIS_dataassim_plugin (Source File: LIS_dataassim_pluginMod.F90)

This is a plugin point for introducing a new data assimilation scheme. As explaind in the `LIS_dataassim_module`, there are three different abstractions associated with the data assimilation implementation. The implementations defined in this subroutine complete the "wirings" required to complete the data assimilation algorithm-related abstractions. Other required interfaces defined in `lsmda_pluginMod` and `dataobs_pluginMod` should be completed to complete a successful implementation.
The following interfaces should be implemented in this routine.

init Initialization of data and memory structures (to be registered using `registerdainit` and later called through `dataassiminit`)

Assimilate Routines to perform data assimilation using observations. (to be registered using `registerassim` and later called through `assimilate`)

Output Routines to perform data assimilation related output (to be registered using `registerdaoutput` and later called through `daoutput`)

Finalize Routines to perform data assimilation related cleanups (to be registered using `registerdafinalize` and later called throuh `dafinalize`)

A single index is used to register a routine, corresponding to the type of data assimilation algorithm used. For example, consider an instance where Ensemble Kalman Filter (EnKF) algorithm is employed, and is denoted by an index 3. The methods should be defined in the registry as follows.

```
call registerdainit(3,enkf_init)
call registerassim(3,enkf_assim)
call registerdaoutput(3,enkf_output)
call registerdafinalize(3,enkf_final)
```

The functions registered above are invoked using generic calls as follows:

```
call dainit(3) -  calls enkf_init
call assimilate(3) - calls enkf_assim
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call dainit(lis%daalg)
call assimilate(lis%daalg)
```

where *lis%daalg* is set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_dataassim_plugin
```

4.8 Fortran: Module Interface LIS_domain_pluginMod (Source File: LIS_domain_pluginMod

This module contains the definition of the functions used for defining routines that initialize various LIS-domains. The user defined functions are incorporated into the appropriate registry to be later invoked through generic calls.

REVISION HISTORY:

```
17 Feb 2004;   Sujay Kumar  Initial Specification
27 Jan 2014;   Shugong Wang Put HRAP from LDT by Kristi
```

4.8.1 LIS_domain_plugin (Source File: LIS_domain_pluginMod.F90)

This is a plugin point for introducing a new LIS-domain. The interface mandates that the following interfaces be implemented and registered for each LIS-domain.

read input Routines to read domain specific options and accordingly set up domain decomposition. (to be registered using `registerinput` and later invoked through `readinput` method)

create grid and tile spaces Routines to create the grid and tile spaces (to be registered using `registerdomain` and later invoked through `makedomain` method)

The user-defined functions are included in the registry using a single index. For example, consider a domain definition using lat/lon projection with a SW to NE data ordering. The methods should be defined in the registry as follows, if the index of the domain is defined to be 1.

```
call registerinput(1,readinput_latlon)
call registerdomain(1,createtiles_latlon)
```

The functions registered above are invoked using generic calls as follows:

```
call readinput(1)   - calls readinput_latlon
call makedomain(1)  - calls createtiles_latlon
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call readinput(lis%domain)
call makedomain(lis%domain)
```

where *lis%domain* is set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_domain_plugin
  use LIS_pluginIndices
```

4.9 Fortran: Module Interface LIS_irrigationmodel_pluginMod (Source File: LIS_irrigationmodel_pluginMod.F90)

REVISION HISTORY:

```
11 Nov 2012    Sujay Kumar  Initial Specification
```

4.9.1 LIS_irrigationmodel_plugin (Source File: LIS_irrigationmodel_pluginMod.F90)

INTERFACE:

```
subroutine LIS_irrigationmodel_plugin
```

4.10 Fortran: Module Interface LIS_lakemodel_pluginMod (Source File: LIS_lakemodel_plug

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 LAKEMODELs used in LIS.

REVISION HISTORY:

```
09 Oct 03    Sujay Kumar  Initial Specification
22 May 13    Shugong Wang Add FLake 1.0
```

4.10.1 LIS_lakemodel_plugin (Source File: LIS_lakemodel_pluginMod.F90)

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

Initialization Definition of LAKEMODEL variables (to be registered using `registerlakemodelinit` and later called using `lakemodelinit`)

Setup Initialization of parameters (to be registered using `registerlakemodelsetup` and later called using `lakemodelsetup`)

Run Routines to execute LAKEMODEL on a single gridcell for single timestep (to be registered using `registerlakemodelrun` and later called using `lakemodelrun`)

Read restart Routines to read a restart file for an LAKEMODEL run (to be registered using `registerlakemodelrestart` and later called using `lakemodelrestart`)

Output Routines to write output (to be registered using `registerlakemodeloutput` and later called using `lakemodeloutput`)

Forcing transfer to model tiles Routines to transfer an array of given forcing to model tiles (to be registered using `registerlakemodelf2t` and later called using `lakemodelf2t`)

Write restart Routines to write a restart file (to be registered using `registerlakemodelwrst` and later called using `lakemodelwrst`)

Finalize Routines to cleanup LAKEMODEL data structures (to be registered using `registerlakemodelfinalize` and later called using `lakemodelfinalize`)

The user-defined functions are included in the registry using a single index. For example, consider the Noah LAKEMODEL is incorporated in the registry with an index of 1 and is invoked later by the following calls

```
call registerlakemodelinit(1,noah_lakemodel_ini)
call registerlakemodelsetup(1,noah_setup)
call registerlakemodelf2t(1,noah_f2t)
call registerlakemodelrun(1,noah_main)
call registerlakemodelrestart(1,noahrst)
call registerlakemodeldynsetup(1,noah_dynsetup)
```

```
call registerlakemodeloutput(1,noah_output)
call registerlakemodelwrst(1,noah_writerst)
call registerlakemodelfinalize(1,noah_finalize)
```

The functions registered above are invoked using generic calls as follows:

```
call lakemodelinit(1)      - calls noah_lakemodel_ini
call lakemodelsetup(1)    - calls noah_setup
call lakemodelf2t(1)      - calls noah_f2t
call lakemodelrun(1)      - calls noah_main
call lakemodeldynsetup(1) - calls noah_dynsetup
call lakemodeloutput(1)   - calls noah_output
call lakemodelwrst(1)     - calls noah_writerst
call lakemodelfinalize(1) - calls noah_finalize
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call lakemodelinit(lis%lakemodel)
call lakemodelsetup(lis%lakemodel)
call lakemodelf2t(lis%lakemodel)
call lakemodelrun(lis%lakemodel)
call lakemodeldynsetup(lis%lakemodel)
call lakemodeloutput(lis%lakemodel)
call lakemodelwrst(lis%lakemodel)
call lakemodelfinalize(lis%lakemodel)
```

where *lis%lakemodel* is set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_lakemodel_plugin
```

4.11 Fortran: Module Interface LIS_landslidemodel_pluginMod (Source File: LIS_landslidemodel_pluginMod.F90)

This module contains the definition of the functions used for LIS initialization, execution, and finalization for different running modes in LIS

REVISION HISTORY:

```
21 Oct 05    Sujay Kumar  Initial Specification
```

4.11.1 LIS_landslidemodel_plugin (Source File: LIS_landslidemodel_pluginMod.F90)

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

Initialization Defining the init routines needed for each running mode in LIS. (to be registered using `registerlisinit` and later called using `lisinit`)

Run Define the execution routines needed for each running mode in LIS (to be registered using `registerlisrun` and later called using `lisrun`)

Finalize Define the finalize routines needed for each LIS running mode. (to be registered using `registerlisfinalize` and later called using `lisfinalize`)

The user-defined functions are included in the registry using a single index. For example, consider the definition of a retrospective running mode in LIS. The methods should be defined in the registry as follows, if the index of the mode is defined to be 1.

```
call registerlisinit(1,lisinit_retrospective)
call registerlisrun(1,lisrun_retrospective)
call registerlisfinalize(1,lisfinal_retrospective)
```

The functions registered above are invoked using generic calls as follows:

```
call lisinit(1)  - calls lisinit_retrospective
call lisrun(1)   - calls lisrun_retrospective
call lisfinal(1)   - calls lisfinal_retrospective
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call lisinit(lis%landslidemodel)
call lisrun(lis%landslidemodel)
call lisfinal(lis%landslidemodel)
```

where *lis%landslidemodel* is set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_landslidemodel_plugin
```

4.12 Fortran: Module Interface LIS_lsm_pluginMod (Source File: LIS_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
```

4.12.1 LIS_lsm_plugin (Source File: LIS_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 `registerlsminit` and later called using `lsminit`)

Setup Initialization of parameters (to be registered using `registerlsmsetup` and later called using `lsmsetup`)

Run Routines to execute LSM on a single gridcell for single timestep (to be registered using `registerlsmrun` and later called using `lsmrun`)

Read restart Routines to read a restart file for an LSM run (to be registered using `registerlsmrestart` and later called using `lsmrestart`)

Forcing transfer to model tiles Routines to transfer an array of given forcing to model tiles (to be registered using `registerlsmf2t` and later called using `lsmf2t`)

Write restart Routines to write a restart file (to be registered using `registerlsmwrst` and later called using `lsmwrst`)

Finalize Routines to cleanup LSM data structures (to be registered using `registerlsmfinalize` and later called using `lsmfinalize`)

The user-defined functions are included in the registry using a single index. For example, consider the Noah LSM is incorporated in the registry with an index of 1 and is invoked later by the following calls

```
call registerlsminit(1,noah_lsm_ini)
call registerlsmsetup(1,noah_setup)
call registerlsmf2t(1,noah_f2t)
call registerlsmrun(1,noah_main)
call registerlsmrestart(1,noahrst)
call registerlsmdynsetup(1,noah_dynsetup)
call registerlsmwrst(1,noah_writerst)
call registerlsmfinalize(1,noah_finalize)
```

The functions registered above are invoked using generic calls as follows:

```
call lsminit(1)      - calls noah_lsm_ini
call lsmsetup(1)    - calls noah_setup
call lsmf2t(1)      - calls noah_f2t
call lsmrun(1)      - calls noah_main
call lsmdynsetup(1) - calls noah_dynsetup
call lsmwrst(1)     - calls noah_writerst
call lsmfinalize(1) - calls noah_finalize
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call lsminit(lis%lsm)
call lsmsetup(lis%lsm)
call lsmf2t(lis%lsm)
call lsmrun(lis%lsm)
call lsmdynsetup(lis%lsm)
call lsmwrst(lis%lsm)
call lsmfinalize(lis%lsm)
```

where *lis%lsm* is set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_lsm_plugin
```

4.13 Fortran: Module Interface LIS_lsmcpl_pluginMod (Source File: LIS_lsmcpl_pluginMod.

This module contains the definition of the functions that need to be defined to enable the use of a land surface model in coupled land-atmosphere setups.

REVISION HISTORY:

```
09 Oct 07    Sujay Kumar  Initial Specification
```

4.13.1 LIS_lsmcpl_plugin (Source File: LIS_lsmcpl_pluginMod.F90)

This is a custom-defined plugin point for introducing a LSM in a coupled land-atmosphere setup. The interface mandates that the following routines be implemented and registered for each of the LSM that is used.
If used for a coupled run with an atmospheric component:

Specify an export state Routines to specify the variables to be sent to the atmos. component (to be registered using `registerlsmsetexport` and later called using `lsmsetexport`)

The user-defined functions are included in the registry using two indices. For example, consider the Noah LSM used in a mode coupled to WRF, the setexport method is incorporated in the registry with an index of 1 corresponding to Noah and 3 corresponding to the runmode coupled to WRF as follows:

```
call registerlsmsetexport(1,3,noah_setwrfexport)
```

The function registered above is invoked using a generic call as follows:

```
call lsmsetexport(1,3)      - calls noah_setwrfexport
```

In the LIS code, the above call is typically invoked in the following manner.

```
call lsmsetexport(lis%lsm, lis%runmode)
```

where *lis%lsm* and utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_lsmcpl_plugin
```

4.14 Fortran: Module Interface LIS_lsmda_pluginMod (Source File: LIS_lsmda_pluginMod.F

This module contains the definition of the functions that need to be defined to enable the use of a land surface model in a data assimilation setup.

REVISION HISTORY:

```
29 Mar 07    Sujay Kumar  Separated from the lsm module implementation.
```

4.14.1 LIS_lsmda_plugin (Source File: LIS_lsmda_pluginMod.F90)

This is a custom-defined plugin point for introducing a new LSM in a data assimilation mode. The interface mandates that a number of routines be implemented and registered for each of the LSM that is used in a data assimilation setup. Currently two algorithms are supported, Direct Insertion (DI), and Ensemble Kalman Filter (EnKF).
For use with EnKF, the following routines need to be implemented.

Return the state prognostic variables Routine that retrieves the specified state prognostic variable(s) (to be registered using `registerlsmdagetstatevar` and later called through `lsmdagetstatevar` method)

Set the state prognostic variables Routine that sets the specified state prognostic variable(s) (to be registered using `registerlsmdasetstatevar` and later called through `lsmdasetstatevar` method)

Retrieve "Obspred" Routine that retrieves the model's estimate of observations. (to be registered using `registerlsmdagetobspred` and later called through `lsmdagetobspred` method)

QC the LSM state Routine that QCs the given LSM state for physical consistency. (to be registered using `registerlsmdaqcstate` and later called through `lsmdaqcstate` method)

QC the OBS state Routine that QCs the given OBS state based on land model states (e.g. filter out observations when dense vegetation is present) (to be registered using `registerlsmdaqcobsstate` and later called through `lsmdaqcobsstate` method)

Scale LSM state Scale the LSM state variables in order to change the variables to similar scales so that the matrices are well-conditioned (to be registered using `registerscalelsmstate` and later called through `scalelsmstate` method)

Descale LSM state Descale the LSM state variables from the scaled state. (to be registered using `registerdescalelsmstate` and later called through `descalelsmstate` method)

Update LSM state updates the state variables using the values of the Increment object (note that the set method actually sets the variables to the lsm state, whereas the update method simply changes the LSM_State object). (to be registered using `registerlsmdaupdatestate` and later called through `lsmdaupdatestate` method)

For use with DI, the following routines are required.

Return the state prognostic variables Routine that retrieves the specified state prognostic variable(s) (to be registered using `registerlsmdagetstatevar` and later called through `lsmdagetstatevar` method)

Set the state prognostic variables Routine that sets the specified state prognostic variable(s) (to be registered using `registerlsmdasetstatevar` and later called through `lsmdasetstatevar` method)

Transform Observations Routine that transforms observations to the LSM state space (could be as simple as a unit conversion). (to be registered using `registerlsmdaobstransform` and later called through `lsmdaobstransform` method)

Map observations to LSM space Routine that maps observations to the LSM state variables. (to be registered using `registerlsmdamapobstolsm` and later called through `lsmdamapobstolsm` method)

The user-defined functions are included in the registry using a user-selected indices. For example, consider the Noah LSM is incorporated in the registry by the following calls. In case of registry functions defined with two indices, the first index refers to Noah LSM and the second index refers to the "assimilation set" (in the following example, soil moisture assimilation using synthetic soil moisture). For the definition of plugin indices, please see LIS_pluginIndices.F90

```
call registerlsmdagetstatevar(1,1,noah271_getsoilm)
call registerlsmdasetstatevar(1,1,noah271_setsoilm)
```

The functions registered above are invoked using generic calls as follows:

```
call lsmdagetstatevar(1,1) - calls noah271_getsoilm
call lsmdasetstatevar(1,1) - calls noah271_setsoilm
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call lsmdagetstatevar(LIS_rc%lsm, LIS_rc%daset)
call lsmdasetstatevar(LIS_rc%lsm, LIS_rc%daset)
call lsmdaobstransform(LIS_rc%lsm, LIS_rc%daset)
```

where *LIS_rc%lsm* and *LIS_rc%daset* are set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_lsmda_plugin
```

4.15 Fortran: Module Interface LIS_lsmirrigation_pluginMod (Source File: LIS_lsmirrigation

REVISION HISTORY:

```
 13 Nov 2012    Sujay Kumar  Initial Specification

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
PUBLIC :: LIS_lsmirrigation_plugin
```

4.15.1 LIS_lsmirrigation_plugin (Source File: LIS_lsmirrigation_pluginMod.F90)

INTERFACE:

```
subroutine LIS_lsmirrigation_plugin
```

4.16 Fortran: Module Interface LIS_lsmoptue_pluginMod (Source File: LIS_lsmoptue_plugin

This module contains the definition of the functions that need to be defined to enable the use of a land surface model in parameter estimation setup.

REVISION HISTORY:

```
 16 Jul 09    Sujay Kumar  Initial Specification

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
PUBLIC :: LIS_lsmoptue_plugin
```

4.16.1 LIS_lsmoptue_plugin (Source File: LIS_lsmoptue_pluginMod.F90)

This is a custom-defined plugin point for introducing a new LSM in a parameter estimation mode. The interface mandates that a number of routines be implemented and registered for each of the LSM that is used in a parameter estimation mode.

INTERFACE:

```
subroutine LIS_lsmoptue_plugin
```

4.17 Fortran: Module Interface LIS_lsmrouting_pluginMod (Source File: LIS_lsmrouting_plu

REVISION HISTORY:

```
 16 Jul 09    Sujay Kumar  Initial Specification

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
PUBLIC :: LIS_lsmrouting_plugin
```

4.17.1 LIS_lsmrouting_plugin (Source File: LIS_lsmrouting_pluginMod.F90)

INTERFACE:

```
subroutine LIS_lsmrouting_plugin
```

4.18 Fortran: Module Interface LIS_lsmrtm_pluginMod (Source File: LIS_lsmrtm_pluginMo

REVISION HISTORY:

```
24 Mar 09    Sujay Kumar  Initial Specification
15 Sep 10    Yudong Tian; Reistered for CRTM2
20 Oct 10    Yudong Tian; Reistered for CRTM2 EMonly
08 Feb 11    Yudong Tian; Reistered for CMEM3
```

4.18.1 LIS_lsmtm_plugin (Source File: LIS_lsmrtm_pluginMod.F90)

INTERFACE:

```
subroutine LIS_lsmrtm_plugin
```

4.19 Fortran: Module Interface LIS_metforcing_pluginMod (Source File: LIS_metforcing_plu

This module contains the definition of the functions used for incorporating a new met forcing scheme. The user defined functions are incorporated into the appropriate registry to be later invoked through generic calls.

REVISION HISTORY:

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

4.19.1 LIS_metforcing_plugin (Source File: LIS_metforcing_pluginMod.F90)

INTERFACE:

```
subroutine LIS_metforcing_plugin
```

DESCRIPTION:

This is a plugin point for introducing a new met forcing scheme. The interface mandates that the following interfaces be implemented and registered for each met forcing scheme.

retrieval of forcing data Routines to retrieve forcing data and to interpolate them. (to be registered using `registerget` and later invoked through `retrieveforcing` method)

definition of native domain Routines to define the native domain (to be registered using registerinitmetforc and later invoked through `initmetforc` method)

temporal interpolation Interpolate forcing data temporally. (to be registered using `registertimeinterp` and later invoked through `timeinterp` method)

The index used in the register calls should be used to select the appropriate met forcing scheme. For example, assume that the GDAS forcing scheme is incorporated in the registry by the following calls.

```
call registerget(1,getgdas)
call registerinitmetforc(1,initmetforcGDAS)
call registertimeinterp(1,time_interp_gdas)
call registerforcingfinal(1,gdasforcing_finalize)
```

The index used here to register these functions is 1. To invoke these methods, the corresponding calls will be:

```
call retrieveforcing(1)  - calls getgdas
call initmetforc(1)  - calls initmetforcGDAS
call timeinterp(1)  - calls time_interp_gdas
call forcingfinalize(1) - calls gdasforcing_finalize
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call retrieveforcing(lis%force,n)
call initmetforc(lis%force)
call timeinterp(lis%force,n)
call forcingfinalize(lis%force)
```

where $lis\%force$ is set through the congiguration utility, enabling the user to select any of the met forcing schemes, at runtime.

4.20 Fortran: Module Interface LIS_openwatermodel_pluginMod (Source File: LIS_openwatermodel_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 OPENWATERs used in LIS.

REVISION HISTORY:

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

4.20.1 LIS_openwatermodel_plugin (Source File: LIS_openwatermodel_pluginMod.F90)

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

Initialization Definition of OPENWATER variables (to be registered using `registeropenwaterinit` and later called using `openwaterinit`)

Setup Initialization of parameters (to be registered using `registeropenwatersetup` and later called using `openwatersetup`)

Run Routines to execute OPENWATER on a single gridcell for single timestep (to be registered using `registeropenwaterrun` and later called using `openwaterrun`)

Read restart Routines to read a restart file for an OPENWATER run (to be registered using `registeropenwaterrestart` and later called using `openwaterrestart`)

Output Routines to write output (to be registered using `registeropenwateroutput` and later called using `openwateroutput`)

Forcing transfer to model tiles Routines to transfer an array of given forcing to model tiles (to be registered using `registeropenwaterf2t` and later called using `openwaterf2t`)

Write restart Routines to write a restart file (to be registered using `registeropenwaterwrst` and later called using `openwaterwrst`)

Finalize Routines to cleanup OPENWATER data structures (to be registered using `registeropenwaterfinalize` and later called using `openwaterfinalize`)

The user-defined functions are included in the registry using a single index. For example, consider the Noah OPENWATER is incorporated in the registry with an index of 1 and is invoked later by the following calls

```
call registeropenwaterinit(1,noah_openwatermodel_ini)
call registeropenwatersetup(1,noah_setup)
call registeropenwaterf2t(1,noah_f2t)
call registeropenwaterrun(1,noah_main)
call registeropenwaterrestart(1,noahrst)
call registeropenwaterdynsetup(1,noah_dynsetup)
call registeropenwateroutput(1,noah_output)
call registeropenwaterwrst(1,noah_writerst)
call registeropenwaterfinalize(1,noah_finalize)
```

The functions registered above are invoked using generic calls as follows:

```
call openwaterinit(1)      - calls noah_openwatermodel_ini
call openwatersetup(1)    - calls noah_setup
call openwaterf2t(1)      - calls noah_f2t
call openwaterrun(1)      - calls noah_main
call openwaterdynsetup(1) - calls noah_dynsetup
call openwateroutput(1)   - calls noah_output
call openwaterwrst(1)     - calls noah_writerst
call openwaterfinalize(1) - calls noah_finalize
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call openwaterinit(lis%openwater)
call openwatersetup(lis%openwater)
call openwaterf2t(lis%openwater)
call openwaterrun(lis%openwater)
call openwaterdynsetup(lis%openwater)
call openwateroutput(lis%openwater)
call openwaterwrst(lis%openwater)
call openwaterfinalize(lis%openwater)
```

where *lis%openwater* is set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_openwatermodel_plugin
```

4.21 Fortran: Module Interface LIS_optUEAlgorithm_pluginMod (Source File: LIS_optUEAlgorithm_pluginMod.F90)

This module contains the definition of the functions used for defining optimization /uncertainty estimation algorithm algorithms The user defined functions are incorporated into the appropriate registry to be later invoked through generic calls.

REVISION HISTORY:

```
2 Feb 2008;   Sujay Kumar  Initial Specification
```

4.21.1 LIS_optUEAlgorithm_plugin (Source File: LIS_optUEAlgorithm_pluginMod.F90)

This is a plugin point for introducing a new optimization /uncertainty estimation scheme.

INTERFACE:

```
subroutine LIS_optUEAlgorithm_plugin
```

4.22 Fortran: Module Interface LIS_param_pluginMod (Source File: LIS_param_pluginMod.

This module contains the definition of the functions used for defining routines to read various sources of parameters maps. The user defined functions are incorporated into the appropriate registry to be later invoked through generic calls.

REVISION HISTORY:

```
11 Dec 03    Sujay Kumar  Initial Specification
17 Jan 2011: David Mocko, added max/min greenness & slope type
```

4.22.1 LIS_laisai_plugin (Source File: LIS_param_pluginMod.F90)

This is a plugin point for introducing new LAI/SAI datasets. The interface mandates that the following routines be implemented and registered for each parameter data source.

read the LAI data Routines to retrieve the LAI data (to be registered using `registerreadlai` and later called using the generic `readlai` method)

read the SAI data Routines to retrieve the SAI data (to be registered using `registerreadsai` and later called using the generic `readsai` method)

The user-defined functions are included in the registry using two indices. For example, consider the incorporation of the LAI/SAI datasets from the AVHRR source with LIS using a latlon projection. The methods should be defined in the registry as follows if the index of the source is defined to 1 and latlon projection in LIS uses and index of 1

```
call registerreadlai(1,1,read_ll_avhrrlai)
call registerreadsai(1,1,read_ll_avhrrsai)
```

The functions registered above are invoked using generic calls as follows:

```
call readlai(1,1)  - calls read_ll_avhrrlai
call readsai(1,1)  - calls read_ll_avhrrsai
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call readlai(lis%domain,lislaisrc)
call readsai(lis%domain,lislaisrc)
```

where *lis%domain* and *lis%laisrc* are set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_laisai_plugin
```

4.22.2 LIS_gfrac_plugin (Source File: LIS_param_pluginMod.F90)

This is a plugin point for introducing new greenness datasets. The interface mandates that the following routines be implemented and registered for each parameter data source.

read the greenness data Routines to retrieve the greenness data (to be registered using `registerreadgfrac` and later called using the generic `readgfrac` method)

The user-defined functions are included in the registry using two indices. For example, consider the incorporation of the greenness datasets from the NCEP source with LIS using a latlon projection. The methods should be defined in the registry as follows if the index of the source is defined to 1 and latlon projection in LIS uses and index of 1

```
call registerreadgfrac(1,1,read_llgfrac)
```

The functions registered above are invoked using generic calls as follows:

```
call readgfrac(1,1)  - calls read_llgfrac
```

In the LIS code, the above calls are typically invoked in the following manner.

```
 call readgfrac(lis%domain,lis%gfracsrc)
```

where *lis%domain* and *lis%gfracsrc* are set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_gfrac_plugin
```

4.22.3 LIS_roughness_plugin (Source File: LIS_param_pluginMod.F90)

This is a plugin point for introducing new greenness datasets. The interface mandates that the following routines be implemented and registered for each parameter data source.

read the greenness data Routines to retrieve the greenness data (to be registered using `registerreadroughness` and later called using the generic `readroughness` method)

The user-defined functions are included in the registry using two indices. For example, consider the incorporation of the greenness datasets from the NCEP source with LIS using a latlon projection. The methods should be defined in the registry as follows if the index of the source is defined to 1 and latlon projection in LIS uses and index of 1

```
call registerreadroughness(1,1,read_llroughness)
```

The functions registered above are invoked using generic calls as follows:

```
call readroughness(1,1)  - calls read_llroughness
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call readroughness(lis%domain,lis%roughnesssrc)
```

where *lis%domain* and *lis%roughnesssrc* are set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_roughness_plugin
```

4.22.4 LIS_emissivity_plugin (Source File: LIS_param_pluginMod.F90)

This is a plugin point for introducing new greenness datasets. The interface mandates that the following routines be implemented and registered for each parameter data source.

read the greenness data Routines to retrieve the greenness data (to be registered using `registerrademissivity` and later called using the generic `rademissivity` method)

The user-defined functions are included in the registry using two indices. For example, consider the incorporation of the greenness datasets from the NCEP source with LIS using a latlon projection. The methods should be defined in the registry as follows if the index of the source is defined to 1 and latlon projection in LIS uses and index of 1

```
call registerrademissivity(1,1,read_llemissivity)
```

The functions registered above are invoked using generic calls as follows:

```
call rademissivity(1,1)  - calls read_llemissivity
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call rademissivity(lis%domain,lis%emissivitysrc)
```

where *lis%domain* and *lis%emissivitysrc* are set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_emissivity_plugin
```

4.22.5 LIS_alb_plugin (Source File: LIS_param_pluginMod.F90)

This is a plugin point for introducing new albedo datasets. The interface mandates that the following routines be implemented and registered for each parameter data source.

read the max snow albedo data Routines to retrieve the max snow albedo data (to be registered using `registerreadmxsnoalb` and later called using the generic `readmxsnoalb` method)

read the albedo climatology data Routines to retrieve the albedo climatology data (to be registered using `registerreadalbedo` and later called using the generic `readalbedo` method)

The user-defined functions are included in the registry using two indices. For example, consider the incorporation of the albedo datasets from the NCEP source with LIS using a latlon projection. The methods should be defined in the registry as follows if the index of the source is defined to 1 and latlon projection in LIS uses and index of 1

```
call registerreadmxsnoalb(1,1,read_llmxsnoalb)
call registerreadalbedo(1,1,read_llalbedo)
```

The functions registered above are invoked using generic calls as follows:

```
call readmxsnoalb(1,1)  - calls read_llmxsnoalb
call readalbedo(1,1)    - calls read_llalbedo
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call readmxsnoalb(lis%domain,lis%albedosrc)
call readalbedo(lis%domain,lis%albedoarc)
```

where *lis%domain* and *lis%albedosrc* are set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_alb_plugin
```

4.23 Fortran: Module Interface LIS_perturb_pluginMod (Source File: LIS_perturb_pluginM

This module contains the definition of the functions used for defining routines that perform pertubations of model states, forcing, or parameters. The user defined functions are incorporated into the appropriate registry to be later invoked through generic calls.

REVISION HISTORY:

```
08Jul2005;   Sujay Kumar  Initial Specification
```

INTERFACE:

4.23.1 LIS_perturb_plugin (Source File: LIS_perturb_pluginMod.F90)

This is a plugin point for introducing a new forcing perturbation scheme. The interface mandates that the following interfaces be implemented for each scheme.

Setup Initialization of data and memory structures (to be registered using `registerperturbsetup` and later called using the generic call `perturbinit`)

Forecast Routines to compute perturbations (to be registered using `registerperturb` and later called using the generic call `perturb`)

The user-defined functions are included in the registry using a single index. For example, consider a new scheme called 'FOO' The methods should be defined in the registry as follows, if the index of the scheme is defined to be 1.

```
call registerperturbsetup(1,foo_setup)
call registerperturb(1,foo_method)
```

The functions registered above are invoked using generic calls as follows:

```
call perturbinit(1)  - calls foo_setup
call perturb(1)      - calls foo_method
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call perturbinit(lis%perturb)
call perturb(lis%perturb)
```

where *lis%perturb* is set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_perturb_plugin
```

4.24 Fortran: Module Interface LIS_pluginIndices (Source File: LIS_pluginIndices.F90)

The code in this file provides values of indices used to to register functions in the plugin modules
The index definitions are simply a convention The user may change these options, and the lis.config should be changed appropriately to ensure that the correct function is called at run time

REVISION HISTORY:

```
 23 Oct 2006: Sujay Kumar  Initial Specification
 17 Jan 2011: David Mocko, added max/min greenness & slope type
 08 Feb 2011: Yudong Tian, added cmem3 rtm.
 01 Jun 2012: Sujay Kumar, changed Ids to character strings
 22 May 2013: Shugong Wang, added FLAKE.1.0
 27 Jan 2014: Shugong Wang, added HRAP projection

!-----------------------------------------------------------------------------
! Run modes
!-----------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_retroId     = "retrospective"
   character*50, public,  parameter :: LIS_RTMforwardId  = "RTM forward"
```

```
   character*50, public,  parameter :: LIS_agrmetrunId = "AGRMET ops"
   character*50, public,  parameter :: LIS_wrfcplId    = "WRF coupling"
   character*50, public,  parameter :: LIS_gcecplId    = "GCE coupling"
   character*50, public,  parameter :: LIS_gfscplId    = "GFS coupling"
   character*50, public,  parameter :: LIS_paramEstimRunId = "parameter estimation"
   character*50, public,  parameter :: LIS_smootherDAId    = "ensemble smoother"
!-----------------------------------------------------------------------------
! Domains
!-----------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_latlonId    = "latlon"
   character*50, public,  parameter :: LIS_mercId      = "mercator"
   character*50, public,  parameter :: LIS_lambertId   = "lambert"
   character*50, public,  parameter :: LIS_gaussId     = "gaussian"
   character*50, public,  parameter :: LIS_polarId     = "polar"
   character*50, public,  parameter :: LIS_utmId       = "UTM"
   character*50, public,  parameter :: LIS_catdomainId = "catchment"
   character*50, public,  parameter :: LIS_hrapId      = "hrap"
!-----------------------------------------------------------------------------
! LSMS
!-----------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_templateLSMId = "none"
   character*50, public,  parameter :: LIS_noah271Id   = "Noah.2.7.1"
   character*50, public,  parameter :: LIS_noah32Id    = "Noah.3.2"
   character*50, public,  parameter :: LIS_noah33Id    = "Noah.3.3"
   character*50, public,  parameter :: LIS_clm2Id      = "CLM.2"
   character*50, public,  parameter :: LIS_vic411Id    = "VIC.4.1.1"
   character*50, public,  parameter :: LIS_vic412Id    = "VIC.4.1.2"
   character*50, public,  parameter :: LIS_mosaicId    = "Mosaic"
   character*50, public,  parameter :: LIS_hyssibId    = "HySSIB"
   character*50, public,  parameter :: LIS_sib2Id      = "SiB2"
   character*50, public,  parameter :: LIS_tessId      = "HTESSEL"
   character*50, public,  parameter :: LIS_julesId     = "JULES"
   character*50, public,  parameter :: LIS_cableId     = "CABLE"
   character*50, public,  parameter :: LIS_fasstId     = "FASST"
   character*50, public,  parameter :: LIS_sheelsId    = "SHEELS"
   character*50, public,  parameter :: LIS_clsmf25Id   = "CLSM F2.5"
   character*50, public,  parameter :: LIS_geowrsi2Id  = "GeoWRSI.2"
   character*50, public,  parameter :: LIS_rdhm356lsmId = "RDHM.3.5.6"

!-----------------------------------------------------------------------------
! Lake models
!-----------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_FLAKE1Id   = "FLAKE.1.0"

!-----------------------------------------------------------------------------
! open water models
!-----------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_templateOpenWaterId    = "template open water"

!-----------------------------------------------------------------------------
! met forcings
!-----------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_metForcTemplateId = "none"
   character*50, public,  parameter :: LIS_gdasId            = "GDAS"
   character*50, public,  parameter :: LIS_geosId            = "GEOS"
```

```
   character*50, public,  parameter :: LIS_geos5fcstId       = "GEOS5 forecast"
   character*50, public,  parameter :: LIS_ecmwfId           = "ECMWF"
   character*50, public,  parameter :: LIS_gswp1Id           = "GSWP1"
   character*50, public,  parameter :: LIS_gswp2Id           = "GSWP2"
   character*50, public,  parameter :: LIS_ecmwfreanalId     = "ECMWF reanalysis"
   character*50, public,  parameter :: LIS_agrmetId          = "AGRMET"
   character*50, public,  parameter :: LIS_princetonId       = "PRINCETON"
   character*50, public,  parameter :: LIS_nldas1Id          = "NLDAS1"
   character*50, public,  parameter :: LIS_nldas2Id          = "NLDAS2"

   character*50, public,  parameter :: LIS_gldasId           = "GLDAS"
   character*50, public,  parameter :: LIS_gfsId             = "GFS"
   character*50, public,  parameter :: LIS_merralandId       = "MERRA-Land"

   character*50, public,  parameter :: LIS_cmapId            = "CMAP"

   character*50, public,  parameter :: LIS_TRMM3B42RTId      = "TRMM 3B42RT"
   character*50, public,  parameter :: LIS_TRMM3B42V6Id      = "TRMM 3B42V6"
   character*50, public,  parameter :: LIS_TRMM3B42V7Id      = "TRMM 3B42V7" ! SY
   character*50, public,  parameter :: LIS_cmorphId          = "CPC CMORPH"
   character*50, public,  parameter :: LIS_stg2Id            = "CPC STAGEII"
   character*50, public,  parameter :: LIS_stg4Id            = "CPC STAGEIV"

   character*50, public,  parameter :: LIS_narrId            = "NARR"
   character*50, public,  parameter :: LIS_ALMIPIIId         = "ALMIPII"
   character*50, public,  parameter :: LIS_RFE2DailyId       = "RFE2(daily)"
   character*50, public,  parameter :: LIS_ceopId            = "CEOP"
   character*50, public,  parameter :: LIS_scanId            = "SCAN"
   character*50, public,  parameter :: LIS_armsId            = "ARMS"
   character*50, public,  parameter :: LIS_gdasLSWGId        = "GDAS(LSWG)"
   character*50, public,  parameter :: LIS_d2pcpcarId        = "D2PCPCAR"
   character*50, public,  parameter :: LIS_d2pcpoklId        = "D2PCPOKL"
   character*50, public,  parameter :: LIS_rdhm356Id         = "RDHM.3.5.6"
   character*50, public,  parameter :: LIS_gdas3dId          = "GDAS(3d)"
   character*50, public,  parameter :: LIS_agrradpsId        = "AGRMET radiation (polar stereographic)"
   character*50, public,  parameter :: LIS_BondvilleId       = "Bondville"
   character*50, public,  parameter :: LIS_FASSTsingleId     = "FASST test"
   character*50, public,  parameter :: LIS_TRIGRSseattleId   = "TRIGRS test"
   character*50, public,  parameter :: LIS_snotelId          = "SNOTEL"
   character*50, public,  parameter :: LIS_coopId            = "COOP"
   character*50, public,  parameter :: LIS_rhoneAGGId        = "Rhone AGG"
   character*50, public,  parameter :: LIS_RFE2gdasId        = "RFE2(GDAS bias-corrected)"
   character*50, public,  parameter :: LIS_vicforcingId      = "VIC processed forcing"
   character*50, public,  parameter :: LIS_PALSmetforcId     = "PALS station forcing"
   character*50, public,  parameter :: LIS_PILDASmetforcId   = "PILDAS"
   character*50, public,  parameter :: LIS_USGSPETforcId     = "PET USGS"
   character*50, public,  parameter :: LIS_agrradId          = "AGRMET radiation (latlon)"
   character*50, public,  parameter :: LIS_capaId            = "CaPA"
   character*50, public,  parameter :: LIS_nam242Id          = "NAM242"
   character*50, public,  parameter :: LIS_WRFoutId          = "WRFout"
!-----------------------------------------------------------------------------
! land surface parameters
!-----------------------------------------------------------------------------
!-----------------------------------------------------------------------------
! LAI/SAI sources
```

```
!-------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_modis_RT_laiId  = "MODIS real-time"
   character*50, public,  parameter :: LIS_ALMIPIIlaiId    = "ALMIPII"
!-------------------------------------------------------------------------
! greenness data sources
!-------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_NESDISgfracId ="NESDIS weekly"
   character*50, public,  parameter :: LIS_SPORTgfracId = "SPORT"
   character*50, public,  parameter :: LIS_ALMIPIIgfracId = "ALMIPII"
!-------------------------------------------------------------------------
! roughness data sources
!-------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_ALMIPIIroughnessId = "ALMIPII"
!-------------------------------------------------------------------------
! albedo data sources
!-------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_ALMIPIIalbedoId = "ALMIPII"
!-------------------------------------------------------------------------
! emissivity data sources
!-------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_ALMIPIIemissId = "ALMIPII"
!-------------------------------------------------------------------------
! data assimilation algorithms
!-------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_diId         = "Direct insertion"
   character*50, public,  parameter :: LIS_enkfId       = "EnKF"
   character*50, public,  parameter :: LIS_enksId       = "EnKS"
!-------------------------------------------------------------------------
! perturbation algorithms
!-------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_gmaopertId = "GMAO scheme"
!-------------------------------------------------------------------------
! Assimilation set
! DA variable being updated with observations
!-------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_noobsId             = "none"
   character*50, public,  parameter :: LIS_synsmId             = "Synthetic SM"
   character*50, public,  parameter :: LIS_synsweId            = "Synthetic SWE"
   character*50, public,  parameter :: LIS_synlstId            = "Synthetic LST"
   character*50, public,  parameter :: LIS_multisynsmobsId     = "Synthetic(Multilayer) sm"
   character*50, public,  parameter :: LIS_isccpTskinId        = "ISCCP LST"
   character*50, public,  parameter :: LIS_NASA_AMSREsmobsId   = "AMSR-E(NASA) soil moisture"
   character*50, public,  parameter :: LIS_LPRM_AMSREsmobsId   = "AMSR-E(LPRM) soil moisture"
   character*50, public,  parameter :: LIS_ECVsmobsId          = "ECV soil moisture"
   character*50, public,  parameter :: LIS_WindSatsmobsId      = "Windsat"
   character*50, public,  parameter :: LIS_WindSatCsmobsId     = "Windsat C-band"
   character*50, public,  parameter :: LIS_snodepobsId         = "SNODEP"
   character*50, public,  parameter :: LIS_ANSASWEsnowobsId    = "ANSA SWE"
   character*50, public,  parameter :: LIS_ANSASCFsnowobsId    = "ANSA SCF"
   character*50, public,  parameter :: LIS_ANSASNWDsnowobsId   = "ANSA snow depth"
   character*50, public,  parameter :: LIS_SMMRSNWDsnowobsId   = "SMMR snow depth"
   character*50, public,  parameter :: LIS_SSMISNWDsnowobsId   = "SSMI snow depth"
   character*50, public,  parameter :: LIS_AMSREsweobsId       = "AMSR-E SWE"
!   character*50, public,  parameter :: LIS_AMSREsnowobsId      = "AMSR-E snow" !yliu
   character*50, public,  parameter :: LIS_PMWsnowobsId        = "PMW snow" !yliu
```

```
   character*50, public,  parameter :: LIS_modisscfId          = "MODIS SCF"
   character*50, public,  parameter :: LIS_GRACEtwsobsId       = "GRACE TWS"
   character*50, public,  parameter :: LIS_synLbandTbobsId     = "Synthetic L-band Tb"
   character*50, public,  parameter :: LIS_RT_SMOPSsmobsId     = "RT SMOPS soil moisture"
   character*50, public,  parameter :: LIS_ASCAT_TUWsmobsId    = "ASCAT (TUW) soil moisture"
   character*50, public,  parameter :: LIS_IMSscaobsId = "IMS snow cover"
!-------------------------------------------------------------------------
! Bias Estimation Algorithms
!-------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_gmaobiasId = "Adaptive bias correction"
!-------------------------------------------------------------------------
! Optimization /Uncertainty estimation Algorithms
!-------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_ESOptId ="Enumerated search"
   character*50, public,  parameter :: LIS_LMOptId ="Levenberg marquardt"
   character*50, public,  parameter :: LIS_GAOptId = "Genetic algorithm"
   character*50, public,  parameter :: LIS_SCEUAOptId = "Shuffled complex evolution"
   character*50, public,  parameter :: LIS_MCSIMId = "Monte carlo sampling"
   character*50, public,  parameter :: LIS_RWMCMCId = "Random walk markov chain monte carlo"
   character*50, public,  parameter :: LIS_DEMCId = "Differential evolution markov chain"
   character*50, public,  parameter :: LIS_DEMCzId = "Differential evolution markov chain z"
!-------------------------------------------------------------------------
! Optimization Set (parameter estimation calibration dataset) definitions
!-------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_EmptyObsId = "No obs"
   character*50, public,  parameter :: LIS_templateObsId = "NONE"
   character*50, public,  parameter :: LIS_wgPBMRsmId = "WG PBMR sm"
   character*50, public,  parameter :: LIS_pesynsm1Id = "Synthetic sm1"
   character*50, public,  parameter :: LIS_pesynsm2Id = "Synthetic sm2"
   character*50, public,  parameter :: LIS_AmerifluxObsId = "Ameriflux obs"
   character*50, public,  parameter :: LIS_ARMObsId = "ARM obs"
   character*50, public,  parameter :: LIS_maconlandslideObsId = "Macon landslide obs"
   character*50, public,  parameter :: LIS_globallandslideObsId = "Global landslide obs"
   character*50, public,  parameter :: LIS_CNRSObsId = "CNRS"
   character*50, public,  parameter :: LIS_CNRS_MPDIObsId = "CNRS MPDI"
   character*50, public,  parameter :: LIS_AMSRE_SRObsId = "AMSRE SR"
   character*50, public,  parameter :: LIS_LPRM_AMSREsmpeObsId = "AMSR-E(LPRM) pe soil moisture"
   character*50, public,  parameter :: LIS_FLUXNETpeObsId = "Gridded FLUXNET"
   character*50, public,  parameter :: LIS_USDA_ARSsmpeObsId = "USDA ARSsm"
!-------------------------------------------------------------------------
! Objective Function Evaluation Criteria
!-------------------------------------------------------------------------
   character*50, public,  parameter :: LIS_LMestimateId = "LM"
   character*50, public,  parameter :: LIS_LSestimateId = "Least squares"
   character*50, public,  parameter :: LIS_LLestimateId = "Likelihood"
   character*50, public,  parameter :: LIS_PestimateId  = "Probability"
!-------------------------------------------------------------------------
! Radiative Transfer Models
!-------------------------------------------------------------------------
   character*50, public, parameter :: LIS_crtmId = "CRTM"
   character*50, public, parameter :: LIS_crtm2Id = "CRTM2"
   character*50, public, parameter :: LIS_crtm2EMId = "CRTM2EM"
   character*50, public, parameter :: LIS_cmem3Id = "CMEM"
   character*50, public, parameter :: LIS_tauomegaRTMId = "Tau Omega"
!-------------------------------------------------------------------------
```

```
! Land Slide Models
!-------------------------------------------------------------------------
   character*50, public, parameter :: LIS_GLSId = "GLS"
   character*50, public, parameter :: LIS_TRIGRSId = "TRIGRS"
!-------------------------------------------------------------------------
! Routing Models
!-------------------------------------------------------------------------
   character*50, public, parameter :: LIS_NLDASrouterId = "NLDAS router"
   character*50, public, parameter :: LIS_HYMAProuterId = "HYMAP router"
!-------------------------------------------------------------------------
!  Irrigation models
!-------------------------------------------------------------------------
   character*50, public, parameter :: LIS_sprinklerIrrigationId = "Sprinkler"
```

4.25 Fortran: Module Interface LIS_routing_pluginMod (Source File: LIS_routing_pluginMo

This module contains the definition of the functions used for routing model initialization, execution, reading and writing of restart files and other relevant routing model computations.

REVISION HISTORY:

```
 6 May 11    Sujay Kumar  Initial Specification
```

4.25.1 LIS_routing_plugin (Source File: LIS_routing_pluginMod.F90)

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

INTERFACE:

```
subroutine LIS_routing_plugin
```

4.26 Fortran: Module Interface LIS_runmode_pluginMod (Source File: LIS_runmode_plugin

This module contains the definition of the functions used for LIS initialization, execution, and finalization for different running modes in LIS

REVISION HISTORY:

```
 21 Oct 05    Sujay Kumar  Initial Specification
```

4.26.1 LIS_runmode_plugin (Source File: LIS_runmode_pluginMod.F90)

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

Initialization Defining the init routines needed for each running mode in LIS. (to be registered using `registerlisinit` and later called using `lisinit`)

Run Define the execution routines needed for each running mode in LIS (to be registered using `registerlisrun` and later called using `lisrun`)

Finalize Define the finalize routines needed for each LIS running mode. (to be registered using `registerlisfinalize` and later called using `lisfinalize`)

The user-defined functions are included in the registry using a single index. For example, consider the definition of a retrospective running mode in LIS. The methods should be defined in the registry as follows, if the index of the mode is defined to be 1.

```
call registerlisinit(1,lisinit_retrospective)
call registerlisrun(1,lisrun_retrospective)
call registerlisfinalize(1,lisfinal_retrospective)
```

The functions registered above are invoked using generic calls as follows:

```
call lisinit(1)  - calls lisinit_retrospective
call lisrun(1)   - calls lisrun_retrospective
call lisfinal(1)   - calls lisfinal_retrospective
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call lisinit(lis%runmode)
call lisrun(lis%runmode)
call lisfinal(lis%runmode)
```

where *lis%runmode* is set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_runmode_plugin
```

Part V
Interpolation Tools in LIS

5 Interpolation tools in LIS

The spatial interpolation tools in LIS are used for the geospatial transformation of an input data to the LIS grid. The tools in LIS are based on the ipolates package developed at NCEP. Currently LIS supports the transformation and interpolation of polar stereographic, lambert conformal, and mercator grid projection data on to an equidistant cylindrical (lat/lon) grid projection. Bilinear, conservative and neighbor search interpolations are supported.

5.1 Fortran: Module Interface UTM_utils (Source File: UTM_utils.F90)

Performs conversions for UTM coordinates
SOURCE: Defense Mapping Agency (DMA) Technical Manual 8358.2 "The Universal Grids: Universal Transverse Mercator (UTM) and Universal Polar Stereographic (UPS)" Defense Mapping Agency, Fairfax VA, September 1989 obtained on the web at http://earth-info.nga.mil/GandG/pubs.html
PURPOSE: primarily for LIS-ARMS on Windows platforms

REVISION HISTORY:

```
    28Oct2004  Matthew Garcia  Initial specification of UTM2Geo subroutine
    29Oct2004  Matthew Garcia  Initial specification of Geo2UTM subroutine
    04Dec2004  Matthew Garcia  Edit/update for incorporation in LISv3.2
 module UTM_utils
   implicit none

   PRIVATE

   public    :: UTM2geo
```

5.1.1 bilinear_interp (Source File: bilinear_interp.F90)

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,li,gi,lo,go,mi,mo, &
      rlat,rlon,w11,w12,w21,w22,n11,n12,n21,n22,udef,iret)
```

USES:

```
  $ use omp_lib

   implicit none
```

ARGUMENTS:

```
   real      :: gridDesco(50)
   integer   :: mi
   integer   :: mo
   logical*1 :: li(mi)
   logical*1 :: lo(mo)
   real      :: gi(mi)
```

```
real      :: go(mo)
real      :: rlat(mo)
real      :: rlon(mo)
real      :: w11(mo),w12(mo), w21(mo),w22(mo)
integer   :: n11(mo),n12(mo),n21(mo),n22(mo)
real      :: udef
integer   :: iret
```

DESCRIPTION:

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. the grids are defined by their grid description arrays.
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
where gridDesc could be defined for either the input grid or the output grid. The routine also returns the the number of output grid points and their latitudes and longitudes are also returned. The 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.
The arguments are:

gridDesco output grid description parameters

li logical input bitmaps

gi real input fields to interpolate

lo logical output bitmaps

go real output fields interpolated

mi integer dimension of input grid fields

mo integer dimension of output grid fields

rlat output latitudes in degrees

rlon output longitudes in degrees

w11,w12,w21,w22 weights to be used for interpolation

n11,n12,n21,n22 index of neighbor points

udef undefined value to be used

iret return code (0-success)

The routines invoked are:

polfixs (5.3.14)
Apply corrections for poles

5.1.2 bilinear_interp_input (Source File: bilinear_interp_input.F90)

INTERFACE:

```
subroutine bilinear_interp_input (gridDesci,gridDesco,npts,&
     rlat,rlon,n11,n12,n21,n22,w11,w12,w21,w22)

  implicit none
```

ARGUMENTS:

```
  real, intent(in)    :: gridDesci(50)
  real                :: gridDesco(50)
  integer             :: npts
  real                :: rlat(npts)
  real                :: rlon(npts)
  integer             :: n11(npts),n12(npts),n21(npts),n22(npts)
  real                :: w11(npts),w12(npts),w21(npts),w22(npts)
```

DESCRIPTION:

This subprogram performs issues calls to compute the interpolation weights and neighbor information for bilinear interpolation,from any grid to any grid for scalar fields. The grids are defined by their grid description arrays.
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
where gridDesc could be defined for either the input grid or the output grid.
The arguments are:

gridDesci input grid description parameters

gridDesco output grid description parameters

npts number of points to in the output field

rlat output latitudes in degrees

rlon output longitudes in degrees

w11,w12,w21,w22 weights to be used for interpolation

n11,n12,n21,n22 index of neighbor points

The routines invoked are:

compute_earth_coord (5.1.3)
Computes the earth coordinates for the output grid

compute_grid_coord (5.1.14)
Computes the grid coordinates of the input grid, based on the earth coordinates of the output grid.

get_fieldpos (5.2.6)
computes the field position for a given point

5.1.3 compute_earth_coord (Source File: compute_earth_coord.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine compute_earth_coord(gridDesc,npts,fill,xpts,ypts,rlon,rlat,nret,&
                               gen_xypts)

  implicit none
```

ARGUMENTS:

```
  real        :: gridDesc(50)
  integer     :: npts
  real        :: fill
  real        :: xpts(npts),ypts(npts)
  real        :: rlat(npts)
  real        :: rlon(npts)
  integer     :: nret
  logical     :: gen_xypts
```

DESCRIPTION:

This subroutine computes the earth coordinates (lat/lon values) of the specified domain. This routine is based on the grid decoding routines in the ipolates interoplation package.
The input options include : 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 grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts input grid x point coordinates

ypts input grid y point coordinates

rlat output latitudes in degrees

rlon output longitudes in degrees

nret return code (0-success)

gen_xypts logical that specifies whether this routine should generate xpts and ypts. Example: This routine is called twice when computing weights and neighbours for the conservative intepolation scheme. The first call must generate xpts and ypts. The second call should not.

The routines invoked are:

compute_earth_coord_latlon (5.1.9)
computes the earth coordinates of a latlon grid

compute_earth_coord_merc (5.1.12)
computes the earth coordinates of a mercator grid

compute_earth_coord_lambert (5.1.8)
computes the earth coordinates of a lambert conformal grid

compute_earth_coord_gauss (5.1.6)
computes the earth coordinates of a gaussian cylindrical grid

compute_earth_coord_polar (5.1.13)
computes the earth coordinates of a polar stereographic grid

5.1.4 compute_earth_coord_UTM (Source File: compute_earth_coord_UTM.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine compute_earth_coord_UTM(gridDesc,npts,fill,xpts,ypts,&
     rlon,rlat,nret)

  use UTM_utils,   only : UTM2geo

  implicit none
```

ARGUMENTS:

```
real            :: gridDesc(50)
integer         :: npts
real            :: fill
real            :: xpts(npts),ypts(npts)
real            :: rlat(npts)
real            :: rlon(npts)
```

DESCRIPTION:

This subroutine computes the earth coordinates of the specified domain for a UTM projection

gridDesc grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts grid x point coordinates

ypts grid y point coordinates

rlat output latitudes in degrees

rlon output longitudes in degrees

5.1.5 compute_earth_coord_ease (Source File: compute_earth_coord_ease.F90)

REVISION HISTORY:

```
04-10-96 Mark Iredell;  Initial Specification
05-27-04 Sujay Kumar; Modified verision with floating point arithmetic.
06-11-07 Bailing Li; adapted from compute_earth_coord_latlon
```

INTERFACE:

```
subroutine compute_earth_coord_ease(gridDesc,npts,fill,xpts,ypts,&
     rlon,rlat,nret)

  implicit none
```

ARGUMENTS:

```
  real            :: gridDesc(50)
  integer         :: npts
  real            :: fill
  real            :: xpts(npts),ypts(npts)
  real            :: rlat(npts)
  real            :: rlon(npts)
```

DESCRIPTION:

This subroutine computes the earth coordinates of the specified domain for an EASE cylindrical projection.

gridDesc grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts grid x point coordinates

ypts grid y point coordinates

rlat output latitudes in degrees

rlon output longitudes in degrees

5.1.6 compute_earth_coord_gauss (Source File: compute_earth_coord_gauss.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine compute_earth_coord_gauss(gridDesc,npts,fill,xpts,ypts,&
     rlon,rlat,nret)

  implicit none
```

ARGUMENTS:

```
real            :: gridDesc(50)
integer         :: npts
real            :: fill
real            :: xpts(npts),ypts(npts)
real            :: rlat(npts)
real            :: rlon(npts)
```

DESCRIPTION:

This subroutine computes the earth coordinates of the specified domain for a gaussian cylindrical projection. This routine is based on the grid decoding routines in the NCEP interoplation package.

gridDesc grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts grid x point coordinates

ypts grid y point coordinates

rlat output latitudes in degrees

rlon output longitudes in degrees

The routines invoked are:

gausslat (5.2.5)
Computes latitude values in gaussian

5.1.7 compute_earth_coord_hrap (Source File: compute_earth_coord_hrap.F90)

REVISION HISTORY:

```
04-10-96 Mark Iredell;  Initial Specification
07-15-05 Sujay Kumar; Modified verision with floating point arithmetic.
```

INTERFACE:

```
subroutine compute_earth_coord_hrap(gridDesc,npts,fill,xpts,ypts,&
     rlon,rlat,nret)
```

USES:

```
use map_utils
implicit none
```

ARGUMENTS:

```
real            :: gridDesc(50)
integer         :: npts
real            :: fill
real            :: xpts(npts),ypts(npts)
real            :: rlat(npts)
real            :: rlon(npts)
integer         :: nret
```

DESCRIPTION:

This subroutine computes the earth coordinates of the specified domain for a hrap projection This routine is based on the decoding routines in the NCEP interoplation package and has been modified the adopted module from the Weather Research and Forecasting (WRF) model.

gridDesc grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts grid x point coordinates

ypts grid y point coordinates

rlat output latitudes in degrees

rlon output longitudes in degrees

nret return code (0-success)

The routines invoked are:

map_set (5.3.7)
Sets the projection to hrap

ij_to_latlon (5.3.9)
Computes the lat lon values for each i,j

5.1.8 compute_earth_coord_lambert (Source File: compute_earth_coord_lambert.F90)

INTERFACE:

```
subroutine compute_earth_coord_lambert(gridDesc,npts,fill,xpts,ypts,&
     rlon,rlat,nret)
```

USES:

```
use map_utils

implicit none
```

ARGUMENTS:

```
real            :: gridDesc(50)
integer         :: npts
real            :: fill
real            :: xpts(npts),ypts(npts)
real            :: rlat(npts)
real            :: rlon(npts)
integer         :: nret
```

DESCRIPTION:

This subroutine computes the earth coordinates of the specified domain for a lambert conformal projection This routine is based on the decoding routines in the NCEP interoplation package and has been modified the adopted module from the Weather Research and Forecasting (WRF) model.

gridDesc grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts grid x point coordinates

ypts grid y point coordinates

rlat output latitudes in degrees

rlon output longitudes in degrees

nret return code (0-success)

The routines invoked are:

map_set (5.3.7)
Sets the projection to lambert conformal

ij_to_latlon (5.3.9)
Computes the lat lon values for each i,j

5.1.9 compute_earth_coord_latlon (Source File: compute_earth_coord_latlon.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine compute_earth_coord_latlon(gridDesc,npts,fill,xpts,ypts,&
     rlon,rlat,nret)

  implicit none
```

ARGUMENTS:

```
  real            :: gridDesc(50)
  integer         :: npts
  real            :: fill
  real            :: xpts(npts),ypts(npts)
  real            :: rlat(npts)
  real            :: rlon(npts)
```

DESCRIPTION:

This subroutine computes the earth coordinates of the specified domain for an equidistant cylindrical projection. This routine is based on the grid decoding routines in the NCEP interoplation package.

gridDesc grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts grid x point coordinates

ypts grid y point coordinates

rlat output latitudes in degrees

rlon output longitudes in degrees

5.1.10 compute_earth_coord_latlon_ll (Source File: compute_earth_coord_latlon_ll.F90)

REVISION HISTORY:

```
20 Sept 08 Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine compute_earth_coord_latlon_ll(gridDesco,gridDesci,npts,fill,xpts,ypts,&
     rlon,rlat,nret)

  implicit none
```

ARGUMENTS:

```
  real            :: gridDesco(50)
  real            :: gridDesci(50)
  integer         :: npts
  real            :: fill
  real            :: xpts(npts),ypts(npts)
  real            :: rlat(npts)
  real            :: rlon(npts)
```

DESCRIPTION:

This subroutine computes the lower left earth coordinates of the specified domain for an equidistant cylindrical projection. This routine is based on the grid decoding routines in the NCEP interoplation package.

gridDesci input grid description parameters

gridDesco output grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts grid x point coordinates

ypts grid y point coordinates

rlat output latitudes in degrees

rlon output longitudes in degrees

5.1.11 compute_earth_coord_ll (Source File: compute_earth_coord_ll.F90)

REVISION HISTORY:

```
20 Sept 08 Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine compute_earth_coord_ll(gridDesco,gridDesci,npts,fill,xpts,ypts,rlon,rlat,nret)

  implicit none
```

ARGUMENTS:

```
  real        :: gridDesco(50)
  real        :: gridDesci(50)
  integer     :: npts
  real        :: fill
  real        :: xpts(npts),ypts(npts)
  real        :: rlat(npts)
  real        :: rlon(npts)
  integer     :: nret
```

DESCRIPTION:

This subroutine computes the lower left earth coordinates (lat/lon values) of the specified domain. This routine is based on the grid decoding routines in the ipolates interoplation package.
The input options include : The current code recognizes the following projections:
(gridDesc(1)=000) equidistant cylindrical

gridDesci input grid description parameters

gridDesco output grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts input grid x point coordinates

ypts input grid y point coordinates

rlat output latitudes in degrees

rlon output longitudes in degrees

nret return code (0-success)

The routines invoked are:

compute_earth_coord_latlon_ll (5.1.10)
computes the lower left earth coordinates of a latlon grid

5.1.12 compute_earth_coord_merc (Source File: compute_earth_coord_merc.F90)

REVISION HISTORY:

```
04-10-96 Mark Iredell;  Initial Specification
07-15-05 Sujay Kumar; Modified verision with floating point arithmetic.
```

INTERFACE:

```
subroutine compute_earth_coord_merc(gridDesc,npts,fill,xpts,ypts,&
     rlon,rlat,nret)
```

USES:

```
use map_utils
implicit none
```

ARGUMENTS:

```
real            :: gridDesc(50)
integer         :: npts
real            :: fill
real            :: xpts(npts),ypts(npts)
real            :: rlat(npts)
real            :: rlon(npts)
integer         :: nret
```

DESCRIPTION:

This subroutine computes the earth coordinates of the specified domain for a mercator projection This routine is based on the decoding routines in the NCEP interoplation package and has been modified the adopted module from the Weather Research and Forecasting (WRF) model.

gridDesc grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts grid x point coordinates

ypts grid y point coordinates

rlat output latitudes in degrees

rlon output longitudes in degrees

nret return code (0-success)

The routines invoked are:

map_set (5.3.7)
Sets the projection to mercator

ij_to_latlon (5.3.9)
Computes the lat lon values for each i,j

5.1.13 compute_earth_coord_polar (Source File: compute_earth_coord_polar.F90)

REVISION HISTORY:

```
04-10-96 Mark Iredell;  Initial Specification
07-15-05 Sujay Kumar; Modified verision with floating point arithmetic.
```

INTERFACE:

```
subroutine compute_earth_coord_polar(gridDesc,npts,fill,xpts,ypts,&
     rlon,rlat,nret)
```

USES:

```
use map_utils
implicit none
```

ARGUMENTS:

```
real            :: gridDesc(50)
integer         :: npts
real            :: fill
real            :: xpts(npts),ypts(npts)
real            :: rlat(npts)
real            :: rlon(npts)
integer         :: nret
```

DESCRIPTION:

This subroutine computes the earth coordinates of the specified domain for a polar stereographic projection This routine is based on the decoding routines in the NCEP interoplation package and has been modified the adopted module from the Weather Research and Forecasting (WRF) model.

gridDesc grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts grid x point coordinates

ypts grid y point coordinates

rlat output latitudes in degrees

rlon output longitudes in degrees

nret return code (0-success)

The routines invoked are:

map_set (5.3.7)
Sets the projection to polar stereographic

ij_to_latlon (5.3.9)
Computes the lat lon values for each i,j

5.1.14 compute_grid_coord (Source File: compute_grid_coord.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine compute_grid_coord(gridDesc,npts,fill,xpts,ypts,rlon,rlat,nret)
  implicit none
```

ARGUMENTS:

```
  real        :: gridDesc(50)
  integer     :: npts
  real        :: fill
  real        :: xpts(npts),ypts(npts)
  real        :: rlat(npts)
  real        :: rlon(npts)
  integer     :: nret
```

DESCRIPTION:

This subroutine computes the grid coordinates (cartesian) of the specified domain. This routine is based on the grid decoding routines in the ipolates interoplation package.
The input options include :
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 grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts output grid x point coordinates

ypts output grid y point coordinates

rlat input latitudes in degrees

rlon input longitudes in degrees

nret return code (0-success)

The routines invoked are:

compute_grid_coord_latlon (5.1.19
computes the grid coordinates of a latlon grid

compute_grid_coord_merc (5.1.20
computes the grid coordinates of a mercator grid

compute_grid_coord_lambert (5.1.18
computes the grid coordinates of a lambert conformal grid

compute_grid_coord_gauss (5.1.16
computes the grid coordinates of a gaussian cylindrical grid

compute_grid_coord_polar (5.1.21
computes the grid coordinates of a polar stereographic grid

5.1.15 compute_grid_coord_ease (Source File: compute_grid_coord_ease.F90)

REVISION HISTORY:

```
04-10-96 Mark Iredell;  Initial Specification
05-27-04 Sujay Kumar; Modified verision with floating point arithmetic.
06-11-07 Bailing Li; adapted from compute_grid_coord_latlon
```

INTERFACE:

```
subroutine compute_grid_coord_ease(gridDesc,npts,fill,xpts,ypts,&
     rlon,rlat,nret)

  implicit none
```

ARGUMENTS:

```
  real            :: gridDesc(50)
  integer         :: npts
  real            :: fill
  real            :: xpts(npts),ypts(npts)
  real            :: rlat(npts)
  real            :: rlon(npts)
  integer         :: nret
```

DESCRIPTION:

This subroutine computes the grid coordinates of the specified domain for an cylindrical EASE projection.

gridDesc grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts output grid x point coordinates

ypts output grid y point coordinates

rlat input latitudes in degrees

rlon input longitudes in degrees

5.1.16 compute_grid_coord_gauss (Source File: compute_grid_coord_gauss.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine compute_grid_coord_gauss(gridDesc,npts,fill,xpts,ypts,&
     rlon,rlat,nret)

  implicit none
```

ARGUMENTS:

```
  real            :: gridDesc(50)
  integer         :: npts
  real            :: fill
  real            :: xpts(npts),ypts(npts)
  real            :: rlat(npts)
  real            :: rlon(npts)
  integer         :: nret
```

DESCRIPTION:

This subroutine computes the grid coordinates of the specified domain for a gaussian cylindrical projection. This routine is based on the grid decoding routines in the NCEP interoplation package.

gridDesc grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts output grid x point coordinates

ypts output grid y point coordinates

rlat input latitudes in degrees

rlon input longitudes in degrees

The routines invoked are:

gausslat (5.2.5
Computes latitude values in gaussian

5.1.17 compute_grid_coord_hrap (Source File: compute_grid_coord_hrap.F90)

REVISION HISTORY:

```
04-10-96 Mark Iredell;  Initial Specification
07-15-05 Sujay Kumar; Modified verision with floating point arithmetic.
07-25-06 Sujay Kumar; Updated version to include Map Utils routines
02-04-14 Shugong Wang; Fix bugs with LIS_XMRG_Reader
```

INTERFACE:

```
subroutine compute_grid_coord_hrap(gridDesc,npts,fill,xpts,ypts,&
     rlon,rlat,nret)

  use map_utils

  implicit none
```

ARGUMENTS:

```
  real            :: gridDesc(50)
  integer         :: npts
  real            :: fill
  real            :: xpts(npts),ypts(npts)
  real            :: rlat(npts)
  real            :: rlon(npts)
  integer         :: nret
```

DESCRIPTION:

This subroutine computes the grid coordinates of the specified domain for a hrap projection. This routine is based on the grid decoding routines in the NCEP interoplation package.

gridDesc grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts output grid x point coordinates

ypts output grid y point coordinates

rlat input latitudes in degrees

rlon input longitudes in degrees

The routines invoked are:

latlonTohrap (5.2.9)
Converts lat, lons to hrap coordinates

map_set (5.3.7)
Determines projection type for grid structure fill

latlon_to_ij (5.3.8)
Converts the input lat/lon values to the Cartesian (i,j) value for the projection type.

5.1.18 compute_grid_coord_lambert (Source File: compute_grid_coord_lambert.F90)

REVISION HISTORY:

```
04-10-96 Mark Iredell;  Initial Specification
07-15-05 Sujay Kumar; Modified verision with floating point arithmetic.
```

INTERFACE:

```
subroutine compute_grid_coord_lambert(gridDesc,npts,fill,xpts,ypts,&
     rlon,rlat,nret)
```

USES:

```
use map_utils

implicit none
```

ARGUMENTS:

```
real            :: gridDesc(50)
integer         :: npts
real            :: fill
real            :: xpts(npts),ypts(npts)
real            :: rlat(npts)
real            :: rlon(npts)
integer         :: nret
```

DESCRIPTION:

This subroutine computes the grid coordinates of the specified domain for a lambert conformal projection. This routine is based on the grid decoding routines in the NCEP interoplation package.

gridDesc grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts output grid x point coordinates

ypts output grid y point coordinates

rlat input latitudes in degrees

rlon input longitudes in degrees

!NOTE: This routine is currently unsupported.

5.1.19 compute_grid_coord_latlon (Source File: compute_grid_coord_latlon.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine compute_grid_coord_latlon(gridDesc,npts,fill,xpts,ypts,&
     rlon,rlat,nret)

implicit none
```

ARGUMENTS:

```
real            :: gridDesc(50)
integer         :: npts
real            :: fill
real            :: xpts(npts),ypts(npts)
real            :: rlat(npts)
real            :: rlon(npts)
integer         :: nret
```

DESCRIPTION:

This subroutine computes the grid coordinates of the specified domain for an equidistant cylindrical rojection. This routine is based on the grid decoding routines in the NCEP interoplation package.

gridDesc grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts output grid x point coordinates

ypts output grid y point coordinates

rlat input latitudes in degrees

rlon input longitudes in degrees

5.1.20 compute_grid_coord_merc (Source File: compute_grid_coord_merc.F90)

REVISION HISTORY:

```
04-10-96 Mark Iredell;  Initial Specification
07-15-05 Sujay Kumar; Modified verision with floating point arithmetic.
```

INTERFACE:

```
subroutine compute_grid_coord_merc(gridDesc,npts,fill,xpts,ypts,&
     rlon,rlat,nret)
```

USES:

```
use LIS_logMod, only : LIS_logunit, LIS_endrun

implicit none
```

ARGUMENTS:

```
real            :: gridDesc(50)
integer         :: npts
real            :: fill
real            :: xpts(npts),ypts(npts)
real            :: rlat(npts)
real            :: rlon(npts)
integer         :: nret
```

DESCRIPTION:

This subroutine computes the grid coordinates of the specified domain for a mercator projection. This routine is based on the grid decoding routines in the NCEP interoplation package.

gridDesc grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts output grid x point coordinates

ypts output grid y point coordinates

rlat input latitudes in degrees

rlon input longitudes in degrees

!NOTE: This routine is currently unsupported.

5.1.21 compute_grid_coord_polar (Source File: compute_grid_coord_polar.F90)

REVISION HISTORY:

```
04-10-96 Mark Iredell;  Initial Specification
07-15-05 Sujay Kumar; Modified verision with floating point arithmetic.
07-25-06 Sujay Kumar; Updated version to include Map Utils routines
```

INTERFACE:

```
subroutine compute_grid_coord_polar(gridDesc,npts,fill,xpts,ypts,&
     rlon,rlat,nret)

  use map_utils

  implicit none
```

ARGUMENTS:

```
real            :: gridDesc(50)
integer         :: npts
real            :: fill
real            :: xpts(npts),ypts(npts)
real            :: rlat(npts)
real            :: rlon(npts)
integer         :: nret
```

DESCRIPTION:

This subroutine computes the grid coordinates of the specified domain for a polar stereographic projection. This routine is based on the grid decoding routines in the NCEP interoplation package.

gridDesc grid description parameters

npts integer maximum number of coordinates

fill fill value to set invalid output data

xpts output grid x point coordinates

ypts output grid y point coordinates

rlat input latitudes in degrees

rlon input longitudes in degrees

The routines invoked are:

latlonTopolar (5.2.10)
Converts lat, lons to polar coordinates

map_set (5.3.7)
Determines projection type for grid structure fill

latlon_to_ij (5.3.8)
Converts the input lat/lon values to the Cartesian (i,j) value for the projection type.

5.1.22 compute_stnwts (Source File: compute_stnwts.F90)

REVISION HISTORY:

```
07-15-05 Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine compute_stnwts(nstns, gridDesc,stnlat, stnlon,&
     npts, stnwt)
  use LIS_logMod, only : LIS_endrun
  implicit none
```

ARGUMENTS:

```
  integer     :: nstns
  real        :: gridDesc(50)
  real        :: stnlat(nstns)
  real        :: stnlon(nstns)
  integer     :: npts
  real        :: stnwt(npts,nstns)
```

DESCRIPTION:

This routine compute the interpolation weights to be applied to a network of stations, to generate a gridded field from observations from stations. It simply uses an inverse distance weighting algorithm to compute the relative weights of each station.

nstns number of stations used in interpolation

gridDesc grid description parameters

stnlat station latitudes in degrees

stnlon station longitudes in degrees

npts integer maximum number of coordinates

stnwt interpolation weights of stations with respect to the each point

The routines invoked are:

compute_earth_coord (5.1.3)
computes the earth coordinates

5.1.23 conserv_interp (Source File: conserv_interp.F90)

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,li,gi,lo,go,mi,mo,&
     rlat,rlon,w11,w12,w21,w22,n11,n12,n21,n22,udef,iret)
 $ use omp_lib
  implicit none
```

ARGUMENTS:

```
integer, parameter :: nb3 = 25, nb4 = 25
real      :: gridDesco(50)
integer   :: mi
integer   :: mo
logical*1 :: li(mi)
logical*1 :: lo(mo)
real      :: gi(mi)
real      :: go(mo)
real      :: rlat(mo)
real      :: rlon(mo)
integer   :: n11(mo,nb4),n21(mo,nb4),n12(mo,nb4),n22(mo,nb4)
real      :: w11(mo,nb4),w21(mo,nb4),w12(mo,nb4),w22(mo,nb4)
real      :: udef
integer   :: iret
```

DESCRIPTION:

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, using 25 points for computing the average.
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

where gridDesc could be defined for either the input grid or the output grid. The routine also returns the the number of output grid points and their latitudes and longitudes are also returned. The 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.
The arguments are:

gridDesco output grid description parameters

ibi integer input bitmap flags

li logical input bitmaps

gi real input fields to interpolate

ibo integer output bitmap flags

lo logical output bitmaps

go real output fields interpolated

mi integer dimension of input grid fields

mo integer dimension of output grid fields

rlat output latitudes in degrees

rlon output longitudes in degrees

w11,w12,w21,w22 weights to be used for interpolation

n11,n12,n21,n22 index of neighbor points

udef undefined value to be used

iret return code (0-success)

The routines invoked are:

polfixs (5.3.14)
Apply corrections for poles

5.1.24 conserv_interp_input (Source File: conserv_interp_input.F90)

INTERFACE:

```
subroutine conserv_interp_input(gridDesci,gridDesco,npts,&
     rlat2,rlon2,n112,n122,n212,n222,w112,w122,w212,w222)

  implicit none
```

ARGUMENTS:

```
  real, intent(in)    :: gridDesci(50)
  real                :: gridDesco(50)
  integer             :: npts
  real                :: rlat2(npts)
  real                :: rlon2(npts)
  integer             :: n112(npts,25),n122(npts,25),&
       n212(npts,25),n222(npts,25)
  real                :: w112(npts,25),w122(npts,25),&
       w212(npts,25),w222(npts,25)
```

DESCRIPTION:

This subprogram performs issues calls to compute the interpolation weights and neighbor information for budget bilinear interpolation,from any grid to any grid for scalar fields. The grids are defined by their grid description arrays.
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
where gridDesc could be defined for either the input grid or the output grid.
The arguments are:

gridDesci input grid description parameters

gridDesco output grid description parameters

npts number of points to in the output field

rlat2 output latitudes in degrees

rlon2 output longitudes in degrees

w112,w122,w212,w222 weights to be used for interpolation

n112,n122,n212,n222 index of neighbor points

The routines invoked are:

compute_earth_coord (5.1.3
Computes the earth coordinates for the output grid

compute_grid_coord (5.1.14
Computes the grid coordinates of the input grid, based on the earth coordinates of the output grid.

get_fieldpos (5.2.6)
computes the field position for a given point

5.1.25 defineLocalDomain (Source File: defineLocalDomain.F90)

REVISION HISTORY:

```
10 July 2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defineLocalDomain(lisGD, inGD, outGD)
```

USES:

```
use LIS_logMod,  only : LIS_logunit, LIS_endrun

implicit none

real,      intent(IN)    :: lisGD(50)
real,      intent(IN)    :: inGD(50)
real,      intent(OUT)   :: outGD(50)
```

DESCRIPTION:

This routine generates a description of the sub-domain based on the specifications of the running domain, native resolution and the output domain. Derives the lat/lon extents from the lis grid description, the map projection and resolution to be used from the input/native grid description.

5.1.26 ezlh_convert (Source File: ezlh_convert.F90)

REVISION HISTORY:

```
30-Jan.-1992 H.Maybee
20-Mar-1992 Ken Knowles  303-492-0644  knowles@kryos.colorado.edu
16-Dec-1993 MJ Brodzik   303-492-8263  brodzik@jokull.colorado.edu
            Copied from nsmconv.f, changed resolutions from
            40-20-10 km to 25-12.5 km
21-Dec-1993 MJ Brodzik   303-492-8263  brodzik@jokull.colorado.edu
            Fixed sign of Southern latitudes in ease_inverse.
12-Sep-1994 David Hoogstrate 303-492-4116 hoogstra@jokull.colorado.edu
            Changed grid cell size. Changed "c","f" to "l","h"
25-Oct-1994 David Hoogstrate 303-492-4116 hoogstra@jokull.colorado.edu
            Changed row size from 587 to 586 for Mercator projection
            Changed function names to "ezlh-.."
07-11-2007  Bailing Li; downloaded this code from NSIDC
```

INTERFACE:

```
integer function ezlh_convert (grid, lat, lon, r, s)

  implicit none
```

ARGUMENTS:

```
character*(*) grid
real lat, lon, r, s
```

DESCRIPTION:

convert geographic coordinates (spherical earth) to azimuthal equal area or equal area cylindrical grid coordinates

```
status = ezlh_convert (grid, lat, lon, r, s)

input : grid - projection name '[NSM][lh]'
        where l = "low"  = 25km resolution
              h = "high" = 12.5km resolution
        lat, lon - geo. coords. (decimal degrees)
```

```
output: r, s - column, row coordinates

result: status = 0 indicates normal successful completion
                -1 indicates error status (point not on grid)
```

5.1.27 ezlh_inverse (Source File: ezlh_inverse.F90)

REVISION HISTORY:

```
30-Jan.-1992 H.Maybee
20-Mar-1992 Ken Knowles  303-492-0644  knowles@kryos.colorado.edu
16-Dec-1993 MJ Brodzik   303-492-8263  brodzik@jokull.colorado.edu
            Copied from nsmconv.f, changed resolutions from
            40-20-10 km to 25-12.5 km
21-Dec-1993 MJ Brodzik   303-492-8263  brodzik@jokull.colorado.edu
            Fixed sign of Southern latitudes in ease_inverse.
12-Sep-1994 David Hoogstrate 303-492-4116 hoogstra@jokull.colorado.edu
            Changed grid cell size. Changed "c","f" to "l","h"
25-Oct-1994 David Hoogstrate 303-492-4116 hoogstra@jokull.colorado.edu
            Changed row size from 587 to 586 for Mercator projection
            Changed function names to "ezlh-.."
07-11-2007  Bailing Li; downloaded this code from NSIDC
```

INTERFACE:

```
integer function ezlh_inverse (grid, r, s, lat, lon)

  implicit none
```

ARGUMENTS:

```
  character*(*) grid
  real r, s, lat, lon
```

DESCRIPTION:

convert azimuthal equal area or equal area cylindrical grid coordinates to geographic coordinates (spherical earth)

```
status = ezlh_inverse (grid, r, s, lat, lon)

input : grid - projection name '[NSM][lh]'
        where l = "low"  = 25km resolution
              h = "high" = 12.5km resolution
        r, s - grid column and row coordinates

output: lat, lon - geo. coords. (decimal degrees)

result: status = 0 indicates normal successful completion
                -1 indicates error status (point not on grid)
```

5.2 Fortran: Module Interface gaussian_mod (Source File: gaussian_mod.F90)

The code in this file provides routines to manage a quasi-regular Gaussian lat/lon grid.

REVISION HISTORY:

```
13 Nov 2006: James Geiger ; Initial Specification
```

```
real,allocatable,dimension(:)   :: gaussian_lat_array
real,allocatable,dimension(:)   :: gaussian_lon_array
 real,allocatable,dimension(:,:) :: gaussian_latitudes
 real,allocatable,dimension(:,:) :: gaussian_longitudes
integer                      :: GAUSSIAN_NR
integer                      :: GAUSSIAN_NC
```

5.2.1 gaussian_find_row (Source File: gaussian_mod.F90)

INTERFACE:

```
function gaussian_find_row(lat)
```

USES:

```
use LIS_logMod , only : LIS_logunit, LIS_endrun

implicit none
```

ARGUMENTS:

```
real :: lat
```

DESCRIPTION:

This function computes the row number within the global quasi-regular Gaussian grid corresponding to the latitude given by lat.

lat latitude to search for

5.2.2 gaussian_find_col (Source File: gaussian_mod.F90)

INTERFACE:

```
function gaussian_find_col(lon)
```

USES:

```
use LIS_logMod , only : LIS_logunit, LIS_endrun

implicit none
```

ARGUMENTS:

```
real :: lon
```

DESCRIPTION:

This function computes the column number within the global quasi-regular Gaussian grid corresponding to the longitude given by lon.

lon longitude to search for

5.2.3 gaussian_comp_lats (Source File: gaussian_mod.F90)

INTERFACE:

```
subroutine gaussian_comp_lats(jmax)
```

USES:

```
use LIS_logMod , only : LIS_logunit

implicit none
```

ARGUMENTS:

```
integer :: jmax
```

DESCRIPTION:

This routine computes the latitudes of the global quasi-regular Gaussian grid corresponding to the total number of latitude circles given by jmax.

jmax the total number of latitude circles

5.2.4 gaussian_comp_lons (Source File: gaussian_mod.F90)

INTERFACE:

```
subroutine gaussian_comp_lons(startlon,dlon)
```

USES:

```
use LIS_logMod , only : LIS_logunit

implicit none
```

ARGUMENTS:

```
real :: startlon
real :: dlon
```

DESCRIPTION:

This routine computes the longitudes of the global quasi-regular Gaussian grid corresponding to starting longitude, startlon, and delta longitude, dlon.

startlon starting longitude of the grid

dlon delta longitude of the grid

5.2.5 gaussian_read_grid (Source File: gaussian_mod.F90)

INTERFACE:

```
subroutine gaussian_read_grid(imax,jmax)
```

USES:

```
use LIS_logMod, only : LIS_logunit

implicit none
```

ARGUMENTS:

```
integer :: imax, jmax
```

DESCRIPTION:

This routine reads in the global quasi-regular Gaussian grid of size imax,jmax.

imax total number of longitude points (along any latitude circle)

jmax total number of latitude circles

!ROUTINE : gausslat

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
```

INTERFACE:

```
subroutine gausslat(jmax,slat,wlat)
  implicit none
```

ARGUMENTS:

```
integer       :: jmax
real          :: slat(jmax)
real          :: wlat(jmax)
```

DESCRIPTION:

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. The arguments are:

jmax input number of latitudes

slat cosines of colatitude

wlat gaussian weights

5.2.6 get_fieldpos (Source File: get_fieldpos.F90)

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
01-17-09  Sujay Kumar; Specified the dlon value separately based on the
          input grid projection
```

INTERFACE:

```
function get_fieldpos(i,j,gridDesc) result(field_pos)

  implicit none
```

ARGUMENTS:

```
  integer     :: field_pos
  real        ::  gridDesc(50)
  integer     ::  i,j
```

DESCRIPTION:

This subprogram returns the field position for a given grid point based on the input grid definition. The arguments are:

i integer x grid point

j integer y grid point

gridDesc grid description parameters

field_pos integer position in grid field to locate grid point

---

5.2.7 hrapToLatLon (Source File: hrapToLatLon.F90)

REVISION HISTORY:

```
86-07-17  R.E.JONES
89-11-01  R.E.JONES   CHANGE TO CRAY CFT77 FORTRAN
05-27-04  Sujay Kumar Incorporated in LIS
```

INTERFACE:

```
subroutine hrapToLatLon(xi,xj,xmeshl,orient,alat,along)

  implicit none
```

ARGUMENTS:

```
  real    :: xi
  real    :: xj
  real    :: xmeshl
  real    :: orient
  real    :: alat
  real    :: along
```

DESCRIPTION:

Converts the coordinates of a location from the grid(i,j) coordinate system overlaid on the hrap map projection true at 60 degrees N or S latitude to the natural coordinate system of latitude/longitude on the earth. The arguments are:

xi I of the point relative to North or South pole

xj J of the point relative to North or South pole

xmeshl mesh length of grid in km at 60deg lat (< 0 if sh)

orient orientation west longitude of the grid

alat latitude in degrees

along longitude in degrees

5.2.8 interp_stndata (Source File: interp_stndata.F90)

REVISION HISTORY:

```
08 Dec 04 Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine interp_stndata(wt,undef,vari,varo,npts,nstns)

  use LIS_coreMod,  only : LIS_rc

  implicit none
```

ARGUMENTS:

```
  integer :: nstns,npts
  real    :: undef
  real    :: wt(npts,nstns)
  real    :: vari(nstns)
  real    :: varo(npts)
```

DESCRIPTION:

This subroutine interpolates station data into a gridded set. Currently works only on a lat/lon grid
The arguments are:

nstns number of stations

npts number of points in the output field

vari input variable (array of observations from the stations)

varo output variable (gridded data on the output field)

wt interpolation weights

5.2.9 latlonTohrap (Source File: latlonTohrap.F90)

REVISION HISTORY:

```
86-07-17  MCDONELL,J.
88-06-07  R.E.JONES   CLEAN UP CODE, TAKE OUT GOTO, USE THEN, ELSE
89-11-02  R.E.JONES   CHANGE TO CRAY CFT77 FORTRAN
05-27-04  Sujay Kumar Incorporated in LIS
```

INTERFACE:

```
subroutine latlontohrap(alat,along,xmeshl,orient,xi,xj)

  implicit none
```

ARGUMENTS:

```
  real     :: alat
  real     :: along
  real     :: xmeshl
  real     :: orient
  real     :: xi
  real     :: xj
```

DESCRIPTION:

Converts the coordinates of a location on earth from the natural coordinate system of latitude/longitude to the grid (i,j) coordinate system overlaid on a hrap map projection true at 60 degrees n or s latitude.
The arguments are:

alat latitude in degrees

along longitude in degrees

xmeshl mesh length of grid in km at 60deg lat (< 0 if sh)

orient orientation west longitude of the grid

xi I of the point relative to North or South pole

xj J of the point relative to North or South pole

5.2.10 latlonTopolar (Source File: latlonTopolar.F90)

REVISION HISTORY:

```
86-07-17  MCDONELL,J.
88-06-07  R.E.JONES   CLEAN UP CODE, TAKE OUT GOTO, USE THEN, ELSE
89-11-02  R.E.JONES   CHANGE TO CRAY CFT77 FORTRAN
05-27-04  Sujay Kumar Incorporated in LIS
```

INTERFACE:

```
subroutine latlontopolar(alat,along,xmeshl,orient,xi,xj)

  implicit none
```

ARGUMENTS:

```
  real     :: alat
  real     :: along
  real     :: xmeshl
  real     :: orient
  real     :: xi
  real     :: xj
```

DESCRIPTION:

Converts the coordinates of a location on earth from the natural coordinate system of latitude/longitude to the grid (i,j) coordinate system overlaid on a polar stereographic map projection true at 60 degrees n or s latitude.
The arguments are:

alat latitude in degrees

along longitude in degrees

xmeshl mesh length of grid in km at 60deg lat (< 0 if sh)

orient orientation west longitude of the grid

xi I of the point relative to North or South pole

xj J of the point relative to North or South pole

5.2.11 lltops (Source File: lltops.F90)

REVISION HISTORY:

```
     05 aug 1998 initial version.....................ssgt mccormick/dnxm
     10 aug 1999 ported to ibm sp-2.  added intent attributes to
                 arguments...............................mr gayno/dnxm
     29 oct 2005; Sujay Kumar; Incorporated into LIS
```

INTERFACE:

```
subroutine lltops( pose, rlat, rlon, mesh, hemi, ri, rj )

  implicit none
```

ARGUMENTS:

```
  integer,      intent(in)     :: pose
  real,         intent(in)     :: rlat
  real,         intent(in)     :: rlon
  integer,      intent(in)     :: mesh
  integer,      intent(out)    :: hemi
  real,         intent(out)    :: ri
  real,         intent(out)    :: rj
```

DESCRIPTION:

converts lat/lon to polar stereographic grid i/j points
method

calculate map related constants/factors.
adjust the longitude according to the user specified sign convention.
convert longitude to radians.
calculate the distance from the pole to the point in radians.
calculate the center point coordinate on the grid.
calculate the grid coordinates for the point.

The arguments are:

pose longitude increment orientation flag (1-positive east, 0-positive west)

rlat input latitude in degrees

rlon input longitude in degrees

mesh mesh factor

hemi index of the hemisphere (1-nh, 2-sh)

ri i-coordinate

rj j-coordinate

5.3 Fortran: Module Interface map_utils (Source File: map_utils.F90)

Module that defines constants, data structures, and subroutines used to convert grid indices to lat/lon and vice versa. (This module is adopted from the Weather Research and Forecasting (WRF) model Standard Initialization(SI) program.

5.3.1 Supported Projections

Cylindrical Lat/Lon (code = PROJ_LATLON)
Mercator (code = PROJ_MERC)
Lambert Conformal (code = PROJ_LC)
Polar Stereographic (code = PROJ_PS)

5.3.2 Remarks

The routines contained within were adapted from routines obtained from the NCEP w3 library. The original NCEP routines were less flexible (e.g., polar-stereo routines only supported truelat of 60N/60S) than what we needed, so modifications based on equations in Hoke, Hayes, and Renninger (AFGWC/TN/79-003) were added to improve the flexibility. Additionally, coding was improved to F90 standards and the routines were combined into this module.

5.3.3 Assumptions

Grid Definition: For mercator, lambert conformal, and polar-stereographic projections, the routines within assume the following:

- Grid is dimensioned (i,j) where i is the East-West direction, positive toward the east, and j is the north-south direction, positive toward the north.

- Origin is at (1,1) and is located at the southwest corner, regardless of hemispere.
- Grid spacing (dx) is always positive.
- Values of true latitudes must be positive for NH domains and negative for SH domains.

For the latlon projection, the grid origin may be at any of the corners, and the deltalat and deltalon values can be signed to account for this using the following convention:

```
Origin Location        Deltalat Sign      Deltalon Sign
---------------        -------------      -------------
 SW Corner                  +                  +
 NE Corner                  -                  -
 NW Corner                  -                  +
 SE Corner                  +                  -
```

5.3.4 Data Definitions:

- Any arguments that are a latitude value are expressed in degrees north with a valid range of -90 to 90
- Any arguments that are a longitude value are expressed in degrees east with a valid range of -180 to 180.
- Distances are in meters and are always positive.
- The standard longitude (stdlon) is defined as the longitude line which is parallel to the y-axis (j-direction), along which latitude increases (NOT the absolute value of latitude, but the actual latitude, such that latitude increases continuously from the south pole to the north pole) as j increases.
- One true latitude value is required for polar-stereographic and mercator projections, and defines at which latitude the grid spacing is true. For lambert conformal, two true latitude values must be specified, but may be set equal to each other to specify a tangent projection instead of a secant projection.

5.3.5 Usage

To use the routines in this module, the calling routines must have the following statement at the beginning of its declaration block: `USE map_utils`
The use of the module not only provides access to the necessary routines, but also defines a structure of `TYPE (proj_info)` that can be used to declare a variable of the same type to hold your map projection information. It also defines some integer parameters that contain the projection codes so one only has to use those variable names rather than remembering the acutal code when using them. The basic steps are as follows:

- Ensure the `USE map_utils` is in your declarations.
- Declare the projection information structure as `type(proj_info): TYPE(proj_info) :: proj`
- Populate your structure by calling the map_set routine:
  `CALL map_set(code,lat1,lon1,dx,stdlon,truelat1,truelat2,nx,ny,proj)` where: code (input) = one of PROJ_LATLON, PROJ_MERC, PROJ_LC, or PROJ_PS
  lat1 (input) = Latitude of grid origin point (i,j)=(1,1) (see assumptions!)
  lon1 (input) = Longitude of grid origin
  dx (input) = grid spacing in meters (ignored for LATLON projections)
  stdlon (input) = Standard longitude for PROJ_PS and PROJ_LC, deltalon (see assumptions) for PROJ_LATLON, ignored for PROJ_MERC
  truelat1 (input) = 1st true latitude for PROJ_PS, PROJ_LC, and PROJ_MERC, deltalat (see assumptions) for PROJ_LATLON

truelat2 (input) = 2nd true latitude for PROJ_LC, ignored for all others.
nx = number of points in east-west direction
ny = number of points in north-south direction
proj (output) = The structure of `type (proj_info)` that will be fully populated after this call

- Now that the proj structure is populated, you may call any of the following routines:

  ```
  latlon_to_ij(proj, lat, lon, i, j)
  ij_to_latlon(proj, i, j, lat, lon)
  truewind_to_gridwind(lon, proj, ugrid, vgrid, utrue, vtrue)
  gridwind_to_truewind(lon, proj, utrue, vtrue, ugrid, vgrid)
  compare_projections(proj1, proj2, same_proj)
  ```

  It is incumbent upon the calling routine to determine whether or not the values returned are within your domain bounds. All values of i, j, lat, and lon are REAL values.

References

Hoke, Hayes, and Renninger, "Map Preojections and Grid Systems for Meteorological Applications." AFGWC/TN-79/003(Rev), Air Weather Service, 1985.

NCAR MM5v3 Modeling System, REGRIDDER program, module_first_guess_map.F NCEP routines w3fb06, w3fb07, w3fb08, w3fb09, w3fb11, w3fb12

REVISION HISTORY:

```
  27 Mar 2001 - Original Version
                Brent L. Shaw, NOAA/FSL (CSU/CIRA)
  02 Apr 2001 - Added routines to rotate winds from true to grid
                and vice versa.
                Brent L. Shaw, NOAA/FSL (CSU/CIRA)
  09 Apr 2001 - Added compare\_projections routine to compare two
                sets of projection parameters.

 implicit none

  ! mean earth radius in m.  the value below is consistent
  ! with nceps routines and grids.
  real, public, parameter     :: earth_radius_m = 6371200.

  ! define public parameters
 mp
  INTEGER, PUBLIC, PARAMETER  :: PROJ_ROTLAT = 203
 mp
 ------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
 ------------------------------------------------------------------------------
  public :: map_init     !initializes map projection structure
  public :: map_set      !completes the initializations
  public :: ij_to_latlon !converts ijs to lat lon values
  public :: latlon_to_ij !converts lat lon values to ijs.
 ------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
 ------------------------------------------------------------------------------
```

```
! projection codes for proj_info structure:
integer, public, parameter  :: proj_latlon = 0
integer, public, parameter  :: proj_merc = 1
integer, public, parameter  :: proj_lc = 3
integer, public, parameter  :: proj_ps = 5
integer, public, parameter  :: proj_hrap = 10
```

5.3.6 map_init (Source File: map_utils.F90)

INTERFACE:

```
subroutine map_init(proj)
 implicit none
```

ARGUMENTS:

```
 type(proj_info), intent(inout)  :: proj
```

DESCRIPTION:

Initializes the map projection structure to missing values
The arguments are:

proj data structure containing the map projection information

---

5.3.7 map_set (Source File: map_utils.F90)

INTERFACE:

```
subroutine map_set(proj_code,lat1,lon1,dx,stdlon,truelat1,truelat2, &
                   idim,jdim,proj)

  implicit none
```

ARGUMENTS:

```
  integer, intent(in)              :: proj_code
  real, intent(in)                 :: lat1
  real, intent(in)                 :: lon1
  real, intent(in)                 :: dx
  real, intent(in)                 :: stdlon
  real, intent(in)                 :: truelat1
  real, intent(in)                 :: truelat2
  integer, intent(in)              :: idim
  integer, intent(in)              :: jdim
  type(proj_info), intent(out)     :: proj
```

DESCRIPTION:

Given a partially filled `proj_info` structure, this routine computes `polei, polej, rsw, and cone` (if LC projection) to complete the structure. This allows us to eliminate redundant calculations when calling the

coordinate conversion routines multiple times for the same map. This will generally be the first routine called when a user wants to be able to use the coordinate conversion routines, and it will call the appropriate subroutines based on the `proj%code` which indicates which projection type this is.
The arguments are:

proj_code code to indicate the type of projection

lat1 latitude of origin

lon1 longitude of origin

dx grid spacing in meters

stdlon standard longitude

truelat1 1st true latitude

truelat2 2nd true latitude

idim number of points in the East-West direction

jdim number of points in the North-South direction

proj data structure containing the map projection information

---

5.3.8 latlon_to_ij (Source File: map_utils.F90)

INTERFACE:

```
subroutine latlon_to_ij(proj, lat, lon, i, j)
  use LIS_XMRG_Reader, only : hrap_to_latlon, latlon_to_hrap
  implicit none
```

ARGUMENTS:

```
  type(proj_info), intent(in)          :: proj
  real, intent(in)                     :: lat
  real, intent(in)                     :: lon
  real, intent(out)                    :: i
  real, intent(out)                    :: j
  real :: hrap_x0, hrap_y0, tmp_i, tmp_j
```

DESCRIPTION:

Converts input lat/lon values to the cartesian (i,j) value for the given projection.
The arguments are:

proj data structure containing the map projection information

lat input latitude

lon input longitude

i output i value

j output j value

5.3.9 ij_to_latlon (Source File: map_utils.F90)

INTERFACE:

```
subroutine ij_to_latlon(proj, i, j, lat, lon)
  use LIS_XMRG_Reader, only : hrap_to_latlon, latlon_to_hrap
  implicit none
```

ARGUMENTS:

```
  type(proj_info),intent(in)          :: proj
  real, intent(in)                    :: i
  real, intent(in)                    :: j
  real, intent(out)                   :: lat
  real, intent(out)                   :: lon
  real :: hrap_x0, hrap_y0, tmp_i, tmp_j
```

DESCRIPTION:

Computes geographical latitude and longitude for a given (i,j) point in a grid with a projection of proj
The arguments are:

proj data structure containing the map projection information

i input i value

j input j value

lat output latitude

lon output longitude

5.3.10 neighbor_interp (Source File: neighbor_interp.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine neighbor_interp(gridDesco,li,gi,lo,go,mi,mo, &
     rlat,rlon,n11,udef, iret)

 $ use omp_lib
  implicit none
```

ARGUMENTS:

```
real      :: gridDesco(50)
integer   :: mi
integer   :: mo
logical*1 :: li(mi)
logical*1 :: lo(mo)
real      :: gi(mi)
```

```
real      :: go(mo)
real      :: rlat(mo)
real      :: rlon(mo)
integer   :: n11(mo)
real      :: udef
integer   :: iret
```

DESCRIPTION:

This subprogram performs neighbor search 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 performs a nearest neighbor search around each output grid point for interpolation. The grids are defined by their grid description arrays.
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
where gridDesc could be defined for either the input grid or the output grid. The routine also returns the the number of output grid points and their latitudes and longitudes are also returned. The 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.
The arguments are:

gridDesco output grid description parameters

ibi integer input bitmap flags

li logical input bitmaps

gi real input fields to interpolate

ibo integer output bitmap flags

lo logical output bitmaps

go real output fields interpolated

mi integer dimension of input grid fields

mo integer dimension of output grid fields

rlat output latitudes in degrees

rlon output longitudes in degrees

n11 index of neighbor points

udef undefined value to be used

iret return code (0-success)

The routines invoked are:

polfixs (5.3.14)
Apply corrections for poles

5.3.11 neighbor_interp_input (Source File: neighbor_interp_input.F90)

INTERFACE:

```
subroutine neighbor_interp_input (gridDesci,gridDesco,npts, rlat2,rlon2,n112)
  implicit none
```

ARGUMENTS:

```
  real, intent(in) :: gridDesci(50)
  real             :: gridDesco(50)
  integer          :: npts
  real             :: rlat2(npts)
  real             :: rlon2(npts)
  integer          :: n112(npts)
```

DESCRIPTION:

This subprogram performs issues calls to compute the neighbor information for neighbor search interpolation,from any grid to any grid for scalar fields. The grids are defined by their grid description arrays.
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
where gridDesc could be defined for either the input grid or the output grid.
The arguments are:

gridDesci input grid description parameters

gridDesco output grid description parameters

npts number of points to in the output field

rlat2 output latitudes in degrees

rlon2 output longitudes in degrees

n112 index of neighbor points

The routines invoked are:

compute_earth_coord (5.1.3)
Computes the earth coordinates for the output grid

compute_grid_coord (5.1.14)
Computes the grid coordinates of the input grid, based on the earth coordinates of the output grid.

get_fieldpos (5.2.6)
computes the field position for a given point

5.3.12 normalize_stnwts (Source File: normalize_stnwts.F90)

REVISION HISTORY:

```
08Dec04 Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine normalize_stnwts(stndata,nstns,npts,undef,wt)
  implicit none
```

ARGUMENTS:

```
integer    :: nstns
integer    :: npts
real       :: stndata(nstns)
real       :: wt(npts,nstns)
real       :: undef
```

DESCRIPTION:

This subroutine normalizes the interpolation weights to convert station data into a gridded set. The arguments are:

nstns number of stations used in interpolation

npts integer maximum number of coordinates

stndata input data (station observations)

undef undefined value used in observations

wt interpolation weights

5.3.13 polarToLatLon (Source File: polarToLatLon.F90)

REVISION HISTORY:

```
86-07-17  R.E.JONES
89-11-01  R.E.JONES   CHANGE TO CRAY CFT77 FORTRAN
05-27-04  Sujay Kumar Incorporated in LIS
```

INTERFACE:

```
subroutine polarToLatLon(xi,xj,xmeshl,orient,alat,along)

  implicit none
```

ARGUMENTS:

```
real    :: xi
real    :: xj
real    :: xmeshl
real    :: orient
real    :: alat
real    :: along
```

DESCRIPTION:

Converts the coordinates of a location from the grid(i,j) coordinate system overlaid on the polar stereographic map projec- tion true at 60 degrees N or S latitude to the natural coordinate system of latitude/longitude on the earth.
The arguments are:

xi I of the point relative to North or South pole

xj J of the point relative to North or South pole

xmeshl mesh length of grid in km at 60deg lat (< 0 if sh)

orient orientation west longitude of the grid

alat latitude in degrees

along longitude in degrees

5.3.14 polfixs (Source File: polfixs.F90)

REVISION HISTORY:

```
04-10-96  Mark Iredell; Initial Specification
```

INTERFACE:

```
subroutine polfixs(nm,nx,km,rlat,rlon,ib,lo,go)
  implicit none
```

ARGUMENTS:

```
  integer         :: nm
  integer         :: nx
  integer         :: km
  real            :: rlat(nm)
  real            :: rlon(nm)
  integer         :: ib
  logical*1       :: lo(nx)
  real            :: go(nx)
```

DESCRIPTION:

This subroutine averages multiple pole scalar values on a latitude/longitude grid. bitmaps may be averaged too.
The arguments are:

nm number of grid points

nx leading dimension of fields

rlat latitudes in degrees

rlon longitudes in degrees

ib integer bitmap flags

lo logical bitmaps

go returned scalar value

5.3.15 pstoll (Source File: pstoll.F90)

REVISION HISTORY:

```
05 aug 1998 initial version....................ssgt mccormick/dnxm
10 aug 1999 ported to ibm sp2.  added intent attributes to
            arguments...............................mr gayno/dnxm
25 jul 2005 Adopted in LIS
```

INTERFACE:

```
subroutine pstoll( hemi, pose, ri, rj, mesh, rlat, rlon )

  implicit none
```

ARGUMENTS:

```
  integer,       intent(in)    :: hemi
  integer,       intent(in)    :: pose
  real,          intent(in)    :: ri
  real,          intent(in)    :: rj
  integer,       intent(in)    :: mesh
  real,          intent(out)   :: rlat
  real,          intent(out)   :: rlon
```

DESCRIPTION:

converts polar stereographic grid i/j points to lat/lon
method:

initialize grid specific constants.
calculate distance between input point and center point of grid.
using this distance, calculate latitude and longitude using trigonometry.
adjust sign of longitude according to user preference.

The arguments are:

hemi index of the hemisphere (1-nh, 2-sh)

pose longitude increment orientation flag (1-positive east, 0-positive west)

ri i-coordinate

rj j-coordinate

mesh mesh factor

rlat output latitude in degrees

rlon output longitude in degrees

5.3.16 snsoidxyTolatlon (Source File: snsoidxyTolatlon.F90)

REVISION HISTORY:

```
 03-14-09  Soni Yatheendradas; Adopted in LIS 5.0 BETA RFS version from GCTP
CAUTION: This code is specific only to RFS project Sierra Nevada basins zone and not generic
```

INTERFACE:

```
subroutine snsoidxyTolatlon(xx,yy,npts,false_easting,false_northing,RRR,lon_center,alat,along)

  implicit none
```

ARGUMENTS:

```
  real,intent(in)     :: xx(npts)
  real,intent(in)     :: yy(npts)
  integer,intent(in)  :: npts
  real,intent(in)     :: false_easting
  real,intent(in)     :: false_northing
  real,intent(in)     :: RRR
  real,intent(in)     :: lon_center
  real     :: alat(npts)
  real     :: along(npts)
```

DESCRIPTION:

Converts the coordinates of a location on earth from the sinusiodal grid (x,y) co-ordinate system of the MODIS tile over the RFS project Sierra Nevada basins, to the natural coordinate system of latitude/longitude grid IMPORTANT: This is a VECTORIZED code
The arguments are:

xx xx of the point/s

yy yy of the point/s

npts number of point/s

false_easting False easting in the same units as the semi-major axis

false_northing False northing in the same units as the semi-major axis

RRR Radius of the reference sphere

lon_center Longitude of the central meridian

alat latitude in degrees

along longitude in degrees

5.3.17 upscaleByAveraging (Source File: upscaleByAveraging.F90)

INTERFACE:

```
subroutine upscaleByAveraging(mi,mo,&
     udef, n11,li, gi, lo, go)

  implicit none
```

ARGUMENTS:

```
integer             :: mi
integer             :: mo
real                :: udef
integer             :: n11(mi)
logical*1           :: li(mi)
real                :: gi(mi)
logical*1           :: lo(mo)
real                :: go(mo)
```

DESCRIPTION:

This subprogram upscales scalar data from a finer grid to a coarser grid using linear averaging. The grids are defined by their grid description arrays.
The grid description arrays are based on the decoding schemes used by NCEP.
The current code recognizes the following projections:
(gridDesc(1)=0) equidistant cylindrical

where gridDesc could be defined for either the input grid or the output grid.
The arguments are:

mi total number of points in the input grid

mo total number of points in the output grid

udef undefine value

n11 array that maps the location of each input grid point in the output grid.

li input bit mask

gi input data (finer grid)

lo output bit mask

go output data (coarser grid)

5.3.18 upscaleByAveraging_input (Source File: upscaleByAveraging_input.F90)

INTERFACE:

```
subroutine upscaleByAveraging_input (gridDesci,gridDesco,mi,mo,&
     n11)

  implicit none
```

ARGUMENTS:

```
real, intent(in)    :: gridDesci(50)
real                :: gridDesco(50)
integer             :: mi
integer             :: mo
integer             :: n11(mi)
```

DESCRIPTION:

This subprogram performs issues calls to compute the neighbor information for upscaling data for scalar fields The grids are defined by their grid description arrays.
The basic logic is as follows: The code first computes the earth coordinates (lat/lon) for the input grid (fine resolution grid). Then it looks up the grid coordinates (x,y) of each lat/lon in the output grid (coarse resolution grid). The code stores these mappings in the n11 array, which represents the location of each input grid point in the output grid.
Note that the input grid needs to be setup either as a global grid or with halos padded around it (if using a subset of the global grid), for the algorithm to work properly in a multiprocessor environment.This is to avoid a case where the required input grid points are split across different processors.
The grid description arrays are based on the decoding schemes used by NCEP.
The current code recognizes the following projections:
(gridDesc(1)=0) equidistant cylindrical
where gridDesc could be defined for either the input grid or the output grid.
The arguments are:

gridDesci input grid description parameters

gridDesco output grid description parameters

mi total number of points in the input grid

mo total number of points in the output grid

n11 array that maps the location of each input grid point in the output grid.

5.3.19 zterp (Source File: zterp.F90)

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
7/24/07 Brian Cosgrove; Fixed bug in the loop which calculates the
        average zenith angle...had been one time interval
        too far into the future, gmt now is not incremented by
        interval until second time through loop
```

INTERFACE:

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

USES:

```
use LIS_PRIV_rcMod
use LIS_logMod,    only : LIS_logunit
implicit none
```

ARGUMENTS:

```
type (lisrcdec)  :: lis
integer        :: iflag
real           :: lat,lon
real           :: mbtime
integer        :: julianb
real           :: interval
real           :: btime
real           :: etime
real           :: weight1
real           :: weight2
```

DESCRIPTION:

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)
The arguments are:

iflag Flag specifying if input data is time averaged or not.

lis instance of the `lis_module`

lat latitude of the current point

lon longitude of the current point

btime beginning time of the original averaged data 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.

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).

mbtime Time of current time step.

julianb Julian day upon which BTIME falls

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

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

czbegdata cosine of zenith angle at the beginning timestep

czenddata cosine of zenith angle at the ending timestep

czmodel cosine of the zenith angle at the current timestep

The routines invoked are:

LIS_localtime (5.3.21)
compute the LIS_localtime based on the GMT

coszenith (5.3.20)
compute the cosine of zenith angle

5.3.20 coszenith (Source File: zterp.F90)

Returns the cosine of the zenith angle from the following parameters
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,dec,omega)
```

USES:

```
use LIS_logMod,    only : LIS_logunit

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)
```

5.3.21 LIS_localtime (Source File: zterp.F90)

Calculates the local time based on GMT

INTERFACE:

```
subroutine LIS_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)
```

Part VI
User-defined extensible components in LIS

6 User-defined extensible components in LIS

The following sections describe the details of user-defined extensible components in LIS. These include the following:

Running Modes

Domains

Land Surface Parameters

Meteorological forcing analyses

Land Surface Models

Data Assimilation

Part VII
Running Modes in LIS

7 Running Modes in LIS

This section contains the implementations of various running modes in LIS. The current implementation includes a retrospective mode and the cycling mode used in the AGRMET system. The interfaces are expected to be extended for other running modes such as coupled, parameter estimation and forecast.

---

8 optUE Running Mode

This section describes the optUE running mode.

8.1 Fortran: Module Interface paramEstim_runMod (Source File: paramEstim_runMod.F90)

This module contains the definition of the functions used for LIS initialization, execution, and finalization for parameter estimation instances.

REVISION HISTORY:

```
 21Oct05    Sujay Kumar  Initial Specification

implicit none

-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
 public :: lis_init_paramEstim  !init method for optimization-uncertainty modeling mode
 public :: lis_run_paramEstim   !run method for optimization-uncertainty modeling mode
 public :: lis_final_paramEstim !finalize method for optimization-uncertainty modeling mode
```

8.1.1 lis_init_paramEstim (Source File: paramEstim_runMod.F90)

INTERFACE:

```
subroutine lis_init_paramEstim
```

USES:

```
  use LIS_coreMod,         only : LIS_core_init
  use LIS_domainMod,       only : LIS_domain_init
  use LIS_surfaceModelMod, only : LIS_surfaceModel_init, &
       LIS_surfaceModel_setup, LIS_surfaceModel_readrestart
  use LIS_metforcingMod,  only : LIS_metforcing_init
  use LIS_perturbMod,     only : LIS_perturb_init, LIS_perturb_readrestart
  use LIS_DAobservationsMod, only : LIS_initDAObservations
  use LIS_dataAssimMod,   only : LIS_dataassim_init
  use LIS_paramsMod,      only : LIS_param_init
  use LIS_routingMod,     only : LIS_routing_init, LIS_routing_readrestart
  use LIS_RTMMod,          only : LIS_RTM_init
  use LIS_appMod,          only : LIS_appModel_init
```

```
use LIS_optUEMod, only : LIS_optUE_init, &
     LIS_objectiveFunc_init,LIS_optUEAlg_init
use LIS_PE_HandlerMod,    only : LIS_PE_init, LIS_PE_restart, &
     LIS_setPEDecisionSpace
```

DESCRIPTION:

This is the initialize method for LIS in a optUE running mode. The following calls are invoked from this method.

LIS_domain_init (3.12.2)
initialize the LIS domains

LIS_param_init (3.28.2)
initialize parameters

LIS_lsm_init (3.22.3)
initialize the land surface model.

LIS_metforcing_init (3.23.2)
initialize the met forcing

LIS_setuplsm (3.22.4)
complete the LSM setups

LIS_lsm_readrestart (3.22.7)
read the restart files

8.1.2 lis_run_paramEstim (Source File: paramEstim_runMod.F90)

INTERFACE:

```
subroutine lis_run_paramEstim
```

USES:

```
use LIS_coreMod,  only : LIS_rc, LIS_endofrun, LIS_timetoRunNest, &
     LIS_ticktime, LIS_core_init
use LIS_surfaceModelMod, only : LIS_surfaceModel_readrestart, LIS_surfaceModel_init, &
     LIS_surfaceModel_setup, LIS_surfaceModel_finalize, LIS_surfaceModel_reset
use LIS_metforcingMod,  only :  LIS_metforcing_init, LIS_metforcing_reset, LIS_metforcing_finalize
use LIS_perturbMod,     only : LIS_perturb_init, LIS_perturb_readrestart
use LIS_DAobservationsMod, only : LIS_initDAObservations
use LIS_dataAssimMod,   only : LIS_dataassim_init
use LIS_routingMod,     only : LIS_routing_init, LIS_routing_readrestart
use LIS_RTMMod,          only : LIS_RTM_init
use LIS_appMod,          only : LIS_appModel_init
use LIS_paramsMod,       only : LIS_param_reset, LIS_param_init, LIS_param_finalize
use LIS_timeMgrMod,      only : LIS_resetClock, LIS_timemgr_init
use LIS_logMod,          only : LIS_flush, LIS_logunit
use LIS_optUEMod, only : LIS_isOptStopCriterionTrue,&
     LIS_runOptUEAlg
use LIS_PE_HandlerMod,   only : LIS_readPEobs, LIS_computePEobjectiveFunc,&
     LIS_resetPEObjectiveFunc, LIS_resetPEobs,LIS_updatePEObjectiveFunc,&
     LIS_setPEDecisionSpace
use LIS_logMod,          only : LIS_logunit, LIS_flush
```

DESCRIPTION:

This is the run method for LIS in a optUE running mode. The following calls are invoked from this method.

8.1.3 lis_final_paramEstim (Source File: paramEstim_runMod.F90)

INTERFACE:

```
subroutine lis_final_paramEstim
```

USES:

```
use LIS_coreMod,       only : lis_finalize
use LIS_surfaceModelMod,          only : LIS_surfaceModel_finalize
use LIS_paramsMod,        only : LIS_param_finalize
use LIS_metforcingMod,  only : LIS_metforcing_finalize
```

DESCRIPTION:

This is the finalize method for LIS in a optUE running mode. The following calls are invoked from this method.

LIS_finalize (3.10.12)
cleanup LIS generic structures

LIS_surfaceModel_finalize (3.34.14)
cleanup land surface model specific structures

LIS_param_finalize (3.28.5)
cleanup parameter specific structures

LIS_metforcing_finalize (3.23.12)
cleanup metforcing specific structures

LIS_dataassim_finalize (3.11.5)
cleanup data assimilation specific structures

---

9 Retrospective Running Mode

This is a typical running mode used in LIS to perform retrospective simulations. The mode assumes that all the meteorological forcing analyes required for the specified time period is archived for the simulation.

9.1 Fortran: Module Interface retrospective_runMod (Source File: retrospective_runMod.F90)

This is a typical running mode used in LIS to perform retrospective simulations. The mode assumes that all the meteorological forcing analyes required for the specified time period is archived for the simulation. This module contains the definition of the functions used for LIS initialization, execution, and finalization for a retrospective runmode in LIS.

REVISION HISTORY:

```
 21Oct05    Sujay Kumar  Initial Specification

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: LIS_init_retrospective  !init method for retrospective mode
public :: LIS_run_retrospective   !run method for retrospective mode
public :: LIS_final_retrospective !finalize method for retrospective mode
```

9.1.1 LIS_init_retrospective (Source File: retrospective_runMod.F90)

INTERFACE:

```
subroutine LIS_init_retrospective
```

USES:

```
use LIS_coreMod,           only : LIS_core_init
use LIS_domainMod,         only : LIS_domain_init
use LIS_surfaceModelMod,   only : LIS_surfaceModel_init,  &
                                  LIS_surfaceModel_setup, &
                                  LIS_surfaceModel_readrestart
use LIS_metforcingMod,     only : LIS_metforcing_init
use LIS_DAobservationsMod, only : LIS_initDAObservations
use LIS_perturbMod,        only : LIS_perturb_init, LIS_perturb_readrestart
use LIS_dataAssimMod,      only : LIS_dataassim_init
use LIS_paramsMod,         only : LIS_param_init
use LIS_routingMod,        only : LIS_routing_init, LIS_routing_readrestart
use LIS_irrigationMod,     only : LIS_irrigation_init
use LIS_RTMMod,            only : LIS_RTM_init
use LIS_appMod,            only : LIS_appModel_init

use LIS_tbotAdjustMod,     only : LIS_createTmnUpdate
```

DESCRIPTION:

This is the initialize method for LIS in a retrospective running mode. The following calls are invoked from this method.

LIS_domain_init (3.12.2)
initialize the LIS domains

LIS_param_init (3.28.2)
initialize parameters

LIS_lsm_init (3.22.3)
initialize the land surface model.

LIS_metforcing_init (3.23.2)
initialize the met forcing

LIS_initDAObservations (3.1.1)
initialize structures needed to read observations for data assimilation

LIS_setuplsm (3.22.4)
complete the LSM setups

LIS_lsm_readrestart (3.22.7)
read the restart files

9.1.2 lis_run_retrospective (Source File: retrospective_runMod.F90)

INTERFACE:

```
subroutine lis_run_retrospective
```

USES:

```
use LIS_coreMod,           only : LIS_rc, LIS_endofrun, LIS_timetoRunNest, &
                                  LIS_ticktime
use LIS_surfaceModelMod,   only : LIS_surfaceModel_f2t, LIS_surfaceModel_run,&
                                  LIS_surfaceModel_output, LIS_surfaceModel_writerestart, &
                                  LIS_surfaceModel_perturb_states
use LIS_paramsMod,         only : LIS_setDynparams
use LIS_metforcingMod,     only : LIS_get_met_forcing, LIS_perturb_forcing
use LIS_perturbMod,        only : LIS_perturb_writerestart
use LIS_DAobservationsMod, only : LIS_readDAobservations, &
                                  LIS_perturb_DAobservations
use LIS_dataAssimMod,      only : LIS_dataassim_run, LIS_dataassim_output
use LIS_routingMod,        only : LIS_routing_run, LIS_routing_writeoutput, &
                                  LIS_routing_writerestart
use LIS_irrigationMod,     only : LIS_irrigation_run,LIS_irrigation_output
use LIS_appMod,            only : LIS_runAppModel, LIS_outputAppModel
use LIS_RTMMod,            only : LIS_RTM_run, LIS_RTM_output
use LIS_logMod,            only : LIS_flush, LIS_logunit
```

DESCRIPTION:

This is the run method for LIS in a retrospective running mode. The following calls are invoked from this method.

LIS_endofrun (3.10.8)
check to see if the end of simulation has reached

LIS_ticktime (3.10.7)
advance model clock

LIS_timeToRunNest (3.10.10)
check to see if the current nest needs to be run.

LIS_setDynparams (3.28.3)
set the time dependent parameters

LIS_get_met_forcing (3.23.7)
retrieve the met forcing

LIS_perturb_forcing (3.23.10)
perturbs the met forcing

LIS_irrigation_run (3.18.2)
run the irrigation model

LIS_surfaceModel_f2t (3.34.9)
transfer forcing to model tiles

LIS_surfaceModel_run (3.34.8)
run the land surface model

LIS_surfaceModel_perturb_states (3.34.12)
perturb the land surface model states

LIS_readDAobservations (3.1.2)
read observations to be used for data assimilation

LIS_perturb_DAobservations (3.1.3)
perturb observations to be used for data assimilation

LIS_dataassim_run (3.11.3)
run the data assimilation algorithm

LIS_dataassim_output (3.11.4)
write da output

LIS_surfaceModel_output (3.34.10)
write surface model output

LIS_surfaceModel_writerestart (3.34.11)
write surface model restart files

LIS_irrigation_output (3.18.3)
write irrigation model output

LIS_routing_run (3.31.3)
run the routing model

LIS_routing_writeoutput (3.31.4)
write routing model output

LIS_routing_writerestart (3.31.5)
write routing model restart data

LIS_rtm_run (**??**)
run the radiative transfer model

LIS_rtm_output (**??**)
write radiative transfer model output

LIS_runAppModel (3.8.2)
run the application model

LIS_outputAppModel (3.8.3)
write the application model output

9.1.3 lis_final_retrospective (Source File: retrospective_runMod.F90)

INTERFACE:

```
subroutine lis_final_retrospective
```

USES:

```
use LIS_coreMod,         only : LIS_finalize
use LIS_logMod,          only : LIS_flush, LIS_logunit
use LIS_surfaceModelMod, only : LIS_surfaceModel_finalize
use LIS_paramsMod,       only : LIS_param_finalize
use LIS_metforcingMod,   only : LIS_metforcing_finalize
use LIS_RTMMod,          only : LIS_RTM_finalize
use LIS_appMod,          only : LIS_appModel_finalize ! SY
```

DESCRIPTION:

This is the finalize method for LIS in a retrospective running mode. The following calls are invoked from this method.

LIS_finalize (3.10.12)
cleanup LIS generic structures

LIS_lsm_finalize (3.22.12)
cleanup land surface model specific structures

LIS_param_finalize (3.28.5)
cleanup parameter specific structures

LIS_metforcing_finalize (3.23.12)
cleanup metforcing specific structures

LIS_RTM_finalize (**??**)
cleanup radiative transfer model specific structures

LIS_dataassim_finalize (3.11.5)
cleanup data assimilation specific structures

9.2 Fortran: Module Interface smootherDA_runMod (Source File: smootherDA_runMod.F9

This module contains the definition of the functions used for LIS initialization, execution, and finalization for an optimization and uncertainty estimation runmode in LIS.

REVISION HISTORY:

```
 21Oct05    Sujay Kumar  Initial Specification

use ESMF

implicit none

-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
 public :: lis_init_smootherDA  !init method for optimization-uncertainty modeling mode
 public :: lis_run_smootherDA   !run method for optimization-uncertainty modeling mode
 public :: lis_final_smootherDA !finalize method for optimization-uncertainty modeling mode
```

9.2.1 lis_init_smootherDA (Source File: smootherDA_runMod.F90)

INTERFACE:

```
subroutine lis_init_smootherDA
```

USES:

```
use LIS_coreMod
use LIS_domainMod
use LIS_surfaceModelMod
use LIS_metforcingMod
use LIS_DAobservationsMod
use LIS_perturbMod
use LIS_dataAssimMod
use LIS_paramsMod
use LIS_irrigationMod
use LIS_routingMod
use LIS_logMod
```

DESCRIPTION:

This is the initialize method for LIS in a retrospective running mode. The following calls are invoked from this method.

LIS_domain_init (3.12.2)
initialize the LIS domains

LIS_param_init (3.28.2)
initialize parameters

LIS_surfaceModel_init (3.34.5)
initialize the land surface model

LIS_metforcing_init (3.23.2)
initialize the met forcing

LIS_initDAObservations (3.1.1)
initialize structures needed to read observations for data assimilation

LIS_surfaceModel_setup (3.34.6)
complete the LSM setups

LIS_surfaceModel_readrestart (3.34.7)
read the surface model restart files

LIS_perturb_readrestart (3.29.2)
read the perturbation restart files

LIS_routing_init (3.31.1)
initialize the routing model

LIS_routing_readrestart (3.31.2)
read the routing model restart files

LIS_core_init (3.10.6)
completes the setting of LIS' clocks and alarms

9.2.2 lis_run_smootherDA (Source File: smootherDA_runMod.F90)

INTERFACE:

```
subroutine lis_run_smootherDA
```

USES:

```
use LIS_coreMod
use LIS_timeMgrMod
use LIS_domainMod
use LIS_surfaceModelMod
use LIS_paramsMod
use LIS_metforcingMod
use LIS_perturbMod
use LIS_DAobservationsMod
use LIS_dataAssimMod
use LIS_routingMod
use LIS_irrigationMod
use LIS_logMod
```

DESCRIPTION:

This is the run method for LIS in a smootherDA running mode. The following calls are invoked from this method.

9.2.3 lis_final_smootherDA (Source File: smootherDA_runMod.F90)

INTERFACE:

```
subroutine lis_final_smootherDA
```

USES:

```
use LIS_coreMod
use LIS_surfaceModelMod
use LIS_paramsMod
use LIS_metforcingMod
```

DESCRIPTION:

This is the finalize method for LIS in a smootherDA running mode. The following calls are invoked from this method.

LIS_finalize (3.10.12)
cleanup LIS generic structures

LIS_surfaceModel_finalize (3.34.14)
cleanup land surface model specific structures

LIS_param_finalize (3.28.5)
cleanup parameter specific structures

LIS_metforcing_finalize (3.23.12)
cleanup metforcing specific structures

9.3 Fortran: Module Interface LISWRFGridCompMod (Source File: LISWRFGridCompMod.F90)

This module defines the handling of the fields being exchanged between LIS and WRF.

REVISION HISTORY:

```
9 Nov 2008: Sujay Kumar, Initial Specification
```

USES:

```
use LISWRFimport_module
use LISWRFexport_module
```

9.4 Fortran: Module Interface LISWRFexport_module (Source File: LISWRFexport_module.F90)

Defines the export state variables from LIS

REVISION HISTORY:

```
9 Nov 2005: Sujay Kumar, Initial Specification
```

USES:

9.5 Fortran: Module Interface LISWRFimport_module (Source File: LISWRFimport_module.F90)

Defines the import state variables into LIS

REVISION HISTORY:

```
9 Nov 2005: Sujay Kumar, Initial Specification
```

USES:

9.5.1 liswrffinalize_coupled (Source File: liswrffinalize_coupled.F90)

REVISION HISTORY:

```
14 Aug 2014  James Geiger  Initial Specification
```

USES:

```
use LISWRFGridCompMod,     only : LISWRF_import
use LIS_coreMod,           only : LIS_rc
use LIS_timeMgrMod,        only : LIS_timemgr_set
use LIS_surfaceModelMod,   only : LIS_surfaceModel_output,         &
                                  LIS_surfaceModel_writerestart
use LIS_logMod,            only : LIS_verify, LIS_flush, LIS_logunit
```

9.5.2 liswrfinit_coupled (Source File: liswrfinit_coupled.F90)

REVISION HISTORY:

```
 14Nov02    Sujay Kumar  Initial Specification
 21Oct05    Sujay Kumar  Modified to include the runmodes. Switched
                         to a init,run,finalize mode
```

USES:

```
use LIS_coreMod
use LIS_domainMod,       only : LIS_domain_init
use LIS_surfaceModelMod, only : LIS_surfaceModel_init, &
     LIS_surfaceModel_setup, LIS_surfaceModel_readrestart
use LIS_metforcingMod,  only : LIS_metforcing_init
use LIS_DAobservationsMod, only : LIS_initDAObservations
use LIS_perturbMod,      only : LIS_perturb_init, LIS_perturb_readrestart
use LIS_dataAssimMod,    only : LIS_dataassim_init
use LIS_paramsMod,        only : LIS_param_init
use LISWRFGridCompMod,    only : LISWRF_alloc_states
use LIS_logMod,           only : LIS_logunit, LIS_flush

use LIS_tbotAdjustMod,    only : LIS_createTmnUpdate

  implicit none
```

ARGUMENTS:

```
  integer, intent(in)     :: nx, ny
```

DESCRIPTION:

This routine defines the set of steps required from LIS during the initialization of the LIS-WRF system.

9.5.3 liswrfrun_coupled (Source File: liswrfrun_coupled.F90)

REVISION HISTORY:

```
14Nov02    Sujay Kumar  Initial Specification
21Oct05    Sujay Kumar  Modified to include the runmodes.
                        Switched to a init,run,finalize mode
July12     C. Carouge   Modified to include CO2 as import.
                        Plus fix for nested domains.
```

USES:

```
use ESMF
use LISWRFGridCompMod
use LIS_coreMod
use LIS_metforcingMod
use LIS_timeMgrMod
use LIS_surfaceModelMod
use LIS_paramsMod
use LIS_metforcingMod
use LIS_perturbMod
use LIS_DAobservationsMod
use LIS_dataAssimMod
use LIS_logMod
use LIS_FORC_AttributesMod
```

Part VIII
Domain implementations in LIS

10 Domain implementations in LIS

This section contains the implementations of interfaces to incorporate a new domain in LIS. Currently LIS supports operations on a lat/lon, polar stereographic, lambert conformal, mercator, and GSWP grids.

---

11 UTM domain

This section contains the interface implementations for a LIS instance using a UTM projection.

11.0.4 readinput_UTM (Source File: readinput_UTM.F90)

REVISION HISTORY:

```
28 Jun 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine readinput_UTM
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc, LIS_domain, LIS_config, &
     LIS_localPet, LIS_npes, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
     LIS_nss_halo_ind, LIS_nse_halo_ind
use LIS_domainMod, only : LIS_quilt_domain, LIS_setParameterDomainSpecs
use LIS_logMod,    only : LIS_logunit, LIS_verify, LIS_endrun
use LIS_fileIOMod, only : LIS_readDomainConfigSpecs
use map_utils
```

DESCRIPTION:

This routine reads the options specified in the LIS configuration file to determine the region of interest in the LIS simulation, in a lat/lon projection. The routine reads the extents of the running domain. The land surface parameters used for the simulation are read from a file that spans the area of interest. The domain specifications of the parameter maps are also read in by this routine. Based on the number of processors and the processor layout specified, this routine also performs the domain decomposition.
The routines invoked are:

listask_for_point (3.51.11)
routine to perform domain decomposition

11.0.5 readinput_hrap (Source File: readinput_hrap.F90)

REVISION HISTORY:

```
15 Jul 2005: Sujay Kumar; Initial Specification
28 Jan 2014: Shugong Wang; Replace run domain specifications from lat-lon
             into HRAP X and Y
```

INTERFACE:

```
subroutine readinput_hrap
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc, LIS_domain, LIS_config, LIS_localPet, &
     LIS_masterproc, LIS_npes, &
     LIS_ews_halo_ind, LIS_nss_halo_ind, LIS_ewe_halo_ind, LIS_nse_halo_ind
use LIS_domainMod, only : LIS_quilt_domain, LIS_setParameterDomainSpecs
use LIS_logMod,    only : LIS_logunit, LIS_endrun, LIS_verify
use LIS_fileIOMod, only : LIS_readDomainConfigSpecs
use LIS_XMRG_Reader, only : hrap_to_latlon
use map_utils
use netcdf

implicit none
```

DESCRIPTION:

This routine reads the options specified in the LIS configuration file to determine the region of interest in the LIS simulation, in a hrap projection. The routine reads the extents of the running domain. The land surface parameters used for the simulation are read from a file that spans the area of interest. The domain specifications of the parameter maps are also read in by this routine. Based on the number of processors and the processor layout specified, this routine also performs the domain decomposition.
The routines invoked are:

listask_for_point (3.51.11)
routine to perform domain decomposition

---

12 Lambert Conformal domain

This section contains the interface implementations for a LIS instance using a lambert conformal projection with SW to NE data ordering

---

12.0.6 readinput_lambert (Source File: readinput_lambert.F90)

REVISION HISTORY:

```
15Jul2005: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine readinput_lambert
```

USES:

```
use ESMF
use map_utils
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_config, &
     LIS_localPet, LIS_masterproc, LIS_npes, &
```

```
       LIS_ews_halo_ind, LIS_nss_halo_ind, LIS_ewe_halo_ind, &
       LIS_nse_halo_ind
  use LIS_domainMod, only : LIS_quilt_domain, LIS_setParameterDomainSpecs
  use LIS_fileIOMod, only : LIS_readDomainConfigSpecs
  use LIS_logMod
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif

  implicit none
```

DESCRIPTION:

This routine reads the options specified in the LIS configuration file to determine the region of interest in the LIS simulation, in a lambert conformal projection. The routine reads the extents of the running domain. The land surface parameters used for the simulation are read from a file that spans the area of interest. The domain specifications of the parameter maps are also read in by this routine. Based on the number of processors and the processor layout specified, this routine also performs the domain decomposition.
The routines invoked are:

listask_for_point (3.51.11)
routine to perform domain decomposition

---

13 Lat/Lon domain

This section contains the interface implementations for a LIS instance using a lat/lon projection with SW to NE data ordering

---

13.0.7 readinput_latlon (Source File: readinput_latlon.F90)

REVISION HISTORY:

```
    11 Apr 2000: Brian Cosgrove; Added Elevation correction and Forcing
                 Mask read statements
    14 Nov 2003: Sujay Kumar; Modified card file that includes regional
                 modeling options
```

INTERFACE:

```
subroutine readinput_latlon
```

USES:

```
  use ESMF
  use LIS_coreMod,   only : LIS_rc, LIS_domain, LIS_config, &
       LIS_localPet, LIS_npes, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
       LIS_nss_halo_ind, LIS_nse_halo_ind
  use LIS_domainMod, only : LIS_quilt_domain, LIS_setParameterDomainSpecs
  use LIS_logMod,    only : LIS_logunit, LIS_endrun, LIS_verify
  use LIS_fileIOMod, only : LIS_readDomainConfigSpecs
  use map_utils
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
```

```
  use netcdf
#endif
```

DESCRIPTION:

This routine reads the options specified in the LIS configuration file to determine the region of interest in the LIS simulation, in a lat/lon projection. The routine reads the extents of the running domain. The land surface parameters used for the simulation are read from a file that spans the area of interest. The domain specifications of the parameter maps are also read in by this routine. Based on the number of processors and the processor layout specified, this routine also performs the domain decomposition.
The routines invoked are:

listask_for_point (3.51.11)
routine to perform domain decomposition

---

14 Mercator domain

This section contains the interface implementations for a LIS instance using a mercator projection with SW to NE data ordering

---

14.0.8 readinput_merc (Source File: readinput_merc.F90)

REVISION HISTORY:

```
    15 Jul 2005: Sujay Kumar; Initial Specification
```

INTERFACE:

```
 subroutine readinput_merc
```

USES:

```
  use ESMF
  use map_utils
  use LIS_coreMod, only :LIS_rc, LIS_domain, LIS_config, LIS_localPet, &
       LIS_masterproc, LIS_npes, &
       LIS_ews_halo_ind, LIS_nss_halo_ind, LIS_ewe_halo_ind, LIS_nse_halo_ind
  use LIS_domainMod, only : LIS_quilt_domain, LIS_setParameterDomainSpecs
  use LIS_fileIOMod, only : LIS_readDomainConfigSpecs
  use LIS_mpiMod
  use LIS_logMod
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
```

DESCRIPTION:

This routine reads the options specified in the LIS configuration file to determine the region of interest in the LIS simulation, in a mercator projection. The routine reads the extents of the running domain. The land surface parameters used for the simulation are read from a file that spans the area of interest. The domain specifications of the parameter maps are also read in by this routine. Based on the number of processors and the processor layout specified, this routine also performs the domain decomposition.
The routines invoked are:

listask_for_point (3.51.11)
routine to perform domain decomposition

---

15 Polar stereographic domain

This section contains the interface implementations for a LIS instance using a polar stereographic projection with SW to NE data ordering

---

15.0.9 readinput_polar (Source File: readinput_polar.F90)

REVISION HISTORY:

```
15 Jul 2005: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine readinput_polar
```

USES:

```
  use ESMF
  use map_utils
  use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_config, &
       LIS_localPet, LIS_masterproc, LIS_npes, &
       LIS_ews_halo_ind, LIS_nss_halo_ind, LIS_ewe_halo_ind, LIS_nse_halo_ind
  use LIS_domainMod, only : LIS_quilt_domain, LIS_setParameterDomainSpecs
  use LIS_fileIOMod, only : LIS_readDomainConfigSpecs
  use LIS_logMod
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif

  implicit none
```

DESCRIPTION:

This routine reads the options specified in the LIS configuration file to determine the region of interest in the LIS simulation, in a polar stereographic projection. The routine reads the extents of the running domain. The land surface parameters used for the simulation are read from a file that spans the area of interest. The domain specifications of the parameter maps are also read in by this routine. Based on the number of processors and the processor layout specified, this routine also performs the domain decomposition.
The routines invoked are:

listask_for_point (3.51.11)
routine to perform domain decomposition

Part IX
Land Surface Parameters in LIS

16 Land Surface Parameters in LIS

This section contains the implementations of various land surface parameter maps. The interfaces are primarily geared towards including satellite and observed maps of parameters such as albedo, landcover, greenness, LAI, and soil parameters.

---

17 Greenness Fraction Data

This section describes the routines to ingest various sources of greenness fraction data in different LIS domains

---

17.0.10 read_NESDISgfrac (Source File: read_NESDISgfrac.F90)

REVISION HISTORY:

```
25 Jul 2005: Sujay Kumar; Initial Specification
20 Feb 2006: Sujay Kumar; Modified to support nesting
```

INTERFACE:

```
subroutine read_NESDISgfrac(n, wt1, wt2, array1, array2)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_logMod,         only : LIS_logunit, LIS_verify, LIS_endrun
use LIS_vegDataMod,     only : LIS_gfrac
use LIS_timeMgrMod

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
real                :: wt1
real                :: wt2
real, intent(inout) :: array1(LIS_rc%ntiles(n))
real, intent(inout) :: array2(LIS_rc%ntiles(n))
```

DESCRIPTION:

This subroutine retrieves the greenness fraction climatology for the specified month and returns the values in the latlon projection
The arguments are:

n index of the nest

mo time index (month or quarter)

array output field with the retrieved greenness fraction

17.0.11 read_SPORTgfrac (Source File: read_SPORTgfrac.F90)

REVISION HISTORY:

```
25 Jul 2005: Sujay Kumar; Initial Specification
20 Feb 2006: Sujay Kumar; Modified to support nesting
10 Aug 2010: Jonathan Case: Modified for daily SPoRT GVF composites.
```

INTERFACE:

```
subroutine read_SPORTgfrac(n, wt1, wt2, array1, array2)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_vegDataMod,     only : LIS_gfrac
use LIS_fileIOMod,     only : LIS_readData
use LIS_timeMgrMod

implicit none
```

ARGUMENTS:

```
integer,          intent(in) :: n
real                         :: wt1
real                         :: wt2
real,          intent(inout) :: array1(LIS_rc%ntiles(n))
real,          intent(inout) :: array2(LIS_rc%ntiles(n))
```

DESCRIPTION:

This subroutine retrieves the greenness fraction climatology for the specified month and returns the values in the latlon projection
The arguments are:

n index of the nest

mo time index (month or quarter)

array output field with the retrieved greenness fraction

17.0.12 setup_SPORTgfrac (Source File: setup_SPORTgfrac.F90)

REVISION HISTORY:

```
16 Jul 2008: Sujay Kumar; Initial Specification
10 Aug 2010: Jonathan Case: Modified for SPORT/MODIS GVF.
```

INTERFACE:

```
subroutine setup_SPORTgfrac(n)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_logMod
use LIS_vegDataMod,  only : LIS_gfrac
use LIS_timeMgrMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine sets up the variables required for reading the greenness fraction daily composite data from SPORT
The arguments are:

n index of the nest

---

18 Leaf Area Index Data

This section describes the routines to ingest various sources of LAI data in different LIS domains

---

18.0.13 read_MODISlai (Source File: read_MODISlai.F90)

REVISION HISTORY:

```
25 Jul 2005: Sujay Kumar; Initial Specification
20 Feb 2006: Sujay Kumar; Modified to support nesting
```

INTERFACE:

```
subroutine read_MODISlai(n, time1, array)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc, LIS_domain, LIS_localPet
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_vegDataMod,    only : LIS_lai
use LIS_fileIOMod,     only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_Time)     :: time1
real, intent(inout) :: array(LIS_rc%ntiles(n))
```

DESCRIPTION:

This subroutine retrieves the LAI climatology for the specified month and returns the values in the latlon projection
The arguments are:

n index of the nest

mo time index (month or quarter)

array output field with the retrieved LAI

18.0.14 read_MODISsai (Source File: read_MODISsai.F90)

REVISION HISTORY:

```
25 Jul 2005: Sujay Kumar; Initial Specification
20 Feb 2006: Sujay Kumar; Modified to support nesting
```

INTERFACE:

```
subroutine read_MODISsai(n, time1, array)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc, LIS_domain, LIS_localPet
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_vegDataMod,     only : LIS_sai
use LIS_fileIOMod,      only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_Time)     :: time1
real, intent(inout) :: array(LIS_rc%ntiles(n))
```

DESCRIPTION:

This subroutine retrieves the SAI climatology for the specified month and returns the values in the latlon projection
The arguments are:

n index of the nest

mo time index (month or quarter)

array output field with the retrieved SAI

18.0.15 setup_MODISlai (Source File: setup_MODISlai.F90)

REVISION HISTORY:

```
16 Jul 2008: Sujay Kumar; Initial Specification
17 Oct 2011: Yudong Tian; Modified from gfrac to lai
```

INTERFACE:

```
subroutine setup_MODISlai(n)
```

USES:

```
use LIS_vegDataMod,  only : LIS_lai

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine sets up the variables required for reading the LAI climatology data from MODIS
The arguments are:

n index of the nest

Part X
Meteorological forcing analyses in LIS

19 TRMM 3B42RT

This section describes the implementation of the precipitation product derived from the TRMM 3B42RT satellite precipitation analysis generated by George Huffman at NASA GSFC.

19.1 Fortran: Module Interface TRMM3B42RT_forcingMod (Source File: TRMM3B42RT_f

This module contains variables and data structures that are used for the implementation of the precipitation product derived from the TRMM real-time multi-satellite precipitation analysis (3B42RT) from NASA GSFC (Huffman et al.2003)
Huffman, G.J.,R.F. Adler, E.F.Stocker, D.T.Bolvin, and E.J.Nelkin (2003): Analysis of TRMM 3-hourly multi-satellite precipitation estimates computed in real and post-real time. Combined preprints CD-ROM, 83rd AMS Annual Meeting, Paper P4.11 in 12th Conference on Sat. Meteor. and Oceanog. 9-13 Feb 2003, Long Beach CA. 6pp.
The implementation in LIS has the derived data type `TRMM3B42RT_struc` that includes the variables to specify the runtime options, and the weights and neighbor information for spatial interpolation
They are desribed below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

TRMM3B42RTdir Directory containing the input data

TRMM3B42RTtime The nearest hourly instance of the incoming data (as a real time).

griduptime1 The time to switch the input resolution or file naming convention

mi Number of points in the input grid

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n112,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

REVISION HISTORY:

```
Yudong Tian, 10/26/2010: Updated to support reading raw .gz files
```

USES:

```
use ESMF

implicit none
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
 ---------------------------------------------------------------------------
  public :: init_TRMM3B42RT      !defines the native resolution of
                                 !the input data
 ---------------------------------------------------------------------------
```

PUBLIC TYPES:

```
 ---------------------------------------------------------------------------
  public :: TRMM3B42RT_struc
```

19.1.1 init_TRMM3B42RT (Source File: TRMM3B42RT_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine init_TRMM3B42RT(findex)
```

USES:

```
  use LIS_coreMod, only: LIS_rc, LIS_config ! SY
  !use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep ! SY
  use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep, &
                             LIS_tick, LIS_parseTimeString, &
                             LIS_registerAlarm ! SY
  use LIS_logMod,    only : LIS_logunit, LIS_endrun, &
                            LIS_getNextUnitNumber, LIS_releaseUnitNumber, LIS_verify ! SY
  use ESMF

  implicit none

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

DESCRIPTION:

Defines the native resolution of the input forcing for TRMM 3B42RT data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). The routines invoked are:

readcrd_TRMM3B42RT (19.1.7)
reads the runtime options specified for TRMM 3B42RT data

LIS_date2time (3.36.24)
converts date to the real time format

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

19.1.2 finalize_TRMM3B42RT (Source File: finalize_TRMM3B42RT.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine finalize_TRMM3B42RT(findex)
```

USES:

```
use TRMM3B42RT_forcingMod, only : TRMM3B42RT_struc
```

DESCRIPTION:

Routine to cleanup TRMM 3B42RT forcing related memory allocations.

19.1.3 get_TRMM3B42RT (Source File: get_TRMM3B42RT.F90)

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
```

INTERFACE:

```
subroutine get_TRMM3B42RT(n,findex)
```

USES:

```
use LIS_coreMod, only           : LIS_rc, LIS_masterproc
use LIS_timeMgrMod,  only       : LIS_time2date, LIS_tick, LIS_get_nstep, &
                                  LIS_isAlarmRinging ! SY
use LIS_logMod, only            : LIS_logunit, LIS_endrun
use TRMM3B42RT_forcingMod, only : TRMM3B42RT_struc
use LIS_metforcingMod,  only    : LIS_forc ! SY

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 3-hrly, TRMM 3B42RT forcing. At the beginning of a simulation, the code reads the most recent past data (nearest 3 hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the TRMM 3B42RT data times

TRMM3B42RTfile (19.1.4)
Puts together appropriate file name for 3 hour intervals

read_TRMM3B42RT (19.1.6)
Interpolates TRMM 3B42RT data to LIS grid

19.1.4 TRMM3B42RTfile (Source File: get_TRMM3B42RT.F90)

INTERFACE:

```
subroutine TRMM3B42RTfile( name, TRMM3B42RTdir, yr, mo, da, hr)
```

USES:

```
use TRMM3B42RT_forcingMod, only : TRMM3B42RT_struc
use LIS_timeMgrMod, only : LIS_date2time
implicit none
```

ARGUMENTS:

```
character(len=*)  :: name
character(len=*)  :: TRMM3B42RTdir
integer           :: yr, mo, da, hr
```

DESCRIPTION:

This subroutine puts together TRMM 3B42RT file name for 3 hour file intervals
The arguments are:

TRMM3B42RTdir Name of the TRMM 3B42RT data directory

yr year

mo month

da day of month

hr hour of day

name name of the timestamped TRMM 3B42RT file

19.1.5 interp_TRMM3B42RT (Source File: interp_TRMM3B42RT.F90)

INTERFACE:

```
subroutine interp_TRMM3B42RT(n, nx, ny, finput, lis_gds, nc, nr, varfield, findex)
```

USES:

```
use TRMM3B42RT_forcingMod, only : TRMM3B42RT_struc
use LIS_coreMod, only : LIS_rc ! SY
use LIS_logMod,    only : LIS_logunit, LIS_endrun ! SY

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer             :: nx
integer             :: ny
integer             :: nc
integer             :: nr
real                :: lis_gds(50)
real, dimension(nx,ny) :: finput
integer, intent(in) :: findex ! SY
```

DESCRIPTION:

This subroutine interpolates a given TRMM 3B42RT field to the LIS grid. The arguments are:

n index of the nest

nx number of columns (in the east-west dimension) in the TRMM 3B42RT grid

ny number of rows (in the north-south dimension) in the TRMM 3B42RT grid

finput input data array to be interpolated

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

---

19.1.6 read_TRMM3B42RT (Source File: read_TRMM3B42RT.F90)

REVISION HISTORY:

```
17 Jul 2001: Jon Gottschalck; Initial code
04 Feb 2002: Jon Gottschalck; Added necessary code to use global precip
             observations with LIS_domain 3 (2x2.5)
```

INTERFACE:

```
subroutine read_TRMM3B42RT (n, name_TRMM3B42RT, findex, order, ferror_TRMM3B42RT)
```

USES:

```
use LIS_coreMod, only           : LIS_rc, LIS_domain
use LIS_logMod, only            : LIS_logunit, LIS_getNextUnitNumber, &
                                  LIS_releaseUnitNumber
use LIS_metforcingMod, only     : LIS_forc
use TRMM3B42RT_forcingMod, only : TRMM3B42RT_struc

 implicit none
```

ARGUMENTS:

```
 integer, intent(in) :: n
 character(len=120)  :: name_TRMM3B42RT
 integer, intent(in) :: findex
 integer, intent(in) :: order
 integer             :: ferror_TRMM3B42RT
```

DESCRIPTION:

For the given time, reads parameters from TRMM 3B42RT data and interpolates to the LIS domain. The arguments are:

n index of the nest

name_TRMM3B42RT name of the 3 hour TRMM 3B42RT forecast file

ferror_TRMM3B42RT flag to indicate success of the call (=0 indicates success)

The routines invoked are:

interp_TRMM3B42RT (19.1.5)
spatially interpolates the TRMM 3B42RT data

19.1.7 readcrd_TRMM3B42RT (Source File: readcrd_TRMM3B42RT.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcrd_TRMM3B42RT()
```

USES:

```
use ESMF
use TRMM3B42RT_forcingMod, only : TRMM3B42RT_struc
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod, only : LIS_logunit
```

DESCRIPTION:

This routine reads the options specific to TRMM 3B42RT forcing from the LIS configuration file.

19.1.8 timeinterp_TRMM3B42RT (Source File: timeinterp_TRMM3B42RT.F90)

REVISION HISTORY:

```
20 Jan 2006:   Yudong Tian: Initial Implementation
10 Oct 2006:   Sujay Kumar: Switched to using ESMF_State for storing
                            forcing data.
```

INTERFACE:

```
subroutine timeinterp_TRMM3B42RT(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod, only: LIS_rc, LIS_domain
use LIS_FORC_AttributesMod
use LIS_metforcingMod, only : LIS_FORC_Base_State, LIS_forc
use LIS_logMod,        only : LIS_verify, LIS_logunit ! SY
use TRMM3B42RT_forcingMod, only : TRMM3B42RT_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep.
The arguments are:

n index of the nest

---

20 TRMM 3B42V6

This section describes the implementation of the precipitation product derived from the TRMM 3B42V6 satellite precipitation analysis generated by George Huffman at NASA GSFC.

20.1 Fortran: Module Interface TRMM3B42V6_forcingMod (Source File: TRMM3B42V6_f

REVISION HISTORY:

```
21 Jun 2013: Soni Yatheendradas; changes from earlier code to avoid
             alternate file skip, jump to previous day TRMM and
             absence of rain rate weighting
```

DESCRIPTION:

This module contains variables and data structures that are used for the implementation of the precipitation data from the NASA TRMM 3B42 (Version 6) merged analysis of precipitation (TRMM 3B42V6). TRMM 3B42 merges satellite passive microwave estimates including TMI, SSM/I, AMSR and AMSU to produce a quasi-global 0.25 degree 3-hourly precipitation analysis.
The implementation in LIS has the derived data type `TRMM3B42V6_struc` that includes the variables to specify the runtime options, and the weights and neighbor information for spatial interpolation
They are desribed below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

TRMM3B42V6dir Directory containing the input data

TRMM3B42V6time The nearest instance of the incoming data (as a real time).

griduptime1 The time to switch the input resolution

mi Number of points in the input grid

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n112,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

USES:

```
    implicit none
    PRIVATE
 ------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
 ------------------------------------------------------------------------------
  public :: init_TRMM3B42V6     !defines the native resolution of
                                !the input data
 ------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
 ------------------------------------------------------------------------------
  public :: TRMM3B42V6_struc
```

20.1.1 init_TRMM3B42V6 (Source File: TRMM3B42V6_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
25Aug2006: Yudong Tian; Implementation for TRMM 3B42 V6
```

INTERFACE:

```
subroutine init_TRMM3B42V6(findex)
```

USES:

```
use LIS_coreMod, only: LIS_rc, LIS_config ! SY
!use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep ! SY
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep, &
                           LIS_tick, LIS_parseTimeString, &
                           LIS_registerAlarm ! SY
use LIS_logMod,    only : LIS_logunit, LIS_endrun, &
                          LIS_getNextUnitNumber, LIS_releaseUnitNumber, LIS_verify ! SY
use ESMF

implicit none

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

DESCRIPTION:

Defines the native resolution of the input forcing for TRMM 3B42V6 data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). The routines invoked are:

readcrd_TRMM3B42V6 (20.1.7)
reads the runtime options specified for TRMM 3B42V6 data

LIS_date2time (3.36.24)
converts date to the real time format

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

20.1.2 finalize_TRMM3B42V6 (Source File: finalize_TRMM3B42V6.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine finalize_TRMM3B42V6(findex)
```

USES:

```
use TRMM3B42V6_forcingMod, only : TRMM3B42V6_struc
```

DESCRIPTION:

Routine to cleanup TRMM3B42V6 forcing related memory allocations.

20.1.3 get_TRMM3B42V6 (Source File: get_TRMM3B42V6.F90)

REVISION HISTORY:

```
25Aug2006: Yudong Tian; Modified for LIS 4.2 release
21 Jun 2013: Soni Yatheendradas; changes from earlier code to avoid
             (a) alternate file skip,
             (b) jump to previous day TRMM, and
             (c) absence of rain rate weighting
```

INTERFACE:

```
subroutine get_TRMM3B42V6(n, findex)
```

USES:

```
use LIS_coreMod, only           : LIS_rc, LIS_masterproc
use LIS_timeMgrMod, only        : LIS_time2date, LIS_tick, LIS_get_nstep, &
                                  LIS_isAlarmRinging ! SY
use LIS_logMod, only            : LIS_logunit, LIS_endrun
use TRMM3B42V6_forcingMod, only : TRMM3B42V6_struc
use LIS_metforcingMod, only     : LIS_forc ! SY

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 3-hrly TRMM 3B42V6 forcing. At the beginning of a simulation, the code reads the most recent past data (nearest 6 hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the TRMM 3B42V6 data times

TRMM3B42V6file (20.1.4)
Puts together appropriate file name for 6 hour intervals

read_TRMM3B42V6 (20.1.6)
Interpolates TRMM 3B42V6 data to LIS grid

20.1.4 TRMM3B42V6file (Source File: get_TRMM3B42V6.F90)

This subroutine puts together 3B42V6 file name for 3 hour file intervals There are two filename formats:
1. original: 3B42.980131.12.6.precipitation, 3B42.980130.3.6.precipitation which was produced after data are converted from hdf to bin format
2. renamed: TRMM3B42V6.2005110809, same data, just different file name
The raw hdf data files are named like this: 3B42.060105.15.6.HDF It is possible that we may implement HDF file reading in LIS code to handle raw input. For now, we do the binary format. – Yudong 8/25/06

INTERFACE:

```
subroutine TRMM3B42V6file( name, n, yr, mo, da, hr)

  use TRMM3B42V6_forcingMod, only : TRMM3B42V6_struc
```

20.1.5 interp_TRMM3B42V6.F90 (Source File: interp_TRMM3B42V6.F90)

REVISION HISTORY:

```
3 Jul 2013: Soni Yatheendradas: Modified to add findex argument towards
            spatial interpolation choice
```

INTERFACE:

```
subroutine interp_TRMM3B42V6(n, nTRMM3B42V6, f, lb, lis_gds, nc, nr, varfield, &
                             findex) ! SY
```

USES:

```
use TRMM3B42V6_forcingMod, only : TRMM3B42V6_struc
use LIS_coreMod, only : LIS_rc ! SY
use LIS_logMod,    only : LIS_logunit, LIS_endrun ! SY

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
integer                :: nc
integer                :: nr
integer                :: nTRMM3B42V6
real                   :: lis_gds(50)
real                   :: f(nTRMM3B42V6)
logical*1              :: lb(nTRMM3B42V6)
real, dimension(nc,nr) :: varfield
integer, intent(in) :: findex ! SY
```

DESCRIPTION:

This subroutine interpolates a given 3B42 V6 field to the LIS grid.
The arguments are:

n index of the nest

nTRMM3B42V6 number of elements in the input grid

f input data array to be interpolated

lb input bitmap

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

20.1.6 read_TRMM3B42V6.F90 (Source File: read_TRMM3B42V6.F90)

REVISION HISTORY:

```
17 Jul 2001: Jon Gottschalck; Initial code
04 Feb 2002: Jon Gottschalck; Added necessary code to use global precip
             observations with LIS_domain 3 (2x2.5)
30 Jul 2002: Jon Gottschalck; Added code to use 3B42V6 and Persiann precip
             data
25 Aug 2006: Yudong Tian; Modified for LIS 4.2 release
19 Apr 2013: Jonathan Case; Corrected some initializations and comparison
             against undefined value
21 Jun 2013: Soni Yatheendradas; changes from earlier code to avoid
             (a) alternate file skip,
             (b) jump to previous day TRMM, and
             (c) absence of rain rate weighting
```

INTERFACE:

```
subroutine read_TRMM3B42V6 (n, fname, findex, order, ferror_TRMM3B42V6)
```

USES:

```
use LIS_coreMod, only           : LIS_rc, LIS_domain
use LIS_logMod, only            : LIS_logunit, LIS_getNextUnitNumber, &
                                  LIS_releaseUnitNumber
use LIS_metforcingMod, only     : LIS_forc
use TRMM3B42V6_forcingMod, only : TRMM3B42V6_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
character(len=80)   :: fname
integer, intent(in) :: findex
integer, intent(in) :: order
integer             :: ferror_TRMM3B42V6
!integer             :: filehr ! SY
```

DESCRIPTION:

For the given time, reads parameters from TRMM 3B42V6 data and interpolates to the LIS domain. The arguments are:

n index of the nest

fname name of the 3-hourly TRMM 3B42V6 file

ferror_TRMM3B42V6 flag to indicate success of the call (=0 indicates success)

The routines invoked are:

interp_TRMM3B42V6 (20.1.5)
spatially interpolates the TRMM 3B42V6 data

20.1.7 readcrd_TRMM3B42V6 (Source File: readcrd_TRMM3B42V6.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
25 Aug 2006; Yudong Tian, Modification for 3B42, LIS 4.2 release
21 Jun 2013: Soni Yatheendradas; change from earlier code
```

INTERFACE:

```
subroutine readcrd_TRMM3B42V6()
```

USES:

```
use ESMF
use TRMM3B42V6_forcingMod, only : TRMM3B42V6_struc
use LIS_coreMod, only : LIS_config,LIS_rc
use LIS_logMod, only : LIS_logunit
```

DESCRIPTION:

This routine reads the options specific to TRMM 3B42V6 forcing from the LIS configuration file.

20.1.8 timeinterp_TRMM3B42V6 (Source File: timeinterp_TRMM3B42V6.F90)

REVISION HISTORY:

```
20 Jan 2006:   Yudong Tian: Initial Implementation
10 Oct 2006:   Sujay Kumar: Switched to using ESMF_State for storing
                            forcing data.
19 Apr 2013: Jonathan Case; Included comparison against undefined value,
             set small -ve values to 0
21 Jun 2013: Soni Yatheendradas; changes from earlier code to avoid
             (a) alternate file skip,
             (b) jump to previous day TRMM, and
             (c) absence of rain rate weighting
```

INTERFACE:

```
subroutine timeinterp_TRMM3B42V6(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod, only           : LIS_rc, LIS_domain
use LIS_FORC_AttributesMod
use LIS_metforcingMod,  only    : LIS_FORC_Base_State, LIS_forc
use LIS_logMod,         only    : LIS_verify, LIS_logunit ! SY
use TRMM3B42V6_forcingMod, only : TRMM3B42V6_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep.
The arguments are:

n index of the nest

---

21 TRMM 3B42V7

This section describes the implementation of the precipitation product derived from the TRMM 3B42V7 satellite precipitation analysis generated by George Huffman and David Bolvin at NASA GSFC.

21.1 Fortran: Module Interface TRMM3B42V7_forcingMod (Source File: TRMM3B42V7_f

REVISION HISTORY:

```
21 Jun 2013: Soni Yatheendradas; changes from earlier code to avoid
             alternate file skip, jump to previous day TRMM and
             absence of rain rate weighting
```

DESCRIPTION:

This module contains variables and data structures that are used for the implementation of the precipitation data from the NASA TRMM 3B42 (Version 7) merged analysis of precipitation (TRMM 3B42V7). TRMM 3B42V7 algorithm consists of two separate steps to produce a quasi-global 0.25 degree 3-hourly precipitation analysis. The first step uses the TRMM VIRS and TMI orbit data (TRMM products 1B01 and 2A12) and the monthly TMI/TRMM Combined Instrument (TCI) calibration parameters (from TRMM product 3B31) to produce monthly IR calibration parameters. The second step uses these derived monthly IR calibration parameters to adjust the merged-IR precipitation data, which consists of GMS, GOES-E, GOES-W, Meteosat-7, Meteosat-5, and NOAA-12 data.
The implementation in LIS has the derived data type `TRMM3B42V7_struc` that includes the variables to specify the runtime options, and the weights and neighbor information for spatial interpolation
They are desribed below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

TRMM3B42V7dir Directory containing the input data

TRMM3B42V7time The nearest instance of the incoming data (as a real time).

griduptime1 The time to switch the input resolution

mi Number of points in the input grid

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n112,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

USES:

```
    implicit none
    PRIVATE
 ---------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
 ---------------------------------------------------------------------------
  public :: init_TRMM3B42V7     !defines the native resolution of
                                !the input data
 ---------------------------------------------------------------------------
```

PUBLIC TYPES:

```
 ---------------------------------------------------------------------------
  public :: TRMM3B42V7_struc
```

21.1.1 init_TRMM3B42V7 (Source File: TRMM3B42V7_forcingMod.F90)

REVISION HISTORY:

```
  11Dec2003: Sujay Kumar; Initial Specification
  25Aug2006: Yudong Tian; Implementation for TRMM 3B42 V7
```

INTERFACE:

```
  subroutine init_TRMM3B42V7(findex)
```

USES:

```
    use LIS_coreMod, only: LIS_rc, LIS_config ! SY
    !use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep ! SY
    use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep, &
                               LIS_tick, LIS_parseTimeString, &
                               LIS_registerAlarm ! SY
    use LIS_logMod,    only : LIS_logunit, LIS_endrun, &
                              LIS_getNextUnitNumber, LIS_releaseUnitNumber, LIS_verify ! SY
    use ESMF

    implicit none

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

DESCRIPTION:

Defines the native resolution of the input forcing for TRMM 3B42V7 data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). The routines invoked are:

readcrd_TRMM3B42V7 (21.1.7)
reads the runtime options specified for TRMM 3B42V7 data

LIS_date2time (3.36.24)
converts date to the real time format

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

21.1.2 finalize_TRMM3B42V7 (Source File: finalize_TRMM3B42V7.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine finalize_TRMM3B42V7(findex)
```

USES:

```
use TRMM3B42V7_forcingMod, only : TRMM3B42V7_struc
```

DESCRIPTION:

Routine to cleanup TRMM3B42V7 forcing related memory allocations.

21.1.3 get_TRMM3B42V7 (Source File: get_TRMM3B42V7.F90)

REVISION HISTORY:

```
21 Jun 2013: Soni Yatheendradas; based on new TRMM 3B42V6 code that has
             changes from earlier code to avoid (a) alternate file skip,
             (b) jump to previous day TRMM, and
             (c) absence of rain rate weighting
```

INTERFACE:

```
subroutine get_TRMM3B42V7(n, findex)
```

USES:

```
use LIS_coreMod, only           : LIS_rc, LIS_masterproc
use LIS_timeMgrMod, only        : LIS_time2date, LIS_tick, LIS_get_nstep, &
                                  LIS_isAlarmRinging ! SY
use LIS_logMod, only            : LIS_logunit, LIS_endrun
use TRMM3B42V7_forcingMod, only : TRMM3B42V7_struc
use LIS_metforcingMod, only     : LIS_forc ! SY

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 3-hrly TRMM 3B42V7 forcing. At the beginning of a simulation, the code reads the most recent past data (nearest 6 hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the TRMM 3B42V7 data times

TRMM3B42V7file (21.1.4)
Puts together appropriate file name for 6 hour intervals

read_TRMM3B42V7 (21.1.6)
Interpolates TRMM 3B42V7 data to LIS grid

21.1.4 TRMM3B42V7file (Source File: get_TRMM3B42V7.F90)

INTERFACE:

```
subroutine TRMM3B42V7file( name, n, yr, mo, da, hr)
```

DESCRIPTION:

This subroutine puts together 3B42V7 file name for 3 hour file intervals There are two filename formats for the TRMM 3B42V7 data:
1. original: 3B42.980131.12.6.precipitation, 3B42.980130.3.6.precipitation which was produced after data are converted from hdf to bin format
2. renamed: TRMM3B42V6.2005110809, same data, just different file name
The raw hdf data files are named like this: 3B42.060105.15.6.HDF It is possible that we may implement HDF file reading in LIS code to handle raw input. For now, we do the binary format. – Yudong 8/25/06

USES:

```
use TRMM3B42V7_forcingMod, only : TRMM3B42V7_struc
```

21.1.5 interp_TRMM3B42V7.F90 (Source File: interp_TRMM3B42V7.F90)

REVISION HISTORY:

```
3 Jul 2013: Soni Yatheendradas: Modified to add findex argument towards
            spatial interpolation choice
```

INTERFACE:

```
subroutine interp_TRMM3B42V7(n, nTRMM3B42V7, f, lb, lis_gds, nc, nr, varfield, &
                             findex) ! SY
```

USES:

```
use TRMM3B42V7_forcingMod, only : TRMM3B42V7_struc
use LIS_coreMod,           only : LIS_rc ! SY
use LIS_logMod,            only : LIS_logunit, LIS_endrun ! SY

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
integer                :: nc
integer                :: nr
integer                :: nTRMM3B42V7
real                   :: lis_gds(50)
real                   :: f(nTRMM3B42V7)
logical*1              :: lb(nTRMM3B42V7)
real, dimension(nc,nr) :: varfield
integer, intent(in) :: findex ! SY
```

DESCRIPTION:

This subroutine interpolates a given 3B42 V7 field to the LIS grid. The arguments are:

n index of the nest

nTRMM3B42V7 number of elements in the input grid

f input data array to be interpolated

lb input bitmap

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

---

21.1.6 read_TRMM3B42V7.F90 (Source File: read_TRMM3B42V7.F90)

REVISION HISTORY:

```
19 Apr 2013: Jonathan Case; Corrected some initializations and comparison
             against undefined value, incorporated here by Soni Yatheendradas
21 Jun 2013: Soni Yatheendradas; Based on new TRMM3B42V6 code, changes from
             earlier code to avoid (a) alternate file skip, (b) jump to
             previous day TRMM, and (c) absence of rain rate weighting
```

INTERFACE:

```
subroutine read_TRMM3B42V7 (n, fname, findex, order, ferror_TRMM3B42V7)
```

USES:

```
use LIS_coreMod,           only : LIS_rc, LIS_domain
use LIS_logMod,            only : LIS_logunit, LIS_getNextUnitNumber, &
                                  LIS_releaseUnitNumber
use LIS_metforcingMod,     only : LIS_forc
use TRMM3B42V7_forcingMod, only : TRMM3B42V7_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
character(len=80)   :: fname
integer, intent(in) :: findex
integer, intent(in) :: order
integer             :: ferror_TRMM3B42V7
!integer             :: filehr ! SY
```

DESCRIPTION:

For the given time, reads parameters from TRMM 3B42V7 data and interpolates to the LIS domain. The arguments are:

n index of the nest

fname name of the 6 hour TRMM 3B42V7 file

ferror_TRMM3B42V7 flag to indicate success of the call (=0 indicates success)

The routines invoked are:

interp_TRMM3B42V7 (21.1.5)
spatially interpolates the TRMM 3B42V7 data

21.1.7 readcrd_TRMM3B42V7 (Source File: readcrd_TRMM3B42V7.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
25 Aug 2006; Yudong Tian, Modification for 3B42, LIS 4.2 release
21 Jun 2013: Soni Yatheendradas; change from earlier 3B42V6 code,
             ported to 3B42V7
```

INTERFACE:

```
subroutine readcrd_TRMM3B42V7()
```

USES:

```
use ESMF
use TRMM3B42V7_forcingMod, only : TRMM3B42V7_struc
use LIS_coreMod, only : LIS_config,LIS_rc
use LIS_logMod, only : LIS_logunit
```

DESCRIPTION:

This routine reads the options specific to TRMM 3B42V7 forcing from the LIS configuration file.

21.1.8 timeinterp_TRMM3B42V7 (Source File: timeinterp_TRMM3B42V7.F90)

REVISION HISTORY:

```
20 Jan 2006:   Yudong Tian: Initial Implementation
10 Oct 2006:   Sujay Kumar: Switched to using ESMF_State for storing
                            forcing data.
19 Apr 2013: Jonathan Case; Included comparison against undefined value,
                            set small -ve values to 0
21 Jun 2013: Soni Yatheendradas; Based on new TRMM3B42V6 code, which is
             based on changes from earlier code to avoid (a) alternate file skip,
             (b) jump to previous day TRMM, and (c) absence of rain rate weighting
```

INTERFACE:

```
subroutine timeinterp_TRMM3B42V7(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod, only           : LIS_rc, LIS_domain
use LIS_FORC_AttributesMod
use LIS_metforcingMod,  only    : LIS_FORC_Base_State, LIS_forc
use LIS_logMod,         only    : LIS_verify, LIS_logunit ! SY
use TRMM3B42V7_forcingMod, only : TRMM3B42V7_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep.
The arguments are:

n index of the nest

---

22 Noah3.1 Bondville forcing

This section describes the implementation of the Bondville test case data.

22.1 Fortran: Module Interface Bondville_forcingMod (Source File: Bondville_forcingMod.F

Contains routines and data structures that are used for the implementation of the station data from various Bondville stations. The stations report estimates of meteorological forcing terms, which is spatially interpolated using the inverse distance weighting scheme (IDW).
The implementation in LIS has the derived data type `Bondville_struc` that includes the variables to specify the runtime options, and the calculation of weights for spatial interpolation.

REVISION HISTORY:

```
05 Oct 2010: David Mocko, Updated for Bondville test case

implicit none
PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: init_Bondville !defines the native resolution of
                                     !the input data
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: Bondville_struc
```

22.1.1 init_Bondville (Source File: Bondville_forcingMod.F90)

INTERFACE:

```
subroutine init_Bondville(findex)
```

USES:

```
use LIS_coreMod,only     : LIS_rc
use LIS_logMod, only     : LIS_logunit,LIS_endrun, &
     LIS_getNextUnitNumber,  &
     LIS_releaseUnitNumber
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep

implicit none
integer, intent(in) :: findex
```

DESCRIPTION:

This routines reads the runtime configurations for using the Bondville station data. Using the metadata provided for the stations, this routine invokes the call to compute the interpolation weights to be later used. The routines invoked are:

readcrd_Bondville (22.1.5)
reads the runtime options specified for Bondville station data

LIS_date2time (3.36.24)
converts date to the real time format

compute_stnwts (5.1.22)
computes the weights for spatial interpolation

22.1.2 finalize_Bondville (Source File: finalize_Bondville.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine finalize_Bondville(findex)
```

USES:

```
use Bondville_forcingMod, only : Bondville_struc
use LIS_logMod, only : LIS_logunit, LIS_endrun
```

DESCRIPTION:

! Routine to cleanup Bondville forcing related memory allocations. !

22.1.3 get_Bondville (Source File: get_Bondville.F90)

REVISION HISTORY:

```
07 Oct 2010: David Mocko, Updated for Bondville test case
```

INTERFACE:

```
subroutine get_Bondville(n,findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod, only : LIS_tick
use LIS_logMod, only : LIS_logunit
use LIS_metforcingMod,  only : LIS_forc
use Bondville_forcingMod, only : Bondville_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates the Bondville station data. At the beginning of a simulation, the code reads the most recent past data, and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the Bondville data times

read_Bondville (22.1.4)
Interpolates the appropriate Bondville station data to LIS grid

---

22.1.4 read_Bondville (Source File: read_Bondville.F90)

REVISION HISTORY:

```
06 Oct 2010: David Mocko, Updated for Bondville test case
```

INTERFACE:

```
subroutine read_Bondville(n,ftn,findex,order,itime)
```

USES:

```
use LIS_logMod, only            : LIS_logunit,LIS_endrun
use LIS_coreMod, only           : LIS_rc,LIS_domain
use LIS_metforcingMod,     only : LIS_forc
use LIS_timeMgrMod, only        : LIS_date2time,LIS_tick
use Bondville_forcingMod, only : Bondville_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: ftn
integer, intent(in) :: itime
integer, intent(in) :: findex
integer, intent(in) :: order
```

DESCRIPTION:

For the given time, reads parameters from the correct Bondville station data (ASCII), transforms into LIS forcing parameters, and interpolates to the LIS domain.
The arguments are:

n index of the nest

ftn unit number for the Bondville station data

itime index for reading current time or reading a future time

The routines invoked are:

normalize_stnwts (5.3.12)
renormalizes the station weights accounting for missing data

interp_stndata (5.2.8)
spatially interpolates the station data onto the LIS grid.

22.1.5 readcrd_Bondville (Source File: readcrd_Bondville.F90)

REVISION HISTORY:

```
13 Apr 2007: Bailing Li, Initial Code
05 Oct 2010: David Mocko, Updated for Bondville test case
```

INTERFACE:

```
subroutine readcrd_Bondville()
```

USES:

```
use ESMF
use LIS_logMod, only : LIS_logunit
use LIS_coreMod, only : LIS_rc, LIS_config
use Bondville_forcingMod, only : Bondville_struc
```

DESCRIPTION:

This routine reads the options specific to the Bondville test case station data forcing from the LIS configuration file.

22.1.6 timeinterp_Bondville (Source File: timeinterp_Bondville.F90)

REVISION HISTORY:

```
05 Oct 2010: David Mocko, Updated for Bondville test case
```

INTERFACE:

```
subroutine timeinterp_Bondville(n,findex)
```

USES:

```
use ESMF
use LIS_logMod, only           : LIS_logunit,LIS_verify,LIS_endrun
use LIS_coreMod, only          : LIS_rc,LIS_domain
use LIS_metforcingMod, only   : LIS_FORC_Base_State, LIS_forc
use Bondville_forcingMod, only : Bondville_struc
use LIS_FORC_AttributesMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

! Temporally interpolates the Bondville forcing data to ! the model timestep. All variables except precipitation ! is linearly interpolated. !

22.2 Fortran: Module Interface PALSmetdata_forcingMod (Source File: PALSmetdata_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of station data from PALS (Protocol for the Analysis of Land Surface Models; http://www.pals.unsw.edu.au) used as forcing within LIS. No spatial interpolation of the data is performed as it is expected to be point data. The LIS domain must be configured to exactly match the station location.

USES:

```
use ESMF

implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init_PALSmetdata      !defines the native resolution of
                                !the input data
```

PUBLIC TYPES:

```
public :: PALSmetdata_struc
```

22.2.1 init_PALSmetdata (Source File: PALSmetdata_forcingMod.F90)

REVISION HISTORY:

```
7 Mar 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine init_PALSmetdata(findex)
```

USES:

```
use LIS_coreMod
use LIS_timeMgrMod
use LIS_logMod

implicit none

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

DESCRIPTION:

Defines the native domain of the input forcing for PALS data (which is assumed to be at the point scale) The routines invoked are:

readcrd_PALSmetdata (22.3.4)
reads the runtime options specified for PALS station data

22.3 Fortran: Module Interface finalize_PALSmetdata (Source File: finalize_PALSmetdata.F

REVISION HISTORY:

```
7 Mar 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine finalize_PALSmetdata(findex)
```

USES:

```
  use LIS_coreMod,       only : LIS_rc
  use PALSmetdata_forcingMod, only : PALSmetdata_struc
```

DESCRIPTION:

Routine to finalize PALS forcing related memory allocations.

22.3.1 get_PALSmetdata (Source File: get_PALSmetdata.F90)

REVISION HISTORY:

```
  7 Mar 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine get_PALSmetdata(n,findex)
```

USES:

```
  use ESMF
  use LIS_coreMod
  use LIS_timeMgrMod
  use LIS_metforcingMod,  only : LIS_forc
  use LIS_logMod
  use PALSmetdata_forcingMod,  only : PALSmetdata_struc
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and assigns 1/2 hourly, PALS met data as forcing. At the beginning of a simulation or at the change of years, the code reads the data (for that year and keeps it in memory). past data (nearest 1./2 hourly interval), and the nearest future data.
NOTES: No spatial interpolation is applied. The LIS domain is expected to be setup over the station of interest.
The arguments are:

n index of the nest

findex index of the forcing

The routines invoked are:

PALSmetdatafile (22.3.2)
Generates the PALS forcing file name

read_PALSvar (22.3.3)
Reads a variable from PALS file, applies the QC and assigns it to the LIS data structures.

22.3.2 PALSmetdatafile (Source File: get_PALSmetdata.F90)

INTERFACE:

```
subroutine PALSmetdatafile(name,PALSmetdatadir,stn_name)

  implicit none
```

ARGUMENTS:

```
  character(len=*), intent(out) :: name
  character(len=*), intent(in)  :: PALSmetdatadir
  character(len=*), intent(in)  :: stn_name
```

DESCRIPTION:

This subroutine puts together PALS met forcing file name
The arguments are:

name name of the PALS met forcing file

PALSmetdatadir Name of the PALS data directory

stn_name name of the station

22.3.3 read_PALSvar (Source File: get_PALSmetdata.F90)

INTERFACE:

```
subroutine read_PALSvar(ftn, varname, tindex1, tindex2, varfield, vardim)
```

USES:

```
  use LIS_coreMod
  use LIS_logMod
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif

  implicit none
```

ARGUMENTS:

```
  integer              :: ftn
  character(len=*)     :: varname
  integer              :: tindex1
  integer              :: tindex2
  real                 :: varfield(tindex2-tindex1)
  logical              :: vardim ! true means 3d else 2d
```

DESCRIPTION:

This routine reads a variable from the station data file and applies the quality control filters.
The arguments are:

ftn netcdf file handle

varname name of the variable to be retrieved

tindex1 starting time index of the data

tindex2 ending time index of the data

varfield retrieved variable field

vardim logical variable indicating if the variable to be read is 2d or 3d (.true. = 3d, .false. = 2d)

22.3.4 readcrd_PALSmetdata (Source File: readcrd_PALSmetdata.F90)

REVISION HISTORY:

```
7 Mar 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine readcrd_PALSmetdata()
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod,  only : LIS_logunit, LIS_verify
use PALSmetdata_forcingMod, only : PALSmetdata_struc
```

DESCRIPTION:

This routine reads the options specific to PALS met forcing from the LIS configuration file.

22.4 Fortran: Module Interface reset_PALSmetdata (Source File: reset_PALSmetdata.F90)

REVISION HISTORY:

```
7 Mar 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine reset_PALSmetdata()
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use PALSmetdata_forcingMod, only : PALSmetdata_struc
```

DESCRIPTION:

Routine to reset PALS forcing related memory allocations.

22.4.1 timeinterp_PALSmetdata (Source File: timeinterp_PALSmetdata.F90)

REVISION HISTORY:

```
7 Mar 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine timeinterp_PALSmetdata(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_constantsMod, only  : LIS_CONST_SOLAR
use LIS_metforcingMod, only : LIS_forc, LIS_FORC_Base_State
use LIS_FORC_AttributesMod
use LIS_timeMgrMod, only : LIS_tick, LIS_time2date
use LIS_logMod, only :LIS_logunit, LIS_verify, LIS_endrun
use PALSmetdata_forcingMod, only : PALSmetdata_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the PALS forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous 3 hourly value is used. All other variables are linearly interpolated between the 3 hourly blocks.
The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

LIS_tick (3.36.27)
advances or retracts time by the specified amount

zterp (5.3.19)
zenith-angle based interpolation

22.4.2 finalize_pildas (Source File: finalize_pildas.F90)

REVISION HISTORY:

```
25 Apr 2013, Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine finalize_pildas(findex)
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use pildas_forcingMod, only : pildas_struc
```

DESCRIPTION:

Routine to cleanup PILDAS forcing related memory allocations.

22.4.3 get_pildas (Source File: get_pildas.F90)

REVISION HISTORY:

```
25 Apr 2013: Sujay Kumar, initial version
```

INTERFACE:

```
subroutine get_pildas(n,findex)
```

USES:

```
use LIS_coreMod
use LIS_timeMgrMod
use LIS_logMod
use LIS_metforcingMod
use pildas_forcingMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 1-hourly PILDAS forcing. At the beginning of a simulation, the code reads the most recent past data (nearest 1 hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
call to advance or retract time

pildasfile (22.4.4)
Puts together appropriate file name for 1 hour intervals

read_pildas (22.5.2)
call to read the PILDAS data and perform spatial interpolation

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

22.4.4 pildasfile (Source File: get_pildas.F90)

INTERFACE:

```
subroutine pildasfile(name,pildasdir,version,yr,mo,da,hr,mn)

  implicit none
```

ARGUMENTS:

```
  character(len=*), intent(out) :: name
  character(len=*), intent(in)  :: pildasdir
  integer, intent(in)           :: version
  integer, intent(in)           :: yr,mo,da,hr,mn
```

DESCRIPTION:

This subroutine puts together PILDAS file name for 1 hour file intervals
The arguments are:

name name of the timestamped GEOS5 forecast file

pildasdir Name of the GEOS5 forecast data directory

yr year

mo month

da day of month

hr hour of day

mn minute

22.4.5 interp_pildas (Source File: interp_pildas.F90)

REVISION HISTORY:

```
  7 Mar 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine interp_pildas(n,findex,input_data,output_2d)
```

USES:

```
  use LIS_coreMod, only : LIS_rc
  use LIS_spatialDownscalingMod
  use pildas_forcingMod, only :pildas_struc

  implicit none
```

ARGUMENTS:

```
  integer, intent(in)   :: n
  integer, intent(in)   :: findex
  real, intent(in)      :: input_data(pildas_struc(n)%nc,pildas_struc(n)%nr)
  real, intent(inout)   :: output_2d(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine interpolates a given GEOS5 data field to the LIS grid.

22.5 Fortran: Module Interface pildas_forcingMod (Source File: pildas_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the PILDAS forcing data. The data is global 1 degree dataset in latlon projection, and at 1 hourly intervals. The derived data type `pildas_struc` includes the variables that specify the runtime options, and the weights and neighbor information to be used for spatial interpolation. They are described below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

nmif Number of forcing variables in the ECMWF data

pildastime1 The nearest, previous 1 hour instance of the incoming data (as a real time).

pildastime2 The nearest, next 1 hour instance of the incoming data (as a real time).

pildasdir Directory containing the input data

mi Number of points in the input grid

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n111,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n122,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

findtime1, findtime2 boolean flags to indicate which time is to be read for temporal interpolation.

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init_pildas      !defines the native resolution of
          !the input data
```

PUBLIC TYPES:

```
------------------------------------------------------------------------------
 public :: pildas_struc
```

22.5.1 init_pildas (Source File: pildas_forcingMod.F90)

REVISION HISTORY:

```
25 Apr 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine init_pildas(findex)
```

USES:

```
   use LIS_coreMod
   use LIS_timeMgrMod
   use LIS_logMod

   implicit none
!AGRUMENTS:
   integer, intent(in) :: findex
```

DESCRIPTION:

Defines the native resolution of the input forcing for PILDAS data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). The routines invoked are:

readcrd_pildas (22.5.3)
reads the runtime options specified for PILDAS data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

22.5.2 read_pildas (Source File: read_pildas.F90)

REVISION HISTORY:

```
25 Apr 2013: Sujay Kumar, Initial code
```

INTERFACE:

```
subroutine read_pildas(n, findex, order, name,ferror)
```

USES:

```
  use LIS_coreMod, only        : LIS_rc, LIS_domain
  use LIS_logMod, only         : LIS_logunit, LIS_verify, LIS_warning
  use LIS_metforcingMod, only  : LIS_forc
  use pildas_forcingMod,only : pildas_struc
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
  implicit none
```

ARGUMENTS:

```
  integer, intent(in)      :: findex
  integer, intent(in)      :: n
  integer, intent(in)      :: order
  character*80, intent(in) :: name
  integer, intent(out)     :: ferror
```

DESCRIPTION:

For the given time, reads the forcing data from the GEOS5 file, transforms into LIS forcing parameters and interpolates to the LIS domain.
The arguments are:

n index of the nest

findex index of the forcing

order flag indicating which data to be read (order=1, read the previous hourly instance, order=2, read the next hourly instance)

name name of the file to be read

ferror return error flag (0-fail, 1-success)

The routines invoked are:

interp_pildas (22.4.5)
Performs spatial interpolation of GEOS5 forecast data to the LIS grid

---

22.5.3 readcrd_pildas (Source File: readcrd_pildas.F90)

REVISION HISTORY:

```
  25 Apr 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine readcrd_pildas()
```

USES:

```
  use ESMF
  use LIS_coreMod, only : LIS_rc, LIS_config
  use LIS_logMod
  use pildas_forcingMod, only : pildas_struc
```

DESCRIPTION:

This routine reads the options specific to Merra-Land forcing from the LIS configuration file.

22.5.4 timeinterp_pildas (Source File: timeinterp_pildas.F90)

REVISION HISTORY:

```
25 Apr 2013: Eric Kemp, initial code
```

INTERFACE:

```
subroutine timeinterp_pildas(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod,          only : LIS_rc, LIS_domain, LIS_localPet
use LIS_metforcingMod,   only : LIS_forc, LIS_FORC_Base_State
use LIS_FORC_AttributesMod
use LIS_constantsMod,     only : LIS_CONST_SOLAR
use LIS_timeMgrMod,       only : LIS_time2date
use LIS_logMod,           only : LIS_logunit, LIS_verify, LIS_endrun
use pildas_forcingMod,      only : pildas_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous 1 hourly value is used. All other variables are linearly interpolated between the 1 hourly blocks.
The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

zterp (5.3.19)
zenith-angle based interpolation

---

23 RFE2Daily

This section describes the implementation of the Climate Prediction Center's (CPC) Rainfall Estimates version 2 (RFE2) daily precipition data, available from ftp://ftp.cpc.ncep.noaa.gov/fews/newalgo_est/ and ftp://ftp.cpc.ncep.noaa.gov/fews/fewsdata/africa/rfe2/.
This precip is a daily precipitation analysis at 0.1 deg lat x 0.1 deg lon produced by merging GTS gauge observations and 3 kinds of satellite estimates (GPI,SSM/I and AMSU). Units are in millimeters (mm). Data is in big endian binary with coverage -40.00S to 40.00N Northward (801 grid points in south - north direction) and 20.00W to 55.00E Eastward (751 grid points in east - west direction)

23.1 Fortran: Module Interface RFE2Daily_forcingMod (Source File: RFE2Daily_forcingMo

This module contains variables, data structures and subroutines used for the implementation of the RFE2.0 rainfall forcing data from the CPC site ftp://ftp.cpc.ncep.noaa.gov/fews/newalgo_est/ and used in USAID/FEWS-NET. This precip is a daily precipitation analysis at 0.1 deg lat x 0.1 deg lon produced by merging GTS gauge observations and 3 kinds of satellite estimates (GPI,SSM/I and AMSU). Units are in millimeters (mm). Data is in big endian binary with coverage -40.00S to 40.00N Northward (801 grid points in south - north direction) and 20.00W to 55.00E Eastward (751 grid points in east - west direction)
The implementation in LIS has the derived data type `RFE2Daily_struc` that includes the variables to specify the runtime options, and the weights and neighbor information for upscaling or spatial interpolation
They are desribed below:

RFE2DailyDir Directory containing the input data

RFE2DailyEndTime The nearest, daily instance of the incoming data (as a real time).

startTime The earliest possible daily availability time of the incoming data (as an ESMF real time).

st_real The earliest possible daily availability time of the incoming data (as a real time).

timeStep Input data time step (as an ESMF real time).

gridDesci Input grid description parameters

mi Number of points in the input grid

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n111,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n112,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

rlat3 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for n. neighbor interpolation)

rlon3 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for n. neighbor interpolation)

n113 Arrays containing the neighbor information of the input grid for each grid point in LIS, for n. neighbor interpolation.

USES:

```
use ESMF

implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init_RFE2Daily      !defines the native resolution of
                                     !the input data
```

PUBLIC TYPES:

```
public :: RFE2Daily_struc
```

23.1.1 init_RFE2Daily (Source File: RFE2Daily_forcingMod.F90)

REVISION HISTORY:

```
26 MAY 2010: Soni Yatheendradas; Initial LIS version for FEWSNET
20 Mar 2013; KR Arsenault, Cleaned up code and documentation
```

INTERFACE:

```
subroutine init_RFE2Daily(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_logMod,     only : LIS_logunit, LIS_endrun
use LIS_timeMgrMod, only : LIS_update_timestep

implicit none
integer, intent(in) :: findex
```

DESCRIPTION:

Defines the native resolution of the input forcing for RFE2 data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation and upscaling schemes V
This routine performs required initializations, including allocating memory, initializing data structures and setting up interpolation weights and upscaling indices.
The arguments are:

findex index of the supplemental forcing source

The routines invoked are:

readcrd_RFE2Daily (23.2.3)
reads the runtime options specified for RFE2Daily data

upscaleByAveraging_input (5.3.18)
computes the neighbor information for upscaling data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

neighbor_interp_input (5.3.11)
computes the neighbor for neighbor interpolation

23.2 Fortran: Module Interface finalize_RFE2Daily (Source File: finalize_RFE2Daily.F90)

REVISION HISTORY:

```
2 June 2010: Soni Yatheendradas; Initial LIS version for FEWSNET
```

INTERFACE:

```
subroutine finalize_RFE2Daily(findex)
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use RFE2Daily_forcingMod, only : RFE2Daily_struc
```

DESCRIPTION:

Routine to cleanup RFE2Daily forcing related memory allocations.

23.2.1 get_RFE2Daily (Source File: get_RFE2Daily.F90)

REVISION HISTORY:

```
30 May 2010; Soni Yatheendradas, Initial LIS version for FEWSNET
20 Mar 2013; KR Arsenault, Cleaned up code and fixed update time issue
```

INTERFACE:

```
subroutine get_RFE2Daily(n, findex)
```

USES:

```
use ESMF
use LIS_coreMod,           only : LIS_rc
use LIS_metforcingMod,     only : LIS_forc
use LIS_timeMgrMod,        only : LIS_calendar, LIS_get_nstep, &
                                  LIS_tick, LIS_date2time, LIS_time2date
use LIS_logMod,            only : LIS_logunit, LIS_endrun, LIS_verify
use RFE2Daily_forcingMod,  only : RFE2Daily_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and upscales/interpolates daily RFE2.0 forcing.
At the beginning of a simulation, the code reads the relevant daily data (nearest daily interval). Since rain value is obtained from total daily accumulation, only one relevant file needs to be read. Current first-cut temporal interpolation implemented is even distribution over sub-daily model run timesteps. The strategy for missing data in spatial upscaling is to retain the corresponding base precip value/s (diurnal precip pattern then follows base precip and is not an even distribution). Upscaled values are missing only if ALL the contained RFE2 pixels are NODATA. The strategy for missing data in spatial interpolation is to set such NODATA values in the RFE2Daily input to 0 (Ronald W Lietzow, USGS, Personal E-mail Communication 05/27/2010).
An upper daily 1000 mm cap is also implemented as per information from USGS staff before being input to the WRSI, MI, and BERM products.
The arguments are:

n index of the nest

findex index of the supplemental forcing source

The routines invoked are:

LIS_tick (3.36.27)
determines the RFE2Daily data times

RFE2Dailyfile (23.2.2)
puts together the RFE2Daily file name

readprecip_RFE2Daily (23.2.4)
reads the RFE2Daily data

23.2.2 RFE2Dailyfile (Source File: get_RFE2Daily.F90)

INTERFACE:

```
subroutine RFE2Dailyfile( filename, RFE2DailyDir, yr, mo, da )

  implicit none
```

ARGUMENTS:

```
  character(len=*)  :: filename
  character(len=*)  :: RFE2DailyDir
  integer           :: yr, mo, da
```

DESCRIPTION:

This subroutine puts together the RFE2Daily file name
The arguments are:

RFE2DailyDir Name of the RFE2Daily directory

yr year

mo month

da day of month

filename name of the timestamped RFE2Daily file

23.2.3 readcrd_RFE2Daily (Source File: readcrd_RFE2Daily.F90)

REVISION HISTORY:

```
26 MAY 2010; Soni Yatheendradas, Initial LIS version for FEWSNET
20 Mar 2013; KR Arsenault, Cleaned up code and added adjustable hour offset
```

INTERFACE:

```
subroutine readcrd_RFE2Daily()
```

USES:

```
use ESMF
use LIS_coreMod,         only : LIS_rc, LIS_config
use LIS_logMod,          only : LIS_logunit, LIS_endrun, LIS_verify
use LIS_timeMgrMod,      only : LIS_calendar, LIS_date2time
use RFE2Daily_forcingMod, only: RFE2Daily_struc
```

DESCRIPTION:

This routine reads the options specific to the CPC-hosted USAID/FEWS-NET RFE2.0 forcing, from the LIS configuration file.

23.2.4 readprecip_RFE2Daily (Source File: readprecip_RFE2Daily.F90)

REVISION HISTORY:

```
30 May 2010; Soni Yatheendradas, Initial LIS version for FEWSNET
20 Mar 2013; KR Arsenault, Cleaned up code and documentation
```

INTERFACE:

```
subroutine readprecip_RFE2Daily( n, fname, findex, order, ferror_RFE2Daily)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain, LIS_masterproc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
        LIS_releaseUnitNumber, LIS_endrun, LIS_verify
use LIS_metforcingMod,  only : LIS_forc
use RFE2Daily_forcingMod, only : RFE2Daily_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
character(len=80)   :: fname
integer, intent(in) :: findex
integer, intent(in) :: order
integer             :: ferror_RFE2Daily
```

DESCRIPTION:

For the given time, reads the RFE2Daily data and interpolates to the LIS domain.
The arguments are:

n index of the nest

fname name of the daily RFE2.0 file

findex index of the supplemental forcing source

ferror_RFE2Daily flag to indicate success of the call (=0 indicates success)

The routines invoked are:

reproject_RFE2Daily (23.2.5)
Upscales/interpolates the RFE2Daily data

23.2.5 reproject_RFE2Daily (Source File: reproject_RFE2Daily.F90)

REVISION HISTORY:

```
30 May 2010; Soni Yatheendradas, Initial LIS version for FEWSNET
20 Mar 2013; KR Arsenault, Cleaned up code and documentation
```

INTERFACE:

```
subroutine reproject_RFE2Daily(n,findex,num_pts,f1d,lb1d, &
                               lis_gds, nc,nr, varfield)
```

USES:

```
use LIS_coreMod,          only : LIS_rc
use LIS_logMod,           only : LIS_logunit
use RFE2Daily_forcingMod, only : RFE2Daily_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
integer, intent(in)    :: findex
integer, intent(in)    :: num_pts
real, intent(in)       :: f1d(num_pts)
logical*1, intent(in)  :: lb1d(num_pts)
real, intent(in)       :: lis_gds(50)
integer, intent(in)    :: nc
integer, intent(in)    :: nr
real, intent(inout)    :: varfield(nc,nr)
```

DESCRIPTION:

This subroutine interpolates a given RFE2Daily field to the LIS grid.
The arguments are:

n index of the nest

findex index of the supplemental forcing source

num_pts number of elements in the input grid

f1d 1-d input data array to be interpolated

lb1d input 1-d bitmap

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

upscaleByAveraging (5.3.17)
upscales scalar data from a finer grid to a coarser grid

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

neighbor_interp (5.3.10)
spatially interpolate the forcing data using nearest neighbor interpolation

---

23.2.6 timeinterp_RFE2Daily (Source File: timeinterp_RFE2Daily.F90)

REVISION HISTORY:

```
2 June 2010: Soni Yatheendradas; Initial LIS version for FEWSNET
```

INTERFACE:

```
subroutine timeinterp_RFE2Daily(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_metforcingMod, only :  LIS_FORC_Base_State, LIS_forc
use LIS_FORC_AttributesMod
use LIS_logMod, only : LIS_verify
use RFE2Daily_forcingMod, only : RFE2Daily_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep (evenly distributed over sub-daily timesteps for now).
The arguments are:

n index of the nest

findex index of the supplemental forcing source

metdata1 previous forcing values

metdata2 next forcing values

---

24 RFE2gdas

This section describes the implementation of the RFE2gdas.0 rainfall precipitation reader. These precipitation data are the Climate Prediction Center's (CPC) Rainfall Estimates version 2 (RFE2) daily data disaggregated to 6-hourly data using GDAS.
The original daily data are available from ftp://ftp.cpc.ncep.noaa.gov/fews/newalgo_est/ and ftp://ftp.cpc.ncep.noaa.gov/fews/fewsdata/africa/rfe2/ and are used in USAID/FEWS-NET.
The CPC precip is a daily precipitation analysis at 0.1 deg lat x 0.1 deg lon produced by merging GTS gauge observations and 3 kinds of satellite estimates (GPI,SSM/I and AMSU). Units are in millimeters (mm). Data is in big endian binary with coverage -40.00S to 40.00N Northward (801 grid points in south - north direction) and 20.00W to 55.00E Eastward (751 grid points in east - west direction)

24.1 Fortran: Module Interface RFE2gdas_forcingMod (Source File: RFE2gdas_forcingMod

This module contains variables, data structures and subroutines used for the implementation of the RFE2gdas.0 rainfall forcing data from the CPC site ftp://ftp.cpc.ncep.noaa.gov/fews/newalgo_est/ and used in USAID/FEWS-NET. This precip is a daily precipitation analysis at 0.1 deg lat x 0.1 deg lon produced by merging GTS gauge observations and 3 kinds of satellite estimates (GPI,SSM/I and AMSU). Units are in millimeters (mm). Data is in big endian binary with coverage -40.00S to 40.00N Northward (801 grid points in south - north direction) and 20.00W to 55.00E Eastward (751 grid points in east - west direction)
The implementation in LIS has the derived data type `RFE2gdas_struc` that includes the variables to specify the runtime options, and the weights and neighbor information for upscaling or spatial interpolation
They are desribed below:

RFE2gdasDir Directory containing the input data

RFE2gdasEndTime The nearest, daily instance of the incoming data (as a real time).

startTime The earliest possible daily availability time of the incoming data (as an ESMF real time).

st_real The earliest possible daily availability time of the incoming data (as a real time).

timeStep Input data time step (as an ESMF real time).

gridDesci Input grid description parameters

mi Number of points in the input grid

stc Starting index (along the east-west direction) of the input grid for each output grid point

str Starting index (along the north-south direction) of the input grid for each output grid point

enc Ending index (along the east-west direction) of the input grid for each output grid point

enr Ending index (along the north-south direction) of the input grid for each output grid point

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n111,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n112,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

rlat3 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for n. neighbor interpolation)

rlon3 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for n. neighbor interpolation)

n113 Arrays containing the neighbor information of the input grid for each grid point in LIS, for n. neighbor interpolation.

USES:

```
use ESMF

implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init_RFE2gdas      ! Defines the native resolution of
                             ! the input data
```

PUBLIC TYPES:

```
public :: RFE2gdas_struc
```

24.1.1 init_RFE2gdas (Source File: RFE2gdas_forcingMod.F90)

REVISION HISTORY:

```
26 MAY 2010: Soni Yatheendradas; Initial LIS version for FEWSNET
```

INTERFACE:

```
subroutine init_RFE2gdas(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_update_timestep
use LIS_logMod,     only : LIS_logunit, LIS_endrun

implicit none

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

DESCRIPTION:

Defines the native resolution of the input forcing for CMAP data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation and upscaling schemes (see Section V).
This routine performs required initializations, including allocating memory, initializing data structures and setting up interpolation weights and upscaling indices.
The arguments are:

findex index of the supplemental forcing source

The routines invoked are:

readRFE2gdascrd (24.2.3)
reads the runtime options specified for RFE2gdas data

upscaleByAveraging_input (5.3.18)
computes the neighbor information for upscaling data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

neighbor_interp_input (5.3.11)
computes the neighbor for neighbor interpolation

24.2 Fortran: Module Interface finalize_RFE2gdas (Source File: finalize_RFE2gdas.F90)

REVISION HISTORY:

```
2 June 2010: Soni Yatheendradas; Initial LIS version for FEWSNET
```

INTERFACE:

```
subroutine finalize_RFE2gdas(findex)
```

USES:

```
use LIS_coreMod,         only : LIS_rc
use RFE2gdas_forcingMod, only : RFE2gdas_struc
```

DESCRIPTION:

Routine to cleanup RFE2gdas forcing related memory allocations.

24.2.1 get_RFE2gdas (Source File: get_RFE2gdas.F90)

REVISION HISTORY:

```
30 May 2010; Soni Yatheendradas, Initial LIS version for FEWSNET
25 Jan 2011: Clement Alo modified for RFE2gdas 6-hourly data
8 Mar 2012: Updated for LIS7
```

INTERFACE:

```
subroutine get_RFE2gdas(n, findex)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_timeMgrMod,     only : LIS_calendar, LIS_get_nstep, LIS_tick
use LIS_logMod,         only : LIS_logunit, LIS_endrun, LIS_verify
use RFE2gdas_forcingMod,  only : RFE2gdas_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and upscales/interpolates 6-hourly RFE2gdas.0 forcing. At the beginning of a simulation, the code reads the relevant daily data (nearest daily interval). Since rain value is obtained from total daily accumulation, only one relevant file needs to be read, not two. Current first-cut temporal interpolation implemented is even distribution over sub-6hourly model run timesteps. The strategy for missing data in spatial upscaling is to retain the corresponding base precip value/s (Diurnal precip pattern then follows base precip and is not an even distribution). Upscaled values are missing only if ALL the contained RFE2gdas pixels are NODATA. The strategy for missing data in spatial interpolation is to set such NODATA values in the RFE2gdas input to 0 (Ronald W Lietzow, USGS, Personal E-mail Communication 05/27/2010). An upper daily 1000 mm cap is also implemented as per Ron's info of current practice before being input to the WRSI, MI, and BERM products
The arguments are:

n index of the nest

findex index of the supplemental forcing source

The routines invoked are:

LIS_tick (3.36.27)
determines the RFE2gdas data times

RFE2gdasfile (24.2.2)
Puts together appropriate file name for 6 hour intervals

readprecip_RFE2gdas (24.2.4)
reads the RFE2gdas data

24.2.2 RFE2gdasfile (Source File: get_RFE2gdas.F90)

INTERFACE:

```
subroutine RFE2gdasfile( name, RFE2gdasDir, yr, mo, da, hr )

  implicit none
```

ARGUMENTS:

```
  character(len=*)   :: name
  character(len=*)   :: RFE2gdasDir
  integer            :: yr, mo, da, hr
```

DESCRIPTION:

This subroutine puts together CMAP file name for 6 hour file intervals
The arguments are:

RFE2gdasDir Name of the RFE2gdas directory

yr year

mo month

da day of month

hr hour of day

name name of the timestamped RFE2gdas file

24.2.3 readRFE2gdascrd (Source File: readRFE2gdascrd.F90)

REVISION HISTORY:

```
  26 MAY 2010; Soni Yatheendradas, Initial LIS version for FEWSNET
  26 JAN 2011; Clement Alo, Modified for 6-hourly RFE2gdas data
```

INTERFACE:

```
subroutine readRFE2gdascrd()
```

USES:

```
  use ESMF
  use LIS_coreMod,   only : LIS_rc, LIS_config
  use LIS_logMod,    only : LIS_logunit, LIS_endrun, LIS_verify
  use LIS_timeMgrMod, only : LIS_calendar, LIS_date2time
  use RFE2gdas_forcingMod, only: RFE2gdas_struc
```

DESCRIPTION:

This routine reads the options specific to the CPC-hosted USAID/FEWS-NET RFE2gdas.0 forcing, from the LIS configuration file.

24.2.4 readprecip_RFE2gdas (Source File: readprecip_RFE2gdas.F90)

REVISION HISTORY:

```
30 May 2010; Soni Yatheendradas, Initial LIS version for FEWSNET
```

INTERFACE:

```
subroutine readprecip_RFE2gdas( n, fname, findex, order, ferror_RFE2gdas, filehr)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain, LIS_masterproc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
        LIS_releaseUnitNumber, LIS_endrun
use LIS_metforcingMod,  only : LIS_forc
use RFE2gdas_forcingMod, only : RFE2gdas_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
character(len=80)   :: fname
integer, intent(in) :: findex
integer, intent(in) :: order
integer             :: ferror_RFE2gdas
integer             :: filehr
```

DESCRIPTION:

For the given time, reads the RFE2gdas data and interpolates to the LIS domain.
The arguments are:

n index of the nest

fname name of the 6 hour RFE2gdas.0 file

findex index of the supplemental forcing source

ferror_RFE2gdas flag to indicate success of the call (=0 indicates success)

filehr current file hour

The routines invoked are:

reproject_RFE2gdas (24.2.5)
Upscales/interpolates the RFE2gdas data

24.2.5 reproject_RFE2gdas (Source File: reproject_RFE2gdas.F90)

INTERFACE:

```
subroutine reproject_RFE2gdas(n,findex,nRFE2gdas,f1d,f2d,lb1d,lb2d, &
     lis_gds, nc,nr, varfield)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_logMod,         only : LIS_logunit, LIS_endrun
use RFE2gdas_forcingMod, only : RFE2gdas_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
integer, intent(in)    :: findex
integer, intent(in)    :: nRFE2gdas
real, intent(in)       :: f1d(nRFE2gdas)
real, intent(in)       :: f2d(NINT(RFE2gdas_struc(n)%gridDesci(2)),&
                              NINT(RFE2gdas_struc(n)%gridDesci(3)))
logical*1, intent(in)  :: lb1d(nRFE2gdas)
logical*1, intent(in)  :: lb2d(NINT(RFE2gdas_struc(n)%gridDesci(2)),&
                               NINT(RFE2gdas_struc(n)%gridDesci(3)))
real,  intent(in)      :: lis_gds(50)
integer, intent(in)    :: nc
integer, intent(in)    :: nr
real, intent(inout)    :: varfield(nc,nr)
```

DESCRIPTION:

This subroutine interpolates a given RFE2gdas field to the LIS grid. The arguments are:

n index of the nest

findex index of the supplemental forcing source

nRFE2gdas number of elements in the input grid

f1d 1-d input data array to be interpolated

f2d 2-d input data array to be interpolated

lb1d input 1-d bitmap

lb2d input 2-d bitmap

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

upscaleByAveraging (5.3.17)
upscales scalar data from a finer grid to a coarser grid

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

neighbor_interp (5.3.10)
spatially interpolate the forcing data using nearest neighbor interpolation

---

24.2.6 timeinterp_RFE2gdas (Source File: timeinterp_RFE2gdas.F90)

REVISION HISTORY:

```
2 June 2010: Soni Yatheendradas; Initial LIS version for FEWSNET
8 Mar 2012: Updated for LIS7
```

INTERFACE:

```
subroutine timeinterp_RFE2gdas(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_metforcingMod, only : LIS_FORC_Base_State, LIS_forc
use LIS_FORC_AttributesMod
use LIS_logMod, only : LIS_verify,LIS_logunit
use RFE2gdas_forcingMod, only : RFE2gdas_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep (evenly distributed over sub-daily timesteps for now).
The arguments are:

n index of the nest

findex index of the supplemental forcing source

suppdata1 previous forcing values

suppdata2 next forcing values

---

25 WRF output

This section describes the implementation of the WRF output forcing reader. This reader extracts meteorological forcing fields from WRF output files.

25.1 Fortran: Module Interface WRFout_forcingMod (Source File: WRFout_forcingMod.F9

This module contains variables and data structures that are used for the implementation of the forcing data extracted from WRF output files. Here WRF output files are consider input data.
The implementation in LIS has the derived data type `WRFout_struc` that includes the variables that specify the runtime options. They are described below:

nest_id Value of the WRF nest

WRFoutdir Directory containing the input data

ts Frequency in seconds of the forcing data

WRFouttime1 The nearest, previous 1 hour instance of the incoming data (as a real time).

WRFouttime2 The nearest, next 1 hour instance of the incoming data (as a real time).

findtime1, findtime2 boolean flags to indicate which time is to be read for temporal interpolation.

USES:

```
implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: init_WRFout      !defines the native resolution of
                           !the input data
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: WRFout_struc
```

25.1.1 init_WRFout (Source File: WRFout_forcingMod.F90)

REVISION HISTORY:

INTERFACE:

```
subroutine init_WRFout(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_update_timestep

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Defines the native resolution of the input forcing for LIS output data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). Note that no interpolation is performed for this forcing. The data is expected to be in the same map projection and resolution as that of the current LIS run.
The arguments are:

findex index of the forcing source

The routines invoked are:

readcrd_WRFout (25.2.4)
reads the runtime options specified for WRF output data

25.2 Fortran: Module Interface finalize_WRFout (Source File: finalize_WRFout.F90)

REVISION HISTORY:

INTERFACE:

```
subroutine finalize_WRFout(findex)
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use WRFout_forcingMod, only : WRFout_struc
```

DESCRIPTION:

Routine to cleanup WRFout forcing related memory allocations.
The arguments are:

findex index of the forcing

25.2.1 get_WRFout (Source File: get_WRFout.F90)

REVISION HISTORY:

```
14 Mar  2013 : Sujay Kumar ; Initial Version in LIS
```

INTERFACE:

```
subroutine get_WRFout(n, findex)
```

USES:

```
use LIS_coreMod,       only : LIS_rc, LIS_domain
use LIS_metforcingMod, only : LIS_forc
use LIS_timeMgrMod,    only : LIS_tick
use LIS_logMod,        only : LIS_logunit, LIS_endrun
use WRFout_forcingMod, only : WRFout_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens and reads 1-hourly WRF output forcing.
The arguments are:

n index of the nest

findex index of the forcing source

The routines invoked are:

LIS_tick (3.36.27)
determines the WRF output data times

WRFoutfile (25.2.2)
Puts together appropriate file name for 1 hour intervals

read_WRFout (25.2.3)
Reads WRF output data to LIS grid

25.2.2 WRFoutfile (Source File: get_WRFout.F90)

REVISION HISTORY:

```
14 Mar 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine WRFoutfile(filename,wrfdir,nest,yr,mo,da,hr,mn,ss)

  implicit none
```

ARGUMENTS:

```
  character*80, intent(out) :: filename
  character*40, intent(in)  :: wrfdir
  integer, intent(in)       :: nest
  integer, intent(in)       :: yr,mo,da,hr,mn,ss
```

DESCRIPTION:

25.2.3 read_WRFout (Source File: read_WRFout.F90)

REVISION HISTORY:

```
14 Mar 2013: Sujay Kumar; Initial Specification in LIS
```

INTERFACE:

```
subroutine read_WRFout(n, findex, order, fname, ferror)
```

USES:

```
  use LIS_coreMod
  use LIS_metforcingMod, only : LIS_forc
  use LIS_logMod
  use WRFout_forcingMod, only : WRFout_struc
#if ( defined USE_NETCDF3 || defined USE_NETCDF4 )
  use netcdf
#endif

  implicit none
```

ARGUMENTS:

```
  integer, intent(in)          :: n
  integer, intent(in)          :: findex
  integer, intent(in)          :: order
  character(len=*), intent(in) :: fname
  integer, intent(out)         :: ferror
```

DESCRIPTION:

For the given time, reads parameters from WRF output files, transforms into 9 LIS forcing parameters and interpolates to the LIS domain.
The arguments are:

n index of the nest

findex index of the forcing source

order flag indicating which data to be read (order=1, read for the previous 1hr bookend, order=2, read for the next 1hr bookend)

fname name of the WRF output file

ferror flag to indicate success of the call (=1 indicates success)

25.2.4 readcrd_WRFout (Source File: readcrd_WRFout.F90)

REVISION HISTORY:

```
14 Mar 2013; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcrd_WRFout()
```

USES:

```
use ESMF
use LIS_coreMod,       only : LIS_rc, LIS_config
use LIS_logMod,        only : LIS_logunit, LIS_verify
use WRFout_forcingMod, only : WRFout_struc

implicit none
```

DESCRIPTION:

This routine reads the options specific to WRF output forcing from the LIS configuration file.

25.3 Fortran: Module Interface reset_WRFout (Source File: reset_WRFout.F90)

REVISION HISTORY:

INTERFACE:

```
subroutine reset_WRFout()
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use WRFout_forcingMod, only : WRFout_struc
```

DESCRIPTION:

Routine to reset WRFout forcing related memory allocations.

25.3.1 timeinterp_WRFout (Source File: timeinterp_WRFout.F90)

REVISION HISTORY:

INTERFACE:

```
subroutine timeinterp_WRFout(n, findex)
```

USES:

```
use ESMF
use LIS_FORC_AttributesMod
use LIS_metforcingMod, only : LIS_forc, LIS_FORC_State
use LIS_coreMod,       only : LIS_rc,LIS_domain
use LIS_constantsMod,  only : LIS_CONST_SOLAR
use LIS_timeMgrMod,    only : LIS_tick, LIS_time2date
use LIS_logMod,        only : LIS_logunit, LIS_verify
use WRFout_forcingMod, only : WRFout_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous 1 hourly value is used. All other variables are linearly interpolated between the 1 hourly blocks.
The arguments are:

n index of the nest

findex index of the forcing source

The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

LIS_tick (3.36.27)
advances or retracts time by the specified amount

zterp (5.3.19)
zenith-angle based interpolation

---

26 AGRMET radiation

This section describes the implementation of the AGRMET radiation product.

26.1 Fortran: Module Interface agrrad_forcingMod (Source File: agrrad_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the AGRMET radiation data. The AGRMET data is in polar stereographic projection at 48km resolution.
The implementation in LIS has the derived data type `agrrad_struc` that includes the variables to specify the runtime options, and the weights and neighbor information for spatial interpolation
They are desribed below:

ts Frequency of the AGRRAD forcing data.

agrdir Directory containing the input data.

agrtime1 The nearest, previous hourly instance of the incoming data (as a real time).

agrtime2 The nearest, next hourly instance of the incoming data (as a real time).

changetime1 The time to switch the input resolution to T170

changetime2 The time to switch the input resolution to T170

findtime1, findtime2 Flags to indicate which time is to be read for temporal interpolation.

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n111,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

fillflag1 Flag used to control initializing smask1.

smask1 Array used to mark mismatches between the AGRRAD forcing and the land/sea mask.

USES:

```
implicit none
PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: init_agrrad      !defines the native resolution of
                           !the input data
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: agrrad_struc
```

26.1.1 init_agrrad (Source File: agrrad_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine init_agrrad(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Defines the native resolution of the input forcing for AGRMET radiation data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V).
The routines invoked are:

readcrd_agrrad (26.1.8)
reads the runtime options specified for AGRMET radiation data

LIS_date2time (3.36.24)
converts date to the real time format

bilinear_interp_input (5.1.2)
computes the neighbor, weights for conservative interpolation

---

26.1.2 finalize_agrrad (Source File: finalize_agrrad.F90)

REVISION HISTORY:

```
29 Jul 2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine finalize_agrrad(findex)
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use agrrad_forcingMod, only : agrrad_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine deallocates and cleans up allocated memory structures for AGRMET radiation forcing implementation.
The arguments are:

findex index of the forcing

26.1.3 get_agrrad (Source File: get_agrrad.F90)

REVISION HISTORY:

```
29 Jul 2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine get_agrrad(n, findex)
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use LIS_metforcingMod, only : LIS_forc
use LIS_timeMgrMod,    only : LIS_tick
use agrrad_forcingMod, only : agrrad_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This is the entry point for calling various AGRMET radiation analyses. At the beginning of a simulation, the code invokes call to read the most recent past data (nearest hourly interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep.
The arguments are:

n index of the nest

findex index of the forcing source

The routines invoked are:

read_agrrad (26.1.4)
routines that reads the AGRMET radiation and interpolates it to the LIS projection.

LIS_tick (3.36.27)
computes the AGRMET read/processing times.

26.1.4 read_agrrad (Source File: read_agrrad.F90)

REVISION HISTORY:

```
29Jul2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine read_agrrad(n,findex,order,yr,mo,da,hr)
```

USES:

```
use LIS_coreMod,        only : LIS_rc,LIS_domain
use LIS_logMod,         only : LIS_logunit, LIS_endrun
use LIS_timeMgrMod,     only : LIS_tick
use LIS_metforcingMod,  only : LIS_forc
use agrrad_forcingMod,  only : agrrad_struc

implicit none
```

ARGUMENTS:

```
integer,   intent(in)    :: n
integer,   intent(in)    :: findex
integer,   intent(in)    :: order
integer,   intent(in)    :: yr,mo,da,hr
```

DESCRIPTION:

This routine opens the AGRMET files and reads the radiation fields. The fields are then spatially interpolated and gaps in the interpolation dues ot mismatches in landmask (between LIS and AGRMET) are filled.
The arguments are:

n index of the nest

findex index of the forcing source

order flag indicating which data to be read (order=1, read the previous hourly instance, order=2, read the next hourly instance)

yr,mo,da,hr current year, month, day and hour.

The routines invoked are:

LIS_tick (3.36.27)
determines the AGRMET data times

agrrad_filename (26.1.5)
generates the name of the AGRMET file to be read

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

agrrad_fillgaps (26.1.6)
fills in the mismatches due to AGRMET and LIS masks

retrieve_agrrad_variables (26.1.7)
retrieves the agrrad variables

26.1.5 agrrad_filename (Source File: read_agrrad.F90)

INTERFACE:

```
subroutine agrrad_filename(name,rootdir,yr,mo,da,hr)

  implicit none
```

ARGUMENTS:

```
  character(len=*)          :: name
  character(len=*)          :: rootdir
  integer, intent(IN)       :: yr
  integer, intent(IN)       :: mo
  integer, intent(IN)       :: da
  integer, intent(IN)       :: hr
```

DESCRIPTION:

This routines generates the name of the surface temperature file by appending the hemisphere and timestamps to the root directory.
The arguments are:

name filename to be generated

rootdir name of the root directory that contains the forcing

yr year of data

mo month of data

da day of data

hr hour of data

26.1.6 agrrad_fillgaps (Source File: read_agrrad.F90)

INTERFACE:

```
subroutine agrrad_fillgaps(n,varfield)
```

USES:

```
use LIS_coreMod,       only : LIS_rc,LIS_domain
use LIS_logMod,      only   : LIS_logunit, LIS_endrun
use agrrad_forcingMod, only : agrrad_struc

implicit none
```

USES:

```
integer, intent(in)    :: n
real, intent(inout)    :: varfield(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine fills in invalid grid points introduced due to reprojection from PS to lat/lon. This routine assumes that the undef or invalid value is the LIS undefined value.
The arguments are:

n index of the nest

ip interpolation option

varfield updated output field

26.1.7 retrieve_agrrad_variables (Source File: read_agrrad.F90)

REVISION HISTORY:

```
04 Dec 2013: James Geiger: initial implementation
```

INTERFACE:

```
subroutine retrieve_agrrad_variables(fname, error, vindex, var)
```

USES:

```
  use LIS_coreMod, only : LIS_rc
  use LIS_logMod,  only : LIS_logunit, LIS_endrun, LIS_verify

#if (defined USE_GRIBAPI)
  use grib_api
#endif

  implicit none
  integer, parameter            :: nc=720, nr=361
```

ARGUMENTS:

```
  character(len=100), intent(in) :: fname
  integer, intent(out)           :: error
  integer, intent(in)            :: vindex
  real, intent(out)              :: var(nc,nr)
```

DESCRIPTION:

This routine retrieves the AGRRAD forcing variable specified by vindex from the file specified by fname. The arguments are:

fname name of the AGRRAD forcing file

error return status of this subroutine error = 0 indicates success error = -1 indicates failure

vindex index of the variable to read

var array to contain AGRRAD forcing variable read from file

26.1.8 readcrd_agrrad (Source File: readcrd_agrrad.F90)

REVISION HISTORY:

```
  29Jul2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcrd_agrrad()
```

USES:

```
  use ESMF
  use LIS_coreMod,       only : LIS_rc, LIS_config
  use agrrad_forcingMod, only : agrrad_struc
  use LIS_logMod,        only : LIS_logunit

  implicit none
```

DESCRIPTION:

This routine reads the options specific to AGRMET algorithms from the LIS configuration file.

26.1.9 timeinterp_agrrad (Source File: timeinterp_agrrad.F90)

REVISION HISTORY:

```
08 Dec 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine timeinterp_agrrad(n, findex)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc,LIS_domain
use LIS_FORC_AttributesMod
use LIS_metforcingMod,  only : LIS_FORC_Base_State, LIS_forc
use LIS_logMod,         only : LIS_verify
use agrrad_forcingMod,  only : agrrad_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the AGRMET radiation forcing data to the model timestep.
The arguments are:

n index of the nest

findex index of the forcing source

---

27 AGRMET polar stereographic radiation

This section describes the implementation of the AGRMET polar stereographic radiation product.

27.0.10 agrradps_filename_sw (Source File: agrradps_filenames.F90)

INTERFACE:

```
subroutine agrradps_filename_sw(nameNH,nameSH,rootdir,yr,mo,da,hr)

  implicit none
```

ARGUMENTS:

```
character(len=*)          :: nameNH
character(len=*)          :: nameSH
character(len=*)          :: rootdir
integer, intent(IN)       :: yr
integer, intent(IN)       :: mo
integer, intent(IN)       :: da
integer, intent(IN)       :: hr
```

DESCRIPTION:

This routines generates the name of the shortwave data file by appending the hemisphere and timestamps to the root directory.
The arguments are:

nameNH filename to be generated for NH

nameSH filename to be generated for SH

rootdir name of the root directory that contains the forcing

yr year of data

mo month of data

da day of data

hr hour of data

27.0.11 agrradps_filename_cloud (Source File: agrradps_filenames.F90)

INTERFACE:

```
subroutine agrradps_filename_cloud(nameNH,nameSH,rootdir,yr,mo,da,hr,layer)

  implicit none
```

ARGUMENTS:

```
  character(len=*)          :: nameNH
  character(len=*)          :: nameSH
  character(len=*)          :: rootdir
  integer, intent(IN)       :: yr
  integer, intent(IN)       :: mo
  integer, intent(IN)       :: da
  integer, intent(IN)       :: hr
  character(len=1), intent(IN)  :: layer
```

DESCRIPTION:

This routines generates the name of the cloud amount data file by appending the hemisphere and timestamps to the root directory.
The arguments are:

name filename to be generated

rootdir name of the root directory that contains the forcing

yr year of data

mo month of data

da day of data

hr hour of data

27.1 Fortran: Module Interface agrradps_forcingMod (Source File: agrradps_forcingMod.F9

This module contains variables and data structures that are used for the implementation of the AGRMET radiation data. The AGRMET data is in polar stereographic projection at 48km resolution in two hemispheres, NH and SH at hourly.
The implementation in LIS has the derived data type `agrradps_struc` that includes the variables to specify the runtime options, and the weights and neighbor information for spatial interpolation
They are desribed below:

agrtime1 The nearest, previous hourly instance of the incoming data (as a real time).

agrtime2 The nearest, next hourly instance of the incoming data (as a real time).

agrpsdir Directory containing the input data

gridspan value indicating the span of the LIS domain (1-NH only, 2-SH only, 3-span includes both NH and SH)

mo1 number of elements in the NH for the LIS grid

mo2 number of elements in the SH for the LIS grid

hemi_nc number of columns (in the east-west dimension) for each hemisphere

hemi_nr number of rows (in the north-south dimension) for each hemisphere

shemi value indicating the starting index of hemisphere loops (1-NH only,2-SH only, 1-span includes both NH and SH)

nhemi value indicating the ending index of hemisphere loops (1-NH only,2-SH only, 2-span includes both NH and SH)

mi number of elements in the input grid

imax x-dimension size of the input grid

jmax y-dimension size of the input grid

interp variable specifying if spatial interpolation needs to be done

rlat1_nh Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1_nh Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n111_nh,n121_nh,n211_nh,n221_nh Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111_nh,w121_nh,w211_nh,w221_nh Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

rlat1_sh Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1_sh Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n111_sh,n121_sh,n211_sh,n221_sh Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111_sh,w121_sh,w211_sh,w221_sh Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

smask landmask to used to fill any gaps in bilinear interpolation due to mismatches in the LIS and AGRMET masks

fillflag1 flag to check for filling gaps due to LIS and AGRMET mask mismatches, for bilinear interpolation

fillflag2 flag to check for filling gaps due to LIS and AGRMET mask mismatches, for neighbor interpolation

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

findtime1, findtime2 boolean flags to indicate which time is to be read for temporal interpolation.

USES:

```
implicit none
```

PUBLIC MEMBER FUNCTIONS:

```
---------------------------------------------------------------------------
 public :: init_agrradps    !defines the native resolution of
                                   !the input data
---------------------------------------------------------------------------
```

PUBLIC TYPES:

```
---------------------------------------------------------------------------
 public :: agrradps_struc
```

27.1.1 init_agrradps (Source File: agrradps_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine init_agrradps(findex)
```

USES:

```
  use LIS_coreMod, only: LIS_rc
  use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep
  use LIS_logMod, only :LIS_logunit, LIS_endrun

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Defines the native resolution of the input forcing for AGRRADPS data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). The routines invoked are:

readcrd_agrradps (27.1.8)
reads the runtime options specified for AGRRADPS data

LIS_date2time (3.36.24)
converts date to the real time format

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

27.1.2 finalize_agrradps (Source File: finalize_agrradps.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine finalize_agrradps
```

USES:

```
use LIS_coreMod,         only : LIS_rc
use agrradps_forcingMod, only : agrradps_struc
```

DESCRIPTION:

Routine to cleanup AGRRADPS forcing related memory allocations.

27.1.3 get_agrradps (Source File: get_agrradps.F90)

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
16 Feb 2007: Chuck Alonge; Changed file name creation to use internal
              files instead of external file (avoids failure in parallel runs)
```

INTERFACE:

```
subroutine get_agrradps(n, findex)
```

USES:

```
use LIS_coreMod,         only : LIS_rc
use LIS_metforcingMod,  only : LIS_forc
use agrradps_forcingMod, only : agrradps_struc
use LIS_timeMgrMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 6-hrly, CMAP forcing. At the beginning of a simulation, the code reads the most recent past data (nearest 6 hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. . The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the data times

read_agrradps (27.1.5)
routines that reads the AGRMET radiation (polar stereographic) and interpolates it to the LIS projection.

27.1.4 interp_agrradpsvar (Source File: interp_agrradpsvar.F90)

INTERFACE:

```
subroutine interp_agrradpsvar(n,ip,gi,udef,varfield)
```

USES:

```
use LIS_coreMod,         only : LIS_rc
use agrradps_forcingMod, only : agrradps_struc
implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
integer, intent(in)    :: ip
real, intent(in)       :: gi(2,agrradps_struc(n)%imax,agrradps_struc(n)%jmax)
real, intent(in)       :: udef
real, intent(inout)    :: varfield(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine interpolates a given AFWA field to the LIS domain (from polar stereographic grid to the LIS grid)
The arguments are:

n index of the nest

ip interpolation option

gi input AGRMET field (both hemispheres)

udef undefined value in the input field

varfield interpolated field in the LIS grid

The routines invoked are:

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

neighbor_interp (5.3.10)
spatially interpolate the forcing data using neighbor interpolation

---

27.1.5 read_agrradps (Source File: read_agrradps.F90)

REVISION HISTORY:

```
29Jul2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine read_agrradps(n,m,order,yr,mo,da,hr)
```

USES:

```
use ESMF
use LIS_coreMod,         only : LIS_rc,LIS_domain
use agrradps_forcingMod, only : agrradps_struc
use LIS_metforcingMod,   only : LIS_forc
use LIS_logMod,          only : LIS_logunit, LIS_verify
use LIS_FORC_AttributesMod
use LIS_metforcingMod,  only : LIS_FORC_Base_State

implicit none
```

ARGUMENTS:

```
integer,   intent(IN)    :: n
integer,   intent(IN)    :: m
integer,   intent(IN)    :: order
integer,   intent(IN)    :: yr,mo,da,hr
```

DESCRIPTION:

This routine opens the AGRMET files and reads the radiation fields. The fields are then spatially interpolated and gaps in the interpolation due to mismatches in landmask (between LIS and AGRMET) are filled.
The arguments are:

n index of the nest

order flag indicating which data to be read (order=1, read the previous hourly instance, order=2, read the next hourly instance)

error output error code to indicate if the data was read and interpolated successfully.

yr,mo,da,hr current year, month, day and hour.

The routines invoked are:

agrradps_filename_sw (27.0.10)
generates the name of the AGRMET file to be read

agrradps_filename_cloud (27.0.11)
generates the name of the AGRMET file to be read

agrmet2latlon1 (27.1.7)
spatially interpolate the forcing data using bilinear interpolation

---

27.1.6 fillgaps_agrradps (Source File: read_agrradps.F90)

INTERFACE:

```
subroutine fillgaps_agrradps(n,varfield)
```

USES:

```
use LIS_coreMod,         only : LIS_rc,LIS_domain
use agrradps_forcingMod, only : agrradps_struc
use LIS_logMod,          only : LIS_logunit, LIS_endrun

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real, intent(inout)    :: varfield(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine fills in invalid grid points introduced due to reprojection from PS to lat/lon. This routine assumes that the undef or invalid value is the LIS undefined value.
The arguments are:

n index of the nest

ip interpolation option

varfield updated output field

27.1.7 agrmet2latlon1 (Source File: read_agrradps.F90)

INTERFACE:

```
subroutine agrmet2latlon1 (n, nameNH, nameSH, varfield, ferror)
```

USES:

```
use LIS_coreMod,         only : LIS_rc,LIS_domain
use LIS_logMod,          only : LIS_logunit
use agrradps_forcingMod, only : agrradps_struc

 implicit none
```

ARGUMENTS:

```
integer,intent(in)          :: n
character(len=*),intent(in) :: nameNH, nameSH
real,intent(out)            :: varfield(LIS_rc%lnc(n),LIS_rc%lnr(n))
integer,intent(out)         :: ferror    !set to 1 if error found
real, parameter             :: udef = -999.  ! input data missing value
```

DESCRIPTION:

Reads in NH and SH data, and interpolates to LIS lat/lon domain. Currently set to proceed only if both files exist even for processing one hemisphere.

27.1.8 readcrd_agrradps (Source File: readcrd_agrradps.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcrd_agrradps()
```

USES:

```
use agrradps_forcingMod, only : agrradps_struc
use LIS_coreMod,         only : LIS_config,LIS_rc
use ESMF
use LIS_logMod,          only : LIS_logunit
```

DESCRIPTION:

This routine reads the options specific to AGRRADPS forcing from the LIS configuration file.

27.1.9 timeinterp_agrradps (Source File: timeinterp_agrradps.F90)

REVISION HISTORY:

```
20 Jan 2006:   Yudong Tian: Initial Implementation
10 Oct 2006:   Sujay Kumar: Switched to using ESMF_State for storing
                            forcing data.
```

INTERFACE:

```
subroutine timeinterp_agrradps(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod,         only : LIS_rc, LIS_domain
use LIS_FORC_AttributesMod
use LIS_metforcingMod,   only : LIS_FORC_Base_State, LIS_forc
use LIS_logMod,          only : LIS_verify
use agrradps_forcingMod, only : agrradps_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep. All variables except precipitation is linearly interpolated.
The arguments are:

n index of the nest

---

28 CEOP forcing

This section describes the implementation of the CEOP station data.

28.1 Fortran: Module Interface ceop_forcingMod (Source File: ceop_forcingMod.F90)

Contains routines and data structures that are used for the implementation of the station data from various CEOP stations. The stations report estimates of meteorological forcing terms, which is spatially interpolated using the inverse distance weighting scheme (IDW).
The implementation in LIS has the derived data type `ceop_struc` that includes the variables to specify the runtime options, and the calculation of weights for spatial interpolation.

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification

implicit none
PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: init_CEOP      !defines the native resolution of
                                !the input data
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: ceop_struc
```

28.1.1 init_CEOP (Source File: ceop_forcingMod.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine init_CEOP(findex)
```

USES:

```
use LIS_coreMod,only : LIS_rc
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep
use LIS_logMod, only : LIS_logunit, LIS_endrun

implicit none
integer, intent(in) :: findex
```

DESCRIPTION:

This routines reads the runtime configurations for using the CEOP station data. Using the metadata provided for the stations, this routine invokes the call to compute the interpolation weights to be later used.
The routines invoked are:

readcrd_ceop (28.1.6)
reads the runtime options specified for CEOP station data

LIS_date2time (3.36.24)
converts date to the real time format

compute_stnwts (5.1.22)
computes the weights for spatial interpolation

28.1.2 finalize_ceop (Source File: finalize_ceop.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine finalize_ceop(findex)
```

USES:

```
use ceop_forcingMod, only : ceop_struc
```

DESCRIPTION:

Routine to cleanup CEOP forcing related memory allocations.

28.1.3 get_ceop (Source File: get_ceop.F90)

REVISION HISTORY:

```
08 Dec 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine get_ceop(n, findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod, only : LIS_tick
use LIS_logMod, only : LIS_logunit
use LIS_metforcingMod, only : LIS_forc
use ceop_forcingMod, only : ceop_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates the CEOP station data. At the beginning of a simulation, the code reads the most recent past data, and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. . The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the CEOP data times

read_ceop (28.1.4)
Interpolates the appropriate CEOP station data to LIS grid

28.1.4 read_ceop (Source File: read_ceop.F90)

REVISION HISTORY:

```
08 Dec 2002: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_ceop(n,findex,order)
```

USES:

```
use LIS_logMod, only        : LIS_logunit
use LIS_coreMod, only     : LIS_rc,LIS_domain
use LIS_metforcingMod,  only : LIS_forc
use ceop_forcingMod, only : ceop_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

For the given time, reads parameters from the correct CEOP station data (ASCII), transforms into LIS forcing parameters and interpolates to the LIS domain.
The arguments are:

order flag indicating which data to be read (order=1, read the previous instance, order=2, read the next instance)

n index of the nest

The routines invoked are:

readbon (28.1.5)
reads the Bondville, IL data

readfpk (28.1.7)
reads the Fort Peck, MT data

readsgp (28.1.8)
reads the SGP data

normalize_stnwts (5.3.12)
renormalizes the station weights accounting for missing data

interp_stndata (5.2.8)
spatially interpolates the station data onto the LIS grid.

28.1.5 readbon (Source File: readbon.F90)

REVISION HISTORY:

```
08 Dec 2002: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine readbon(n,order,ftn,tair,qair,swdown,lwdown,u,v,psurf,pcp)
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use LIS_timeMgrMod,  only : LIS_tick
use LIS_logMod,      only : LIS_logunit, LIS_endrun
use ceop_forcingMod, only : ceop_struc

implicit none
```

ARGUMENTS:

```
integer,  intent (IN)     :: n
integer , intent (IN)     :: ftn
integer , intent (IN)     :: order
real    , intent (INOUT)  :: psurf(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: tair(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: qair(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: u(ceop_struc(n)%nstns),v(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: pcp(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: swdown(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: lwdown(ceop_struc(n)%nstns)
```

DESCRIPTION:

Reads the CEOP station data for Bondville, IL. The format of the data is ASCII.
The arguments are:

n index of the nest

order flag indicating which data to be read (order=1, read the previous instance, order=2, read the next instance)

ftn unit number used to open the file

tair array for storing the 2m air temperature for all stations

qair array for storing the 2m relative humidity for all stations

swdown array for storing the downward shortwave radiation for all stations

lwdown array for storing the downward longwave radiation for all stations

u array for storing the u component of wind for all stations

v array for storing the v component of wind for all stations

psurf array for storing the surface pressure for all stations

pcp array for storing the precipitation for all stations

28.1.6 readcrd_ceop (Source File: readcrd_ceop.F90)

REVISION HISTORY:

```
08 Dec 2004; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcrd_ceop()
```

USES:

```
use ESMF
use LIS_logMod, only : LIS_logunit
use LIS_coreMod, only : LIS_rc,LIS_config
use ceop_forcingMod, only : ceop_struc
```

DESCRIPTION:

This routine reads the options specific to CEOP station data forcing from the LIS configuration file.

28.1.7 readfpk (Source File: readfpk.F90)

REVISION HISTORY:

```
08 Dec 2002: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine readfpk(n,order,ftn,tair,qair,swdown,lwdown,u,v,psurf,pcp)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod, only : LIS_tick
use LIS_logMod, only : LIS_logunit, LIS_endrun
use ceop_forcingMod, only :ceop_struc

implicit none
```

ARGUMENTS:

```
integer,  intent (IN)     :: n
integer , intent (IN)     :: ftn
integer , intent (IN)     :: order
real    , intent (INOUT)  :: psurf(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: tair(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: qair(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: u(ceop_struc(n)%nstns),v(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: pcp(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: swdown(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: lwdown(ceop_struc(n)%nstns)
```

DESCRIPTION:

Reads the CEOP station data for Fort Peck, MT. The format of the data is ASCII.
The arguments are:

n index of the nest

order flag indicating which data to be read (order=1, read the previous instance, order=2, read the next instance)

ftn unit number used to open the file

tair array for storing the 2m air temperature for all stations

qair array for storing the 2m relative humidity for all stations

swdown array for storing the downward shortwave radiation for all stations

lwdown array for storing the downward longwave radiation for all stations

u array for storing the u component of wind for all stations

v array for storing the v component of wind for all stations

psurf array for storing the surface pressure for all stations

pcp array for storing the precipitation for all stations

28.1.8 readsgp (Source File: readsgp.F90)

REVISION HISTORY:

```
08 Dec 2002: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine readsgp(n,tair,qair,swdown,lwdown,u,v,psurf,pcp, &
     errorcode)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod, only : LIS_tick, LIS_date2time
use LIS_logMod, only : LIS_logunit, LIS_endrun
use ceop_forcingMod, only : ceop_struc

implicit none
```

ARGUMENTS:

```
integer,  intent (IN)     :: n
real    , intent (INOUT)  :: psurf(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: tair(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: qair(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: u(ceop_struc(n)%nstns),v(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: pcp(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: swdown(ceop_struc(n)%nstns)
real    , intent (INOUT)  :: lwdown(ceop_struc(n)%nstns)
integer , intent (INOUT)  :: errorcode
```

DESCRIPTION:

Reads the CEOP station data for SGP. The format of the data is ASCII.
The arguments are:

n index of the nest

order flag indicating which data to be read (order=1, read the previous instance, order=2, read the next instance)

tair array for storing the 2m air temperature for all stations

qair array for storing the 2m relative humidity for all stations

swdown array for storing the downward shortwave radiation for all stations

lwdown array for storing the downward longwave radiation for all stations

u array for storing the u component of wind for all stations

v array for storing the v component of wind for all stations

psurf array for storing the surface pressure for all stations

pcp array for storing the precipitation for all stations

28.1.9 timeinterp_ceop (Source File: timeinterp_ceop.F90)

REVISION HISTORY:

```
08 Dec 2004: Sujay Kumar; Initial Specification
10 Oct 2006:   Sujay Kumar: Switched to using ESMF_State for storing
                            forcing data.
```

INTERFACE:

```
subroutine timeinterp_ceop(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_metforcingMod, only : LIS_FORC_Base_State, LIS_forc
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use ceop_forcingMod, only : ceop_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the CEOP forcing data to the model timestep. All variables except precipitation is linearly interpolated.

---

29 CMAP

This section describes the implementation of the precipitation forcing from the Climate Prediction Center's (CPC) Merged Analysis of Precipitation (CMAP)

29.1 Fortran: Module Interface cmap_forcingMod (Source File: cmap_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the precipitation data from the Climate Prediction Center (CPC)'s Merged Analysis of Precipitation (CMAP). CMAP merges gauge measurements and satellite estimates including GPI, OPI, SSM/I to produce a global 2.5 degree pentad precipitation analysis. The 6-hourly product obtained by disaggregating CMAP using the GDAS forcing is used in this implementation.
The implementation in LIS has the derived data type `cmap_struc` that includes the variables to specify the runtime options, and the weights and neighbor information for spatial interpolation
They are desribed below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

cmapdir Directory containing the input data

cmaptime The nearest, hourly instance of the incoming data (as a real time).

griduptime1 The time to switch the input resolution to T170

mi Number of points in the input grid

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n112,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

USES:

```
implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: init_CMAP      !defines the native resolution of
                                !the input data
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: cmap_struc
```

29.1.1 init_CMAP (Source File: cmap_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine init_CMAP(findex)
```

USES:

```
use LIS_coreMod, only: LIS_rc
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep
use LIS_logMod,  only : LIS_logunit, LIS_endrun

implicit none
integer, intent(in) :: findex
```

DESCRIPTION:

Defines the native resolution of the input forcing for CMAP data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V).
The routines invoked are:

readcrd_cmap (29.1.8)
reads the runtime options specified for CMAP data

LIS_date2time (3.36.24)
converts date to the real time format

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

29.1.2 finalize_cmap (Source File: finalize_cmap.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine finalize_cmap(findex)
```

USES:

```
use cmap_forcingMod, only : cmap_struc
```

DESCRIPTION:

Routine to cleanup CMAP forcing related memory allocations.

29.1.3 get_cmap (Source File: get_cmap.F90)

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
16 Feb 2007: Chuck Alonge; Changed file name creation to use internal
              files instead of external file (avoids failure in parallel runs)
```

INTERFACE:

```
subroutine get_cmap(n,findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod, only : LIS_tick, LIS_get_nstep
use LIS_logMod, only : LIS_logunit
use cmap_forcingMod, only : cmap_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 6-hrly, CMAP forcing. At the beginning of a simulation, the code reads the most recent past data (nearest 6 hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the CMAP data times

cmapfile (29.1.4)
Puts together appropriate file name for 6 hour intervals

read_cmap (29.1.7)
Interpolates CMAP data to LIS grid

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

29.1.4 cmapfile (Source File: get_cmap.F90)

INTERFACE:

```
subroutine cmapfile( name, cmapdir, yr, mo, da, hr)
```

```
implicit none
```

ARGUMENTS:

```
character(len=*)   :: name
character(len=*)   :: cmapdir
character(len=100) :: temp
integer            :: yr, mo, da, hr
```

DESCRIPTION:

This subroutine puts together CMAP file name for 6 hour file intervals.
The arguments are:

cmapdir Name of the CMAP directory

yr year

mo month

da day of month

hr hour of day

name name of the timestamped CMAP file

29.1.5 cmapfile_old (Source File: get_cmap.F90)

This subroutine puts together CMAP file name for 6 hour file intervals.

INTERFACE:

```
subroutine cmapfile_old( name, cmapdir, yr, mo, da, hr)
```

29.1.6 interp_cmap (Source File: interp_cmap.F90)

INTERFACE:

```
subroutine interp_cmap(n,ncmap,f,lb,lis_gds,nc,nr, &
     varfield)
```

USES:

```
use cmap_forcingMod, only : cmap_struc
```

```
implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
integer                :: nc
integer                :: nr
integer                :: ncmap
real                   :: lis_gds(50)
real                   :: f(ncmap)
logical*1              :: lb(ncmap)
real, dimension(nc,nr) :: varfield
```

DESCRIPTION:

This subroutine interpolates a given CMAP field to the LIS grid. The arguments are:

n index of the nest

ncmap number of elements in the input grid

f input data array to be interpolated

lb input bitmap

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

---

29.1.7 read_cmap (Source File: read_cmap.F90)

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)
16 Feb 2007: Chuck Alonge; Changed baopen to baopenr for grib
25 Jan 2012: Sujay Kumar; Switched to the use of grib-api library
```

INTERFACE:

```
subroutine read_cmap( n, fname, findex, order, ferror_cmap, filehr)
```

USES:

```
  use LIS_coreMod,        only : LIS_rc, LIS_domain
  use LIS_logMod,         only : LIS_logunit, LIS_warning, LIS_verify
  use LIS_metforcingMod, only : LIS_forc
  use cmap_forcingMod,    only : cmap_struc
#if (defined USE_GRIBAPI)
  use grib_api
#endif

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: n
  character(len=80)   :: fname
  integer, intent(in) :: findex
  integer, intent(in) :: order
  integer             :: ferror_cmap
  integer             :: filehr
```

DESCRIPTION:

For the given time, reads parameters from CMAP data and interpolates to the LIS domain.
The arguments are:

n index of the nest

fname name of the 6 hour CMAP file

ferror_cmap flag to indicate success of the call (=0 indicates success)

filehr current file hour

The routines invoked are:

interp_cmap (29.1.6)
spatially interpolates the CMAP data

29.1.8 readcrd_cmap (Source File: readcrd_cmap.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcrd_cmap()
```

USES:

```
use ESMF
use cmap_forcingMod, only : cmap_struc
use LIS_coreMod, only : LIS_config,LIS_rc
use LIS_logMod, only : LIS_logunit
```

DESCRIPTION:

This routine reads the options specific to CMAP forcing from the LIS configuration file.

29.1.9 timeinterp_cmap (Source File: timeinterp_cmap.F90)

REVISION HISTORY:

```
20 Jan 2006:   Yudong Tian: Initial Implementation
10 Oct 2006:   Sujay Kumar: Switched to using ESMF_State for storing
                            forcing data.
```

INTERFACE:

```
subroutine timeinterp_cmap(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod, only: LIS_rc, LIS_domain
use LIS_metforcingMod, only : LIS_FORC_Base_State, LIS_forc
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use cmap_forcingMod, only : cmap_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep.
The arguments are:

n index of the nest

---

30 CMORPH

This section describes the implementation of the precipitation product from CPC's MORPHing technique, known as CMORPH.

30.1 Fortran: Module Interface cmorph_forcingMod (Source File: cmorph_forcingMod.F90)

REVISION HISTORY:

```
05Jan2006: Yudong Tian; start from Jon G.'s code for older versions of LIS.
```

DESCRIPTION:

This module contains variables and data structures that are used for the implementation of the precipitation data from the Climate Prediction Center (CPC)'s MORPHing technique (CMORPH). CMORPH products are produced as global fields from 60 N-60S at 8km resolution at 30 minute intervals.
The implementation in LIS has the derived data type `cmorph_struc` that includes the variables to specify the runtime options, and the weights and neighbor information for spatial interpolation
They are desribed below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

cmordir Directory containing the input data

cmortime The nearest, hourly instance of the incoming data (as a real time).

griduptime1 The time to switch the input resolution to T170

mi Number of points in the input grid

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n112,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

USES:

```
implicit none
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init_cmorph      !defines the native resolution of
                                !the input data
```

PUBLIC TYPES:

```
public :: cmorph_struc
```

30.1.1 init_cmorph (Source File: cmorph_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
06Jan2006: Yudong Tian; modification for LISv4.2
```

INTERFACE:

```
subroutine init_cmorph(findex)
```

USES:

```
use LIS_coreMod, only: LIS_rc
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep

implicit none
integer, intent(in) :: findex

real :: gridDesci(50)
integer :: updoy, yr1,mo1,da1,hr1,mn1,ss1
real :: upgmt
integer :: n
```

DESCRIPTION:

Defines the native resolution of the input forcing for CMORPH data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes V
The routines invoked are:

readcrd_cmorph (30.1.7)
reads the runtime options specified for CMORPH data

LIS_date2time (3.36.24)
converts date to the real time format

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

30.1.2 finalize_cmorph (Source File: finalize_cmorph.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine finalize_cmorph(findex)
```

USES:

```
use cmorph_forcingMod, only : cmorph_struc
```

DESCRIPTION:

Routine to cleanup CMORPH forcing related memory allocations.

30.1.3 get_cmorph (Source File: get_cmorph.F90)

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
29 Dec 2003: Luis Goncalves; Added CMORPH global observed precip data sources
06 Jan 2005: Yudong Tian; Modified for LISv4.2
```

INTERFACE:

```
subroutine get_cmorph(n, findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_timeMgrMod, only : LIS_tick, LIS_get_nstep
use cmorph_forcingMod, only :cmorph_struc
use LIS_logMod, only : LIS_logunit

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 30 minute CMORPH forcing. At the beginning of a simulation, the code reads the most recent past data (nearest 30min interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the CMORPH data times

cmorphfile (30.1.4)
Puts together appropriate file name for 6 hour intervals

read_cmorph (30.1.6)
Interpolates CMORPH data to LIS grid

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

30.1.4 cmorphfile (Source File: get_cmorph.F90)

INTERFACE:

```
subroutine cmorphfile( name, cmorphdir, yr, mo, da, hr)

  implicit none
```

ARGUMENTS:

```
  character(len=*) :: name
  character(len=*) :: cmorphdir
  integer            :: yr, mo, da, hr
```

DESCRIPTION:

This subroutine puts together CMORPH file name for 30min file intervals.
The arguments are:

cmorphdir Name of the CMORPH directory

yr year

mo month

da day of month

hr hour of day

name name of the timestamped CMORPH file

30.1.5 interp_cmorph (Source File: interp_cmorph.F90)

INTERFACE:

```
subroutine interp_cmorph(n, nx, ny, finput, lis_gds, nc, nr, varfield)
```

USES:

```
use cmorph_forcingMod, only : cmorph_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer             :: nx
integer             :: ny
integer             :: nc
integer             :: nr
real, dimension(nx,ny) :: finput
real, dimension(nc,nr) :: varfield
```

DESCRIPTION:

This subroutine interpolates a given CMORPH field to the LIS grid. The arguments are:

n index of the nest

nx number of columns (in the east-west dimension) in the CMORPH grid

ny number of rows (in the north-south dimension) in the CMORPH grid

finput input data array to be interpolated

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

---

30.1.6 read_cmorph (Source File: read_cmorph.F90)

REVISION HISTORY:

```
17 Jul 2001: Jon Gottschalck; Initial code
04 Feb 2002: Jon Gottschalck; Added necessary code to use global precip
             observations with LIS_domain 3 (2x2.5)
29 Dec 2003: Luis Goncalves; Added code to use CMORPH precip data
06 Jan 2006: Yudong Tian; modified for LISv4.2
```

INTERFACE:

```
subroutine read_cmorph (n, name_cmorph, findex, order, ferror_cmorph, iflg )
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_masterproc
use LIS_logMod, only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_metforcingMod, only : LIS_forc
use cmorph_forcingMod, only : cmorph_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
character(len=*)    :: name_cmorph
integer, intent(in) :: findex
integer, intent(in) :: order
integer             :: ferror_cmorph
integer             :: iflg
```

DESCRIPTION:

For the given time, reads parameters from CMORPH data and interpolates to the LIS domain. The arguments are:

n index of the nest

name_cmorph name of the CMORPH file

ferror_cmorph flag to indicate success of the call (=0 indicates success)

iflg flag indicating which 1/2 hour to read

The routines invoked are:

interp_cmorph (30.1.5)
spatially interpolates the CMORPH data

30.1.7 readcrd_cmorph (Source File: readcrd_cmorph.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
29 Dec 2003; Luis Gustavo, adapted for CMORPH
06 Jan 2005; Yudong Tian, modified for LISv4.2
```

INTERFACE:

```
subroutine readcrd_cmorph()
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_config, LIS_rc
use LIS_logMod, only : LIS_logunit
use cmorph_forcingMod, only : cmorph_struc
```

DESCRIPTION:

This routine reads the options specific to CMORPH forcing from the LIS configuration file.

30.1.8 timeinterp_cmorph (Source File: timeinterp_cmorph.F90)

REVISION HISTORY:

```
 20 Jan 2006:   Yudong Tian: Initial Implementation
 10 Oct 2006:   Sujay Kumar: Switched to using ESMF_State for storing
                             forcing data.
```

INTERFACE:

```
subroutine timeinterp_cmorph(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc,LIS_domain
use LIS_FORC_AttributesMod
use LIS_metforcingMod, only : LIS_FORC_Base_State, LIS_forc
use LIS_logMod,         only : LIS_verify
use cmorph_forcingMod,  only : cmorph_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep.
The arguments are:

n index of the nest

30.2 Fortran: Module Interface coop_forcingMod (Source File: coop_forcingMod.F90)

Contains routines and data structures that are used for the implementation of the station data from various COOP stations. The stations report estimates of meteorological forcing terms, which is spatially interpolated using the inverse distance weighting scheme (IDW).
The implementation in LIS has the derived data type `coop_struc` that includes the variables to specify the runtime options, and the calculation of weights for spatial interpolation.

REVISION HISTORY:

```
13Jul2011: Yuqiong Liu:  Initial Specification

implicit none
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init_COOP      !defines the native resolution of
                                !the input data
```

PUBLIC TYPES:

```
public :: coop_struc
```

30.2.1 init_COOP (Source File: coop_forcingMod.F90)

INTERFACE:

```
subroutine init_COOP(findex)
```

USES:

```
use LIS_coreMod,only : LIS_rc
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep
use LIS_logMod, only : LIS_logunit, LIS_endrun

implicit none
integer, intent(in) :: findex
```

DESCRIPTION:

This routines reads the runtime configurations for using the COOP station data. Using the metadata provided for the stations, this routine invokes the call to compute the interpolation weights to be later used. The routines invoked are:

readcrd_coop (30.2.5)
reads the runtime options specified for COOP station data

LIS_date2time (3.36.24)
converts date to the real time format

compute_stnwts (5.1.22)
computes the weights for spatial interpolation

30.2.2 finalize_coop (Source File: finalize_coop.F90)

REVISION HISTORY:

```
08Jun2011: Yuqiong Liu; Initial Specification
```

INTERFACE:

```
subroutine finalize_coop(findex)
```

USES:

```
use coop_forcingMod, only : coop_struc
```

DESCRIPTION:

Routine to cleanup COOP forcing related memory allocations.

30.2.3 get_coop (Source File: get_coop.F90)

REVISION HISTORY:

```
13 Jul 2011: Yuqiong Liu; Initial Specification
```

INTERFACE:

```
subroutine get_coop(n,findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod, only : LIS_tick
use LIS_logMod, only : LIS_logunit
use LIS_metforcingMod, only : LIS_forc
use coop_forcingMod, only : coop_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates the COOP station data. At the beginning of a simulation, the code reads the most recent past data, and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the COOP data times

read_coop (30.2.4)
Interpolates the appropriate COOP station data to LIS grid

30.2.4 read_coop (Source File: read_coop.F90)

REVISION HISTORY:

```
13 Jul 2011: Yuqiong Liu; Initial Specification, based on LVT COOP obs reader by S. Kumar
```

INTERFACE:

```
subroutine read_coop(n,ftn,findex,order)
```

USES:

```
use LIS_logMod, only         : LIS_logunit, LIS_endrun
use LIS_coreMod, only        : LIS_rc,LIS_domain
use LIS_metforcingMod,  only : LIS_forc
use coop_forcingMod,    only : coop_struc
use map_utils,    only : latlon_to_ij


implicit none
```

ARGUMENTS:

```
integer, intent(in)           :: n
integer, intent(in)           :: ftn
integer, intent(in)           :: findex
integer, intent(in)           :: order
```

DESCRIPTION:

For the given time, reads parameters from the correct COOP station data (ASCII), transforms into LIS forcing parameters and interpolates to the LIS domain.
The arguments are:

order flag indicating which data to be read (order=1, read the previous instance, order=2, read the next instance)

n index of the nest

ftn unit number for the COOP station data

The routines invoked are:

normalize_stnwts (5.3.12)
renormalizes the station weights accounting for missing data

interp_stndata (5.2.8)
spatially interpolates the station data onto the LIS grid.

30.2.5 readcrd_coop (Source File: readcrd_coop.F90)

REVISION HISTORY:

```
13Jul2011; Yuqiong Liu, Initial Specification
```

INTERFACE:

```
subroutine readcrd_coop()
```

USES:

```
use ESMF
use LIS_logMod, only : LIS_logunit
use LIS_coreMod, only : LIS_rc, LIS_config
use coop_forcingMod, only : coop_struc
```

DESCRIPTION:

This routine reads the options specific to COOP station data forcing from the LIS configuration file.

30.2.6 timeinterp_coop (Source File: timeinterp_coop.F90)

REVISION HISTORY:

```
13 Jul 2011: Yuqiong Liu; Initial Specification
```

INTERFACE:

```
subroutine timeinterp_coop(n, findex)
```

USES:

```
use ESMF
use LIS_logMod,  only        : LIS_logunit
use LIS_coreMod,        only : LIS_rc,LIS_domain
use LIS_metforcingMod, only : LIS_FORC_Base_State, LIS_forc
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use coop_forcingMod,    only : coop_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the SCAN forcing data to the model timestep. All variables except precipitation is linearly interpolated.

---

31 ECMWF

The operational, global analysis products from the European Center for Medium-Range Weather Forecasts (ECMWF) is available on a T_L511 triangular truncation, linear reduced gaussian grid for four synoptic hours: 00, 06, 12, and 18 UTC, which are used in LIS. The radiation products from the data assimilation system at the ECMWF is based on the scheme simulated after Morcrette's work [10]. The estimation of shortwave and longwave radiation includes schemes to include effects of adsorption by water vapor and other gases and aerosol scattering based on different parameterizations [13, 9].

31.1 Fortran: Module Interface ecmwf_forcingMod (Source File: ecmwf_forcingMod.F90)

The operational, global analysis products from the European Center for Medium-Range Weather Forecasts (ECMWF) is available on a T_L511 triangular truncation, linear reduced gaussian grid for four synoptic hours: 00, 06, 12, and 18 UTC, which are used in LIS. The radiation products from the data assimilation system at the ECMWF is based on the scheme simulated after Morcrette's work [10]. The estimation of shortwave and longwave radiation includes schemes to include effects of adsorption by water vapor and other gases and aerosol scattering based on different parameterizations [13, 9].
This module contains variables and data structures that are used for the implementation of the ECMWF forcing data. The data is global 0.25 degree dataset in latlon projection, and at 3 hourly intervals. The derived data type `ecmwf_struc` includes the variables that specify the runtime options, and the weights and neighbor information to be used for spatial interpolation. They are described below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

ecmwftime1 The nearest, previous 3 hour instance of the incoming data (as a real time).

ecmwftime2 The nearest, next 3 hour instance of the incoming data (as a real time).

griduptime1 The time to switch Geopotential field

griduptime2 The time to switch Geopotential field

ecmwfdir Directory containing the input data

elevfile File with the elevation definition for the input data

mi Number of points in the input grid

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n11,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n12,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

findtime1, findtime2 boolean flags to indicate which time is to be read for temporal interpolation.

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init_ecmwf      !defines the native resolution of
                                !the input data
```

PUBLIC TYPES:

```
public :: ecmwf_struc
```

31.1.1 init_ecmwf (Source File: ecmwf_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine init_ecmwf(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep

implicit none

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

DESCRIPTION:

Defines the native resolution of the input forcing for ECMWF data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). The routines invoked are:

readcrd_ecmwf (31.1.10)
reads the runtime options specified for ECMWF data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

31.1.2 finalize_ecmwf (Source File: finalize_ecmwf.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine finalize_ecmwf(findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use ecmwf_forcingMod, only : ecmwf_struc
```

DESCRIPTION:

Routine to cleanup allocated structures for ECMWF forcing.

31.1.3 get_ecmwf (Source File: get_ecmwf.F90)

REVISION HISTORY:

```
18 Jun 2003: Urszula Jambor; original code based on getreanlecmwf.F90
20 Feb 2006: Sujay Kumar; Modified with nesting options
```

INTERFACE:

```
subroutine get_ecmwf(n,findex)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_metforcingMod, only : LIS_forc
use LIS_logMod,         only : LIS_logunit
use LIS_timeMgrMod,     only : LIS_get_nstep, LIS_tick
use ecmwf_forcingMod,   only : ecmwf_struc

implicit none

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

DESCRIPTION:

Opens, reads, and interpolates 3-hrly, 1/4 degree ECMWF forcing. At the beginning of a simulation, the code reads the most recent past data (nearest 3 hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
call to advance or retract time

read_ecmwf (31.1.5)
call to read the ECMWF data and perform spatial interpolation

read_ecmwf_elev (31.1.9)
reads the native elevation of the ECMWF data to be used for topographic adjustments to the forcing

---

31.1.4 create_ecmwf_filename (Source File: get_ecmwf.F90)

INTERFACE:

```
subroutine create_ecmwf_filename(dir,avgfilename1, avgfilename2, instfilename,&
     yr,mo,da,hr)
```

USES:

```
use LIS_timeMgrMod, only : LIS_tick

implicit none
```

ARGUMENTS:

```
character(len=*)   :: dir
character(len=*)   :: avgfilename1
character(len=*)   :: avgfilename2
character(len=*)   :: instfilename
integer            :: yr, mo, da, hr
```

DESCRIPTION:

creates the ECMWF filenames

31.1.5 read_ecmwf (Source File: read_ecmwf.F90)

REVISION HISTORY:

```
18 Jun 2003: Urszula Jambor; original code
22 Jun 2011: Hiroko Beaudoing; Updated to GRIB_API routines
             to handle both GRIB1 and GRIB2 data
```

INTERFACE:

```
subroutine read_ecmwf( order, n, findex, &
     avgfile1, avgfile2, instfile, yr, mon, da, hr, ferror )
```

USES:

```
use LIS_coreMod, only        : LIS_rc, LIS_domain
use LIS_logMod, only         : LIS_logunit, LIS_endrun
use LIS_metforcingMod, only : LIS_forc
use ecmwf_forcingMod,   only : ecmwf_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: order     ! lower(1) or upper(2) time interval bdry
integer, intent(in)    :: n         ! lower(1) or upper(2) time interval bdry
integer, intent(in)    :: findex
integer, intent(in)    :: yr,mon,da,hr ! data and hour (multiple of 3)
integer, intent(inout) :: ferror    ! set to zero if there's an error
character(len=*)       :: avgfile1
character(len=*)       :: avgfile2
character(len=*)       :: instfile
```

DESCRIPTION:

For the given time, reads the parameters from 1/4 degree ECMWF data, transforms into 9 LIS forcing parameters and interpolates to the LIS domain.
ECMWF model output variables used to force LIS fall into 2 categories: 1) inst3, Instantaneous values, available every 3 hours

2) 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.
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 LIS 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. LSP accum, large scale precipitation [m]
9. CP accum, convective precipitation [m]

order flag indicating which data to be read (order=1, read the previous 3 hourly instance, order=2, read the next 3 hourly instance)

n index of the nest

yr current year

mon current month

da current day of the year

hr current hour of day

ferror flag to indicate success of the call (=0 indicates success)

The routines invoked are:

retrieve_inst_ecmwfvars (31.1.6)
retrieves the instantaneous variable

retrieve_accum_ecmwfvars (31.1.7)
retrieves the accumulated variable

interp_ecmwf (31.1.8)
spatial interpolation from the ECMWF grid to the LIS grid

31.1.6 retrieve_inst_ecmwfvars (Source File: read_ecmwf.F90)

INTERFACE:

```
subroutine retrieve_inst_ecmwfvars(n, findex, instfile, glbdata, fret)
  use LIS_coreMod,      only : LIS_rc, LIS_domain
  use LIS_logMod,       only : LIS_logunit, LIS_verify
  use ecmwf_forcingMod, only : ecmwf_struc
#if (defined USE_GRIBAPI)
  use grib_api
#endif

  implicit none
```

ARGUMENTS:

```
integer,   intent(in)        :: n
integer,   intent(in)        :: findex
character(len=*), intent(in) :: instfile
real                         :: glbdata(9,LIS_rc%ngrid(n))
integer, intent(inout)       :: fret
```

DESCRIPTION:

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.

31.1.7 retrieve_accum_ecmwfvars (Source File: read_ecmwf.F90)

INTERFACE:

```
subroutine retrieve_accum_ecmwfvars(n, findex, avgfile1, avgfile2, glbdata1, &
                                    hr, fret)
  use LIS_coreMod,      only : LIS_rc, LIS_domain
  use LIS_logMod,       only : LIS_logunit, LIS_verify
  use ecmwf_forcingMod, only : ecmwf_struc
#if (defined USE_GRIBAPI)
  use grib_api
#endif

  implicit none
```

ARGUMENTS:

```
integer,   intent(in)        :: n
integer,   intent(in)        :: findex
character(len=*), intent(in) :: avgfile1
character(len=*), intent(in) :: avgfile2
real                         :: glbdata1(9,LIS_rc%ngrid(n))
integer                      :: hr
integer, intent(inout)       :: fret
```

DESCRIPTION:

This routine opens the corresponding ECMWF data file to extract the specified variable, which represents an accumulated value. Should be used for shortwave, longwave, lsp, and cp.

31.1.8 interp_ecmwf (Source File: read_ecmwf.F90)

INTERFACE:

```
subroutine interp_ecmwf(n, findex, pcp_flag, necmwf,f,lb,lis_gds,nc,nr, &
     varfield)
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use ecmwf_forcingMod,   only : ecmwf_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
logical, intent(in) :: pcp_flag
integer, intent(in) :: necmwf
real, intent(out)   :: f(necmwf)
logical*1           :: lb(necmwf)
real                :: lis_gds(50)
integer, intent(in) :: nc
integer, intent(in) :: nr
real, intent(out)   :: varfield(nc,nr)
```

DESCRIPTION:

This subroutine interpolates a given ECMWF field to the LIS grid. The arguments are:

n index of the nest

necmwf number of elements in the input grid

f input data array to be interpolated

lb input bitmap

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

---

31.1.9 read_ecmwf_elev (Source File: read_ecmwf_elev.F90)

REVISION HISTORY:

```
17Dec2004; Sujay Kumar; Initial Specificaton
```

INTERFACE:

```
subroutine read_ecmwf_elev(n, findex, change)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_metforcingMod, only : LIS_forc
use LIS_fileIOMod,     only : LIS_read_param
use LIS_logMod,         only : LIS_logunit, LIS_endrun
use ecmwf_forcingMod,   only : ecmwf_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates ECMWF model elevation to the LIS grid. The data will be used to perform any topographical adjustments to the forcing.
The arguments are:

n index of the nest

31.1.10 readcrd_ecmwf (Source File: readcrd_ecmwf.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcrd_ecmwf()
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_config
use ecmwf_forcingMod, only : ecmwf_struc
use LIS_logMod, only : LIS_logunit
```

DESCRIPTION:

This routine reads the options specific to ECMWF forcing from the LIS configuration file.

31.1.11 timeinterp_ecmwf (Source File: timeinterp_ecmwf.F90)

REVISION HISTORY:

```
 5 Nov 2001: Urszula Jambor; Initial Specification
10 Oct 2006: Sujay Kumar: Switched to using ESMF_State for storing
             forcing data.
```

INTERFACE:

```
subroutine timeinterp_ecmwf(n, findex)
```

USES:

```
use ESMF
use LIS_coreMod,            only : LIS_rc, LIS_domain
use LIS_FORC_AttributesMod
use LIS_metforcingMod,      only : LIS_forc, LIS_FORC_Base_State
use LIS_constantsMod,       only : LIS_CONST_SOLAR
use LIS_timeMgrMod,         only : LIS_time2date, LIS_tick
use LIS_logMod,             only : LIS_logunit, LIS_verify
use ecmwf_forcingMod,       only : ecmwf_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous 3 hourly value is used. All other variables are linearly interpolated between the 3 hourly blocks.
The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

LIS_tick (3.36.27)
advances or retracts time by the specified amount

zterp (5.3.19)
zenith-angle based interpolation

---

32 ECMWF Reanalysis

This section describes the implementation of the bias-corrected atmospheric reanalysis data developed by Aaron Berg. The data is global 0.5 degree in latlon projection.

32.1 Fortran: Module Interface ecmwfreanal_forcingMod (Source File: ecmwfreanal_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the bias-corrected ECMWF atmospheric reanalysis data (Berg et al. 2003). The data is global 0.5 degree dataset in latlon projection, and at 6 hourly intervals. The derived data type `ecmwfreanal_struc` includes the variables that specify the runtime options, and the weights and neighbor information to be used for spatial interpolation. They are described below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

fmodeltime1 The nearest, previous 6 hour instance of the incoming data (as a real time).

fmodeltime2 The nearest, next 6 hour instance of the incoming data (as a real time).

remask1d The data used to remask the input data to the LIS mask.

ecmwfreanaldir Directory containing the input data

emaskfile File containing the 0.5 deg land-sea mask used in the input data.

elevfile File with the elevation definition for the input data.

mi Number of points in the input grid

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n11,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n12,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

findtime1, findtime2 boolean flags to indicate which time is to be read for temporal interpolation.

Berg, A.A., J.S. Famiglietti, J.P. Walker, and P.R. Houser, 2003: Impact of bias correction to reanalysis products on simulations of North American soil moisture and hydrological fluxes. Journal of Geophysical Research, 108, 4490, DOI: 10.1029/2002JD003334.

USES:

```
implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: init_ECMWFREANAL      !defines the native resolution of
                                !the input data
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: ecmwfreanal_struc
```

32.1.1 init_ECMWFREANAL (Source File: ecmwfreanal_forcingMod.F90)

REVISION HISTORY:

```
26Jan2004: Sujay Kumar; Init_ial Specification
```

INTERFACE:

```
subroutine init_ECMWFREANAL(findex)
```

USES:

```
    use LIS_coreMod, only : LIS_rc
    use LIS_timeMgrMod, only : LIS_update_timestep

    implicit none
  !AGRUMENTS:
    integer, intent(in) :: findex
```

DESCRIPTION:

Defines the native resolution of the input forcing for ECMWFREANAL data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). The routines invoked are:

readcrd_ecmwfreanal (32.1.9)
reads the runtime options specified for ECMWFREANAL data

readmask_ecmwfreanal (32.1.10)
reads the 0.5 degree land sea mask

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

read_ecmwfreanal_elev (32.1.8)
reads the native elevation of the ecmwfreanal data to be used for topographic adjustments to the forcing

---

32.1.2 finalize_ecmwfreanal (Source File: finalize_ecmwfreanal.F90)

REVISION HISTORY:

```
25 Oct 2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine finalize_ecmwfreanal(findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use ecmwfreanal_forcingMod, only : ecmwfreanal_struc
```

DESCRIPTION:

Routine to cleanup allocated structures for ecmwf reanalysis forcing.

32.1.3 geogfill2_ecmwfreanal (Source File: geogfill2_ecmwfreanal.F90)

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_ecmwfreanal(n, nc,nr,geogmask,fdata,v,tmask)
```

USES:

```
use LIS_coreMod, only : LIS_domain

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer             :: nc
integer             :: nr
logical*1           :: geogmask(nc,nr)
real                :: fdata(nc,nr)
integer             :: v
logical*1           :: tmask(nc,nr)
```

DESCRIPTION:

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.
The arguments are:

n index of the nest

nc number of columns of the grid (along the east-west dimension)

nr number of rows of the grid (along the north-south dimension)

geogmask interpolated forcing bitmap

fdata interpolated forcing

v index of the variable

tmask valid temperature points (used as the land mask)

32.1.4 get_ecmwfreanal (Source File: get_ecmwfreanal.F90)

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 the scheme in LIS
```

INTERFACE:

```
subroutine get_ecmwfreanal(n,findex)
```

USES:

```
use LIS_coreMod,            only : LIS_rc
use LIS_metforcingMod,      only : LIS_forc
use LIS_logMod,             only : LIS_logunit
use LIS_timeMgrMod,         only : LIS_get_nstep, LIS_tick
use ecmwfreanal_forcingMod, only : ecmwfreanal_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 6-hrly, 1/2 degree Reanalysis ECMWF forcing. At the beginning of a simulation, the code reads the most recent past data (nearest 6 hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
call to advance or retract time

read_ecmwfreanal (32.1.5)
call to read the ECMWF Reanalysis data and perform spatial interpolation

---

32.1.5 read_ecmwfreanal (Source File: read_ecmwfreanal.F90)

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_ECMWFREANAL.
24 Nov 2003; Sujay Kumar; Included in LIS
15 Mar 2004; Matt Rodell; Fix test for beginning of restart run
```

INTERFACE:

```
subroutine read_ecmwfreanal(order, n, findex, yr, mon, da, hr, ferror)
  USES:
  use LIS_coreMod,        only : LIS_rc,LIS_domain, LIS_localPet, LIS_masterproc
  use LIS_metforcingMod, only : LIS_forc
  use LIS_logMod,         only : LIS_logunit, LIS_endrun
  use ecmwfreanal_forcingMod, only : ecmwfreanal_struc


  implicit none
```

ARGUMENTS:

```
  integer, intent(in)    :: order
  integer, intent(in)    :: n
  integer, intent(in)    :: findex
  integer, intent(in)    :: yr
  integer, intent(in)    :: mon
  integer, intent(in)    :: da
  integer, intent(in)    :: hr
  integer, intent(inout) :: ferror
```

DESCRIPTION:

For the given time, reads 8 parameters from 1/2 degree Reanalysis ECMWF data, transforms into 9 LIS forcing parameters and interpolates to the LIS 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/m2s], accumulation
4. STRD Downward longwave radiation, surface [W/m2s], 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

order flag indicating which data to be read (order=1, read the previous 6 hourly instance, order=2, read the next 6 hourly instance)

n index of the nest

yr current year

mon current month

da current day of the year

hr current hour of day

ferror flag to indicate success of the call (=0 indicates success)

The routines invoked are:

ecmwfreanalgrid_2_lisgrid (32.1.6)
transform the ECMWF REANALYSIS data to the LIS grid

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

geogfill2_ecmwfreanal (32.1.3)
fill in any grid points due to mismatched grids.

32.1.6 ecmwfreanalgrid_2_lisgrid (Source File: read_ecmwfreanal.F90)

REVISION HISTORY:

```
    10 Apr 2002: Urszula Jambor;  Code adapted from
                 ecmwfgrid_2_grid2catgrid, by R. Reichle
```

INTERFACE:

```
 subroutine ecmwfreanalgrid_2_lisgrid( nx, ny, grid_data )

   implicit none
```

ARGUMENTS:

```
   integer, intent(in)                   :: nx, ny
   real, intent(inout), dimension(nx,ny) :: grid_data
```

DESCRIPTION:

Changes grid data from ECMWF data convention to LIS 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
```

LIS: South-to-North around Date Line Full global grid. Starts at the southernmost latitude and date line, going east and then north.

32.1.7 fillgaps (Source File: read_ecmwfreanal.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 )
```

32.1.8 read_ecmwfreanal_elev (Source File: read_ecmwfreanal_elev.F90)

REVISION HISTORY:

```
17Dec2004; Sujay Kumar; Initial Specificaton
3  Dec 2007; Sujay Kumar; Added the abstract method to read the
             EDAS data.
```

INTERFACE:

```
subroutine read_ecmwfreanal_elev(n,findex)
```

USES:

```
use LIS_coreMod,            only : LIS_rc, LIS_domain
use LIS_metforcingMod,     only : LIS_forc
use LIS_logMod,             only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,          only : LIS_read_param
use ecmwfreanal_forcingMod, only : ecmwfreanal_struc
```

32.1.9 readcrd_ecmwfreanal (Source File: readcrd_ecmwfreanal.F90)

REVISION HISTORY:

```
26Jan2004; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcrd_ecmwfreanal()
```

USES:

```
use ESMF
use LIS_logMod, only : LIS_logunit
use LIS_coreMod, only : LIS_rc, LIS_config
use ecmwfreanal_forcingMod, only : ecmwfreanal_struc
```

DESCRIPTION:

This routine reads the options specific to ECMWF reanalysis forcing from the LIS configuration file.

32.1.10 readmask_ecmwfreanal (Source File: readmask_ecmwfreanal.F90)

REVISION HISTORY:

```
16 Apr 2002: Urszula Jambor; Initial Code
24 Nov 2003: Sujay Kumar; Included in LIS
```

INTERFACE:

```
subroutine readmask_ecmwfreanal(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc  ! LIS non-model-specific 1-D variables
use LIS_logMod, only : LIS_logunit
use ecmwfreanal_forcingMod, only : ecmwfreanal_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Reads in land-sea mask for 0.5 degree Reanalysis ECMWF data set, changes grid from ECMWF convention to LIS convention by calling `ecmwfreanalgrid_2_lisgrid`, and transforms grid array into 1D vector for later use.
The arguments are:

n index of the nest

The routines invoked are:

ecmwfreanalgrid_2_lisgrid (32.1.6)
transform the ECMWF Reanalysis data to the LIS grid

32.1.11 timeinterp_ecmwfreanal (Source File: timeinterp_ecmwfreanal.F90)

REVISION HISTORY:

```
24 Nov 2003: Sujay Kumar; Initial version of code adapted from the GLDAS
                          version
 10 Oct 2006: Sujay Kumar: Switched to using ESMF_State for storing
              forcing data.
```

INTERFACE:

```
subroutine timeinterp_ecmwfreanal(n, findex)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_FORC_AttributesMod
use LIS_constantsMod, only  : LIS_CONST_SOLAR
use LIS_metforcingMod, only : LIS_forc, LIS_FORC_Base_State
use LIS_timeMgrMod,     only : LIS_time2date
use ecmwfreanal_forcingMod, only : ecmwfreanal_struc
use LIS_logMod, only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous 6 hourly value is used. All other variables are linearly interpolated between the 6 hourly blocks.
The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

zterp (5.3.19)
zenith-angle based interpolation

---

33 GDAS

The Global Data Assimilation System (GDAS) is the global, operational weather forecast model developed at the Environmental Modeling Center (EMC) of NOAA/NCEP [6]. The radiation algorithms in GDAS include parameterizations for SW↓ and LW↓ radiation interactions between clouds and radiation. The shortwave radiation parameterization follows the the scheme developed by [3] and the longwave radiation scheme is based on the work of [14]. GDAS forcing variables are produced on a quadratic T170 gaussian grid. LIS uses the 00, 03, and as needed, the 06 (hours after current) forecasts, which are produced at 6 hour intervals. From October 2002 onwards, the GDAS outputs are generated at a higher resolution T254 gaussian grid.

33.0.12 create_gdasf9_filename (Source File: create_gdasf9_filename.F90)

INTERFACE:

```
subroutine create_gdasf9_filename(option, name00, name03, &
                                  gdasdir, yr, mo, da, hr, status )
```

USES:

```
use LIS_timeMgrMod, only : LIS_tick

implicit none
```

ARGUMENTS:

```
integer,          intent(in)    :: option
character(len=*), intent(in)    :: gdasdir
integer, intent(in)             :: yr, mo, da, hr
character(len=*), intent (out)  :: name00
character(len=*), intent (out)  :: name03
integer                         :: status
```

DESCRIPTION:

This subroutine puts together 9hr forecast GDAS file name
First read the notes in create_gdasfilename.F90 (33.0.13).
This routine is used to create the GDAS file name for the backup 9hr forecast file. At analysis hours (hr = 00, 06, 12, and 18), LIS reads the hr.f03 GDAS file for both the new instantaneous and the new averaged forcing values (here, name00 == name03). Should this file be missing, the 9-hour forecast of the previous analysis hour contains comparable forcing values.

An hr.f09 file contains instantaneous forcing values valid at hour hr+9, and it contains 3-hourly averaged forcing values valid for [hr+6, hr+9]
For example, when LIS is at hour 06, it reads ahead to get instantaneous forcing values at hour 09 and to get averaged forcing data for the period [06, 09]. The GDAS file 06.f03 contains these data. Should this file be missing, the GDAS file 00.f09 also contains instantaneous forcing values at hour 09 and averged forcing values for the period [06, 09].
Note that backup 9-hour forecast files exist only for anaylsis hours. Should a name00 or name03 file be missing when LIS is at a non-analysis hour (hr = 03, 09, 15, or 21), then there is no previous hour with a 9-hour forecast that contains comparable forcing values. Another strategy must be developed to replace those missing files.
Note that the logic for reading and creating names for bookend1, which is only done at initialization, is the same as described above except imagine that you are at start hour - 3 reading ahead to the start hour.
Note that the case where LIS is reading name00, name03, and name06 data to read ahead is not handled by this routine. Example, consider the case when reading ahead to hour 18. Here LIS is at hour 15, and LIS needs 18f00 for instantaneous values at hour 18, and LIS needs 12f03 and 12f06 for averaged values for the period [15, 18]. 12f03 contains averaged values for [12, 15]. If it is missing, LIS could use 06f09 to obtain averaged values for [12, 15]. Again, this scenario is not handled by this routine. LIS will simply roll back to the the previous day to look for data.
The arguments are:

option bookend option (1 or 2)

name00 name of the timestamped 9hr GDAS file

name03 name of the timestamped 9hr GDAS file

gdasdir Name of the GDAS directory

yr year

mo month

da day of month

hr hour of day

status flag indicating if the 9hr forecast is available.

33.0.13 create_gdasfilename (Source File: create_gdasfilename.F90)

INTERFACE:

```
subroutine create_gdasfilename(option, name00, name03, name06, &
     F06flag, gdasdir, yr, mo, da, hr )
```

USES:

```
use LIS_timeMgrMod, only : LIS_tick

implicit none
```

ARGUMENTS:

```
integer,          intent(in)    :: option
character(len=*), intent(in)    :: gdasdir
integer, intent(in)             :: yr, mo, da, hr
character(len=*), intent (out)  :: name00
character(len=*), intent (out)  :: name03
character(len=*), intent (out)  :: name06
logical                         :: F06flag
```

DESCRIPTION:

This subroutine puts together GDAS file names for different (0,3,6hr) forecast instances
LIS uses a mix of instantaneous forcing values and averaged forcing values from the GDAS forcing data. LIS processes the GDAS forcing at a 3-hourly period. When it is time to read new GDAS forcing data, LIS reads 3 hours ahead. So when LIS' clock is at hour hr, LIS processes data for the period]hr, hr+3]. Here, LIS reads instantaneous forcing valid at hour hr+3 and averaged forcing valid for [hr, hr+3]
GDAS files "named" 00.f00, 06.f00, 12.f00, and 18.f00 are the analysis files, and they contain only instantaneous forcing values at hours 00, 06, 12, and 18 respectively. The files hr.f03, hr.f06, and hr.f09 are, respectively, 3-hour, 6-hour, and 9-hour forecasts from the analysis hour hr, where hr is either 00, 06, 12, or 18.
The hr.f03 files contain instantaneous forcing values at hour hr+3, and they contain forcing values averaged over the 3-hour period [hr, hr+3].
The hr.f06 files contain instantaneous forcing values at hour hr+6, and they contain forcing values averaged over the 6-hour period [hr, hr+6].
The hr.f09 files contain instantaneous forcing values at hour hr+9, and they contain forcing values averaged over the 3-hour period [hr+6, hr+9].
Again, LIS processes the GDAS forcing at a 3-hourly period. When the hour of LIS' clock, hr, is one of the analysis hours (00, 06, 12, 18) then LIS reads ahead to extract instantaneous forcing values from the hr.f03 file, and it reads ahead to extract averaged forcing values from the hr.f03 file.
When the hour of LIS' clock, hr, is not one of the analysis hours (in this case when hr is 03, 09, 15, or 21), then LIS reads instantaneous forcing values from the from the next analysis hour. E.g., when hr is 03, read from 06.f00; when hr is 09, read from 12.f00. LIS computes averaged forcing values by reading the averaged values from the 3-hour and 6-hour forecasts of the previous analysis hour. E.g., when hr is 03, read from 00.f03 and 00.f06; when hr is 09, read from 06.f03 and 06.f06. For this case, the averaged forcing values are computed by subtracting the 3-hour forecast from the 6-hour forecast both of the previous analysis hour thusly:
Consider LIS' hour to be 15. LIS needs averaged forcing valid for the period [15, 18]. The previous analysis hour is then 12. We want the averaged forcing values from 12.f03 and from 12.f06.
hr.f06 contains averaged forcing, denoted f_6, for the period [hr, hr+6] and hr.f03 contains averaged forcing, denoted f_3, for the period [hr, hr+3]. Thus for forcing, f_6, $6*f_6$ is the total f at hour hr+6 and $3*f_3$ is the total f at hour hr+3. $(6*f_6-3*f_3)$ represents the total f from hr+3 to hr+6, so $(6*f_6-3*f_3)/3 \equiv 2*f_6-f_3$ is the averaged f for the period [hr+3, hr+6].
So to compute the averaged forcing for period [15, 15+3], LIS must read 12.f03 and 12.f06. f_6 represents averaged f for [12, 18]; f_3 represents averaged f for [12, 15]; $2*f_6-f_3$ represents avergage f for [15, 18], the desired period.
Note that the variable, F06flag, is used to indicate the above situation when LIS must subtract f_3 values from f_6 values to obtain the desired averaged forcing values.
Note that the logic for reading and creating names for bookend1, which is only done at initialization, is the same as described above except imagine that you are at start hour - 3 reading ahead to the start hour.
The arguments are:

option bookend option (1 or 2)

gdasdir Name of the GDAS directory

yr year

mo month

da day of month

hr hour of day

name00 name of the GDAS file for reading instantaneous fields

name03 name of the 3hr GDAS file for reading time averaged fields

name06 name of the 6hr GDAS file for reading time averaged fields

F06flag flag to indicate if 6hr forecast data is required for this interval

---

33.1 Fortran: Module Interface finalize_gdas (Source File: finalize_gdas.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine finalize_gdas(findex)
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use gdas_forcingMod
```

DESCRIPTION:

Routine to cleanup allocated structures for GDAS forcing.
The arguments are:

findex index of the forcing

33.2 Fortran: Module Interface gdas_forcingMod (Source File: gdas_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the forcing data from the Global Data Assimilation System (GDAS) developed at the Environmental Modeling Center (EMC) of NOAA/NCEP. GDAS forcing variables are produced on a quadratic gaussian grid. LIS uses the 00, 03, 06 and as needed, the 09 forecasts. The forecasts are produced at 6 hr intervals. The resolution of GDAS forcing varies as follows:
upto 2000/1/24 : T126 (384x190) grid
2001/1/24 - 2002/10/29 : T170 (512x256) grid
2002/10/29 - 2005/5/31 : T254 (768x384) grid
2005/5/31 - 2010/7/27 : T382 (1152x576) grid
2010/7/28 onwards : T574 (1760x880) grid

The implementation in LIS has the derived data type `gdas_struc` that includes the variables that specify the runtime options, and the weights and neighbor information to be used for spatial interpolation. They are described below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

nmif Number of forcing variables in the GDAS data

gdastime1 The nearest, previous 3 hour instance of the incoming data (as a real time).

gdastime2 The nearest, next 3 hour instance of the incoming data (as a real time).

griduptime1 The time to switch GDAS resolution to T126

griduptime2 The time to switch GDAS resolution to T170

griduptime3 The time to switch GDAS resolution to T254

griduptime4 The time to switch GDAS resolution to T382

griduptime5 The time to switch GDAS resolution to T574

findtime1, findtime2 boolean flags to indicate which time is to be read for temporal interpolation.

gdasdir Directory containing the input data

elevfile File with the elevation definition for the input data.

mi Number of points in the input grid

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n11,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n112,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

USES:

```
implicit none

PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
public :: init_GDAS      !defines the native resolution of
                                !the input data
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
public :: gdas_struc
```

33.2.1 init_GDAS (Source File: gdas_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine init_GDAS(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_logMod,     only : LIS_logunit, LIS_endrun
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Defines the native resolution of the input forcing for GDAS data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). Based on the GDAS data map projection and resolution, this routine sets up the spatial interpolation weights. The dates of the GDAS resolution switches are also defined in this routine.
The arguments are:

findex index of the forcing source

The routines invoked are:

readcrd_gdas (33.2.7)
reads the runtime options specified for GDAS data

LIS_date2time (3.36.24)
converts date to the real time format

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

33.2.2 get_gdas (Source File: get_gdas.F90)

REVISION HISTORY:

```
08 Dec 2000: Urszula Jambor; Rewrote geteta.f in fortran90 to
             use GDAS in GLDAS
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)
11 Dec 2003: Sujay Kumar; Specification of elevation correction options
16 Mar 2008: Sujay Kumar; Enabled the capability to read the 6 and 9hr
             forecasts as backups
29 Apr 2010: Sujay Kumar: Fixed the problems in mixing the instantaneous
             and time averaged GDAS forecast files.
```

INTERFACE:

```
subroutine get_gdas(n, findex)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_metforcingMod,  only : LIS_forc
use LIS_timeMgrMod,     only : LIS_tick, LIS_get_nstep
use LIS_logMod,         only : LIS_logunit, LIS_endrun
use gdas_forcingMod,    only : gdas_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 3-hrly, GDAS forcing. The idea is to open either the 00, 03 and 06 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. At the beginning of a simulation, the code reads the most recent past data (nearest 3 hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. The strategy for missing data is to first look for 9hr forecasts and if not go backwards up to 10 days to get forcing at the same time of day.
See additional notes in create_gdasfilename(33.0.13).
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the GDAS data times

create_gdasfilename (33.0.13)
Puts together appropriate file name for either 3 or 6 hour intervals

read_gdas (33.2.3)
Interpolates GDAS data to LIS grid

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

read_gdas_elev (33.2.6)
reads the native elevation of the GDAS data to be used for topographic adjustments to the forcing

33.2.3 read_gdas (Source File: read_gdas.F90)

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
    29 Apr 2010: Sujay Kumar: Fixed the mixing of instantaneous and time
                 averaged fields.
    25 Jan 2012: Sujay Kumar; Switched to the use of grib-api library
```

INTERFACE:

```
 subroutine read_gdas( order, n, findex, &
      name00, name03, name06, F06flag, ferror,try )
```

USES:

```
   use LIS_coreMod,        only : LIS_rc, LIS_domain
   use LIS_timeMgrMod,     only : LIS_get_nstep, LIS_date2time
   use LIS_metforcingMod,  only : LIS_forc
   use gdas_forcingMod,    only : gdas_struc
   use LIS_logMod,         only : LIS_logunit

   implicit none
```

ARGUMENTS:

```
   integer, intent(in)          :: order
   integer, intent(in)          :: n
   integer, intent(in)          :: findex
   character(len=*), intent(in) :: name00
   character(len=*), intent(in) :: name03
   character(len=*), intent(in) :: name06
   logical, intent(in)          :: F06flag
   integer, intent(out)         :: ferror
   integer, intent(inout)       :: try
```

DESCRIPTION:

For the given time, reads parameters from GDAS forecast datasets, transforms into 9 LIS forcing parameters and interpolates to the LIS domain.
The arguments are:

order flag indicating which data to be read (order=1, read for the previous 3hr bookend, order=2, read for the next 3 hr bookend)

n index of the nest

name00 name of the instantaneous forecast file

name03 name of the 3 hour GDAS forecast file

name06 name of the 6 hour GDAS forecast file

F06flag flag to indicate if 6hr forecast data is required for this interval

ferror flag to indicate success of the call (=0 indicates success)

try index of the tries (in case of missing data)

The routines invoked are:

interp_gdas (33.2.5)
spatially interpolates a GDAS variable

33.2.4 retrieve_gdas_variables (Source File: read_gdas.F90)

INTERFACE:

```
subroutine retrieve_gdas_variables(n, findex, fname, glbdata, errorcode)
```

USES:

```
  use LIS_coreMod,        only : LIS_rc, LIS_domain
  use LIS_logMod,         only : LIS_logunit,LIS_getNextUnitNumber,&
       LIS_releaseUnitNumber, LIS_verify, LIS_warning
  use gdas_forcingMod,    only : gdas_struc

#if (defined USE_GRIBAPI)
  use grib_api
#endif

  implicit none
```

ARGUMENTS:

```
  integer               :: n
  integer               :: findex
  character(len=*)      :: fname
  real                  :: glbdata(10,LIS_rc%ngrid(n))
  integer               :: errorcode
```

DESCRIPTION:

This subroutine retrieves GDAS forcing variables, and interpolates them to the LIS grid.

33.2.5 interp_gdas (Source File: read_gdas.F90)

INTERFACE:

```
subroutine interp_gdas(n, findex, pcp_flag, ngdas,f,lb,lis_gds,nc,nr, &
     varfield)
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use gdas_forcingMod,   only : gdas_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
logical, intent(in) :: pcp_flag
integer, intent(in) :: ngdas
real, intent(out)   :: f(ngdas)
logical*1           :: lb(ngdas)
real                :: lis_gds(50)
integer, intent(in) :: nc
integer, intent(in) :: nr
real, intent(out)   :: varfield(nc,nr)
```

DESCRIPTION:

This subroutine interpolates a given GDAS field to the LIS grid. The arguments are:

n index of the nest

ngdas number of elements in the input grid

f input data array to be interpolated

lb input bitmap

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

---

33.2.6 read_gdas_elev (Source File: read_gdas_elev.F90)

REVISION HISTORY:

```
17Dec2004; Sujay Kumar; Initial Specificaton
```

INTERFACE:

```
subroutine read_gdas_elev(n, findex, change)
```

USES:

```
use LIS_coreMod,       only : LIS_rc, LIS_domain
use LIS_metforcingMod, only : LIS_forc
use LIS_fileIOMod,     only : LIS_read_param
use LIS_logMod,        only : LIS_logunit, LIS_getNextUnitNumber, &
                              LIS_releaseUnitNumber, LIS_endrun
use gdas_forcingMod,   only : gdas_struc
 use gaussian_mod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates GDAS model elevation to the LIS grid. The data will be used to perform any topographical adjustments to the forcing.
The arguments are:

n index of the nest

The routines invoked are:

LIS_readData (3.14.1)
Abstract method to read the elevation of the forcing data in the same map projection used in LIS.

33.2.7 readcrd_gdas (Source File: readcrd_gdas.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcrd_gdas()
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc, LIS_config
use LIS_logMod
use gdas_forcingMod, only: gdas_struc
```

DESCRIPTION:

This routine reads the options specific to GDAS forcing from the LIS configuration file.

33.3 Fortran: Module Interface reset_gdas (Source File: reset_gdas.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine reset_gdas
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use LIS_timeMgrMod, only : LIS_date2time
use gdas_forcingMod
```

DESCRIPTION:

Routine to cleanup allocated structures for GDAS forcing.

33.3.1 timeinterp_gdas (Source File: timeinterp_gdas.F90)

REVISION HISTORY:

```
 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.
 10 Oct 2006: Sujay Kumar: Switched to using ESMF_State for storing
              forcing data.
 25 Jan 2012: Sujay Kumar; added the tweaks to enable ensemble forcings
```

INTERFACE:

```
subroutine timeinterp_gdas(n, findex)
```

USES:

```
use ESMF
use LIS_coreMod,        only  : LIS_rc, LIS_domain
use LIS_FORC_AttributesMod
use LIS_timeMgrMod,     only  : LIS_time2date, LIS_tick
use LIS_constantsMod,   only  : LIS_CONST_SOLAR
use LIS_metforcingMod, only  : LIS_forc, LIS_FORC_Base_State
use LIS_logMod,         only  : LIS_logunit, LIS_verify, LIS_endrun
use gdas_forcingMod,    only  : gdas_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous 3 hourly value is used. All other variables are linearly interpolated between the 3 hourly blocks.
If the ensemble of forcing option is specified, then the code picks out the ensemble member (in each subgrid tile) to assign the forcing.
The arguments are:

n index of the nest

The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

LIS_tick (3.36.27)
advances or retracts time by the specified amount

zterp (5.3.19)
zenith-angle based interpolation

---

34 GDAS LSWG forcing

This section describes the implementation of the GDAS LSWG data.

34.1 Fortran: Module Interface finalize_gdasLSWG (Source File: finalize_gdasLSWG.F90)

REVISION HISTORY:

```
17 March 2009; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine finalize_gdasLSWG
```

USES:

DESCRIPTION:

Routine to cleanup allocated structures for GDAS forcing.

34.2 Fortran: Module Interface gdasLSWG_forcingMod (Source File: gdasLSWG_forcingMo

This module contains variables and subroutines used for the implementation of the forcing data generated from the GDAS output for the LSWG activity.

USES:

```
use ESMF

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
 ---------------------------------------------------------------------------
  public :: init_GDASLSWG      !defines the native resolution of
                                  !the input data
 ---------------------------------------------------------------------------
```

PUBLIC TYPES:

```
 ---------------------------------------------------------------------------
  public :: gdasLSWG_struc
```

34.2.1 init_gdasLSWG (Source File: gdasLSWG_forcingMod.F90)

REVISION HISTORY:

```
30 APR 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine init_gdasLSWG(findex)
```

USES:

```
  use LIS_coreMod,    only : LIS_rc
  use LIS_timeMgrMod, only : LIS_calendar, LIS_date2time, LIS_update_timestep
  use LIS_logMod,     only : LIS_logunit, LIS_endrun, LIS_verify

  implicit none

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

DESCRIPTION:

This routine performs required initializations, including reading the configurable options, allocating memory and initializing data structures and setting up interpolation weights.

34.2.2 get_gdasLSWG (Source File: get_gdasLSWG.F90)

REVISION HISTORY:

```
 20 Oct 2009; Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine get_gdasLSWG(n, findex, metdata1, metdata2)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc
use LIS_metforcingMod, only : LIS_forc
use LIS_timeMgrMod,     only : LIS_calendar, LIS_get_nstep, LIS_tick
use LIS_logMod,         only : LIS_logunit, LIS_endrun, LIS_verify
use gdasLSWG_forcingMod,  only : gdasLSWG_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: metdata1(LIS_rc%met_nf(findex), LIS_rc%ngrid(n))
real                :: metdata2(LIS_rc%met_nf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

This routine issues the calls to open and read the GDAS (LSWG) data. The data interval is assumed to be 6 hours.

34.2.3 retgdasLSWG (Source File: read_gdasLSWG.F90)

REVISION HISTORY:

```
20 Oct 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine retgdasLSWG(n, order, m, metdata)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_timeMgrMod, only : LIS_calendar, LIS_doy2date
use LIS_logMod,     only : LIS_logunit, LIS_endrun, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_verify
use gdasLSWG_forcingMod, only : gdasLSWG_struc

implicit none
```

ARGUMENTS:

```
integer,   intent(in) :: n
integer,   intent(in) :: order
integer,   intent(in) :: m
real                  :: metdata(LIS_rc%met_nf(m), LIS_rc%ngrid(n))
```

DESCRIPTION:

This subroutine reads the GDAS (ASCII) data specied for the LSWG work. At the start of the simulation, the code reads the entire dataset into memory and then indexes to the right location at other times. The code also performs spatial interpolation to the LIS grid using a neighor search approach.
The data has 26 levels - We assume that there are 25 layers of data. The rh data is only specified for 21 levels. The remaining levels are assumed to be zero.

34.2.4 readcrd_gdasLSWG (Source File: readcrd_gdasLSWG.F90)

REVISION HISTORY:

```
20 Oct 2009; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcrd_gdasLSWG()
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc, LIS_config
use LIS_logMod,    only : LIS_logunit, LIS_verify
use gdasLSWG_forcingMod, only: gdasLSWG_struc
```

DESCRIPTION:

This routine reads the options specific to GDASLSWG forcing from the LIS configuration file.

34.2.5 timeinterp_gdasLSWG (Source File: timeinterp_gdasLSWG.F90)

REVISION HISTORY:

```
20 Oct 2009: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine timeinterp_gdasLSWG(n, findex)
```

USES:

```
  use ESMF
  use LIS_coreMod,    only : LIS_rc, LIS_domain
  use LIS_logMod,     only : LIS_verify
  use LIS_FORC_AttributesMod
  use LIS_metforcingMod, only : LIS_FORC_Base_State
  use gdasLSWG_forcingMod,  only : gdasLSWG_struc
#if (defined RTMS)
  USE units_conversion
  USE Type_Kinds, ONLY: fp
#endif

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: n
  integer, intent(in) :: findex
  real                :: metdata1(LIS_rc%met_nf(findex), LIS_rc%ngrid(n))
  real                :: metdata2(LIS_rc%met_nf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

This routine performs the temporal interpolation of the GDAS data to the LIS model timestep

---

35 GEOS

This section describes the implementation of the forcing data available from the Goddard Earth Observing System (GEOS) data assimilation system produced by the NASA GSFC's Global Modeling and Assimilation office (GMAO).

35.0.6 finalize_geos (Source File: finalize_geos.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine finalize_geos(findex)
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use geos_forcingMod, only : geos_struc
```

DESCRIPTION:

Routine to cleanup GEOS forcing related memory allocations.

35.1 Fortran: Module Interface geos_forcingMod (Source File: geos_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the GEOS forcing data. The data is global 1 degree dataset in latlon projection, and at 3 hourly intervals. The derived data type `geos_struc` includes the variables that specify the runtime options, and the weights and neighbor information to be used for spatial interpolation. They are described below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

nmif Number of forcing variables in the ECMWF data

geostime1 The nearest, previous 3 hour instance of the incoming data (as a real time).

geostime2 The nearest, next 3 hour instance of the incoming data (as a real time).

griduptime The time to switch GEOS resolution

geosdir Directory containing the input data

mi Number of points in the input grid

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n111,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n122,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

findtime1, findtime2 boolean flags to indicate which time is to be read for temporal interpolation.

USES:

```
implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: init_geos      !defines the native resolution of
                                !the input data
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: geos_struc
```

35.1.1 init_geos (Source File: geos_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine init_geos(findex)
```

USES:

```
use LIS_coreMod,    only:  LIS_rc
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Defines the native resolution of the input forcing for GEOS data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V).
The routines invoked are:

readcrd_geos (35.1.5)
reads the runtime options specified for GEOS data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

35.1.2 get_geos (Source File: get_geos.F90)

REVISION HISTORY:

```
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
 5 Nov 2001: Urszula Jambor; Reset tiny negative SW values to zero.
10 Feb 2003: Sujay Kumar: Initial Specification in LIS.
 2 Dec 2008: Yudong Tian, added support for GEOS5 from 00Z2MAR2007
```

INTERFACE:

```
subroutine get_geos(n, findex)
```

USES:

```
use LIS_coreMod,        only  : LIS_rc
use LIS_timeMgrMod,     only  : LIS_get_nstep, LIS_tick
use LIS_logMod,         only  : LIS_logunit, LIS_endrun
use LIS_metforcingMod, only  : LIS_forc
use geos_forcingMod,    only  : geos_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 3-hrly, 1 degree GEOS forcing. At the beginning of a simulation, the code reads the most recent past data (nearest 3 hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
call to advance or retract time

geosfile (35.1.3)
Puts together appropriate file name for 3 hour intervals

read_geos (35.1.4)
call to read the GEOS data and perform spatial interpolation

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

35.1.3 geosfile (Source File: get_geos.F90)

INTERFACE:

```
subroutine geosfile(name,geosdir,yr,mo,da,hr,ncold)

  implicit none
```

ARGUMENTS:

```
  character(len=*), intent(in)  :: geosdir
  integer, intent(in)           :: yr,mo,da,hr,ncold
  character(len=*), intent(out) :: name
```

DESCRIPTION:

This subroutine puts together GEOS file name for 3 hour file intervals.
The arguments are:

geosdir Name of the GEOS directory

yr year

mo month

da day of month

hr hour of day

ncold Number of columns (used as to determine the GEOS resolution)

name name of the timestamped GEOS file

35.1.4 read_geos (Source File: read_geos.F90)

REVISION HISTORY:

```
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
10 Feb 2003: Sujay Kumar: Initial Specification in LIS.
```

INTERFACE:

```
subroutine read_geos(order,n, findex, &
     name,tscount,ferror)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, LIS_masterproc
use LIS_logMod,only : LIS_logunit, LIS_endrun, LIS_verify, &
                      LIS_getNextUnitNumber, LIS_releaseUnitNumber
use LIS_metforcingMod, only: LIS_forc
use geos_forcingMod, only : geos_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)          :: n
integer, intent(in)          :: findex
character(len=*), intent(in) :: name
integer, intent(in)          :: order, tscount
integer, intent(out)         :: ferror
```

DESCRIPTION:

For the given time, reads parameters from GEOS data, transforms into 9 LIS forcing parameters and interpolates to the LIS domain.

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)

The arguments are:

order flag indicating which data to be read (order=1, read the previous 3 hourly instance, order=2, read the next 3 hourly instance)

n index of the nest

name name of the 3 hour GDAS forecast file

tscount time step count

ferror return error code (0 indicates success)

The routines invoked are:

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

35.1.5 readcrd_geos (Source File: readcrd_geos.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcrd_geos()
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod, only : LIS_logunit
use geos_forcingMod, only : geos_struc
```

DESCRIPTION:

This routine reads the options specific to GEOS forcing from the LIS configuration file.

35.1.6 timeinterp_geos (Source File: timeinterp_geos.F90)

REVISION HISTORY:

```
 30 Nov 2000: Jon Radakovich; Initial code based on geteta.f
 17 Apr 2001: Jon Gottschalck; A few changes to allow model init.
  5 Nov 2001: Urszula Jambor; Reset tiny negative SW values to zero
 10 Feb 2003: Sujay Kumar: Initial Specification in LIS.
 10 Oct 2006: Sujay Kumar: Switched to using ESMF_State for storing
              forcing data.
```

INTERFACE:

```
subroutine timeinterp_geos(n, findex)
```

USES:

```
use ESMF
use LIS_coreMod,          only : LIS_rc, LIS_domain, LIS_localPet
use LIS_metforcingMod,   only : LIS_forc, LIS_FORC_Base_State
use LIS_FORC_AttributesMod
use LIS_constantsMod,     only : LIS_CONST_SOLAR
use LIS_timeMgrMod,       only : LIS_time2date
use LIS_logMod,           only : LIS_logunit, LIS_verify, LIS_endrun
use geos_forcingMod,      only : geos_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous 3 hourly value is used. All other variables are linearly interpolated between the 3 hourly blocks.
The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

zterp (5.3.19)
zenith-angle based interpolation

35.2 Fortran: Module Interface finalize_geos5fcst (Source File: finalize_geos5fcst.F90)

REVISION HISTORY:

```
7 Mar 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine finalize_geos5fcst(findex)
```

USES:

```
use LIS_coreMod,          only : LIS_rc
use geos5fcst_forcingMod, only : geos5fcst_struc, geos5fcst_initialized
```

DESCRIPTION:

Routine to finalize GEOS5 forcing related memory allocations.

35.3 Fortran: Module Interface geos5fcst_forcingMod (Source File: geos5fcst_forcingMod.F9

This module contains variables and data structures that are used for the implementation of GEOS5 ensemble forecast data used as forcing within LIS. The input spatial resolution of the data is at 1.25x1 deg. This plugin supports either the use of any chosen ensemble member within the GEOS5 forecast output or the full ensemble.

USES:

```
implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: init_GEOS5FCST      !defines the native resolution of
                              !the input data
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: geos5fcst_struc
```

35.3.1 init_GEOS5FCST (Source File: geos5fcst_forcingMod.F90)

REVISION HISTORY:

```
7 Mar 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine init_GEOS5FCST(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_update_timestep
use LIS_logMod,     only : LIS_logunit,LIS_endrun
use LIS_spatialDownscalingMod, only : LIS_init_pcpclimo_native

implicit none

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

DESCRIPTION:

Defines the native resolution of the input forcing for GEOS5 forecast data (which is assumed to be at 1.25x1 deg). The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). Currently only bilinear interpolation is supported for this forcing plugin.
The routines invoked are:

readcrd_geos5fcst (35.3.6)
reads the runtime options specified for GEOS5 forecast data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

35.3.2 get_geos5fcst (Source File: get_geos5fcst.F90)

REVISION HISTORY:

```
7 Mar 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine get_geos5fcst(n,findex)
```

USES:

```
use LIS_coreMod,          only : LIS_rc, LIS_domain
use LIS_timeMgrMod,       only : LIS_tick
use LIS_metforcingMod,    only : LIS_forc
use LIS_logMod,           only : LIS_logunit, LIS_endrun
use geos5fcst_forcingMod, only : geos5fcst_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates hourly, GEOS5 forecast forcing. At the beginning of a simulation, the code reads the most recent past data (nearest hourly interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.
The arguments are:

n index of the nest

findex index of the forcing

The routines invoked are:

LIS_tick (3.36.27)
determines the GEOS5 data times

geos5fcstfile (35.3.3)
Puts together appropriate timestamped GEOS5 data

read_geos5fcst (35.3.5)
Reads and Interpolates GEOS5 forecast data to the LIS grid

35.3.3 geos5fcstfile (Source File: get_geos5fcst.F90)

INTERFACE:

```
subroutine geos5fcstfile(name,geos5fcstdir,yr,mo,da,hr,mn,ens_id)

  implicit none
```

ARGUMENTS:

```
character(len=*), intent(out) :: name
character(len=*), intent(in)  :: geos5fcstdir
integer, intent(in)           :: yr,mo,da,hr,mn
integer                       :: ens_id
```

DESCRIPTION:

This subroutine puts together GEOS5FCST file name for 1 hour file intervals
The arguments are:

name name of the timestamped GEOS5 forecast file

geos5fcstdir Name of the GEOS5 forecast data directory

yr year

mo month

da day of month

hr hour of day

mn minute

ens_id id of the ensemble member

35.3.4 interp_geos5fcst (Source File: interp_geos5fcst.F90)

REVISION HISTORY:

```
7 Mar 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine interp_geos5fcst(n,findex,input_data,output_2d)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_spatialDownscalingMod
use geos5fcst_forcingMod, only :geos5fcst_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: n
integer, intent(in)   :: findex
real, intent(in)      :: input_data(geos5fcst_struc(n)%nc,geos5fcst_struc(n)%nr)
real, intent(inout)   :: output_2d(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine interpolates a given GEOS5 data field to the LIS grid.

35.3.5 read_geos5fcst (Source File: read_geos5fcst.F90)

REVISION HISTORY:

```
7 Mar 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine read_geos5fcst(n, findex, order, name, ferror)
```

USES:

```
  use LIS_coreMod,          only : LIS_rc, LIS_domain
  use LIS_logMod,           only : LIS_logunit, LIS_verify, LIS_warning
  use LIS_metforcingMod,    only : LIS_forc
  use geos5fcst_forcingMod, only : geos5fcst_struc
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
  implicit none
```

ARGUMENTS:

```
integer, intent(in)       :: findex
integer, intent(in)       :: n
integer, intent(in)       :: order
character*100, intent(in) :: name
integer, intent(out)      :: ferror
```

DESCRIPTION:

For the given time, reads the forcing data from the GEOS5 file, transforms into LIS forcing parameters and interpolates to the LIS domain.
The arguments are:

n index of the nest

findex index of the forcing

order flag indicating which data to be read (order=1, read the previous hourly instance, order=2, read the next hourly instance)

name name of the file to be read

ferror return error flag (0-fail, 1-success)

The routines invoked are:

interp_geos5fcst (35.3.4)
Performs spatial interpolation of GEOS5 forecast data to the LIS grid

---

35.3.6 readcrd_geos5fcst (Source File: readcrd_geos5fcst.F90)

REVISION HISTORY:

```
7 Mar 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine readcrd_geos5fcst()
```

USES:

```
use ESMF
use LIS_coreMod,          only : LIS_rc, LIS_config
use LIS_logMod,           only : LIS_logunit, LIS_verify
use geos5fcst_forcingMod, only : geos5fcst_struc
```

DESCRIPTION:

This routine reads the options specific to GEOS5 forecast forcing from the LIS configuration file.

35.4 Fortran: Module Interface reset_geos5fcst (Source File: reset_geos5fcst.F90)

REVISION HISTORY:

```
7 Mar 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine reset_geos5fcst()
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use geos5fcst_forcingMod, only : geos5fcst_struc
```

DESCRIPTION:

Routine to reset GEOS5 forecast forcing related memory allocations.

35.4.1 timeinterp_geos5fcst (Source File: timeinterp_geos5fcst.F90)

REVISION HISTORY:

```
7 Mar 2013: Sujay Kumar, initial specification
```

INTERFACE:

```
subroutine timeinterp_geos5fcst(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod,          only : LIS_rc,LIS_domain
use LIS_constantsMod,     only : LIS_CONST_SOLAR
use LIS_metforcingMod,    only : LIS_forc, LIS_FORC_Base_State
use LIS_FORC_AttributesMod
use LIS_timeMgrMod,       only : LIS_tick, LIS_time2date
use LIS_logMod,           only : LIS_logunit, LIS_verify, LIS_endrun
use geos5fcst_forcingMod, only : geos5fcst_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the GEOS5 forecast forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous 3 hourly value is used. All other variables are linearly interpolated between the 3 hourly blocks.
The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

LIS_tick (3.36.27)
advances or retracts time by the specified amount

zterp (5.3.19)
zenith-angle based interpolation

---

36 GFS

This section describes the implementation of the GFS forcing data.

36.0.2 create_gfs_f0backup_filename (Source File: create_gfs_f0backup_filename.F90)

INTERFACE:

```
subroutine create_gfs_f0backup_filename(option, name00, gfsdir, yr, mo, da, hr, status)
```

USES:

```
use LIS_timeMgrMod, only : LIS_tick

implicit none
```

ARGUMENTS:

```
integer,          intent(in)    :: option
character(len=*), intent(in)    :: gfsdir
integer, intent(in)             :: yr, mo, da, hr
character(len=*), intent (out)  :: name00
integer                         :: status
```

DESCRIPTION:

This subroutine puts together the backup file for f00 forecast
The arguments are:

option bookend option (1 or 2)

name00 name of the timestamped F00 backup file

gfsdir Name of the GFS directory

yr year

mo month

da day of month

hr hour of day

status flag indicating if the filename was generated successfully

---

36.0.3 create_gfsfilename (Source File: create_gfsfilename.F90)

REVISION HISTORY:

```
16 Mar 2008: Sujay Kumar; Initial specification
```

INTERFACE:

```
subroutine create_gfsfilename(option, name00, name03, name06, &
     F06flag, gfsdir, yr, mo, da, hr )
```

USES:

```
use LIS_timeMgrMod, only : LIS_tick

implicit none
```

ARGUMENTS:

```
integer,          intent(in)    :: option
character(len=*), intent(in)    :: gfsdir
integer, intent(in)             :: yr, mo, da, hr
character(len=*), intent (out)  :: name00
character(len=*), intent (out)  :: name03
character(len=*), intent (out)  :: name06
logical                         :: F06flag
```

DESCRIPTION:

This subroutine puts together GFS file names for different (0,3,6hr) forecast instances
The arguments are:

option bookend option (1 or 2)

gfsdir Name of the GFS directory

yr year

mo month

da day of month

hr hour of day

name00 name of the GFS file for reading instantaneous fields

name03 name of the 3hr GFS file for reading time averaged fields

name06 name of the 6hr GFS file for reading time averaged fields

F06flag flag to indicate if 6hr forecast data is required for this interval

---

36.1 Fortran: Module Interface finalize_gfs (Source File: finalize_gfs.F90)

REVISION HISTORY:

```
16 Mar 2008: Sujay Kumar; Initial specification
```

INTERFACE:

```
subroutine finalize_gfs(findex)
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use gfs_forcingMod
```

DESCRIPTION:

Routine to cleanup allocated structures for GFS forcing.

36.1.1 get_gfs (Source File: get_gfs.F90)

REVISION HISTORY:

```
16 Mar 2008: Sujay Kumar; Initial specification
29 Apr 2010: Sujay Kumar: Fixed the problems in mixing the instantaneous
             and time averaged GFS forecast files.
```

INTERFACE:

```
subroutine get_gfs(n,findex)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_metforcingMod, only : LIS_forc
use LIS_timeMgrMod,     only : LIS_tick, LIS_get_nstep
use LIS_logMod,         only : LIS_logunit, LIS_endrun
use gfs_forcingMod,    only : gfs_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 3-hrly, GFS forcing. The idea is to open either the 00, 03 and 06 forecast file associated with the most recent GFS assimilation (available every 6 hours). Precipitation rates and radiation fluxes will be taken from the F03 and F06 files, since averages are provided. At the beginning of a simulation, the code reads the most recent past data (nearest 3 hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. The strategy for missing data is to first look for 9hr forecasts and if not go backwards up to 10 days to get forcing at the same time of day. Note that when run in forecast mode (for future days), the F00 files are not available and the code defaults to using F03 and F06 files.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the GFS data times

create_gfsfilename (36.0.3)
Puts together appropriate file name for 3 hour intervals

create_gfs_f0backup_filename (36.0.2)
Puts together the backup filenames when F00 forecasts are not available.

read_gfs (36.2.2)
Interpolates GFS data to LIS grid

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

read_gfs_elev (36.2.5)
reads the native elevation of the GFS data to be used for topographic adjustments to the forcing

---

36.2 Fortran: Module Interface gfs_forcingMod (Source File: gfs_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the forcing data from the Global Forecast System (GFS) developed at the Environmental Modeling Center (EMC) of NOAA/NCEP. GFS forcing variables are produced on a quadratic gaussian grid. LIS uses the 00, 03, and 06 forecast files. The forecasts are produced at 6 hr intervals. The resolution of GFS forcing varies as follows:
upto 2000/1/24 : T126 (384x190) grid
2001/1/24 - 2002/10/29 : T170 (512x256) grid
2002/10/29 - 2005/5/31 : T254 (768x384) grid
2005/5/31 - 2010/7/28 : T382 (1152x576) grid
2010/7/28 - : T574 (1760x880) grid

The implementation in LIS has the derived data type `gfs_struc` that includes the variables that specify the runtime options, and the weights and neighbor information to be used for spatial interpolation. They are described below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

nmif Number of forcing variables in the GFS data

gfstime1 The nearest, previous 3 hour instance of the incoming data (as a real time).

gfstime2 The nearest, next 3 hour instance of the incoming data (as a real time).

griduptime1 The time to switch GFS resolution to T126

griduptime2 The time to switch GFS resolution to T170

griduptime3 The time to switch GFS resolution to T254

griduptime4 The time to switch GFS resolution to T382

griduptime5 The time to switch GFS resolution to T574

gfsdir Directory containing the input data

elevfile File with the elevation definition for the input data.

mi Number of points in the input grid

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n11,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n112,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

findtime1, findtime2 boolean flags to indicate which time is to be read for temporal interpolation.

USES:

```
implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: init_GFS      !defines the native resolution of
                                !the input data
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: gfs_struc
```

36.2.1 init_GFS (Source File: gfs_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine init_GFS(findex)
```

USES:

```
  use LIS_coreMod,    only : LIS_rc
  use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep

  implicit none
!AGRUMENTS:
  integer, intent(in) :: findex
```

DESCRIPTION:

Defines the native resolution of the input forcing for GFS data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). Based on the GFS data map projection and resolution, this routine sets up the spatial interpolation weights. The dates of the GFS resolution switches are also defined in this routine.
The routines invoked are:

readcrd_gfs (36.2.6)
reads the runtime options specified for GFS data

LIS_date2time (3.36.24)
converts date to the real time format

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

36.2.2 read_gfs (Source File: read_gfs.F90)

REVISION HISTORY:

```
16 Mar 2008: Sujay Kumar; Initial specification
25 Jan 2012: Sujay Kumar; Switched to the use of grib-api library
```

INTERFACE:

```
subroutine read_gfs( order, n, findex, name00, name03, name06, F06flag, ferror,try )
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_timeMgrMod,     only : LIS_get_nstep, LIS_date2time
use LIS_metforcingMod, only : LIS_forc
use LIS_logMod,         only : LIS_logunit
use gfs_forcingMod,    only : gfs_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)           :: order
integer, intent(in)           :: n
integer, intent(in)           :: findex
character(len=*), intent(in)  :: name00
character(len=*), intent(in)  :: name03
character(len=*), intent(in)  :: name06
logical, intent(in)           :: F06flag
integer, intent(out)          :: ferror
integer, intent(inout)        :: try
```

DESCRIPTION:

For the given time, reads parameters from GFS forecast datasets, transforms into 9 LIS forcing parameters and interpolates to the LIS domain.
The arguments are:

order flag indicating which data to be read (order=1, read for the previous 3hr bookend, order=2, read for the next 3 hr bookend)

n index of the nest

name00 name of the instantaneous forecast file

name03 name of the 3 hour GFS forecast file

name06 name of the 6 hour GFS forecast file

F06flag flag to indicate if 6hr forecast data is required for this interval

ferror flag to indicate success of the call (=0 indicates success)

try index of the tries (in case of missing data)

The routines invoked are:

interp_gfs (36.2.4)
spatially interpolates a GFS variable

36.2.3 retrieve_gfs_variables (Source File: read_gfs.F90)

INTERFACE:

```
subroutine retrieve_gfs_variables(n, findex, fname, glbdata, errorcode)
```

USES:

```
  use LIS_coreMod,        only : LIS_rc, LIS_domain
  use LIS_logMod,         only : LIS_logunit,LIS_getNextUnitNumber,&
       LIS_releaseUnitNumber, LIS_verify, LIS_warning
  use gfs_forcingMod,    only : gfs_struc

#if (defined USE_GRIBAPI)
  use grib_api
#endif

  implicit none
```

ARGUMENTS:

```
integer               :: n
integer               :: findex
character(len=*)      :: fname
real                  :: glbdata(10,LIS_rc%ngrid(n))
integer               :: errorcode
```

DESCRIPTION:

This subroutine retrieves GFS forcing variables, and interpolates them to the LIS grid.

36.2.4 interp_gfs (Source File: read_gfs.F90)

INTERFACE:

```
subroutine interp_gfs(n, findex, pcp_flag,ngfs,f,lb,lis_gds,nc,nr, &
     varfield)
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use gfs_forcingMod,   only : gfs_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
logical, intent(in) :: pcp_flag
integer, intent(in) :: ngfs
real, intent(out)   :: f(ngfs)
logical*1           :: lb(ngfs)
real                :: lis_gds(50)
integer, intent(in) :: nc
integer, intent(in) :: nr
real, intent(out)   :: varfield(nc,nr)
```

DESCRIPTION:

This subroutine interpolates a given GFS field to the LIS grid. The arguments are:

n index of the nest

pcp_flag flag indicating precipitation variables

ngfs number of elements in the input grid

f input data array to be interpolated

lb input bitmap

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

---

36.2.5 read_gfs_elev (Source File: read_gfs_elev.F90)

REVISION HISTORY:

```
16 Mar 2008: Sujay Kumar; Initial specification
```

INTERFACE:

```
subroutine read_gfs_elev(n, findex, change)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_metforcingMod, only : LIS_forc
use LIS_fileIOMod,      only : LIS_readData
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use gfs_forcingMod,    only : gfs_struc
 use gaussian_mod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates GFS model elevation to the LIS grid. The data will be used to perform any topographical adjustments to the forcing.
The arguments are:

n index of the nest

The routines invoked are:

LIS_readData (3.14.1)
Abstract method to read the elevation of the forcing data in the same map projection used in LIS.

36.2.6 readcrd_gfs (Source File: readcrd_gfs.F90)

REVISION HISTORY:

```
16 Mar 2008: Sujay Kumar; Initial specification
```

INTERFACE:

```
subroutine readcrd_gfs()
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc, LIS_config
use LIS_logMod,    only : LIS_logunit
use gfs_forcingMod, only: gfs_struc
```

DESCRIPTION:

This routine reads the options specific to GFS forcing from the LIS configuration file.

36.2.7 timeinterp_gfs (Source File: timeinterp_gfs.F90)

REVISION HISTORY:

```
16 Mar 2008: Sujay Kumar; Initial specification
10 Oct 2006: Sujay Kumar: Switched to using ESMF_State for storing
             forcing data.
```

INTERFACE:

```
subroutine timeinterp_gfs(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod,        only  : LIS_rc, LIS_domain
use LIS_FORC_AttributesMod
use LIS_timeMgrMod,     only  : LIS_time2date, LIS_tick
use LIS_constantsMod,   only  : LIS_CONST_SOLAR
use LIS_metforcingMod, only  : LIS_forc, LIS_FORC_Base_State
use LIS_logMod,         only  : LIS_logunit, LIS_verify, LIS_endrun
use gfs_forcingMod,    only  : gfs_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous 3 hourly value is used. All other variables are linearly interpolated between the 3 hourly blocks.
The arguments are:

n index of the nest

The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

LIS_tick (3.36.27)
advances or retracts time by the specified amount

zterp (5.3.19)
zenith-angle based interpolation

---

37 GLDAS

This section describes the implementation of the forcing data available from the Global Land Data Assimilation System (GLDAS) produced by merging GEOS, AGRMET radiation and CMAP precipitaion by the NASA GSFC's Global Modeling and Assimilation office (GMAO).

37.0.8 fillgaps_gldas (Source File: fillgaps_gldas.F90)

INTERFACE:

```
subroutine fillgaps_gldas(n,ip,varfield)
```

USES:

```
use LIS_coreMod,       only : LIS_rc,LIS_domain
use LIS_logMod,        only : LIS_logunit, LIS_endrun
use gldas_forcingMod,  only : gldas_struc

implicit none
```

USES:

```
integer, intent(in)    :: n
integer, intent(in)    :: ip
real, intent(inout)    :: varfield(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine fills in invalid grid points introduced due to mismatches in landmasks in LIS and GLDAS. This routine assumes that the undef or invalid value is the LIS undefined value.
The arguments are:

n index of the nest

ip interpolation option

varfield updated output field

37.1 Fortran: Module Interface finalize_gldas (Source File: finalize_gldas.F90)

REVISION HISTORY:

```
19 Sept 2008: Sujay Kumar: Initial Implementation
```

INTERFACE:

```
subroutine finalize_gldas(findex)
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use gldas_forcingMod
```

DESCRIPTION:

Routine to cleanup allocated structures for GDAS forcing.

37.1.1 get_gldas (Source File: get_gldas.F90)

REVISION HISTORY:

```
19 Sept 2008: Sujay Kumar: Initial Implementation
```

INTERFACE:

```
subroutine get_gldas(n,findex)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_metforcingMod, only : LIS_forc
use LIS_timeMgrMod,     only : LIS_tick, LIS_get_nstep
use LIS_logMod,         only : LIS_logunit, LIS_endrun
use gldas_forcingMod,    only : gldas_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 3-hrly, GLDAS forcing. The temporal frequency of the GLDAS data is 3 hours. At the beginning of a simulation, the code reads the most recent past data (nearest 3 hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the GDAS data times

gldasfile (37.1.2)
Puts together a time stamped filename for GLDAS data.

read_gldas (37.2.2)
reads and spatially interpolates GLDAS data

37.1.2 gldasfile (Source File: get_gldas.F90)

INTERFACE:

```
subroutine gldasfile(name,gldasdir,yr,mo,da,hr,ncold)

  implicit none
```

ARGUMENTS:

```
  character(len=*), intent(in)  :: gldasdir
  integer, intent(in)       :: yr,mo,da,hr,ncold
  character(len=*), intent(out) :: name
```

DESCRIPTION:

This subroutine puts together GLDAS file name for 3 hour file intervals
The arguments are:

name name of the timestamped GLDAS file

gldasdir Name of the GLDAS directory

yr year

mo month

da day of month

hr hour of day

ncold Number of columns (used as to determine the GLDAS resolution)

37.2 Fortran: Module Interface gldas_forcingMod (Source File: gldas_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the GLDAS forcing data produced by merging GEOS, AGRMET radiation and CMAP precipitation
The forcing data consists of the following meteorological variables.

2m air temperature

2m specific humidity

incident downward shortwave radiation

incident downward longwave radiation

eastward wind

northward wind

surface pressure

rainfall rate

snowfall rate

convective rainfall rate

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init_gldas      !defines the native resolution of
                                !the input data
```

PUBLIC TYPES:

```
public :: gldas_struc
```

37.2.1 init_gldas (Source File: gldas_forcingMod.F90)

REVISION HISTORY:

```
19 Sept 2008: Sujay Kumar: Initial Implementation
```

INTERFACE:

```
subroutine init_gldas(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_update_timestep
implicit none

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

DESCRIPTION:

Defines the native resolution of the input forcing for GLDAS data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V).
The routines invoked are:

readcrd_gldas (37.2.4)
reads the runtime options specified for GLDAS data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

neighbor_interp_input (5.3.11)
computes the computational weights for neighbor interpolation

---

37.2.2 read_gldas (Source File: read_gldas.F90)

REVISION HISTORY:

```
19 Sept 2008: Sujay Kumar: Initial Implementation
25 Jan 2012: Sujay Kumar; Switched to the use of grib-api library
```

INTERFACE:

```
subroutine read_gldas( order, n, findex, name, ferror, try )
```

USES:

```
  use LIS_coreMod,         only : LIS_rc, LIS_domain
  use LIS_timeMgrMod,      only : LIS_get_nstep
  use LIS_metforcingMod,   only : LIS_forc
  use LIS_logMod,          only : LIS_logunit, LIS_verify, LIS_warning
  use gldas_forcingMod,    only : gldas_struc

#if (defined USE_GRIBAPI)
  use grib_api
#endif

  implicit none
```

ARGUMENTS:

```
  integer, intent(in)           :: order
  integer, intent(in)           :: n
  integer, intent(in)           :: findex
  character(len=*),  intent(in) :: name
  integer, intent(out)          :: ferror
  integer, intent(inout)        :: try
```

DESCRIPTION:

For the given time, reads parameters from GLDAS data and spatially interpolates to the LIS domain. The arguments are:

order flag indicating which data to be read (order=1, read the previous 3 hourly instance, order=2, read the next 3 hourly instance)

n index of the nest

name name of the 3 hour GLDAS forecast file

ferror flag to indicate success of the call (=0 indicates success)

try index of the tries (in case of missing data)

The routines invoked are:

interp_gldas (37.2.3)
spatially interpolates a GLDAS variable

fillgaps_gldas (37.0.8)
fills the data gaps due to mismatches in landmask of LIS domain and GLDAS mask.

37.2.3 interp_gldas (Source File: read_gldas.F90)

INTERFACE:

```
subroutine interp_gldas(n,findex,pcp_flag,ngldas,f,lb,lis_gds,nc,nr, &
                        varfield)
```

USES:

```
use LIS_coreMod,      only : LIS_rc
use gldas_forcingMod, only : gldas_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
logical, intent(in) :: pcp_flag
integer, intent(in) :: ngldas
real, intent(out)   :: f(ngldas)
logical*1           :: lb(ngldas)
real                :: lis_gds(50)
integer, intent(in) :: nc
integer, intent(in) :: nr
real, intent(out)   :: varfield(nc,nr)
```

DESCRIPTION:

This subroutine spatially interpolates a given GLDAS field to the LIS grid. The arguments are:

n index of the nest

findex index of the forcing source

pcp_flag flag indicating if precip variables are being interpolated

ngldas number of elements in the input grid

f input data array to be interpolated

lb input bitmap

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

neighbor_interp (5.3.10)
spatially interpolate the forcing data using neighbor interpolation

---

37.2.4 readcrd_gldas (Source File: readcrd_gldas.F90)

REVISION HISTORY:

```
19 Sept 2008: Sujay Kumar: Initial Implementation
```

INTERFACE:

```
subroutine readcrd_gldas()
```

USES:

```
use ESMF
use LIS_coreMod,      only : LIS_rc, LIS_config
use LIS_logMod,       only : LIS_logunit
use gldas_forcingMod, only : gldas_struc
```

DESCRIPTION:

This routine reads the runtime options specific to GLDAS forcing from the LIS configuration file.

37.2.5 timeinterp_gldas (Source File: timeinterp_gldas.F90)

REVISION HISTORY:

```
19 Sept 2008: Sujay Kumar: Initial Implementation
```

INTERFACE:

```
subroutine timeinterp_gldas(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod,        only  : LIS_rc, LIS_domain, LIS_localPet
use LIS_timeMgrMod,     only  : LIS_time2date, LIS_tick
use LIS_constantsMod,   only  : LIS_CONST_SOLAR
use LIS_FORC_AttributesMod
use LIS_metforcingMod, only  : LIS_forc, LIS_FORC_Base_State
use LIS_logMod,         only  : LIS_logunit, LIS_verify, LIS_endrun
use gldas_forcingMod,    only  : gldas_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous 3 hourly value is used. All other variables are linearly interpolated between the 3 hourly blocks.
The arguments are:

n index of the nest

The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

LIS_tick (3.36.27)
advances or retracts time by the specified amount

zterp (5.3.19)
zenith-angle based interpolation

---

38 GSWP1

This section describes the implementation of the forcing data used in the Global Soil Wetness Project Phase I.

38.1 Fortran: Module Interface finalize_gswp1 (Source File: finalize_gswp1.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine finalize_gswp1
```

USES:

```
use gswp1_forcingMod, only : gswp1_struc
```

DESCRIPTION:

Routine to cleanup allocated structures for GSWP1 forcing.

38.1.1 get_gswp1 (Source File: get_gswp1.F90)

REVISION HISTORY:

```
11 Dec 2003: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine get_gswp1(n, findex)
```

USES:

```
use LIS_coreMod,         only : LIS_rc
use LIS_metforcingMod,   only : LIS_forc
use LIS_timeMgrMod,      only : LIS_get_nstep, LIS_tick
use LIS_logMod,          only : LIS_logunit, LIS_endrun
use gswp1_forcingMod,    only : gswp1_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates hourly, GSWP1 forcing. At the beginning of a simulation, the code reads the most recent past data (nearest hourly interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day . The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the GSWP data times

gswp1file (38.1.2)
determines the GSWP1 data file

read_gswp1 (38.2.2)
reads GSWP1 data to LIS grid

38.1.2 gswp1file (Source File: get_gswp1.F90)

INTERFACE:

```
subroutine gswp1file(name,gswp1dir,yr,mo,da,hr,ncold)

implicit none
```

INPUT PARAMETERS:

```
character(len=*), intent(in) :: gswp1dir
integer, intent(in)      :: yr,mo,da,hr,ncold
```

OUTPUT PARAMETERS:

```
character(len=*), intent(out) :: name
```

DESCRIPTION:

This subroutine puts together GSWP-1 file name

38.2 Fortran: Module Interface gswp1_forcingMod (Source File: gswp1_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the forcing data from the Global Soil Wetness Project (GSWP). GSWP forcing variables are produced on a latlon 1degree grid at 3 hour intervals.
The implemenatation in LIS has the derived data type `gswp_struc` that includes the variables that specify the runtime options They are described below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

gswptime1 The nearest, previous 3 hour instance of the incoming data (as a real time).

gswptime2 The nearest, next 3 hour instance of the incoming data (as a real time).

tair Directory containing the 2m air temperature data

qair Directory containing the 2m specific humidity data

psurf Directory containing the surface pressure data

wind Directory containing the wind data

rainf Directory containing the total precipitation data

snowf Directory containing the total snowfall data

swdown Directory containing the downward shortwave radiation data

swdown Directory containing the downward longwave radiation data

mi Number of points in the input grid

findtime1, findtime2 boolean flags to indicate which time is to be read for temporal interpolation.

38.2.1 init_gswp1 (Source File: gswp1_forcingMod.F90)

REVISION HISTORY:

```
11 Dec 2003: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine init_gswp1(findex)
```

USES:

```
    use LIS_coreMod, only : LIS_rc
    use LIS_timeMgrMod, only : LIS_update_timestep
    use LIS_logMod,  only : LIS_logunit, LIS_endrun

    implicit none
  !AGRUMENTS:
    integer, intent(in) :: findex
```

DESCRIPTION:

Defines the native resolution of the input forcing for GSWP1 data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V).
The routines invoked are:

readcrd_gswp1 (38.3.1)
reads the runtime options specified for GSWP1 data

38.2.2 read_gswp1 (Source File: read_gswp1.F90)

REVISION HISTORY:

```
11 Dec 2003: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine read_gswp1(order,n,findex, name,tscount,ferror)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain, LIS_localPet
use LIS_metforcingMod, only : LIS_forc
use LIS_logMod, only         : LIS_logunit, LIS_endrun, LIS_verify
use gswp1_forcingMod,   only : gswp1_struc
```

DESCRIPTION:

Reads in GSWP-1 data and performs interpolation to the LIS domain.

38.3 Core Functions of read_gswp1

bilinear_interp Interpolates GSWP-1 data to LIS grid using bilinear interpolation

GSWP-1 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. wind speed wind speed
6. wind east dummy field, always 0
7. ps Instantaneous Surface Pressure $[Pa]$
8. preacc Total precipitation $[mm/s]$
9. precon Convective precipatation $[mm/s]$

38.3.1 readcrd_gswp1 (Source File: readcrd_gswp1.F90)

REVISION HISTORY:

```
11 Dec 2003: Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcrd_gswp1()
```

USES:

```
use ESMF
use LIS_logMod, only : LIS_logunit
use LIS_coreMod, only : LIS_rc, LIS_config
use gswp1_forcingMod, only : gswp1_struc
```

DESCRIPTION:

This routine reads the options specific to GSWP1 forcing from the LIS configuration file.

38.3.2 timeinterp_gswp1 (Source File: timeinterp_gswp1.F90)

REVISION HISTORY:

```
11 Dec 2003: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine timeinterp_gswp1(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_metforcingMod, only : LIS_forc, LIS_FORC_Base_State
use LIS_FORC_AttributesMod
use LIS_timeMgrMod, only        : LIS_get_nstep, LIS_time2date
use LIS_constantsMod, only       : LIS_CONST_SOLAR
use gswp1_forcingMod, only : gswp1_struc
use LIS_logMod, only         : LIS_logunit, LIS_verify, LIS_endrun
```

ARGUMENTS:

```
implicit none

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous hourly value is used. All other variables are linearly interpolated between the hourly blocks.
The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

zterp (5.3.19)
zenith-angle based interpolation

---

39 GSWP

This section describes the implementation of the forcing data used in the Global Soil Wetness Project Phase II.

39.0.3 fillgaps_gswp2 (Source File: fillgaps_gswp2.F90)

INTERFACE:

```
subroutine fillgaps_gswp2(n,ip,varfield)
```

USES:

```
use LIS_coreMod,       only : LIS_rc,LIS_domain
use LIS_logMod,        only : LIS_logunit, LIS_endrun
use gswp2_forcingMod,   only : gswp2_struc

implicit none
```

USES:

```
integer, intent(in)    :: n
integer, intent(in)    :: ip
real, intent(inout)    :: varfield(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine fills in invalid grid points introduced due to mismatches in landmasks in LIS and GSWP2. This routine assumes that the undef or invalid value is the LIS undefined value.
The arguments are:

n index of the nest

ip interpolation option

varfield updated output field

39.1 Fortran: Module Interface finalize_gswp2 (Source File: finalize_gswp2.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine finalize_gswp2
```

USES:

```
use gswp2_forcingMod, only : gswp2_struc
```

DESCRIPTION:

Routine to cleanup allocated structures for GSWP forcing.

39.1.1 finterp_gswp2 (Source File: finterp_gswp2.F90)

REVISION HISTORY:

```
2002/08/16  15:55:18  guo
```

INTERFACE:

```
subroutine finterp_gswp2(ip,trp_flag,val0,val1,val2,val3,madtt,istep,vout)
```

USES:

```
use LIS_logMod, only : LIS_logunit

implicit none

integer,intent(in)           :: ip
integer,intent(in)           :: istep
character(len=1),intent(in)  :: trp_flag
real,intent(in)              :: val0
real,intent(in)              :: val1
real,intent(in)              :: val2
real,intent(in)              :: val3
integer,intent(in)           :: madtt
real                         :: vout
```

DESCRIPTION:

Introduced a new interpolation type "P" which specifies a temporal disaggregation of interpolated precipitation to improve partitioning of infiltration/runoff when forcing data interval is large compared to the typical length of a convective (or total) rain event.

** This routine interpolates atmospheric adtt seconds period forcing data to dtt seconds valtrp time-step for one adtt interval. Interpolation is performed based on the value of 'trp_flag':

"L" or "l" = value represents average over interval ending at current time
"N" or "n" = value represents average over interval beginning at current time
"C" or "c" = value represents average over interval centered on current time
"I" or "i" = instantaneous value at current time (linear interpolation)
"P" or "p" = PDF disaggregation applied in time (for precip)
"0" (zero) = no interpolation, centered on current time
Otherwise = no interpolation, applied beginning at current time

* NOTE:
* For "L", "N", and "C" to conserve the mean over the interval after
* interpolation, 'madtt' MUST BE A MULTIPLE OF 2!

The arguments are:

ip grid point in question

trp_flag type of interpolation

val0 last forcing value (for cases N and C)

val1 current forcing value (all cases)

val2 next forcing value (for all cases but default)

val3 ueber-next forcing value (for all cases C and L)

madtt number of valtrp timesteps in a forcing timestep

vout output forcing value

39.1.2 get_gswp2 (Source File: get_gswp2.F90)

REVISION HISTORY:

```
20Feb2004; Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine get_gswp2(n, findex)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_metforcingMod, only : LIS_forc
use LIS_timeMgrMod,     only : LIS_get_nstep, LIS_tick
use LIS_logMod,         only : LIS_logunit
use gswp2_forcingMod,    only : gswp2_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 3-hrly, GSWP2 forcing. At the beginning of a simulation, the code reads the most recent past data (nearest 3 hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day . The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the GSWP2 data times

read_gswp2 (39.2.2)
Interpolates GSWP2 data to LIS grid

39.2 Fortran: Module Interface gswp2_forcingMod (Source File: gswp2_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the forcing data from the Global Soil Wetness Project (GSWP2). GSWP2 forcing variables are produced on a latlon 1degree grid at 3 hour intervals.
The implemenatation in LIS has the derived data type `gswp2_struc` that includes the variables that specify the runtime options They are described below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

gswp2time1 The nearest, previous 3 hour instance of the incoming data (as a real time).

gswp2time2 The nearest, next 3 hour instance of the incoming data (as a real time).

tair Directory containing the 2m air temperature data

qair Directory containing the 2m specific humidity data

psurf Directory containing the surface pressure data

wind Directory containing the wind data

rainf Directory containing the total precipitation data

snowf Directory containing the total snowfall data

swdown Directory containing the downward shortwave radiation data

swdown Directory containing the downward longwave radiation data

mi Number of points in the input grid

findtime1, findtime2 boolean flags to indicate which time is to be read for temporal interpolation.

USES:

```
implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: init_GSWP2      !defines the native resolution of
                                !the input data
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: gswp2_struc
```

39.2.1 init_GSWP2 (Source File: gswp2_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine init_GSWP2(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_update_timestep
use LIS_logMod,     only : LIS_logunit, LIS_endrun

implicit none
```

USES:

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

DESCRIPTION:

Defines the native resolution of the input forcing for GSWP2 data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V).
The routines invoked are:

readcrd_gswp2 (39.2.8)
reads the runtime options specified for GSWP2 data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

gswp2_mask (39.2.7)
reads the GSWP2 mask

39.2.2 read_gswp2 (Source File: read_gswp2.F90)

REVISION HISTORY:

```
20Feb2004; Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_gswp2(order,n, findex, yr,mo,da,hr,mn,ss)
```

USES:

```
#if ( defined USE_NETCDF3 || defined USE_NETCDF4 )
  use netcdf
#endif
  use LIS_coreMod, only : LIS_rc,LIS_domain
  use LIS_metforcingMod, only:LIS_forc
  use LIS_logMod, only : LIS_logunit
  use LIS_gswpMod, only : getgswp_timeindex
  use gswp2_forcingMod, only : gswp2_struc

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

For the given time, reads parameters from the GSWP2 files, transforms into 9 LIS forcing parameters and interpolates to the LIS domain.
The arguments are:

order flag indicating which data to be read (order=1, read the previous 3 hourly instance, order=2, read the next 3 hourly instance)

n index of the nest

yr current year

mo current month

da current day of the month

hr current hour of day

mn current minute

ss current second

The routines invoked are:

getgswp_timeindex (3.15.2)
computes the GSWP2 time index

gswp2_process_forcing (39.2.3)
reads the GSWP2 forcing from the native NETCDF file

interp_gswp2 (39.2.4)
interpolates GSWP2 forcing to the LIS grid

gswp2_fillgaps (39.0.3)
fills in the mismatches due to GSWP2 and LIS masks

39.2.3 gswp2_process_forcing (Source File: read_gswp2.F90)

INTERFACE:

```
subroutine gswp2_process_forcing(n, gswp21d, gswp2_filled)
```

USES:

```
use gswp2_forcingMod, only : gswp2_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)                                        :: n
real, dimension(gswp2_struc(n)%vector_len), intent(in)      :: gswp21d
real, dimension(gswp2_struc(n)%ncold*gswp2_struc(n)%nrold), intent(out) :: gswp2_filled

real, dimension(gswp2_struc(n)%ncold, gswp2_struc(n)%nrold) :: gswp22d
```

DESCRIPTION:

Converts the native GSWP2 vector data to a gridded data, reordered to the LIS format. The arguments are:

n index of the nest

gswp21d native GSWP2 data as a 1-d vector.

gswp2_filled Filled, 1-d vector GSWP2 field

gswp22d reordered gridded GSWP2 field

The routines invoked are:

gswp2_expand_to_grid (39.2.5)
expand the gswp2 native data, change to the LIS ordering style.

gspw_2d_to_1d (39.2.6)
convert the 2d field to a 1d vector

39.2.4 interp_gswp2 (Source File: read_gswp2.F90)

INTERFACE:

```
subroutine interp_gswp2(n,findex, land_len,f,tempgswp2)
```

USES:

```
use LIS_coreMod,     only : LIS_rc, LIS_domain
use gswp2_forcingMod, only : gswp2_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)                        :: n
integer, intent(in)                        :: findex
integer, intent(in)                        :: land_len
real, dimension(land_len), intent(in)      :: f
real, dimension(LIS_rc%ngrid(n)), intent(out) :: tempgswp2
```

DESCRIPTION:

Interpolates the GSWP2 data onto the LIS grid. The arguments are:

n index of the nest

land_len size of the GSWP2 data

f native GSWP2 data

tempgswp2 interpolated GSWP2 data

The routines invoked are:

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

---

39.2.5 gswp2_expand_to_grid (Source File: read_gswp2.F90)

INTERFACE:

```
subroutine gswp2_expand_to_grid(n, array1, array2)
```

USES:

```
use gswp2_forcingMod, only : gswp2_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)                                          :: n
real, dimension(gswp2_struc(n)%vector_len), intent(in)        :: array1
real, dimension(gswp2_struc(n)%ncold, gswp2_struc(n)%nrold), &
      intent(out) :: array2
```

DESCRIPTION:

changes the 1d native GSWP2 data to a 2-d gridded data, reordered to N-S, E-W style. The arguments are:

n index of the nest

array1 native 1-d GSWP2 data

array2 reordered 2-d GSWP2 data.

39.2.6 gswp2_2d_to_1d (Source File: read_gswp2.F90)

INTERFACE:

```
subroutine gswp2_2d_to_1d(n, array2, array1)
```

USES:

```
use gswp2_forcingMod, only : gswp2_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)                                          :: n
real, dimension(gswp2_struc(n)%ncold, gswp2_struc(n)%nrold),          &
      intent(in)                                             :: array2
real, dimension(gswp2_struc(n)%ncold*gswp2_struc(n)%nrold),           &
      intent(out)                                            :: array1
```

DESCRIPTION:

reshapes the 2d-array to 1d array.
The arguments are:

n index of the nest

array1 input 2-d array

array2 input 1-d array

39.2.7 gswp2_mask (Source File: read_gswp2.F90)

INTERFACE:

```
subroutine gswp2_mask()
#if ( defined USE_NETCDF3 ||  defined USE_NETCDF4 )
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use gswp2_forcingMod, only : gswp2_struc
use netcdf
```

DESCRIPTION:

Reads the GSWP2 mask and computes the masking index.

39.2.8 readcrd_gswp2 (Source File: readcrd_gswp2.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcrd_gswp2()
```

USES:

```
use LIS_logMod, only : LIS_logunit
use LIS_coreMod, only : LIS_rc, LIS_config
use gswp2_forcingMod, only : gswp2_struc
use ESMF
```

DESCRIPTION:

This routine reads the options specific to GSWP2 forcing from the LIS configuration file.

39.2.9 timeinterp_gswp2 (Source File: timeinterp_gswp2.F90)

REVISION HISTORY:

```
20 Feb 2004; Sujay Kumar : Initial Specification
10 Oct 2006: Sujay Kumar: Switched to using ESMF_State for storing
             forcing data.
```

INTERFACE:

```
subroutine timeinterp_gswp2(n, findex)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc,LIS_domain
use LIS_constantsMod,   only : LIS_CONST_SOLAR
use LIS_metforcingMod, only : LIS_forc, LIS_FORC_Base_State
use LIS_FORC_AttributesMod
use LIS_timeMgrMod,     only : LIS_time2date
use LIS_logMod,         only : LIS_logunit, LIS_verify, LIS_endrun
use gswp2_forcingMod,    only : gswp2_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous 3 hourly value is used. All other variables are linearly interpolated between the 3 hourly blocks.
The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

zterp (5.3.19)
zenith-angle based interpolation

finterp_gswp2 (39.1.1)
routine to compute temporal disaggregation of forcing

---

40 MERRA-Land

This section describes the implementation of the MERRA-Land forcing data.

40.0.10 finalize_merraland (Source File: finalize_merraland.F90)

REVISION HISTORY:

```
12 Oct 2009, Eric Kemp
```

INTERFACE:

```
subroutine finalize_merraland(findex)
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use merraland_forcingMod, only : merraland_struc
```

DESCRIPTION:

Routine to cleanup MERRA-Land forcing related memory allocations.

40.0.11 get_merraland (Source File: get_merraland.F90)

REVISION HISTORY:

```
12 Oct 2009: Eric Kemp, initial version
27 May 2010: David Mocko, changed to hourly forcing for LIS6.0
25 Oct 2010: David Mocko, updated for LIS6.1 public release
 5 Apr 2013: Sujay Kumar, updated for the direct use of files from GES-DISC
31 May 2013: David Mocko, changes to match MERRA-Land interpolation strategy
```

INTERFACE:

```
subroutine get_merraland(n,findex)
```

USES:

```
use LIS_coreMod
use LIS_timeMgrMod
use LIS_logMod
use LIS_metforcingMod
use merraland_forcingMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 1-hourly MERRA-Land forcing. At the beginning of a simulation, the code reads the most recent past data (nearest 1 hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
call to advance or retract time

merralandfiles (40.0.12)
Puts together appropriate file name for 1 hour intervals

read_merraland (40.1.2)
call to read the MERRA-Land data and perform spatial interpolation

---

40.0.12 merralandfiles (Source File: get_merraland.F90)

INTERFACE:

```
subroutine merralandfiles(merralanddir,yr,mo,da, &
     slvname,flxname,radname,mldname,lndname)

  implicit none
```

ARGUMENTS:

```
  character(len=*), intent(in)  :: merralanddir
  integer, intent(in)           :: yr,mo,da
  character(len=*), intent(out) :: slvname
  character(len=*), intent(out) :: flxname
  character(len=*), intent(out) :: radname
  character(len=*), intent(out) :: mldname
  character(len=*), intent(out) :: lndname
```

DESCRIPTION:

This subroutine puts together MERRA-land file names for daily netcdf files
The arguments are:

merralanddir Name of the MERRA-Land directory

yr year

mo month

da day of month

slvname name of the timestamped met forcing file

flxname name of the timestamped precip flux file

radname name of the timestamped radidation file

40.1 Fortran: Module Interface merraland_forcingMod (Source File: merraland_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the MERRA-Land forcing data. The data is global 1 degree dataset in latlon projection, and at 1 hourly intervals. The derived data type `merraland_struc` includes the variables that specify the runtime options, and the weights and neighbor information to be used for spatial interpolation. They are described below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

nmif Number of forcing variables in the ECMWF data

merralandtime1 The nearest, previous 1 hour instance of the incoming data (as a real time).

merralandtime2 The nearest, next 1 hour instance of the incoming data (as a real time).

merralanddir Directory containing the input data

mi Number of points in the input grid

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n111,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n122,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

rlat3 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for n. neighbor interpolation)

rlon3 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for n. neighbor interpolation)

n113 Arrays containing the neighbor information of the input grid for each grid point in LIS, for n. neighbor interpolation.

findtime1, findtime2 boolean flags to indicate which time is to be read for temporal interpolation.

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init_MERRALAND      !defines the native resolution of
          !the input data
```

PUBLIC TYPES:

```
public :: merraland_struc
```

40.1.1 init_MERRALAND (Source File: merraland_forcingMod.F90)

REVISION HISTORY:

```
12 Oct 2009: Eric Kemp
22 Jul 2010: David Mocko, changed to hourly forcing
```

INTERFACE:

```
subroutine init_MERRALAND(findex)
```

USES:

```
  use LIS_coreMod
  use LIS_timeMgrMod
  use LIS_logMod

  implicit none
!AGRUMENTS:
  integer, intent(in) :: findex
```

DESCRIPTION:

Defines the native resolution of the input forcing for MERRA-Land data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). The routines invoked are:

readcrd_merraland (40.1.4)
reads the runtime options specified for MERRA-Land data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

40.1.2 read_merraland (Source File: read_merraland.F90)

REVISION HISTORY:

```
12 Oct 2009: Eric Kemp, Initial code
22 Jul 2010: David Mocko, changed to hourly forcing
 5 Apr 2013: Sujay Kumar, updated for the direct use of files from GES-DISC
```

INTERFACE:

```
subroutine read_merraland(n,order,findex,&
     slvname, flxname, radname, mldname, lndname, &
     merraforc,ferror)
```

USES:

```
  use LIS_coreMod, only : LIS_rc,LIS_domain, LIS_masterproc
  use LIS_logMod
  use LIS_FORC_AttributesMod
  use LIS_metforcingMod, only: LIS_forc
  use merraland_forcingMod, only : merraland_struc
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif

  implicit none
```

ARGUMENTS:

```
  integer, intent(in)          :: n
  integer, intent(in)          :: order
  integer, intent(in)          :: findex
  character(len=*), intent(in) :: slvname
  character(len=*), intent(in) :: flxname
  character(len=*), intent(in) :: radname
  character(len=*), intent(in) :: mldname
  character(len=*), intent(in) :: lndname
  real                         :: merraforc(&
       merraland_struc(n)%nvars,24,&
       LIS_rc%lnc(n)*LIS_rc%lnr(n))
  integer, intent(out)         :: ferror
```

DESCRIPTION:

For the given time, reads parameters from MERRA-Land data, transforms into 9 LIS forcing parameters and interpolates to the LIS domain.

MERRALAND FORCING VARIABLES (unless noted, fields are 1-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)

The arguments are:

order flag indicating which data to be read (order=1, read the previous 1 hourly instance, order=2, read the next 1 hourly instance)

n index of the nest

name name of the 1 hour MERRA-Land analysis file

tscount time step count

ferror return error code (0 indicates success)

The routines invoked are:

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

---

40.1.3 interp_merraland_var (Source File: read_merraland.F90)

INTERFACE:

```
subroutine interp_merraland_var(n,findex, input_var,  var_index, &
     pcp_flag, merraforc)
```

USES:

```
use LIS_coreMod
use LIS_logMod
use merraland_forcingMod, only : merraland_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
integer, intent(in)    :: findex
real,    intent(in)    :: input_var(merraland_struc(n)%ncold, &
     merraland_struc(n)%nrold, 24)
integer, intent(in)    :: var_index
logical, intent(in)    :: pcp_flag
real,    intent(inout) :: merraforc(merraland_struc(n)%nvars, &
     24, LIS_rc%lnc(n)*LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine spatially interpolates a MERRA-Land field to the LIS running domain

---

40.1.4 readcrd_merraland (Source File: readcrd_merraland.F90)

REVISION HISTORY:

```
12 Oct 2009; Eric Kemp, Initial code
22 Jul 2010: David Mocko, changed to hourly forcing
```

INTERFACE:

```
subroutine readcrd_merraland()
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod
use merraland_forcingMod, only : merraland_struc
```

DESCRIPTION:

This routine reads the options specific to MERRA-Land forcing from the LIS configuration file.

40.2 Fortran: Module Interface reset_merraland (Source File: reset_merraland.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine reset_merraland
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use LIS_timeMgrMod, only : LIS_date2time
use merraland_forcingMod
```

DESCRIPTION:

Routine to cleanup allocated structures for MERRALAND forcing.

40.2.1 timeinterp_merraland (Source File: timeinterp_merraland.F90)

REVISION HISTORY:

```
12 Oct 2009: Eric Kemp, initial code
22 Jul 2010: David Mocko, changed to hourly forcing
31 May 2013: David Mocko, changes to match MERRA-Land interpolation strategy
```

INTERFACE:

```
subroutine timeinterp_merraland(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod,          only : LIS_rc, LIS_domain, LIS_localPet
use LIS_metforcingMod,   only : LIS_forc, LIS_FORC_Base_State
use LIS_FORC_AttributesMod
use LIS_constantsMod,     only : LIS_CONST_SOLAR
use LIS_timeMgrMod,       only : LIS_tick
use LIS_logMod,           only : LIS_logunit, LIS_verify, LIS_endrun
use merraland_forcingMod,      only : merraland_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous 1 hourly value is used. All other variables are linearly interpolated between the 1 hourly blocks.
The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

zterp (5.3.19)
zenith-angle based interpolation

---

41 NAM242

The North American Mesoscale Forecast System (NAM) is the North American operational weather forecast model developed at the Environmental Modeling Center (EMC) of NOAA/NCEP.
See http://www.emc.ncep.noaa.gov/index.php?branch=NAM
This reader supports the "NAM 242 AWIPS Grid – Over Alaska" product in GRIB2 format with filenames of the form fh.xxxx_tl.press_gr.awp242. See http://www.nco.ncep.noaa.gov/pmb/products/nam/

41.0.2 create_nam242f9_filename (Source File: create_nam242f9_filename.F90)

INTERFACE:

```
subroutine create_nam242f9_filename(option, name00, name03, &
     namdir, yr, mo, da, hr,status )
```

USES:

```
! Dagang add begin
use LIS_timeMgrMod, only : LIS_tick, LIS_date2time
use LIS_coreMod,    only : LIS_rc
use LIS_logMod,     only : LIS_logunit
! Dagang add end

implicit none
```

ARGUMENTS:

```
integer,          intent(in)    :: option
character(len=*), intent(in)    :: namdir
integer, intent(in)             :: yr, mo, da, hr
character(len=*), intent (out)  :: name00
character(len=*), intent (out)  :: name03
integer                         :: status
```

DESCRIPTION:

This subroutine puts together 9hr forecast NAM file name
The arguments are:

option bookend option (1 or 2)

name00 name of the timestamped 9hr NAM file

name03 name of the timestamped 9hr NAM file

namdir Name of the NAM directory

yr year

mo month

da day of month

hr hour of day

status flag indicating if the 9hr forecast is available.

41.0.3 create_nam242filename (Source File: create_nam242filename.F90)

INTERFACE:

```
subroutine create_nam242filename(option, name00, name03, name06, &
     F06flag, namdir, yr, mo, da, hr )
```

USES:

```
   ! Dagang add begin
   use LIS_timeMgrMod, only : LIS_tick, LIS_date2time
   use LIS_coreMod,    only : LIS_rc
   use LIS_logMod,     only : LIS_logunit
   ! Dagang add end

   implicit none
```

ARGUMENTS:

```
   integer,          intent(in)    :: option
   character(len=*), intent(in)    :: namdir
   integer, intent(in)             :: yr, mo, da, hr
   character(len=*), intent (out)  :: name00
   character(len=*), intent (out)  :: name03
   character(len=*), intent (out)  :: name06
   logical                         :: F06flag
```

DESCRIPTION:

This subroutine puts together NAM file names for different (0,3,6hr) forecast instances
The arguments are:

option bookend option (1 or 2)

namdir Name of the NAM directory

yr year

mo month

da day of month

hr hour of day

name00 name of the NAM file for reading instantaneous fields

name03 name of the 3hr NAM file for reading time averaged fields

name06 name of the 6hr NAM file for reading time averaged fields

F06flag flag to indicate if 6hr forecast data is required for this interval

---

41.1 Fortran: Module Interface finalize_nam242 (Source File: finalize_nam242.F90)

REVISION HISTORY:

```
Sep 2012: NOHRSC/NOAA: Initial specification
```

INTERFACE:

```
subroutine finalize_nam242(findex)
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use nam242_forcingMod
```

DESCRIPTION:

Routine to cleanup allocated structures for NAM forcing.
The arguments are:

findex index of the forcing

41.1.1 get_nam242 (Source File: get_nam242.F90)

REVISION HISTORY:

```
    Sep 2012: NOHRSC/NOAA: Initial specification
 12 Mar 2013: James Geiger: Committed into LIS
 10 Apr 2013: James Geiger: Rewrote to use GRIBAPI
```

INTERFACE:

```
subroutine get_nam242(n, findex)
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use LIS_metforcingMod, only : LIS_forc
use LIS_timeMgrMod,    only : LIS_tick, LIS_get_nstep
use LIS_logMod,        only : LIS_logunit, LIS_endrun
use nam242_forcingMod, only : nam242_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 3-hrly, NAM forcing. The idea is to open either the 00, 03 and 06 forecast file associated with the most recent NAM assimilation (available every 6 hours). Precipitation rates and radiation fluxes will be taken from the F03 and F06 files, since averages are provided. At the beginning of a simulation, the code reads the most recent past data (nearest 3 hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. The strategy for missing data is to first look for 9hr forecasts and if not go backwards up to 10 days to get forcing at the same time of day.
The arguments are:

n index of the nest

findex index of the forcing source

The routines invoked are:

LIS_tick (3.36.27)
determines the NAM data times

create_nam242filename (41.0.3)
Puts together appropriate file name for 3 hour intervals

create_nam242f9_filename (41.0.2)
Puts together appropriate file name for 9 hour intervals

read_nam242 (41.2.2)
Interpolates NAM data to LIS grid

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

read_nam242_elev (41.2.4)
reads the native elevation of the NAM data to be used for topographic adjustments to the forcing

---

41.2 Fortran: Module Interface nam242_forcingMod (Source File: nam242_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the forcing data from the Global Data Assimilation System (NAM) developed at the Environmental Modeling Center (EMC) of NOAA/NCEP. NAM forcing variables are produced on a quadratic gaussian grid. LIS uses the 00, 03, 06 and as needed, the 09 forecasts. The forecasts are produced at 6 hr intervals.
The implementation in LIS has the derived data type `nam242_struc` that includes the variables that specify the runtime options, and the weights and neighbor information to be used for spatial interpolation. They are described below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

nmif Number of forcing variables in the NAM data

ts Frequency in seconds of the forcing data

namtime1 The nearest, previous 3 hour instance of the incoming data (as a real time).

namtime2 The nearest, next 3 hour instance of the incoming data (as a real time).

findtime1, findtime2 boolean flags to indicate which time is to be read for temporal interpolation.

namdir Directory containing the input data

elevfile File with the elevation definition for the input data.

mi Number of points in the input grid

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n11,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n112,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

USES:

```
implicit none

PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
public :: init_nam242    !defines the native resolution of
                         !the input data
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
public :: nam242_struc
```

41.2.1 init_nam242 (Source File: nam242_forcingMod.F90)

REVISION HISTORY:

```
Sep 2012: NOHRSC/NOAA: Initial specification
```

INTERFACE:

```
subroutine init_nam242(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Defines the native resolution of the input forcing for NAM data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). Based on the NAM data map projection and resolution, this routine sets up the spatial interpolation weights. The dates of the NAM resolution switches are also defined in this routine.
The arguments are:

findex index of the forcing source

The routines invoked are:

readcrd_nam242 (41.2.5)
reads the runtime options specified for NAM data

LIS_date2time (3.36.24)
converts date to the real time format

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

41.2.2 read_nam242 (Source File: read_nam242.F90)

REVISION HISTORY:

```
   Sep 2012: NOHRSC/NOAA: Initial specification
12 Mar 2013: James Geiger: Committed into LIS
10 Apr 2013: James Geiger: Rewrote to use GRIBAPI
```

INTERFACE:

```
subroutine read_nam242(n, findex, order, name00, name03, name06, &
                       F06flag, ferror, try)
```

USES:

```
  use LIS_coreMod,        only : LIS_rc, LIS_domain
  use LIS_metforcingMod,  only : LIS_forc
  use LIS_logMod,         only : LIS_logunit, LIS_endrun, LIS_verify
  use nam242_forcingMod,  only : nam242_struc

#if (defined USE_GRIBAPI)
  use grib_api
#endif

  implicit none
```

ARGUMENTS:

```
integer, intent(in)          :: n
integer, intent(in)          :: findex
integer, intent(in)          :: order
character(len=*), intent(in) :: name00
character(len=*), intent(in) :: name03
character(len=*), intent(in) :: name06
logical, intent(in)          :: F06flag
integer, intent(out)         :: ferror
integer, intent(inout)       :: try
```

DESCRIPTION:

For the given time, reads parameters from NAM forecast datasets, transforms into 9 LIS forcing parameters and interpolates to the LIS domain.
The arguments are:

n index of the nest

findex index of the forcing source

order flag indicating which data to be read (order=1, read for the previous 3hr bookend, order=2, read for the next 3 hr bookend)

name00 name of the instantaneous forecast file

name03 name of the 3 hour NAM forecast file

name06 name of the 6 hour NAM forecast file

F06flag flag to indicate if 6hr forecast data is required for this interval

ferror flag to indicate success of the call (=1 indicates success)

try index of the tries (in case of missing data)

The routines invoked are:

interp_nam242 (41.2.3)
spatially interpolates a NAM variable

41.2.3 interp_nam242 (Source File: read_nam242.F90)

INTERFACE:

```
subroutine interp_nam242(n,findex,iparam,nnam,f,lb,lis_gds,nc,nr,varfield)
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use nam242_forcingMod, only : nam242_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
integer, intent(in) :: iparam
integer, intent(in) :: nnam
real, intent(out)   :: f(nnam)
logical*1           :: lb(nnam)
real                :: lis_gds(50)
integer, intent(in) :: nc
integer, intent(in) :: nr
real, intent(out)   :: varfield(nc,nr)
```

DESCRIPTION:

This subroutine interpolates a given NAM field to the LIS grid. The arguments are:

n index of the nest

n index of the forcing source

iparam paramter indicator

nnam number of elements in the input grid

f input data array to be interpolated

lb input bitmap

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

---

41.2.4 read_nam242_elev (Source File: read_nam242_elev.F90)

REVISION HISTORY:

```
Sep 2012: NOHRSC/NOAA: Initial specification
```

INTERFACE:

```
subroutine read_nam242_elev(n, findex, change)
```

USES:

```
use LIS_coreMod,       only : LIS_rc, LIS_domain
use LIS_metforcingMod, only : LIS_forc
use LIS_fileIOMod,     only : LIS_read_param
use LIS_logMod,        only : LIS_logunit, LIS_getNextUnitNumber, &
                              LIS_releaseUnitNumber, LIS_endrun
use nam242_forcingMod, only : nam242_struc
 use gaussian_mod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates NAM model elevation to the LIS grid. The data will be used to perform any topographical adjustments to the forcing.
The arguments are:

n index of the nest

The routines invoked are:

LIS_readData (3.14.1)
Abstract method to read the elevation of the forcing data in the same map projection used in LIS.

41.2.5 readcrd_nam242 (Source File: readcrd_nam242.F90)

REVISION HISTORY:

```
Sep 2012: NOHRSC/NOAA: Initial specification
```

INTERFACE:

```
subroutine readcrd_nam242()
```

USES:

```
use ESMF
use LIS_coreMod,       only : LIS_rc, LIS_config
use LIS_logMod,        only : LIS_logunit
use nam242_forcingMod, only: nam242_struc

implicit none
```

DESCRIPTION:

This routine reads the options specific to NAM242 forcing from the LIS configuration file.

41.3 Fortran: Module Interface reset_nam242 (Source File: reset_nam242.F90)

REVISION HISTORY:

```
Sep 2012: NOHRSC/NOAA: Initial specification
```

INTERFACE:

```
subroutine reset_nam242
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use nam242_forcingMod
```

DESCRIPTION:

Routine to cleanup allocated structures for NAM forcing.

41.3.1 timeinterp_nam242 (Source File: timeinterp_nam242.F90)

REVISION HISTORY:

```
Sep 2012: NOHRSC/NOAA: Initial specification
```

INTERFACE:

```
subroutine timeinterp_nam242(n, findex)
```

USES:

```
use ESMF
use LIS_coreMod,       only : LIS_rc, LIS_domain
use LIS_FORC_AttributesMod
use LIS_timeMgrMod,    only : LIS_time2date, LIS_tick
use LIS_constantsMod,  only : LIS_CONST_SOLAR
use LIS_metforcingMod, only : LIS_forc, LIS_FORC_State
use LIS_logMod,        only : LIS_logunit, LIS_verify, LIS_endrun
use nam242_forcingMod, only : nam242_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous 3 hourly value is used. All other variables are linearly interpolated between the 3 hourly blocks.
The arguments are:

n index of the nest

findex index of the forcing source

The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

LIS_tick (3.36.27)
advances or retracts time by the specified amount

zterp (5.3.19)
zenith-angle based interpolation

---

42 NARR forcing

This section describes the implementation of the NARR data.

42.1 Fortran: Module Interface finalize_narr (Source File: finalize_narr.F90)

REVISION HISTORY:

```
17 March 2009; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine finalize_narr
```

USES:

DESCRIPTION:

Routine to cleanup allocated structures for GDAS forcing.

42.1.1 get_narr (Source File: get_narr.F90)

REVISION HISTORY:

```
30 APR 2009; Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine get_narr(n, findex)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_metforcingMod, only : LIS_forc
use LIS_timeMgrMod,     only : LIS_get_nstep, LIS_tick
use LIS_logMod,         only : LIS_logunit
use narr_forcingMod,  only : narr_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

42.1.2 interp_narrfield (Source File: interp_narrfield.F90)

REVISION HISTORY:

```
30 APR 2009: Sujay Kumar; Initial Specification
25 Jan 2012: Sujay Kumar; switched to the use of grib_api
```

INTERFACE:

```
subroutine interp_narrfield(n, narrfile, iv,pds5,pds6,pds7,metdata)
```

USES:

```
  use LIS_coreMod,        only : LIS_rc, LIS_domain
  use LIS_metforcingMod,  only : LIS_forc
  use LIS_logMod,         only : LIS_logunit,LIS_warning, LIS_verify
  use narr_forcingMod,    only : narr_struc

#if (defined USE_GRIBAPI)
  use grib_api
#endif
  implicit none
```

ARGUMENTS:

```
  integer,          intent(in) :: n
  character(len=*), intent(in) :: narrfile
  integer,          intent(in) :: iv
  integer,          intent(in) :: pds5
  integer,          intent(in) :: pds6
  integer,          intent(in) :: pds7
  real                         :: metdata(LIS_rc%ngrid(n))
```

DESCRIPTION:

42.2 Fortran: Module Interface narr_forcingMod (Source File: narr_forcingMod.F90)

USES:

```
  implicit none

  PRIVATE
 ------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
 ------------------------------------------------------------------------------
  public :: init_NARR      !defines the native resolution of
                                  !the input data
 ------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
 ------------------------------------------------------------------------------
  public :: narr_struc
```

42.2.1 init_narr (Source File: narr_forcingMod.F90)

REVISION HISTORY:

```
30 APR 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine init_narr(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep

implicit none

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

DESCRIPTION:

42.2.2 read_narr (Source File: read_narr.F90)

REVISION HISTORY:

```
30 APR 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_narr(n, m, order, narrfile)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_metforcingMod, only : LIS_forc
use LIS_logMod,  only : LIS_logunit, LIS_endrun
use narr_forcingMod, only : narr_struc

implicit none
```

ARGUMENTS:

```
integer,   intent(in) :: n
integer,   intent(in) :: m
integer,   intent(in) :: order
character(len=*)      :: narrfile
```

DESCRIPTION:

42.2.3 readcrd_narr (Source File: readcrd_narr.F90)

REVISION HISTORY:

```
30 APR 2009; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcrd_narr()
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc, LIS_config
use LIS_logMod,    only : LIS_logunit, LIS_verify
use narr_forcingMod, only: narr_struc
```

DESCRIPTION:

This routine reads the options specific to NARR forcing from the LIS configuration file.

42.2.4 timeinterp_narr (Source File: timeinterp_narr.F90)

REVISION HISTORY:

```
30 APR 2009: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine timeinterp_narr(n, findex)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_logMod,     only : LIS_verify
use LIS_FORC_AttributesMod
use LIS_metforcingMod, only : LIS_FORC_Base_State, LIS_forc
use narr_forcingMod,  only : narr_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine temporally interpolates the forcing variables from NARR data to the LIS timestep.

---

43 NLDAS1

The atmospheric forcing used in the North American Land Data Assimilation System (NLDAS) [5] features products at an hourly 0.125° spatial resolution, over the continental United States. The SW↓ is generated from a 1/2° product derived at the University of Maryland from NOAA's Geostationary Operational Environmental Satellites (GOES). The LW↓ is derived from 3 hourly NCEP Eta Data Assimilation System (EDAS) output fields [12], and from 3 hourly and 6 hourly Eta mesoscale model forecast fields when EDAS data is unavailable. This dataset is for NLDAS Phase 1 (NLDAS-1); see here for more: http://ldas.gsfc.nasa.gov/nldas/NLDAS1forcing.php

---

43.0.5 apply_conusmask (Source File: apply_conusmask.F90)

INTERFACE:

```
subroutine apply_conusmask(n, nc,nr,varfield)
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use nldas1_forcingMod,  only : nldas1_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: n
integer, intent(in)   :: nc
integer, intent(in)   :: nr
real, intent(inout)   :: varfield(nc,nr)
```

DESCRIPTION:

43.1 Fortran: Module Interface finalize_nldas1 (Source File: finalize_nldas1.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine finalize_nldas1(findex)
```

USES:

```
use LIS_coreMod,      only : LIS_rc
use nldas1_forcingMod, only : nldas1_struc
```

DESCRIPTION:

Routine to cleanup nldas1 forcing related memory allocations.

43.1.1 get_nldas1 (Source File: get_nldas1.F90)

REVISION HISTORY:

```
27 Apr 2000: Initial Specification
02 Feb 2004: Sujay Kumar ; Initial Version in LIS
06 Mar 2007: Kristi Arsenault; Implemented EDAS-Elevation Correction Var
15 Feb 2012: K. Arsenault; Accommodate GES DISC, NCEP filename conventions
14 Mar 2014: David Mocko: Updated filename options for config file
```

INTERFACE:

```
subroutine get_nldas1(n,findex)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_timeMgrMod,     only : LIS_tick
use LIS_metforcingMod,  only : LIS_forc
use LIS_logMod,         only : LIS_logunit, LIS_endrun
use nldas1_forcingMod,  only : nldas1_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates hourly, NLDAS-1 forcing. At the beginning of a simulation, the code reads the most recent past data (nearest hourly interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the NLDAS-1 data times

get_nldas1_filename (43.1.2)
Puts together appropriate timestamped filename

read_nldas1 (43.2.4)
Interpolates NLDAS-1 data to LIS grid

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

43.1.2 get_nldas1_filename (Source File: get_nldas1_filename.F90)

REVISION HISTORY:

```
04 Sep 2001: Brian Cosgrove; Use of NASA data enabled, updated
             reading of data directory structure to read new format
15 Feb 2012: K. Arsenault; Accommodate GES DISC, NCEP filename conventions
14 Mar 2014: David Mocko: Updated filename options for config file
```

INTERFACE:

```
subroutine get_nldas1_filename(n,filename,nldas1dir,yr,mo,da,doy,hr)

  use nldas1_forcingMod,only : nldas1_struc

  implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
character(len=*),intent(out)  :: filename
character(len=*), intent(in)  :: nldas1dir
integer,    intent(in)        :: yr,mo,da,doy,hr
```

DESCRIPTION:

This subroutine puts together NLDAS-1 filename structures for 1 hour file intervals. Two options for naming file conventions are given:
1: NASA GES DISC filename type (from http://disc.sci.gsfc.nasa.gov/hydrology/data-holdings)
2: NASA-Original NLDAS-1 filename type (also NCEP-style)

The arguments are:

nldas1dir Name of the NLDAS-1 directory

yr year

mo month

da day of month

doy Julian day of year (needed for subdirectory structure)

hr hour of day

filename name of the timestamped NLDAS-1 file

43.1.3 interp_nldas1 (Source File: interp_nldas1.F90)

INTERFACE:

```
subroutine interp_nldas1(n, findex, pcp_flag, nldas1,f,lb,lis_gds,nc,nr, &
     varfield)
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use nldas1_forcingMod,  only : nldas1_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: n
integer, intent(in)   :: findex
logical, intent(in)   :: pcp_flag
integer, intent(in)   :: nldas1
real, intent(in)      :: f(nldas1)
logical*1, intent(in) :: lb(nldas1)
real, intent(in)      :: lis_gds(50)
integer, intent(in)   :: nc
integer, intent(in)   :: nr
real, intent(inout)   :: varfield(nc,nr)
```

DESCRIPTION:

This subroutine interpolates a given NLDAS1 field to the LIS grid. The arguments are:

n index of the nest

kpds grib decoding array

ngdas number of elements in the input grid

f input data array to be interpolated

lb input bitmap

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

neighbor_interp (5.3.10)
spatially interpolate the forcing data using neighbor interpolation

---

43.1.4 nldas1_ec_removal (Source File: nldas1_ec_removal.F90)

REVISION HISTORY:

```
20 Oct 2006: Kristi Arsenault; Adapted elevation correction code
             to remove such a correction from NLDAS1 fields
```

INTERFACE:

```
subroutine nldas1_ec_removal( nest, point, force_tmp, force_hum, &
                              force_lwd, force_prs )
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use nldas1_forcingMod,   only : nldas1_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Removes Temperature, Pressure, Humidity and Longwave Radiation forcing correction.
The corrections are based on the lapse-rate and hypsometric adjustments to these variables described in Cosgrove et. al (2003).
Cosgrove, B.A. et.al, Real-time and retrospective forcing in the North American Land Data Assimilation (NLDAS1) project, Journal of Geophysical Research, 108(D22), 8842, DOI: 10.1029/2002JD003118, 2003.
The arguments are:

nest index of the domain or nest.

point index of the grid point

force_tmp temperature value for the grid point

force_hum specific humidity for the grid point

force_lwd downward longwave radiation for the grid point

force_prs surface pressure for the grid point

43.2 Fortran: Module Interface nldas1_forcingMod (Source File: nldas1_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the forcing data used in the North American Land Data Assimilation System (NLDAS; Cosgrove et al.(2003)). The variables are produced at 0.125 degree spatial resolution, and at hourly intervals, over the continental United States. The SW↓ is generated from a 1/2° product derived at the University of Maryland from NOAA's Geostationary Operational Environmental Satellites (GOES). The LW↓ is derived from 3 hourly NCEP Eta Data Assimilation System (EDAS) output fields [12], and from 3 hourly and 6 hourly Eta mesoscale model forecast fields when EDAS data is unavailable.
This dataset represents NLDAS Phase 1 (NLDAS-1); for more see: http://ldas.gsfc.nasa.gov/nldas/NLDAS1forcing.php
Cosgrove, B. Real-time and retrospective forcing in the North American Land Data Assimilation System (NLDAS) project. Journal of Geophysical Research, 108 (D22), 8842, DOI:10.1029/2002JD003118
The implemenatation in LIS has the derived data type `nldas1_struc` that includes the variables that specify the runtime options, and the weights and neighbor information to be used for spatial interpolation. They are described below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

nldas1time1 The nearest, previous hourly instance of the incoming data (as a real time).

nldas1time2 The nearest, next hourly instance of the incoming data (as a real time).

nldas1dir Directory containing the input data

nldas1_filesrc Center(GES-DISC—NCEP)-based NLDAS-1 filename source option

elevfile File with the elevation definition for the input data.

mi Number of points in the input grid

orig_ediff Original NLDAS-1 elevation difference field

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n111,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n122,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

rlat3 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for nearest neighbor interpolation)

rlon3 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for nearest neighbor interpolation)

n113 Array containing the neighbor information of the input grid for each grid point in LIS, for nearest neighbor interpolation.

findtime1, findtime2 boolean flags to indicate which time is to be read for temporal interpolation.

REVISION HISTORY:

```
02 Feb 2004: Sujay Kumar; Initial Specification
20 Oct 2007: Kristi Arsenault; Changed to EDAS elev file for new diff
15 Feb 2012: K. Arsenault; Accommodate GES DISC, NCEP filename conventions
14 Mar 2014: David Mocko: Added NLDAS-1 precipitation and radiation
                             field choices to the lis.config file
                          Changed flag from "1"/"2" to "NCEP"/"GES-DISC".
```

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init_nldas1      !defines the native resolution of
                                !the input data
```

PUBLIC TYPES:

```
public :: nldas1_struc
```

43.2.1 init_NLDAS1 (Source File: nldas1_forcingMod.F90)

INTERFACE:

```
subroutine init_NLDAS1(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_timeMgrMod, only : LIS_update_timestep
use LIS_logMod,     only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use map_utils,      only : ij_to_latlon

implicit none

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

DESCRIPTION:

Defines the native resolution of the input forcing for NLDAS-1 data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). The routines invoked are:

readcrd_nldas1 (43.2.5)
reads the runtime options specified for NLDAS-1 data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

neighbor_interp_input (5.3.11)
computes the neighbor, weights for nearest neighbor interpolation

read_edas_elev (43.2.3)
reads the native elevation of the EDAS data to be used for topographic adjustments to the metforcing data

---

43.2.2 orig_elevdiff (Source File: orig_elevdiff.F90)

REVISION HISTORY:

```
20 Oct 2006: Kristi Arsenault; Adapted elevation correction code
             to remove such a correction from NLDAS fields
```

INTERFACE:

```
subroutine orig_elevdiff(nest)
```

USES:

```
use nldas1_forcingMod, only : nldas1_struc
use LIS_logmod, only       : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Open and read in original NLDAS elevation difference file.
The arguments are:

nest index of the domain or nest.

The routines invoked are:

LIS_getNextUnitNumber (3.21.10)
obtain the next available unit number

LIS_releaseUnitNumber (3.21.11)
release the unit number after use.

43.2.3 read_edas_elev (Source File: read_edas_elev.F90)

REVISION HISTORY:

```
17 Dec 2004; Sujay Kumar; Initial Specificaton
20 Dec 2006; Kristi Arsenault; Changed to read EDAS elevation only
3  Dec 2007; Sujay Kumar; Added the abstract method to read the
             EDAS data.
```

INTERFACE:

```
subroutine read_edas_elev(n,findex)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_metforcingMod, only : LIS_forc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,      only : LIS_readData
use nldas1_forcingMod,   only : nldas1_struc
implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates NLDAS1-EDAS model elevation to the LIS grid. The data will be used to perform any topographical adjustments to the forcing.
The arguments are:

n index of the nest

The routines invoked are:

LIS_readData (3.14.1)
abstract method to read EDAS elevation data in the map projection used in the LIS model grid.

43.2.4 read_nldas1 (Source File: read_nldas1.F90)

REVISION HISTORY:

```
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
20 Oct 2007: Kristi Arsenault; Updated NLDAS-1 Code to Read EDAS Height Field
25 Jan 2012: Sujay Kumar; Switched to the use of grib-api library
15 Feb 2012: Kristi Arsenault; Accommodate GES DISC filename conventions
14 Mar 2014: David Mocko: Fixed the use of NLDAS-1 GOES SWdown radiation.
                          Added NLDAS-1 precipitation and radiation
                          field choices to the lis.config file.
```

INTERFACE:

```
subroutine read_nldas1(n, findex, order, name, ferror)
```

USES:

```
  use LIS_coreMod, only        : LIS_rc, LIS_domain
  use LIS_metforcingMod, only  : LIS_forc
  use LIS_logMod, only         : LIS_logunit, LIS_verify, LIS_warning, LIS_endrun
  use nldas1_forcingMod, only : nldas1_struc
#if (defined USE_GRIBAPI)
  use grib_api
#endif

  implicit none
```

ARGUMENTS:

```
  integer, intent(in)      :: n
  integer, intent(in)      :: findex
  integer, intent(in)      :: order
  character*80, intent(in) :: name
  integer, intent(out)     :: ferror
```

DESCRIPTION:

For the given time, reads parameters from NLDAS-1 data, transforms into 9 LIS forcing parameters and interpolates to the LIS spatial domain. The routine reads the gauge-corrected precip and GOES-based radiation fields by default. If these fields are not present, then the code defaults to the corresponding EDAS fields.
The arguments are:

n index of the nest

findex sequential index of the NLDAS-1 forcing among various forcings

order flag indicating which data to be read (order=1, read the previous hourly instance, order=2, read the next hourly instance)

name name of the NLDAS-1 forcing file

ferror flag to indicate success of the call (=1 indicates success)

The routines invoked are:

interp_nldas1 (43.1.3)
spatially interpolates a NLDAS-1 variable

nldas1_ec_removal (43.1.4)
removes the native elevation correction applied to the NLDAS fields. This is performed so that the elevation correction can be reapplied at the spatial resolution of the LIS domain

apply_conusmask (43.0.5)
applies the CONUS landmask to exclude NLDAS-1 data over Canada and Mexico.

43.2.5 readcrd_nldas1 (Source File: readcrd_nldas1.F90)

REVISION HISTORY:

```
02 Feb 2004; Sujay Kumar, Initial Code
04 Mar 2007; Kristi Arsenault, Implemented EDAS Elevation Correction
15 Feb 2012: K. Arsenault; Accommodate GES DISC, NCEP filename conventions
14 Mar 2014: David Mocko: Added NLDAS-1 precipitation and radiation
                             field choices to the lis.config file.
                           Changed flag from "1"/"2" to "NCEP"/"GES-DISC".
```

INTERFACE:

```
subroutine readcrd_nldas1()
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod,  only : LIS_logunit, LIS_verify, LIS_endrun
use nldas1_forcingMod, only : nldas1_struc
```

DESCRIPTION:

This routine reads the options specific to NLDAS-1 forcing from the LIS configuration file.

43.2.6 reset_nldas1 (Source File: reset_nldas1.F90)

REVISION HISTORY:

```
02Feb2004; Sujay Kumar, Initial Code
04Mar2007; Kristi Arsenault, Implemented EDAS Elevation Correction
```

INTERFACE:

```
subroutine reset_nldas1()
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use nldas1_forcingMod, only : nldas1_struc
```

DESCRIPTION:

This routine reads the options specific to NLDAS1 forcing from the LIS configuration file.

43.2.7 timeinterp_nldas1 (Source File: timeinterp_nldas1.F90)

REVISION HISTORY:

```
02Feb2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine timeinterp_nldas1(n, findex)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_constantsMod, only  : LIS_CONST_SOLAR
use LIS_FORC_AttributesMod
use LIS_metforcingMod, only : LIS_FORC_Base_State, LIS_forc
use LIS_timeMgrMod, only : LIS_tick, LIS_time2date
use LIS_logMod, only :LIS_logunit, LIS_verify
use nldas1_forcingMod, only : nldas1_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous 3 hourly value is used. All other variables are linearly interpolated between the 3 hourly blocks.
The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

LIS_tick (3.36.27)
advances or retracts time by the specified amount

zterp (5.3.19)
zenith-angle based interpolation

---

44 NLDAS2

The atmospheric forcing used in the North American Land Data Assimilation System Phase 2 (NLDAS-2) features products at an hourly 0.125° spatial resolution, from 25 to 53 North and from -125 to -67 West, from January 1979 to present. For details, see Xia et al. (2012, JGR) and visit:
http://ldas.gsfc.nasa.gov/nldas/NLDAS2forcing.php

44.1 Fortran: Module Interface finalize_nldas2 (Source File: finalize_nldas2.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
24 Aug 2007: Chuck Alonge; Modified for use with NLDAS2 data
```

INTERFACE:

```
subroutine finalize_nldas2(findex)
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use nldas2_forcingMod, only : nldas2_struc
```

DESCRIPTION:

Routine to cleanup nldas2 forcing related memory allocations.

44.1.1 gesdisc_nldas2filea (Source File: get_gesdisc_filenames.F90)

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
24 Aug 2007: Chuck Alonge; Modified for use with NLDAS-2 data
22 Jan 2012: K. Arsenault; Accommodate GES DISC, NCEP/EMC filename conventions
```

INTERFACE:

```
subroutine gesdisc_nldas2filea(filename,nldas2dir,yr,mo,da,doy,hr)

  implicit none
```

ARGUMENTS:

```
  character*80, intent(out) :: filename
  character*40, intent(in)  :: nldas2dir
  integer, intent(in)       :: yr,mo,da,doy,hr
```

DESCRIPTION:

This subroutine puts together GES DISC NLDAS-2 "A" file name for 1 hour file intervals
The arguments are:

nldas2dir Name of the NLDAS-2 directory

yr year

mo month

da day of month

doy Julian day of year (needed for subdirectory structure)

hr hour of day

filename name of the timestamped GES DISC NLDAS-2 file

44.1.2 gesdisc_nldas2fileb (Source File: get_gesdisc_filenames.F90)

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
24 Aug 2007: Chuck Alonge; Modified for use with NLDAS-2 data
22 Jan 2012: K. Arsenault; Accommodate GES DISC, GES DISC filename conventions
```

INTERFACE:

```
subroutine gesdisc_nldas2fileb(filename,nldas2dir,yr,mo,da,doy,hr)

  implicit none
```

ARGUMENTS:

```
  character*80, intent(out) :: filename
  character*40, intent(in)  :: nldas2dir
  integer, intent(in)       :: yr,mo,da,doy,hr
```

DESCRIPTION:

This subroutine puts together GES DISC NLDAS-2 "B" file name for 1 hour file intervals
The arguments are:

nldas2dir Name of the NLDAS-2 directory

yr year

mo month

da day of month

doy Julian day of year (needed for subdirectory structure)

hr hour of day

filename name of the timestamped GES DISC NLDAS-2 file

44.1.3 ncep_nldas2filea (Source File: get_ncep_filenames.F90)

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
24 Aug 2007: Chuck Alonge; Modified for use with NLDAS-2 data
22 Jan 2012: K. Arsenault; Accommodate GES DISC, NCEP filename conventions
```

INTERFACE:

```
subroutine ncep_nldas2filea(filename,nldas2dir,yr,mo,da,hr)

  implicit none
```

ARGUMENTS:

```
character*80, intent(out) :: filename
character*40, intent(in)  :: nldas2dir
integer, intent(in)       :: yr,mo,da,hr
```

DESCRIPTION:

This subroutine puts together NCEP NLDAS-2 "A" file name for 1 hour file intervals
The arguments are:

nldas2dir Name of the NLDAS-2 directory

yr year

mo month

da day of month

hr hour of day

filename name of the timestamped NCEP NLDAS-2 file

44.1.4 ncep_nldas2fileb (Source File: get_ncep_filenames.F90)

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
24 Aug 2007: Chuck Alonge; Modified for use with NLDAS-2 data
22 Jan 2012: K. Arsenault; Accommodate GES DISC, NCEP filename conventions
```

INTERFACE:

```
subroutine ncep_nldas2fileb(filename,nldas2dir,yr,mo,da,hr)

implicit none
```

ARGUMENTS:

```
character*80, intent(out) :: filename
character*40, intent(in)  :: nldas2dir
integer, intent(in)       :: yr,mo,da,hr
```

DESCRIPTION:

This subroutine puts together NCEP NLDAS-2 "B" file name for 1 hour file intervals
The arguments are:

nldas2dir Name of the NLDAS-2 directory

yr year

mo month

da day of month

hr hour of day

filename name of the timestamped NCEP NLDAS-2 file

44.1.5 get_nldas2 (Source File: get_nldas2.F90)

REVISION HISTORY:

```
02 Feb 2004: Sujay Kumar; Initial Version in LIS
22 Aug 2007: Chuck Alonge; Updated for NLDAS2 Forcing
22 Jan 2012: K. Arsenault; Accommodate GESDISC, NCEP filename conventions
14 Mar 2014: David Mocko: Trim writing of filenames to log file
```

INTERFACE:

```
subroutine get_nldas2(n,findex)
```

USES:

```
use LIS_coreMod, only        : LIS_rc, LIS_domain
use LIS_timeMgrMod,     only : LIS_tick
use LIS_metforcingMod,  only : LIS_forc
use LIS_logMod,         only : LIS_logunit, LIS_endrun
use nldas2_forcingMod,  only : nldas2_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates hourly, NLDAS-2 forcing (NARR based). At the beginning of a simulation, the code reads the most recent past data (nearest hourly interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the NLDAS-2 data times

gesdisc_nldas2filea (44.1.1)
Puts together appropriate timestamped GES DISC filename - a data

gesdisc_nldas2fileb (44.1.2)
Puts together appropriate timestamped GES DISC filename - b data

ncep_nldas2filea (44.1.3)
Puts together appropriate timestamped NCEP/EMC filename - a data

ncep_nldas2fileb (44.1.4)
Puts together appropriate timestamped NCEP/EMC filename - b data

read_nldas2a (44.2.3)
Reads and Interpolates NLDAS-2 A data to LIS grid

read_nldas2b (44.2.5)
Reads and Interpolates NLDAS-2 B data to LIS grid

44.1.6 nldas2_ec_removal (Source File: nldas2_ec_removal.F90)

REVISION HISTORY:

```
20 Oct 2006: Kristi Arsenault; Adapted elevation correction code
             to remove such a correction from NLDAS2 fields
```

INTERFACE:

```
subroutine nldas2_ec_removal( nest, point, force_tmp, force_hum, &
                              force_lwd, force_prs )
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use nldas2_forcingMod,   only : nldas2_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Removes Temperature, Pressure, Humidity and Longwave Radiation forcing correction.
The corrections are based on the lapse-rate and hypsometric adjustments to these variables described in Cosgrove et. al (2003).
Cosgrove, B.A. et.al, Real-time and retrospective forcing in the North American Land Data Assimilation (NLDAS2) project, Journal of Geophysical Research, 108(D22), 8842, DOI: 10.1029/2002JD003118, 2003.
The arguments are:

nest index of the domain or nest.

point index of the grid point

force_tmp temperature value for the grid point

force_hum specific humidity for the grid point

force_lwd downward longwave radiation for the grid point

force_prs surface pressure for the grid point

44.2 Fortran: Module Interface nldas2_forcingMod (Source File: nldas2_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the forcing data used in the North American Land Data Assimilation System Phase II. The variables are produced at 0.125 degree spatial resolution, and at hourly intervals. For more details please view the forcing files manual available at the following URL:
http://ldas.gsfc.nasa.gov//nldas/NLDAS2forcing.php
The implemenatation in LIS has the derived data type `nldas2_struc` that includes the variables that specify the runtime options, and the weights and neighbor information to be used for spatial interpolation. They are described below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

nldas2time1 The nearest, previous hourly instance of the incoming data (as a real time).

nldas2time2 The nearest, next hourly instance of the incoming data (as a real time).

nldas2dir Directory containing the input data

nldas2_filesrc Center(GES-DISC—NCEP)-based NLDAS-2 filename source option

mi Number of points in the input grid

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n111,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n122,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

rlat3 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for nearest neighbor interpolation)

rlon3 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for nearest neighbor interpolation)

n113 Arrays containing the neighbor information of the input grid for each grid point in LIS, for nearest neighbor interpolation.

findtime1, findtime2 boolean flags to indicate which time is to be read for temporal interpolation.

REVISION HISTORY:

```
02 Feb 2004: Sujay Kumar; Initial Specification
24 Aug 2007: Chuck Alonge; Modified for use with NLDAS-2 data
14 Mar 2014: David Mocko: Added CAPE and PET forcing from NLDAS-2
```

USES:

```
implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
 ------------------------------------------------------------------------------
  public :: init_NLDAS2      !defines the native resolution of
                                  !the input data
 ------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
 ------------------------------------------------------------------------------
  public :: nldas2_struc
```

44.2.1 init_NLDAS2 (Source File: nldas2_forcingMod.F90)

INTERFACE:

```
subroutine init_NLDAS2(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_update_timestep
use LIS_logMod,     only : LIS_logunit,LIS_endrun
use LIS_spatialDownscalingMod, only : LIS_init_pcpclimo_native

implicit none

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

DESCRIPTION:

Defines the native resolution of the input forcing for NLDAS-2 data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). The routines invoked are:

readcrd_nldas2 (44.2.7)
reads the runtime options specified for NLDAS-2 data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

read_nldas2_elev (44.2.2)
reads the native elevation of the NLDAS-2 data to be used for topographic adjustments to the forcing

---

44.2.2 read_nldas2_elev (Source File: read_nldas2_elev.F90)

REVISION HISTORY:

```
17Dec2004; Sujay Kumar; Initial Specificaton
24 Aug 2007: Chuck Alonge; Modified for use with NLDAS2 data
```

INTERFACE:

```
subroutine read_nldas2_elev(n,findex)
```

USES:

```
  use LIS_coreMod
  use LIS_metforcingMod
  use LIS_logMod
  use nldas2_forcingMod
  use LIS_fileIOMod
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates NLDAS2 model elevation to the LIS grid. The data will be used to perform any topographical adjustments to the forcing.
The arguments are:

n index of the nest

The routines invoked are:

ij_to_latlon (5.3.9)
computes the lat lon values in LIS grid projection

44.2.3 read_nldas2a (Source File: read_nldas2a.F90)

REVISION HISTORY:

```
    11 Apr 2000: Brian Cosgrove; changed code to use Forcing Mask (With
                 inland water filled in).  Deleted 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
    24 Aug 2007: Chuck Alonge; Modified for use with NLDAS2 data
    25 Jan 2012: Sujay Kumar; Switched to the use of grib-api library
    14 Mar 2014: David Mocko: Added CAPE and PET forcing from NLDAS-2
```

INTERFACE:

```
 subroutine read_nldas2a(n, findex, order, month, name,ferror)
```

USES:

```
   use LIS_coreMod, only        : LIS_rc, LIS_domain
   use LIS_logMod, only         : LIS_logunit, LIS_verify, LIS_warning
   use LIS_metforcingMod, only  : LIS_forc
   use nldas2_forcingMod, only  : nldas2_struc
 #if (defined USE_GRIBAPI)
   use grib_api
 #endif

   implicit none
```

ARGUMENTS:

```
   integer, intent(in)      :: findex
   integer, intent(in)      :: n
   integer, intent(in)      :: order
   integer, intent(out)     :: month
   character*80, intent(in) :: name
   integer, intent(out)     :: ferror
```

DESCRIPTION:

For the given time, reads parameters from NLDAS2 data, transforms into 11 LIS forcing parameters and interpolates to the LIS domain.
The arguments are:

order flag indicating which data to be read (order=1, read the previous hourly instance, order=2, read the next hourly instance)

n index of the nest

name name of the hourly NLDAS2 forecast file

ferror flag to indicate success of the call (=1 indicates success)

The routines invoked are:

interp_nldas2 (44.2.4)
spatially interpolates a NLDAS2 variable

44.2.4 interp_nldas2 (Source File: read_nldas2a.F90)

INTERFACE:

```
subroutine interp_nldas2(n,findex, month, pcp_flag, &
     input_size,input_data,input_bitmap,&
     lis_gds,nc,nr, &
     output_2d)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_spatialDownscalingMod
use nldas2_forcingMod, only :nldas2_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: n
integer, intent(in)   :: findex
integer, intent(in)   :: month
logical, intent(in)   :: pcp_flag
integer, intent(in)   :: input_size
real, intent(in)      :: input_data(input_size)
logical*1, intent(in) :: input_bitmap(input_size)
real, intent(in)      :: lis_gds(50)
integer, intent(in)   :: nc
integer, intent(in)   :: nr
real, intent(inout)   :: output_2d(nc,nr)
```

DESCRIPTION:

This subroutine interpolates a given NLDAS2 field to the LIS grid. The arguments are:

n index of the nest

kpds grib decoding array

input_size number of elements in the input grid

f input data array to be interpolated

input_bitmap input bitmap

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

output_2d output interpolated field

The routines invoked are:

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

neighbor_interp (5.3.10)
spatially interpolate the forcing data using nearest neighbor interpolation

44.2.5 read_nldas2b (Source File: read_nldas2b.F90)

REVISION HISTORY:

```
    11 Apr 2000: Brian Cosgrove; changed code to use Forcing Mask (With
                 inland water filled in).  Deleted 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
    24 Aug 2007: Chuck Alonge; Modified for use with NLDAS2 data
    25 Jan 2012: Sujay Kumar; Switched to the use of grib-api library
    14 Mar 2014: David Mocko: Fixed the use of Forcing "B" file data
                              of NARR meteorology, radiation, precip,
                              and/or forcing height fields
```

INTERFACE:

```
 subroutine read_nldas2b(n, findex, order, name,ferror)
```

USES:

```
   use LIS_coreMod, only        : LIS_rc, LIS_domain
   use LIS_logMod, only         : LIS_logunit, LIS_verify, LIS_warning
   use LIS_metforcingMod, only  : LIS_forc
   use nldas2_forcingMod,only : nldas2_struc
 #if (defined USE_GRIBAPI)
   use grib_api
 #endif

   implicit none
```

ARGUMENTS:

```
   integer, intent(in)      :: findex
   integer, intent(in)      :: n
   integer, intent(in)      :: order
   character*80, intent(in) :: name
   integer, intent(out)     :: ferror
```

DESCRIPTION:

For the given time, reads parameters from NLDAS2 data, transforms into 10 LIS forcing parameters and interpolates to the LIS domain.
The arguments are:

order flag indicating which data to be read (order=1, read the previous hourly instance, order=2, read the next hourly instance)

n index of the nest

name name of the hourly NLDAS2 forecast file

ferror flag to indicate success of the call (=1 indicates success)

The routines invoked are:

interp_nldas2 (44.2.4)
spatially interpolates a NLDAS2 variable

44.2.6 read_orig_nldas2_elevdiff (Source File: read_orig_nldas2_elevdiff.F90)

REVISION HISTORY:

```
7 Nov 2012: Sujay Kumar - initial specification based on the NLDAS1
                          implementation
```

INTERFACE:

```
subroutine read_orig_nldas2_elevdiff(n)
```

USES:

```
  use LIS_coreMod
  use nldas2_forcingMod, only : nldas2_struc
  use LIS_logmod
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Open and read in original NLDAS elevation difference file.
The arguments are:

n index of the domain or nest.

44.2.7 readcrd_nldas2 (Source File: readcrd_nldas2.F90)

REVISION HISTORY:

```
02Feb2004; Sujay Kumar, Initial Code
24 Aug 2007: Chuck Alonge; Modified for use with NLDAS2 data
22 Jan 2012: K. Arsenault; Accommodate GES DISC, NCEP filename conventions
14 Mar 2014: David Mocko: Removed elevation file line in config file,
                          as this data is now in the LDT input file.
```

INTERFACE:

```
subroutine readcrd_nldas2()
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod,  only : LIS_logunit, LIS_verify, LIS_endrun
use nldas2_forcingMod, only : nldas2_struc
```

DESCRIPTION:

This routine reads the options specific to NLDAS-2 forcing from the LIS configuration file.

44.3 Fortran: Module Interface reset_nldas2 (Source File: reset_nldas2.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine reset_nldas2()
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use nldas2_forcingMod, only : nldas2_struc
```

DESCRIPTION:

Routine to reset nldas2 forcing related memory allocations.

44.3.1 timeinterp_nldas2 (Source File: timeinterp_nldas2.F90)

REVISION HISTORY:

```
02 Feb 2004: Sujay Kumar; Initial Specification
24 Aug 2007: Chuck Alonge; Modified for use with NLDAS-2 data
14 Mar 2014: David Mocko: Added CAPE and PET forcing from NLDAS-2
```

INTERFACE:

```
subroutine timeinterp_nldas2(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_constantsMod, only  : LIS_CONST_SOLAR
use LIS_metforcingMod, only : LIS_forc, LIS_FORC_Base_State
use LIS_FORC_AttributesMod
use LIS_timeMgrMod, only : LIS_tick, LIS_time2date
use LIS_logMod, only :LIS_logunit, LIS_verify
use nldas2_forcingMod, only : nldas2_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous 3 hourly value is used. All other variables are linearly interpolated between the 3 hourly blocks.
The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

LIS_tick (3.36.27)
advances or retracts time by the specified amount

zterp (5.3.19)
zenith-angle based interpolation

---

45 PET_USGS

This section describes the implementation of the potential evapotranspiration product from the USGS FAO-PET method, using GDAS forcing fields as inputs.

45.0.2 finalize_petusgs (Source File: finalize_petusgs.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
08Mar2012: Kristi Arsenault;  Modified for USGS PET dataset
25Oct2013: Kristi Arsenault;  Added PET USGS to LIS7
```

INTERFACE:

```
subroutine finalize_petusgs(findex)
```

USES:

```
use petusgs_forcingMod, only : petusgs_struc
```

DESCRIPTION:

Routine to cleanup PET USGS forcing related memory allocations.

45.0.3 get_petusgs (Source File: get_petusgs.F90)

REVISION HISTORY:

```
17 Jul 2001: Jon Gottschalck; Initial code
06 Jan 2005: Yudong Tian; Modified for LISv4.2
12 Mar 2012: K. Arsenault;  Updated for USGS PET data
25 Oct 2013: K. Arsenault;  Added PET USGS to LIS7
```

INTERFACE:

```
subroutine get_petusgs(n, findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_timeMgrMod, only : LIS_tick, LIS_get_nstep
use LIS_logMod,     only : LIS_logunit
use petusgs_forcingMod, only : petusgs_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates daily USGS PET forcing. At the beginning of a simulation, the code reads the most recent "past" data file (current 1-day), and nearest "future" data file (next 1-day). These two datasets are used to temporally interpolate the data to the current model timestep. . The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the USGS PET data times

petusgsfile (45.0.4)
Puts together appropriate file name for 6 hour intervals

read_petusgs (45.1.2)
Interpolates USGS PET data to LIS grid

conserv_interp_input (5.1.24)
Computes the neighbor, weights for conservative interpolation

---

45.0.4 petusgsfile (Source File: get_petusgs.F90)

INTERFACE:

```
subroutine petusgsfile( filename, filetype, petusgsdir, yr, mo, da )

  implicit none
```

ARGUMENTS:

```
  character(len=*), intent(out) :: filename
  character(len=*), intent(in)  :: petusgsdir
  character(len=*), intent(in)  :: filetype
  integer         , intent(in)  :: yr, mo, da
```

DESCRIPTION:

This subroutine puts together daily USGS PET filenames.
The arguments are:

petusgsdir Name of the USGS PET directory

yr year

mo month

da day of month

filename name of the timestamped USGS PET file

45.0.5 interp_petusgs (Source File: interp_petusgs.F90)

INTERFACE:

```
subroutine interp_petusgs(n, findex, nx, ny, finput, lis_gds, nc, nr, varfield)
```

USES:

```
  use LIS_coreMod,        only : LIS_rc
  use LIS_logMod,         only : LIS_logunit
  use petusgs_forcingMod, only : petusgs_struc

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: n
  integer, intent(in) :: findex
  integer             :: nx
  integer             :: ny
  integer             :: nc
  integer             :: nr
  real, dimension(nx,ny) :: finput
  real, dimension(nc,nr) :: varfield
```

DESCRIPTION:

This subroutine interpolates a given USGS PET field to the LIS grid.
The arguments are:

n index of the nest

nx number of columns (in the east-west dimension) in the USGS PET grid

ny number of rows (in the north-south dimension) in the USGS PET grid

finput input data array to be interpolated

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

---

45.1 Fortran: Module Interface petusgs_forcingMod (Source File: petusgs_forcingMod.F90)

REVISION HISTORY:

```
05Mar2012: K. Arsenault;  Added FEWSNET PET dataset as a forcing option
25Oct2013: K. Arsenault;  Added PET USGS to LIS7
02May2014; K. Arsenault;  Added climatological PET file option
```

DESCRIPTION:

This module contains variables and data structures that are used for the implementation of the potential evapotranspiration (PET) data from the US Geological Survey (USGS) method for estimating PET from NCEP GDAS 1.0 files and fields. The files are global and summed daily (6Z GDAS forecast to next day forecast), reflecting a 00Z timestamp.
The implementation in LIS has the derived data type `petusgs_struc` that includes the variables to specify the runtime options, and the weights and neighbor information for spatial interpolation
They are desribed below:

petdir Directory containing the input data

pettype Option to select PET file type: climatology or current (retrospective)

pettime The nearest (daily) instance of the incoming data (as a real time).

griduptime1 The time to switch the input resolution to T170

mi Number of points in the input grid

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n112,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

USES:

```
implicit none
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init_petusgs     !defines the native resolution of
                                  !the input data
```

PUBLIC TYPES:

```
public :: petusgs_struc
```

45.1.1 init_petusgs (Source File: petusgs_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar;  Initial Specification
11Mar2012: K. Arsenault; Expanded for use with USGSP PET data
```

INTERFACE:

```
subroutine init_petusgs(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep
use LIS_logMod,     only : LIS_logunit, LIS_endrun

implicit none
integer, intent(in) :: findex

real    :: gridDesci(50)
integer :: updoy,r1,mo1,da1,hr1,mn1,ss1
real    :: upgmt
integer :: n
```

DESCRIPTION:

Defines the native resolution of the input forcing for USGS PET data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). The routines invoked are:

readpetusgscrd (45.1.3)
reads the runtime options specified for USGS PET data

LIS_date2time (3.36.24)
converts date to the real time format

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

45.1.2 read_petusgs (Source File: read_petusgs.F90)

REVISION HISTORY:

```
17 Jul 2001: Jon Gottschalck; Initial code
06 Jan 2006: Yudong Tian; modified for LISv4.2
10 Mar 2012: K. Arsenault;  Added USGS PET Dataset
25 Oct 2013: K. Arsenault;  Added PET USGS to LIS7
```

INTERFACE:

```
subroutine read_petusgs (n, pet_filename, findex, ferror_petusgs )
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_masterproc
use LIS_logMod,  only : LIS_logunit, LIS_getNextUnitNumber, &
                        LIS_releaseUnitNumber
use LIS_metforcingMod, only  : LIS_forc
use petusgs_forcingMod, only : petusgs_struc
use fbil_module

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
character(99), intent(in) :: pet_filename
integer,intent(out) :: ferror_petusgs
```

DESCRIPTION:

For the given time, reads parameters from USGS PET data and interpolates to the LIS domain.
The arguments are:

n index of the nest

pet_filename name of the USGS PET file

ferror_petusgs flag to indicate success of the call (=0 indicates success)

The routines invoked are:

interp_petusgs (45.0.5)
spatially interpolates the USGS PET data

45.1.3 readpetusgscrd (Source File: readpetusgscrd.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
10 Mar 2012; K. Arsenault, modified for PET USGS
25 Oct 2013; K. Arsenault, added PET USGS to LIS7
02 May 2014; K. Arsenault, added climatological PET file option
```

INTERFACE:

```
subroutine readpetusgscrd()
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_config, LIS_rc
use LIS_logMod,  only : LIS_logunit, LIS_verify, LIS_endrun
use petusgs_forcingMod, only : petusgs_struc
```

DESCRIPTION:

This routine reads the options specific to USGS PET forcing from the LIS configuration file.

45.1.4 timeinterp_petusgs (Source File: timeinterp_petusgs.F90)

REVISION HISTORY:

```
20 Jan 2006:  Yudong Tian: Initial Implementation
10 Oct 2006:  Sujay Kumar: Switched to using ESMF_State for storing
                           forcing data.
10 Mar 2012:  K. Arsenault: Applied to USGS PET dataset
25 Oct 2013:  K. Arsenault: added PET USGS to LIS7
```

INTERFACE:

```
subroutine timeinterp_petusgs(n, findex )
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc,LIS_domain
use LIS_FORC_AttributesMod
use LIS_metforcingMod,  only : LIS_FORC_State, LIS_forc
use LIS_logMod,         only : LIS_verify
use petusgs_forcingMod, only : petusgs_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep.
Also, daily PET values are scaled up (i.e. multiplied) by a factor of 100 to preserve the precision, so here the read-in values are divided by 100 to obtain actual PET summary.
The arguments are:

n index of the nest

---

46 Princeton

This is the 50 year forcing data [**?**] produced at Princeton University. The data is global at 1 degree spatial resolution, at 3 hourly intervals.

46.0.5 finalize_princeton (Source File: finalize_princeton.F90)

REVISION HISTORY:

```
26 Jan 2007; Hiroko Kato, Initial Code
```

INTERFACE:

```
subroutine finalize_princeton(findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use princeton_forcingMod, only : princeton_struc
```

DESCRIPTION:

Routine to cleanup allocated structures for princeton forcing.

46.0.6 get_princeton (Source File: get_princeton.F90)

REVISION HISTORY:

```
26 Jan 2007: Hiroko Kato; Initial Specification adopted from LIS
```

INTERFACE:

```
subroutine get_princeton(n,findex)
```

USES:

```
use LIS_coreMod,          only : LIS_rc
use LIS_metforcingMod,   only : LIS_forc
use LIS_timeMgrMod,       only : LIS_get_nstep, LIS_tick
use LIS_logMod,           only : LIS_logunit
use princeton_forcingMod, only : princeton_struc
```

```
implicit none

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

DESCRIPTION:

Opens, reads, and interpolates 3-hrly, 1 degree Princeton forcing. At the beginning of a simulation, the code reads the most recent past data (nearest 3 hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep. The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day . The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
call to advance or retract time

read_princeton (46.1.2)
call to read the Princeton data and perform spatial interpolation

read_princeton_elev (46.1.4)
reads the native elevation of the Princeton data to be used for topographic adjustments to the forcing

---

46.1 Fortran: Module Interface princeton_forcingMod (Source File: princeton_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the 50-yr Princeton data (Sheffield et al. 2006). The data is global 1 degree dataset in latlon projection, and at 3 hourly intervals. The derived data type `princeton_struc` includes the variables that specify the runtime options, and the weights and neighbor information to be used for spatial interpolation. They are described below:

ncold Number of columns (along the east west dimension) for the input data

nrold Number of rows (along the north south dimension) for the input data

fmodeltime1 The nearest, previous 6 hour instance of the incoming data (as a real time).

fmodeltime2 The nearest, next 6 hour instance of the incoming data (as a real time).

princetondir Directory containing the input data

elevfile File with the elevation definition for the input data.

mi Number of points in the input grid

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n11,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

rlat2 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

rlon2 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for conservative interpolation)

n12,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

findtime1, findtime2 boolean flags to indicate which time is to be read for temporal interpolation.

Sheffield, J., G. Goteti, and E. F. Wood, 2006: Development of a 50-yr high-resolution global dataset of meteorological forcings for land surface modeling, J. Climate, 19 (13), 3088-3111

USES:

```
implicit none

PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
public :: init_PRINCETON      !defines the native resolution of
                                !the input data
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
public :: princeton_struc
```

46.1.1 init_PRINCETON (Source File: princeton_forcingMod.F90)

REVISION HISTORY:

```
26Jan2007: Hiroko Kato; Initial Specification
```

INTERFACE:

```
subroutine init_PRINCETON(findex)
```

USES:

```
  use LIS_coreMod, only : LIS_rc
  use LIS_timeMgrMod, only : LIS_update_timestep
  implicit none

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

DESCRIPTION:

Defines the native resolution of the input forcing for PRINCETON data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). The routines invoked are:

readcrd_princeton (46.1.5)
reads the runtime options specified for PRINCETON data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

read_princeton_elev (46.1.4)
reads the native elevation of the princeton data to be used for topographic adjustments to the forcing

---

46.1.2 read_princeton (Source File: read_princeton.F90)

REVISION HISTORY:

```
26 Jan 2007: Hiroko Kato; Initial Specification adopted from LIS/retberg.F90
```

INTERFACE:

```
subroutine read_princeton( order, n, findex, yr, mon, da, hr, ferror )
```

USES:

```
  use LIS_coreMod,          only : LIS_rc,LIS_domain, LIS_localPet, &
                                   LIS_masterproc
  use LIS_metforcingMod,    only : LIS_forc
  use LIS_timeMgrMod,       only : LIS_tick
  use LIS_logMod,           only : LIS_logunit, LIS_endrun
  use princeton_forcingMod, only : princeton_struc
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif


  implicit none
```

ARGUMENTS:

```
  integer, intent(in)    :: n  ! nest
  integer, intent(in)    :: order  ! lower(1) or upper(2) time interval bdry
  integer, intent(in)    :: findex
  integer, intent(in)    :: yr,mon,da,hr      ! data and hour (multiple of 3)
  integer, intent(inout) :: ferror           ! set to zero if there's an error
```

DESCRIPTION:

For the given time, reads the parameters from 1 degree PRINCETON data, transforms into 9 LIS forcing parameters and interpolates to the LIS domain.

PRINCETON variables used to force LIS are: mean values starting at timestep, available every 3 hours

NOTE-1: be aware that PRINCETON has only total precipitation. NOTE 2: only one wind component, it is magnitude

PRINCETON FORCING VARIABLES:
1. TAS Air Temperature [K]
2. SHUM Specific humidity [kg kg-1]
3. DSWRF Downward shortwave radiation [W m-2]
4. DLWRF Downward longwave radiation [W m-2]
5. WIND Wind speed [m s-1]
6. PRES Surface pressure [Pa]
7. PRCP Precipitation [kg m-2 s-1]

order flag indicating which data to be read (order=1, read the previous 3 hourly instance, order=2, read the next 3 hourly instance)

n index of the nest

yr current year

mon current month

da current day of the year

hr current hour of day

ferror flag to indicate success of the call (=0 indicates success)

The routines invoked are:

princetongrid_2_lisgrid (46.1.3)
transform the PRINCETON data to the LIS grid

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

---

46.1.3 princetongrid_2_lisgrid (Source File: read_princeton.F90)

REVISION HISTORY:

```
    10 Apr 2002: Urszula Jambor;  Code adapted from
                 ecmwfgrid_2_grid2catgrid, by R. Reichle
    29 Jan 2007: Hiroko Kato; modified berggrid_2_lisgrid for Princeton data
```

INTERFACE:

```
 subroutine princetongrid_2_lisgrid( nx, ny, grid_data )

   implicit none
```

ARGUMENTS:

```
integer, intent(in)                   :: nx, ny
real, intent(inout), dimension(nx,ny) :: grid_data
```

DESCRIPTION:

Changes grid data from PRINCETON data convention to LIS convention
PRINCETON: Sorth-to-Nouth around Greenwich Meridian Global grid. Data are written in NetCDF from "lower left to upper right" starting at 0.5-degree grid point center coordinates: 0.5E,89.5S and going to 0.5W,89.5N.
LIS: South-to-North around Date Line Full global grid. Starts at the southernmost latitude and date line, going east and then north.

46.1.4 read_princeton_elev (Source File: read_princeton_elev.F90)

REVISION HISTORY:

```
1 Feb 2007; Hiroko Kato; Initial Specificaton adopted from
                         read_ecmwf_elev.F90
```

INTERFACE:

```
subroutine read_princeton_elev(n,findex)
```

USES:

```
use LIS_coreMod,         only : LIS_rc, LIS_domain
use LIS_metforcingMod,  only : LIS_forc
use LIS_fileIOMod,       only : LIS_read_param
use LIS_logMod,          only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use princeton_forcingMod,only : princeton_struc
implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens and reads PRINCETON model elevation to the LIS grid. The data will be used to perform any topographical adjustments to the forcing. The elevation file needs to be preprocessed to fit the running resolution and domain.
The arguments are:

n index of the nest

The routines invoked are:

LIS_readData (3.14.1)
call the abstract method to get the elevation data in the LIS projection.

46.1.5 readcrd_princeton (Source File: readcrd_princeton.F90)

REVISION HISTORY:

```
26 Jan 2007; Hiroko Kato, Initial Code adopted from readcrd_princeton.F90
25 Jun 2007; Hiroko Kato, upgraded to LISv5.0
```

INTERFACE:

```
subroutine readcrd_princeton()
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod,  only : LIS_logunit
use princeton_forcingMod, only : princeton_struc
```

DESCRIPTION:

This routine reads the options specific to PRINCETON forcing from the LIS configuration file.

46.2 Fortran: Module Interface reset_princeton (Source File: reset_princeton.F90)

REVISION HISTORY:

```
21/08/2014 Bailing Li;initial creation
```

INTERFACE:

```
subroutine reset_princeton()
```

USES:

```
use LIS_coreMod,          only : LIS_rc
use princeton_forcingMod, only : princeton_struc
```

DESCRIPTION:

Routine to reset nldas2 forcing related memory allocations.

46.2.1 timeinterp_princeton (Source File: timeinterp_princeton.F90)

REVISION HISTORY:

```
26 Jan 2007: Hiroko Kato; Initial version of code adapted from LIS
25 Jun 2007: Hiroko Kato; upgraded for LISv5.0
```

INTERFACE:

```
subroutine timeinterp_princeton(n, findex)
```

USES:

```
use ESMF
use LIS_coreMod,         only : LIS_rc,LIS_domain
use LIS_constantsMod,    only : LIS_CONST_SOLAR
use LIS_FORC_AttributesMod
use LIS_metforcingMod,  only : LIS_forc, LIS_FORC_Base_State
```

```
use LIS_timeMgrMod,      only : LIS_time2date
use LIS_logMod,          only : LIS_logunit, LIS_verify, LIS_endrun
use princeton_forcingMod,only : princeton_struc


implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep. Downward shortwave radiation is interpolated using a zenith-angled based approach. Precipitation and longwave radiation are not temporally interpolated, and the previous 3 hourly value is used. All other variables are linearly interpolated between the 3 hourly blocks. Only total precipitation is available (no convective).
The routines invoked are:

LIS_time2date (3.36.25)
converts the time to a date format

zterp (5.3.19)
zenith-angle based interpolation

---

47 Rhone-AGG

This is the forcing data used in the PILPS experiments for the Rhone-AGG domain.

47.0.2 finalize_rhoneAGG (Source File: finalize_rhoneAGG.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine finalize_rhoneAGG
```

USES:

```
use LIS_coreMod, only : LIS_rc
use rhoneAGG_forcingMod, only : rhoneAGG_struc
```

DESCRIPTION:

Routine to cleanup RHONEAGG forcing related memory allocations.

47.0.3 get_rhoneAGG (Source File: get_rhoneAGG.F90)

REVISION HISTORY:

```
5 Nov 2003: Dave Mocko, Initial Specification
```

INTERFACE:

```
subroutine get_rhoneAGG(n,findex)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_metforcingMod, only : LIS_forc
use LIS_timeMgrMod,     only : LIS_get_nstep, LIS_tick
use LIS_logMod,         only : LIS_logunit, LIS_endrun
use rhoneAGG_forcingMod,   only : rhoneAGG_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates RHONEAGG 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.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
Determines RHONEAGG data times

rhoneAGGfile (47.0.4)
Puts together appropriate file name for 3 hour intervals

readrhoneAGG (47.0.5)
Interpolates RHONEAGG data to LIS grid

47.0.4 rhoneAGGfile (Source File: get_rhoneAGG.F90)

This subroutine puts together RHONEAGG file name

INTERFACE:

```
subroutine rhoneAGGfile(name,rhoneAGGdir,yr,mo,da,hr,ncold)

implicit none
```

INPUT PARAMETERS:

```
character*40, intent(in) :: rhoneAGGdir
integer, intent(in)      :: yr,mo,da,hr,ncold
```

OUTPUT PARAMETERS:

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

47.0.5 readrhoneAGG (Source File: read_rhoneAGG.F90)

REVISION HISTORY:

```
5 Nov 2003: Dave Mocko, Initial Specification
```

INTERFACE:

```
subroutine readrhoneAGG(n, order,findex, name,tscount,ferror)
```

USES:

```
use LIS_coreMod,        only : LIS_rc,LIS_domain
use LIS_metforcingMod, only : LIS_forc
use LIS_logMod,         only : LIS_logunit, LIS_verify, LIS_endrun
use rhoneAGG_forcingMod,   only : rhoneAGG_struc
```

DESCRIPTION:

Reads in RHONEAGG data and performs interpolation to the LIS domain.

47.1 Core Functions of readrhoneAGG

bilinear_interp Interpolates RHONEAGG data to LIS grid using bilinear interpolation

RHONEAGG 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)

47.1.1 readcrd_rhoneAGG (Source File: readcrd_rhoneAGG.F90)

REVISION HISTORY:

```
11 Dec 2003: Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcrd_rhoneAGG()
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod, only    : LIS_logunit
use rhoneAGG_forcingMod, only : rhoneAGG_struc
```

DESCRIPTION:

Routine to read RHONEAGG specific parameters from the LIS configuration file.

47.2 Fortran: Module Interface rhoneAGG_forcingMod (Source File: rhoneAGG_forcingMo

Contains routines and variables that define the native domain for RHONEAGG model forcing.

47.2.1 init_RHONEAGG (Source File: rhoneAGG_forcingMod.F90)

REVISION HISTORY:

```
11 Dec 2003: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine init_RHONEAGG(findex)
```

USES:

```
  use LIS_coreMod, only: LIS_rc
  use LIS_timeMgrMod, only : LIS_update_timestep
  implicit none

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

DESCRIPTION:

Defines the native resolution of the input forcing for RHONEAGG AGG data.
The routines invoked are:

readcrd_rhoneAGG (47.1.1)
reads the runtime options specified for RhoneAGGAGG data

47.2.2 timeinterp_rhoneAGG (Source File: timeinterp_rhoneAGG.F90)

REVISION HISTORY:

```
5 Nov 2003: Dave Mocko, Initial Specification
```

INTERFACE:

```
subroutine timeinterp_rhoneAGG(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod, only :LIS_rc, LIS_domain, LIS_localPet
use LIS_constantsMod, only  : LIS_CONST_SOLAR
use LIS_FORC_AttributesMod
use LIS_metforcingMod, only: LIS_forc, LIS_FORC_Base_State
use LIS_timeMgrMod,     only: LIS_time2date
use LIS_logMod, only : LIS_logunit, LIS_verify, LIS_endrun
use rhoneAGG_forcingMod, only : rhoneAGG_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates RHONEAGG 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.

47.3 Core Functions of time_interp_rhoneAGG

zterp Performs zenith angle-based temporal interpolation

---

48 SCAN forcing

This section describes the implementation of the SCAN station data.

48.0.1 finalize_scan (Source File: finalize_scan.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine finalize_scan(findex)
```

USES:

```
use scan_forcingMod, only : scan_struc
```

DESCRIPTION:

Routine to cleanup SCAN forcing related memory allocations.

48.0.2 get_scan (Source File: get_scan.F90)

REVISION HISTORY:

```
13 Apr 2007: Bailing Li; Initial Specification
```

INTERFACE:

```
subroutine get_scan(n,findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod, only : LIS_tick
use LIS_logMod, only : LIS_logunit
use LIS_metforcingMod,  only : LIS_forc
use scan_forcingMod, only : scan_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates the SCAN station data. At the beginning of a simulation, the code reads the most recent past data, and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the SCAN data times

read_scan (48.0.3)
Interpolates the appropriate SCAN station data to LIS grid

48.0.3 read_scan (Source File: read_scan.F90)

REVISION HISTORY:

```
13 Apr 2007: Bailing Li; Initial Specification
```

INTERFACE:

```
subroutine read_scan(n,ftn,findex,order)
```

USES:

```
use LIS_logMod, only         : LIS_logunit
use LIS_metforcingMod,  only : LIS_forc
use LIS_coreMod, only        : LIS_rc,LIS_domain
use scan_forcingMod,    only : scan_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)          :: n
integer, intent(in)          :: ftn
integer, intent(in)          :: findex
integer, intent(in)          :: order
```

DESCRIPTION:

For the given time, reads parameters from the correct SCAN station data (ASCII), transforms into LIS forcing parameters and interpolates to the LIS domain.
The arguments are:

order flag indicating which data to be read (order=1, read the previous instance, order=2, read the next instance)

n index of the nest

ftn unit number for the SCAN station data

The routines invoked are:

normalize_stnwts (5.3.12)
renormalizes the station weights accounting for missing data

interp_stndata (5.2.8)
spatially interpolates the station data onto the LIS grid.

48.0.4 readcrd_scan (Source File: readcrd_scan.F90)

REVISION HISTORY:

```
13 Apr 2007; Bailing Li, Initial Code
```

INTERFACE:

```
subroutine readcrd_scan()
```

USES:

```
use ESMF
use LIS_logMod, only : LIS_logunit
use LIS_coreMod, only : LIS_rc, LIS_config
use scan_forcingMod, only : scan_struc
```

DESCRIPTION:

This routine reads the options specific to SCAN station data forcing from the LIS configuration file.

48.1 Fortran: Module Interface scan_forcingMod (Source File: scan_forcingMod.F90)

Contains routines and data structures that are used for the implementation of the station data from various SCAN stations. The stations report estimates of meteorological forcing terms, which is spatially interpolated using the inverse distance weighting scheme (IDW).
The implementation in LIS has the derived data type `scan_struc` that includes the variables to specify the runtime options, and the calculation of weights for spatial interpolation.

REVISION HISTORY:

```
13Apr2007: Bailing Li:  Initial Specification

implicit none
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init_SCAN      !defines the native resolution of
                                !the input data
```

PUBLIC TYPES:

```
public :: scan_struc
```

48.1.1 init_SCAN (Source File: scan_forcingMod.F90)

INTERFACE:

```
subroutine init_SCAN(findex)
```

USES:

```
use LIS_coreMod,only : LIS_rc
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep
use LIS_logMod, only : LIS_logunit, LIS_endrun

implicit none
integer, intent(in) :: findex
```

DESCRIPTION:

This routines reads the runtime configurations for using the SCAN station data. Using the metadata provided for the stations, this routine invokes the call to compute the interpolation weights to be later used.
The routines invoked are:

readcrd_scan (48.0.4)
reads the runtime options specified for SCAN station data

LIS_date2time (3.36.24)
converts date to the real time format

compute_stnwts (5.1.22)
computes the weights for spatial interpolation

48.1.2 timeinterp_scan (Source File: timeinterp_scan.F90)

REVISION HISTORY:

```
13 Apr 2007: Bailing Li; Initial Specification
```

INTERFACE:

```
subroutine timeinterp_scan(n, findex)
```

USES:

```
use ESMF
use LIS_logMod,  only        : LIS_logunit
use LIS_coreMod,        only : LIS_rc,LIS_domain
use LIS_metforcingMod, only : LIS_FORC_Base_State, LIS_forc
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use scan_forcingMod,    only : scan_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the SCAN forcing data to the model timestep. All variables except precipitation is linearly interpolated.

48.1.3 finalize_snotel (Source File: finalize_snotel.F90)

REVISION HISTORY:

```
08Jun2011: Yuqiong Liu; Initial Specification
```

INTERFACE:

```
subroutine finalize_snotel(findex)
```

USES:

```
use snotel_forcingMod, only : snotel_struc
```

DESCRIPTION:

Routine to cleanup SCAN forcing related memory allocations.

48.1.4 get_snotel (Source File: get_snotel.F90)

REVISION HISTORY:

```
09 Jun 2011: Yuqiong Liu; Initial Specification
```

INTERFACE:

```
subroutine get_snotel(n,findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod, only : LIS_tick
use LIS_logMod, only : LIS_logunit
use LIS_metforcingMod,  only : LIS_forc
use snotel_forcingMod, only : snotel_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates the SNOTEL station data. At the beginning of a simulation, the code reads the most recent past data, and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.36.27)
determines the SNOTEL data times

read_snotel (48.1.5)
Interpolates the appropriate SNOTEL station data to LIS grid

48.1.5 read_snotel (Source File: read_snotel.F90)

REVISION HISTORY:

```
08 Jun 2011: Yuqiong Liu; Initial Specification, based on LIS SNOTEL obs reader by S. Kumar
```

INTERFACE:

```
subroutine read_snotel(n,ftn,findex,order)
```

USES:

```
use LIS_logMod, only         : LIS_logunit, LIS_endrun
use LIS_coreMod, only        : LIS_rc,LIS_domain
use LIS_metforcingMod, only : LIS_forc
use snotel_forcingMod,    only : snotel_struc
use map_utils,    only : latlon_to_ij

implicit none
```

ARGUMENTS:

```
integer, intent(in)           :: n
integer, intent(in)           :: ftn
integer, intent(in)           :: findex
integer, intent(in)           :: order
```

DESCRIPTION:

For the given time, reads parameters from the correct SNOTEL station data (ASCII), transforms into LIS forcing parameters and interpolates to the LIS domain.
The arguments are:

order flag indicating which data to be read (order=1, read the previous instance, order=2, read the next instance)

n index of the nest

ftn unit number for the SNOTEL station data

The routines invoked are:

normalize_stnwts (5.3.12)
renormalizes the station weights accounting for missing data

interp_stndata (5.2.8)
spatially interpolates the station data onto the LIS grid.

48.1.6 readcrd_snotel (Source File: readcrd_snotel.F90)

REVISION HISTORY:

```
08Jun2011; Yuqiong Liu, Initial Specification
```

INTERFACE:

```
subroutine readcrd_snotel()
```

USES:

```
use ESMF
use LIS_logMod, only : LIS_logunit
use LIS_coreMod, only : LIS_rc, LIS_config
use snotel_forcingMod, only : snotel_struc
```

DESCRIPTION:

This routine reads the options specific to SNOTEL station data forcing from the LIS configuration file.

48.2 Fortran: Module Interface snotel_forcingMod (Source File: snotel_forcingMod.F90)

Contains routines and data structures that are used for the implementation of the station data from various SNOTEL stations. The stations report estimates of meteorological forcing terms, which is spatially interpolated using the inverse distance weighting scheme (IDW).
The implementation in LIS has the derived data type `snotel_struc` that includes the variables to specify the runtime options, and the calculation of weights for spatial interpolation.

REVISION HISTORY:

```
08Jun2010: Yuqiong Liu:  Initial Specification

implicit none
PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
 ------------------------------------------------------------------------------
  public :: init_SNOTEL      !defines the native resolution of
                                  !the input data
 ------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
 ------------------------------------------------------------------------------
  public :: snotel_struc
```

48.2.1 init_SNOTEL (Source File: snotel_forcingMod.F90)

INTERFACE:

```
subroutine init_SNOTEL(findex)
```

USES:

```
use LIS_coreMod,only : LIS_rc
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep
use LIS_logMod, only : LIS_logunit, LIS_endrun

implicit none
integer, intent(in) :: findex
```

DESCRIPTION:

This routines reads the runtime configurations for using the SNOTEL station data. Using the metadata provided for the stations, this routine invokes the call to compute the interpolation weights to be later used. The routines invoked are:

readcrd_snotel (48.1.6)
reads the runtime options specified for SNOTEL station data

LIS_date2time (3.36.24)
converts date to the real time format

compute_stnwts (5.1.22)
computes the weights for spatial interpolation

48.2.2 timeinterp_snotel (Source File: timeinterp_snotel.F90)

REVISION HISTORY:

```
09 Jun 2011: Yuqiong Liu; Initial Specification, based on code for SCAN
```

INTERFACE:

```
subroutine timeinterp_snotel(n, findex)
```

USES:

```
use ESMF
use LIS_logMod,  only        : LIS_logunit
use LIS_coreMod,        only : LIS_rc,LIS_domain
use LIS_metforcingMod, only : LIS_FORC_Base_State, LIS_forc
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use snotel_forcingMod,    only : snotel_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the SNOTEL forcing data to the model timestep. All variables except precipitation is linearly interpolated.

---

49 STAGEII

This section describes the implementation of the precipitation product from NCEP, known as STAGEII

49.1 Fortran: Module Interface finalize_stg2 (Source File: finalize_stg2.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
17Jul2006; K. Arsenault, Added Stage II
```

INTERFACE:

```
subroutine finalize_stg2(findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use stg2_forcingMod
```

DESCRIPTION:

Routine to cleanup Stage II forcing related memory allocations.

49.1.1 get_stg2 (Source File: get_stg2.F90)

REVISION HISTORY:

```
25 May 2006: Kristi Arsenault;  Data and Code implementation
```

INTERFACE:

```
subroutine get_stg2(n, findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod, only : LIS_tick, LIS_get_nstep
use LIS_logMod, only : LIS_logunit
use stg2_forcingMod, only : stg2_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates hourly STAGE2 forcing. At the beginning of a simulation, the code reads the most recent past data (nearest the hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep.
The arguments are:

n index of the nest

lis_time Current LIS Time

stg2_file_time End boundary time of Stage II file

file_name Stage II filename - passed back to getstg

The routines invoked are:

LIS_tick (3.36.27)
determines the STAGE2 data times

bilinear_interp_input (5.1.2)
Computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
Computes the neighbor, weights for conservative interpolation

stg2file (49.2.2)
Puts together appropriate file name for 1-hour intervals

read_stg2 (49.1.3)
Interpolates STAGE2 data to LIS grid

49.1.2 interp_stg2 (Source File: interp_stg2.F90)

INTERFACE:

```
subroutine interp_stg2 (n, findex, nstg2, ppt_field, lb, lis_gds, &
                        nc, nr, varfield )
```

USES:

```
use stg2_forcingMod, only : stg2_struc
use LIS_coreMod, only: LIS_rc

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
integer, intent(in)    :: findex
integer                :: nc      ! Number of columns (in the E-W dimension) in the LIS grid
integer                :: nr      ! Number of rows (in the N-S dimension) in the LIS grid
integer                :: nstg2   ! Number of points in original STAGE II grid
real                   :: lis_gds(50)
real                   :: ppt_field(nstg2)
logical*1              :: lb(nstg2)
real, dimension(nc,nr) :: varfield
```

DESCRIPTION:

This subroutine interpolates a given STG2 field to the LIS grid. The arguments are:

n index of the nest

nstg2 number of elements in the input grid

ppt_field input data array to be interpolated

lb input bitmap

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

bilinear_interp (5.1.1)
Spatially interpolate the forcing data using bilinear interpolation, or

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative (budget bil.) interpolation

---

49.1.3 read_stg2 (Source File: read_stg2.F90)

REVISION HISTORY:

```
25 May 2006: Kristi Arsenault;  Data and code implementation
25 Jan 2012: Sujay Kumar; Switched to the use of grib-api library
```

INTERFACE:

```
subroutine read_stg2( n, fname, findex, order, ferror_stg2 )
```

USES:

```
  use LIS_coreMod,        only : LIS_rc, LIS_domain
  use LIS_logMod,         only : LIS_logunit, LIS_verify
  use LIS_metforcingMod,  only : LIS_forc
  use stg2_forcingMod,    only : stg2_struc

#if (defined USE_GRIBAPI)
  use grib_api
#endif

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: n
  character(len=80)   :: fname
  integer, intent(in) :: findex
  integer, intent(in) :: order
  integer             :: ferror_stg2
```

DESCRIPTION:

For the given time, reads parameters from STAGE2 datasets and interpolates to a designated user-domain. NOTE:: These subroutines use the READ_GRIB routines for for opening and reading the STAGE II grib files.
The arguments are:

n index of the nest

fname name of the hourly STAGE2 file

ferror_stg2 flag to indicate success of the call (=0 indicates success)

filehr current file hour

The routines invoked are:

interp_stg2 (49.1.2)
spatially interpolates the STG2 data

49.1.4 readcrd_stg2 (Source File: readcrd_stg2.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
05 Jun 2006; Kristi Arsenault, Code and data implementation
```

INTERFACE:

```
subroutine readcrd_stg2()
```

USES:

```
  use ESMF
  use stg2_forcingMod, only : stg2_struc
  use LIS_coreMod, only : LIS_config, LIS_rc
  use LIS_logMod, only : LIS_logunit
```

DESCRIPTION:

This routine reads the options specific to STAGE2 forcing from the LIS configuration file.

49.2 Fortran: Module Interface stg2_forcingMod (Source File: stg2_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the precipitation data from the National Center for Environmental Prediction (NCEP) Stage II (STAGE2) Doppler Radar+gage product. The Stage II is a national level product, on an hourly interval, and supplements mainly the which base forcing precipitation (e.g., NLDAS) is being used.
The implementation in LIS has the derived data type `stg2_struc` that includes the variables to specify the runtime options, and the weights and neighbor information for spatial interpolation
They are desribed below:

ncol Number of columns (along the east west dimension) for the input data

nrow Number of rows (along the north south dimension) for the input data

stg2dir Directory containing the input data

stg2time The nearest, hourly instance of the incoming data (as a real time).

mi Number of points in the input grid

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n111,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

n112,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

USES:

```
implicit none
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init_STG2      !defines the native resolution of
                                !the input data
```

PUBLIC TYPES:

```
public :: stg2_struc
```

49.2.1 init_STG2 (Source File: stg2_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
25May2006: Kristi Arsenault; Data and code implementation
```

INTERFACE:

```
subroutine init_STG2(findex)
```

USES:

```
use LIS_coreMod, only: LIS_rc
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep

implicit none
integer,   intent(in) :: findex
```

DESCRIPTION:

Defines the native resolution of the input forcing for STAGE2 data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). The routines invoked are:

readcrd_stg2 (49.1.4)
reads the runtime options specified for STAGE2 data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

LIS_date2time (3.36.24)
converts date to the real time format - time of grid change

49.2.2 stg2file (Source File: stg2file.F90)

INTERFACE:

```
subroutine stg2file( name, stg2dir, yr, mo, da, hr)

implicit none
```

DESCRIPTION:

This subroutine puts together a STAGE2 filename for one hour file intervals
The arguments are:

stg2dir Name of the STAGE II directory

yr year

mo month

da day of month

hr hour of day

name name of the time-stamped STAGE II file

49.2.3 timeinterp_stg2 (Source File: timeinterp_stg2.F90)

REVISION HISTORY:

```
25May2006: Kristi Arsenault; Initial implementation
10 Oct 2006:   Sujay Kumar: Switched to using ESMF_State for storing
                            forcing data.
```

INTERFACE:

```
subroutine timeinterp_stg2(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_metforcingMod, only : LIS_FORC_Base_State, LIS_forc
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use stg2_forcingMod,    only : stg2_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep.
The arguments are:

n index of the nest

---

50 STAGEIV

This section describes the implementation of the precipitation product from NCEP, known as STAGEIV

50.1 Fortran: Module Interface finalize_stg4 (Source File: finalize_stg4.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
17Jul2006; K. Arsenault, Added Stage IV
```

INTERFACE:

```
subroutine finalize_stg4(findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use stg4_forcingMod
```

DESCRIPTION:

Routine to cleanup Stage IV forcing related memory allocations.

50.1.1 get_stg4 (Source File: get_stg4.F90)

REVISION HISTORY:

```
25 May 2006: Kristi Arsenault;  Data and Code implementation
```

INTERFACE:

```
subroutine get_stg4(n, findex)
```

USES:

```
use LIS_coreMod, only     : LIS_rc
use LIS_logMod, only      : LIS_logunit
use LIS_timeMgrMod, only  : LIS_tick, LIS_get_nstep
use stg4_forcingMod, only : stg4_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates hourly STAGE4 forcing. At the beginning of a simulation, the code reads the most recent past data (nearest the hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep.
The arguments are:

n index of the nest

lis_time Current LIS Time

stg4_file_time End boundary time of Stage IV file

file_name Stage IV filename - passed back to getstg

The routines invoked are:

LIS_tick (3.36.27)
determines the STAGE4 data times

bilinear_interp_input (5.1.2)
Computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
Computes the neighbor, weights for conservative interpolation

stg4file (50.3.2)
Puts together appropriate file name for 1-hour intervals

read_stg4 (50.1.3)
Interpolates STAGE4 data to LIS grid

50.1.2 interp_stg4 (Source File: interp_stg4.F90)

INTERFACE:

```
subroutine interp_stg4 (n, findex, nstg4, ppt_field, lb, lis_gds, &
                        nc, nr, varfield )
```

USES:

```
use stg4_forcingMod, only : stg4_struc
use LIS_coreMod, only: LIS_rc

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
integer, intent(in)    :: findex
integer                :: nc      ! Number of columns (in the E-W dimension) in the LIS grid
integer                :: nr      ! Number of rows (in the N-S dimension) in the LIS grid
integer                :: nstg4   ! Number of points in original STAGE IV grid
real                   :: lis_gds(50)
real                   :: ppt_field(nstg4)
logical*1              :: lb(nstg4)
real, dimension(nc,nr) :: varfield
```

DESCRIPTION:

This subroutine interpolates a given STAGE4 field to the LIS grid. The arguments are:

n index of the nest

nstg4 number of elements in the input grid

ppt_field input data array to be interpolated

lb input bitmap

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

bilinear_interp (5.1.1)
Spatially interpolate the forcing data using bilinear interpolation, or

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative (budget bil.) interpolation

---

50.1.3 read_stg4 (Source File: read_stg4.F90)

REVISION HISTORY:

```
25 May 2006: Kristi Arsenault;  Data and code implementation
25 Jan 2012: Sujay Kumar; Switched to the use of grib-api library
```

INTERFACE:

```
subroutine read_stg4( n, fname,findex,order, ferror_stg4 )
```

USES:

```
  use LIS_coreMod,        only : LIS_rc, LIS_domain
  use LIS_logMod,         only : LIS_logunit, LIS_verify
  use LIS_metforcingMod,  only : LIS_forc
  use stg4_forcingMod,    only : stg4_struc

#if (defined USE_GRIBAPI)
  use grib_api
#endif

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: n
  character(len=80)   :: fname
  integer, intent(in) :: findex
  integer, intent(in) :: order
  integer             :: ferror_stg4
```

DESCRIPTION:

For the given time, reads parameters from STAGE4 datasets and interpolates to a designated user-domain. NOTE:: These subroutines use the READ_GRIB routines for for opening and reading the STAGE IV grib files.
The arguments are:

n index of the nest

fname name of the hourly STAGE4 file

ferror_stg4 flag to indicate success of the call (=0 indicates success)

filehr current file hour

The routines invoked are:

interp_stg4 (50.1.2)
spatially interpolates the STAGE4 data

50.1.4 readcrd_stg4 (Source File: readcrd_stg4.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
05 Jun 2006; Kristi Arsenault, Code and data implementation
```

INTERFACE:

```
subroutine readcrd_stg4()
```

USES:

```
use ESMF
use stg4_forcingMod, only : stg4_struc
use LIS_coreMod, only : LIS_config, LIS_rc
use LIS_logMod, only : LIS_logunit
```

DESCRIPTION:

This routine reads the options specific to STAGE4 forcing from the LIS configuration file.

50.2 Fortran: Module Interface reset_stg4 (Source File: reset_stg4.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine reset_stg4()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use stg4_forcingMod
```

DESCRIPTION:

Routine to cleanup Stage IV forcing related memory allocations.

50.3 Fortran: Module Interface stg4_forcingMod (Source File: stg4_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the precipitation data from the National Center for Environmental Prediction (NCEP) Stage IV (STAGE4) Doppler Radar+gage product. The Stage IV is a national level product, on an hourly interval, and supplements mainly the which base forcing precipitation (e.g., NLDAS) is being used.
The implementation in LIS has the derived data type `stg4_struc` that includes the variables to specify the runtime options, and the weights and neighbor information for spatial interpolation
They are desribed below:

ncol Number of columns (along the east west dimension) for the input data

nrow Number of rows (along the north south dimension) for the input data

stg4dir Directory containing the input data

stg4time The nearest, hourly instance of the incoming data (as a real time).

mi Number of points in the input grid

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n111,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

n112,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init_STG4      !defines the native resolution of
                                !the input data
```

PUBLIC TYPES:

```
public :: stg4_struc
```

50.3.1 init_STG4 (Source File: stg4_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
25May2006: Kristi Arsenault; Data and code implementation
```

INTERFACE:

```
subroutine init_STG4(findex)
```

USES:

```
use LIS_coreMod, only: LIS_rc
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep

implicit none
integer,   intent(in) :: findex
```

DESCRIPTION:

Defines the native resolution of the input forcing for STAGE4 data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes (see Section V). The routines invoked are:

readcrd_stg4 (50.1.4)
reads the runtime options specified for STAGE4 data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

LIS_date2time (3.36.24)
converts date to the real time format - time of grid change

50.3.2 stg4file (Source File: stg4file.F90)

INTERFACE:

```
subroutine stg4file( name, stg4dir, yr, mo, da, hr)

implicit none
```

DESCRIPTION:

This subroutine puts together a STAGE4 filename for one hour file intervals
The arguments are:

stg4dir Name of the STAGE IV directory

yr year

mo month

da day of month

hr hour of day

name name of the time-stamped STAGE IV file

50.3.3 timeinterp_stg4 (Source File: timeinterp_stg4.F90)

REVISION HISTORY:

```
25May2006: Kristi Arsenault; Initial implementation
10 Oct 2006:   Sujay Kumar: Switched to using ESMF_State for storing
                            forcing data.
```

INTERFACE:

```
subroutine timeinterp_stg4(n,findex)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_metforcingMod,  only : LIS_FORC_Base_State, LIS_forc
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use stg4_forcingMod,    only : stg4_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep.
The arguments are:

n index of the nest

---

51 Template

This is a dummy placeholder implementation of a baseforcing, to be used as a template for new implementations.

51.1 Fortran: Module Interface finalize_metForcTemplate (Source File: finalize_metForcTemplate.F90)

INTERFACE:

```
subroutine finalize_metForcTemplate
```

USES:

51.1.1 get_metForcTemplate (Source File: get_metForcTemplate.F90)

INTERFACE:

```
subroutine get_metForcTemplate(n, findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

The arguments are:

n index of the nest

51.2 Fortran: Module Interface metForcTemplate_forcingMod (Source File: metForcTemplate_forcingMod.F90)

USES:

```
none

implicit none
```

```
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: init_metForcTemplate      !defines the native resolution of
                                 !the input data
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: metForcTemplate_struc
```

51.2.1 init_metForcTemplate (Source File: metForcTemplate_forcingMod.F90)

INTERFACE:

```
subroutine init_metForcTemplate(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
```

51.3 Fortran: Module Interface reset_metForcTemplate (Source File: reset_metForcTemplat

INTERFACE:

```
subroutine reset_metForcTemplate
```

USES:

51.3.1 timeinterp_metForcTemplate (Source File: timeinterp_metForctemplate.F90)

forcing data. INTERFACE:

```
subroutine timeinterp_metForcTemplate(n, findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

```
implicit none

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

DESCRIPTION:

The arguments are:

n index of the nest

findex index of the forcing

---

52 VIC-processed Forcing

This section describes the implementation of the reader for VIC-processed forcing data.
The purpose of this reader is to read forcing data that has been processed by a stand-alone run of VIC.
The VIC land surface model's forcing data processing routines are aware of 22 input forcing variables. These routines process the provided input forcing variables to generate the 10 final atmospheric forcing variables that drive the land surface physics.
The logic for processing the input forcing variables is complex, and depending on which input forcing variables are supplied, these routines

- compute final atmospheric forcing variables from the supplied input forcing variables,
- derive final atmospheric forcing variables from the supplied input forcing variables and computed atmospheric forcing variables,
- estimate final atmospheric forcing variables.

For example the Stehekin test-case supplies four input forcing variables, and the Pilps test-case supplies seven. From these, the 10 final atmospheric forcing variables are generated. These two test-cases are found at http://www.hydro.washington.edu/Lettenmaier/Models/VIC/index.shtml
Due to several technical difficulties, logic was written to read in the VIC-processed final atmospheric forcing data rather than to try to process the input forcing data within LIS.
To produce the VIC-processed final atmospheric forcing data, one must first run a stand-alone VIC simulation.

1. Add the write_bin_atmosdata.c file to the stand-alone VIC source code.
2. Call the write_bin_atmosdata routine in vicNl after the call to initialize_atmos.
3. Run stand-alone VIC. Stand-alone VIC will then process all the raw forcing input data and produce the entries for the atmos data structure. The write_bin_atmosdata routine will write these entries to disk as 4-byte values (to be consistent with LIS).
4. Run the forcing regridding utility. This reads the output from the write_bin_atmosdata routine and grids it.

52.0.2 getvicforcing (Source File: getvicforcing.F90)

INTERFACE:

```
subroutine getvicforcing(n, findex)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use vic_forcingMod,     only : vicforcing_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

The arguments are:

n index of the nest

This is the entry point for the routines that read VIC-processed forcing data.
Note that VIC-processed forcing data are in lock-step with either VIC's model time-step (for energy-balance runs) or VIC's snow time-step (for water-balance runs).
Note that VIC-processed forcing data are considered valid at the given time-stamp of the corresponding forcing data files.

52.0.3 get_vicforcing_filename (Source File: getvicforcing.F90)

INTERFACE:

```
subroutine get_vicforcing_filename(filename, dir, year, month, day, hour)
```

USES:

```
implicit none
```

ARGUMENTS:

```
character(len=140), intent(out) :: filename
character(len=100), intent(in) :: dir
integer, intent(in) :: year
integer, intent(in) :: month
integer, intent(in) :: day
integer, intent(in) :: hour
```

DESCRIPTION:

The arguments are:

filename the name of the forcing file to be read

dir directory containing the VIC-processed forcing data

year year

month month

day day

hour hour

This routine generates the filename of the VIC-processed forcing file to be read.

52.0.4 readviccrd (Source File: readviccrd.F90)

REVISION HISTORY:

```
16 Nov 2011; James Geiger, Initial Code
```

INTERFACE:

```
subroutine readviccrd()
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_config
use LIS_logMod,     only : LIS_logunit
use vic_forcingMod, only : vicforcing_struc
```

DESCRIPTION:

This routine reads the options specific to VIC formatted forcing from the LIS configuration file.

52.0.5 time_interp_vicforcing (Source File: time_interp_vicforcing.F90)

INTERFACE:

```
subroutine time_interp_vicforcing(n, findex)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_FORC_AttributesMod
use LIS_metforcingMod,  only : LIS_forc, LIS_FORC_Base_State
use LIS_logMod,         only : LIS_verify
use vic_forcingMod,     only : vicforcing_struc
```

ARGUMENTS:

```
implicit none

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

DESCRIPTION:

This routine performs the temporal interpolation of the VIC-processed forcing data to the LIS model time-step.
The arguments are:

n index of the nest

findex index of the supplemental forcing scheme

52.0.6 vic411_read_gridded_forcing_data (Source File: vic411_read_gridded_forcing_data.F90)

INTERFACE:

```
subroutine vic411_read_gridded_forcing_data(n, findex, filename, ferror)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_logMod,         only : LIS_getNextUnitNumber, &
                               LIS_releaseUnitNumber, &
                               LIS_logunit, LIS_endrun
use LIS_metforcingMod,  only : LIS_forc
use vic_forcingMod,     only : vicforcing_struc
use vic411_lsmMod,      only : vic411_struc
```

ARGUMENTS:

```
implicit none

integer, intent(in)            :: n
integer, intent(in)            :: findex
character(len=140), intent(in) :: filename
integer, intent(out)           :: ferror
```

DESCRIPTION:

This routine reads the VIC-processed forcing data from disk and stores it within the suppdata arrays. The arguments are:

n index of the nest

findex index of the supplemental forcing scheme

52.0.7 interp_vic (Source File: vic411_read_gridded_forcing_data.F90)

INTERFACE:

```
subroutine interp_vic(n,findex,var,NLEN,f,tempvic)
```

USES:

```
use LIS_coreMod,     only : LIS_rc, LIS_domain
use vic_forcingMod,  only : vicforcing_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)                           :: n
integer, intent(in)                           :: findex
integer, intent(in)                           :: var
integer, intent(in)                           :: NLEN
real, dimension(NLEN), intent(in)             :: f
real, dimension(LIS_rc%ngrid(n)), intent(out) :: tempvic
```

DESCRIPTION:

Interpolates the VIC formatted forcing data onto the LIS grid. The arguments are:

n index of the nest

NLEN size of the VIC forcing data

var variable index

f native VIC forcing data

g interpolated VIC forcing data

The routines invoked are:

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

---

52.1 Fortran: Module Interface vic_forcingMod (Source File: vic_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of VIC formatted forcing data.
The implementation in LIS has the derived data type `vicforcing_struc` that includes the variables that specify the runtime options.
They are described below:

vicdir Directory containing the input data

forcingInterval Interval of the forcing data [hours]. This value should be either VIC's model time-step for the energy-balance mode or the snow time-step for the water-balance mode.

NC specifies the number of columns (longitude grid-cells) for the gridded VIC data (not used when read_gridded == 0)

NR specifies the number of rows (latitude grid-cells) for the gridded VIC data (not used when read_gridded == 0)

slat specifies the lower left latitude for the gridded VIC forcing domain

slon specifies the lower left longitude for the gridded VIC forcing domain

elat specifies the upper right latitude for the gridded VIC forcing domain

elon specifies the upper right longitude for the gridded VIC forcing domain

dlon specifies the domain resolution (dx) for the gridded VIC forcing domain

dlat specifies the domain resolution (dy) for the gridded VIC forcing domain

time1 The nearest, previous forcing interval instance of the incoming data (as a real time).

time2 The nearest, next forcing interval instance of the incoming data (as a real time).

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: defineNativevicforcing      !defines the native resolution of
                                      !the input data
```

PUBLIC TYPES:

```
public :: vicforcing_struc
```

52.1.1 defineNativevicforcing (Source File: vic_forcingMod.F90)

INTERFACE:

```
subroutine defineNativevicforcing(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
```

DESCRIPTION:

Defines the native resolution of the input forcing for the VIC formatted data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes V. Based on the specified data map projection and resolution, this routine sets up the spatial interpolation weights. The routines invoked are:

readviccrd (52.0.4)
reads the runtime options specified for VIC formatted data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

52.2 Fortran: Module Interface vicforcing_finalize (Source File: vicforcing_finalize.F90)

INTERFACE:

```
subroutine vicforcing_finalize
```

USES:

52.3 Fortran: Module Interface vicforcing_reset (Source File: vicforcing_reset.F90)

INTERFACE:

```
subroutine vicforcing_reset
```

USES:

---

53 VIC-processed Forcing

This section describes the implementation of the reader for VIC-processed forcing data.
The purpose of this reader is to read forcing data that has been processed by a stand-alone run of VIC.
The VIC land surface model's forcing data processing routines are aware of 22 input forcing variables. These routines process the provided input forcing variables to generate the 10 final atmospheric forcing variables that drive the land surface physics.
The logic for processing the input forcing variables is complex, and depending on which input forcing variables are supplied, these routines

- compute final atmospheric forcing variables from the supplied input forcing variables,
- derive final atmospheric forcing variables from the supplied input forcing variables and computed atmospheric forcing variables,
- estimate final atmospheric forcing variables.

For example the Stehekin test-case supplies four input forcing variables, and the Pilps test-case supplies seven. From these, the 10 final atmospheric forcing variables are generated. These two test-cases are found at http://www.hydro.washington.edu/Lettenmaier/Models/VIC/index.shtml
Due to several technical difficulties, logic was written to read in the VIC-processed final atmospheric forcing data rather than to try to process the input forcing data within LIS.
To produce the VIC-processed final atmospheric forcing data, one must first run a stand-alone VIC simulation.

1. Add the write_bin_atmosdata.c file to the stand-alone VIC source code.
2. Call the write_bin_atmosdata routine in vicNl after the call to initialize_atmos.
3. Run stand-alone VIC. Stand-alone VIC will then process all the raw forcing input data and produce the entries for the atmos data structure. The write_bin_atmosdata routine will write these entries to disk as 4-byte values (to be consistent with LIS).
4. Run the forcing regridding utility. This reads the output from the write_bin_atmosdata routine and grids it.

53.0.1 getvicforcing (Source File: getvicforcing.F90)

INTERFACE:

```
subroutine getvicforcing(n, findex)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use vic_forcingMod,     only : vicforcing_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

The arguments are:

n index of the nest

This is the entry point for the routines that read VIC-processed forcing data.
Note that VIC-processed forcing data are in lock-step with either VIC's model time-step (for energy-balance runs) or VIC's snow time-step (for water-balance runs).
Note that VIC-processed forcing data are considered valid at the given time-stamp of the corresponding forcing data files.

53.0.2 get_vicforcing_filename (Source File: getvicforcing.F90)

INTERFACE:

```
subroutine get_vicforcing_filename(filename, dir, year, month, day, hour)
```

USES:

```
implicit none
```

ARGUMENTS:

```
character(len=140), intent(out) :: filename
character(len=100), intent(in) :: dir
integer, intent(in) :: year
integer, intent(in) :: month
integer, intent(in) :: day
integer, intent(in) :: hour
```

DESCRIPTION:

The arguments are:

filename the name of the forcing file to be read

dir directory containing the VIC-processed forcing data

year year

month month

day day

hour hour

This routine generates the filename of the VIC-processed forcing file to be read.

53.0.3 readviccrd (Source File: readviccrd.F90)

REVISION HISTORY:

```
16 Nov 2011; James Geiger, Initial Code
```

INTERFACE:

```
subroutine readviccrd()
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_config
use LIS_logMod,     only : LIS_logunit
use vic_forcingMod, only : vicforcing_struc
```

DESCRIPTION:

This routine reads the options specific to VIC formatted forcing from the LIS configuration file.

53.0.4 time_interp_vicforcing (Source File: time_interp_vicforcing.F90)

INTERFACE:

```
subroutine time_interp_vicforcing(n, findex)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_FORC_AttributesMod
use LIS_metforcingMod,  only : LIS_forc, LIS_FORC_Base_State
use LIS_logMod,         only : LIS_verify
use vic_forcingMod,     only : vicforcing_struc
```

ARGUMENTS:

```
implicit none

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

DESCRIPTION:

This routine performs the temporal interpolation of the VIC-processed forcing data to the LIS model time-step.
The arguments are:

n index of the nest

findex index of the supplemental forcing scheme

53.0.5 vic412_read_gridded_forcing_data (Source File: vic412_read_gridded_forcing_data.F90)

INTERFACE:

```
subroutine vic412_read_gridded_forcing_data(n, findex, filename, ferror)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_logMod,         only : LIS_getNextUnitNumber, &
                               LIS_releaseUnitNumber, &
                               LIS_logunit, LIS_endrun
```

```
use LIS_metforcingMod,  only : LIS_forc
use vic_forcingMod,     only : vicforcing_struc
use vic412_lsmMod,      only : vic412_struc
```

ARGUMENTS:

```
implicit none

integer, intent(in)            :: n
integer, intent(in)            :: findex
character(len=140), intent(in) :: filename
integer, intent(out)           :: ferror
```

DESCRIPTION:

This routine reads the VIC-processed forcing data from disk and stores it within the suppdata arrays. The arguments are:

n index of the nest

findex index of the supplemental forcing scheme

53.0.6 interp_vic (Source File: vic412_read_gridded_forcing_data.F90)

INTERFACE:

```
subroutine interp_vic(n,findex,var,NLEN,f,tempvic)
```

USES:

```
use LIS_coreMod,     only : LIS_rc, LIS_domain
use vic_forcingMod,  only : vicforcing_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)                          :: n
integer, intent(in)                          :: findex
integer, intent(in)                          :: var
integer, intent(in)                          :: NLEN
real, dimension(NLEN), intent(in)            :: f
real, dimension(LIS_rc%ngrid(n)), intent(out) :: tempvic
```

DESCRIPTION:

Interpolates the VIC formatted forcing data onto the LIS grid. The arguments are:

n index of the nest

NLEN size of the VIC forcing data

var variable index

f native VIC forcing data

g interpolated VIC forcing data

The routines invoked are:

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative interpolation

---

53.1 Fortran: Module Interface vic_forcingMod (Source File: vic_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of VIC formatted forcing data.
The implementation in LIS has the derived data type `vicforcing_struc` that includes the variables that specify the runtime options.
They are described below:

vicdir Directory containing the input data

forcingInterval Interval of the forcing data [hours]. This value should be either VIC's model time-step for the energy-balance mode or the snow time-step for the water-balance mode.

NC specifies the number of columns (longitude grid-cells) for the gridded VIC data (not used when read_gridded == 0)

NR specifies the number of rows (latitude grid-cells) for the gridded VIC data (not used when read_gridded == 0)

slat specifies the lower left latitude for the gridded VIC forcing domain

slon specifies the lower left longitude for the gridded VIC forcing domain

elat specifies the upper right latitude for the gridded VIC forcing domain

elon specifies the upper right longitude for the gridded VIC forcing domain

dlon specifies the domain resolution (dx) for the gridded VIC forcing domain

dlat specifies the domain resolution (dy) for the gridded VIC forcing domain

time1 The nearest, previous forcing interval instance of the incoming data (as a real time).

time2 The nearest, next forcing interval instance of the incoming data (as a real time).

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: defineNativevicforcing      !defines the native resolution of
                                      !the input data
```

PUBLIC TYPES:

```
public :: vicforcing_struc
```

53.1.1 defineNativevicforcing (Source File: vic_forcingMod.F90)

INTERFACE:

```
subroutine defineNativevicforcing(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
```

DESCRIPTION:

Defines the native resolution of the input forcing for the VIC formatted data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes V. Based on the specified data map projection and resolution, this routine sets up the spatial interpolation weights. The routines invoked are:

readviccrd (52.0.4)
reads the runtime options specified for VIC formatted data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
computes the neighbor, weights for conservative interpolation

---

53.2 Fortran: Module Interface vicforcing_finalize (Source File: vicforcing_finalize.F90)

INTERFACE:

```
subroutine vicforcing_finalize
```

USES:

53.3 Fortran: Module Interface vicforcing_reset (Source File: vicforcing_reset.F90)

INTERFACE:

```
subroutine vicforcing_reset
```

USES:

54 rdhm356 forcing

This section describes the implementation of the rdhm356 data.

54.1 Fortran: Module Interface LIS_XMRG_Reader (Source File: LIS_XMRG_Reader.F90)

This module defines routines to read NOAA XMRG and XMRG-like data. It automatically recognize big endian and little endian, compressed (gzip) and uncompressed (plain binary), XMRG and XMRG-like formats. All meta data XMRG and XMRG-like data formats are defined as module variables, which are described as below:

xor hrap-x coordinate of southwest corner of grid

yor hrap-x coordinate of southwest corner of grid

maxx number of hrap grid boxes in x direction

maxy number of hrap grid boxes in y direction

scale_factor scale factor to convert integer into real number (XMRG-like)

cell_size scale factor of cell size (XMRG-like)

nodata_value default value for missing data (XMRG-like)

num_byte number of bytes of data (2 for short integer and 4 for real, XMRG-like)

reverse_byte flag for little endian (0) and big endian (1)

rec2_len length of the second record for format recognition

REVISION HISTORY:

```
28 Nov 2013: Shugong Wang, Initial Code
02 Dec 2013: Shugong Wang, Add support to Sub-HRAP coordinate
  implicit none
  private
```

PUBLIC MEMBER FUNCTIONS:

```
  public :: xmrg_read_header
  public :: xmrg_read_data
  public :: xmrg_hrap_xy
  public :: xmrg_grid_latlon
  public :: latlon_to_hrap
  public :: hrap_to_latlon
```

54.1.1 xmrg_read_header (Source File: LIS_XMRG_Reader.F90)

INTERFACE:

```
  subroutine xmrg_read_header(xmrg_name,     &
                              x_origin,      &
                              y_origin,      &
                              num_row,       &
                              num_col,       &
                              nodata_value,  &
                              len_rec2,      &
                              cell_size,     &
                              status)
```

DESCRIPTION:

This routine reads the meta information of a XMRG or XMRG-like file from its headers. It retrieves the HRAP origin, numbers of colum and rows, and nodata value of XMRG or XMRG-lile data.
Description of arguments:

xmrg_name file name of XMRG/XMRG-like file

x_origin HRAP X coordiante of the lower left corner of data domain, optional output

y_origin HRAP Y coordinate of the lower left corner of data domain, optional output

num_row row number of data domain, optional output

num_col col number of data domain, optional output

nodata_value no-data value, optional output

len_rec2 length of the second record, optional output

cell_size HRAP unit of cell size, optional output

status return status: 0 - success, 1 - fail

The routine invoked is:

xmrg_read_header_c C function reading XMRG/XMRG-like header records (1st and 2nd records). (54.3.10

54.1.2 xmrg_read_data (Source File: LIS_XMRG_Reader.F90)

INTERFACE:

```
subroutine xmrg_read_data(xmrg_name,   &
                          xmrg_data,   &
                          xmrg_nrow,   &
                          xmrg_ncol,   &
                          nodata_value,&
                          status)
```

DESCRIPTION:

This routine reads the data of a XMRG/XMRG-like file. It retrieves data in a 2D array. The row number and the column number of XMRG/XMRG-lie data are optional output. No-data value is an optional input to exclude no-data values from scaling operation. Default no-data value is -1 if nodata_value is not presented.
Description of arguments:

xmrg_name name of XMRG/XMRG-like file

xmrg_data 2D array of XMRG/XMRG-like data starting from the lower left corner of the data domain

xmrg_nrow row number of XMRG/XMRG-like data, optional output

xmrg_ncol column number of XMRG/XMRG-like data, optional output

nodata_value no-data value of XMRG/XMRG-like data, optional input. default value is -1 when not explicitly specified.

status return status: 0 - success, 1 - fail

The routine invoked is:

xmrg_read_data_c (54.3.9) C function reading data from XMRG/XMRG-like file.

xmrg_read_header (54.1.1) read row and column numbers of XMRG/XMRG-lie data

---

54.1.3 xmrg_hrap_xy (Source File: LIS_XMRG_Reader.F90)

INTERFACE:

```
subroutine xmrg_hrap_xy(xmrg_name, hrapx, hrapy, resolution, status)
```

DESCRIPTION:

This routine retrieves HRAP coordinates (HRAPX, HRAPY) of XMRG/XMRG-like data boxes. HRAP resolution is an optional input. The default resolution is 1.0 HRAP unit.
Description of arguments:

xmrg_name name of XMRG/XMRG-like file

hrapx HRAP X coordinates, allocatable 2D array

hrapy HRAP Y coordinates, allocatable 2D array

resolution HRAP resolution of XMRG/XMRG-like data, optional input

status return status: 0 - success, 1 - fail, optional output

The routine invoked is:

xmrg_read_header (54.1.1) Read header information of XMRG/XMRG-like file.

54.1.4 xmrg_grid_latlon (Source File: LIS_XMRG_Reader.F90)

INTERFACE:

```
subroutine xmrg_grid_latlon(xmrg_name, lat, lon, resolution, status)
```

DESCRIPTION:

This routine retrieves lower left latitude-longitude coordinates of XMRG/XMRG-like data boxes. HRAP resolution is an optional input. The default resolution is 1.0 HRAP unit.
Description of arguments:

xmrg_name name of XMRG/XMRG-like file

lat latitude of data domain, allocatable 2D array

lon longitude of data domain, allocatable 2D array

resolution HRAP resolution of XMRG/XMRG-like data, optional input

status return status: 0 - success, 1 - fail, optional output

The routines invoked are:

xmrg_read_header (54.1.1)

hrap_to_latlon (54.1.6)

54.1.5 latlon_to_hrap (Source File: LIS_XMRG_Reader.F90)

INTERFACE:

```
subroutine latlon_to_hrap(rlon, rlat, hrap_x, hrap_y)
```

DESCRIPTION:

This routine convert latitude and longitude into HRAP coordinates (HRAPX and HRAPY). The code is adapted from `http://www.nws.noaa.gov/oh/hrl/dmip/lat_lon.txt`

54.1.6 hrap_to_latlon (Source File: LIS_XMRG_Reader.F90)

INTERFACE:

```
subroutine hrap_to_latlon(hrap_x, hrap_y, rlon, rlat)
```

DESCRIPTION:

This routine converts HRAP coordinates into latitude and longitude The code is adapted from `http://www.nws.noaa.gov/oh/hrl/dmip/lat_lon.txt`

54.1.7 rdhm356_temper_file (Source File: rdhm356_file_name.F90)

INTERFACE:

```
subroutine rdhm356_temper_file( name, rdhm356_temper_dir, yr, mo, da, hr)

  implicit none
```

DESCRIPTION:

This subroutine puts together a STAGE2 filename for one hour file intervals
The arguments are:

rdhm356_temper_dir Name of the STAGE II directory

yr year

mo month

da day of month

hr hour of day

name name of the time-stamped STAGE II file

54.1.8 rdhm356_precip_file (Source File: rdhm356_file_name.F90)

INTERFACE:

```
subroutine rdhm356_precip_file( name, rdhm356_precip_dir, yr, mo, da, hr)

  implicit none
```

DESCRIPTION:

This subroutine puts together a STAGE2 filename for one hour file intervals
The arguments are:

rdhm356_precip_dir Name of the STAGE II directory

yr year

mo month

da day of month

hr hour of day

name name of the time-stamped STAGE II file

54.2 Fortran: Module Interface finalize_rdhm356 (Source File: rdhm356_finalize.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
17Jul2006; K. Arsenault, Added dmip II
19Dec2013; Shugong Wang, RDHM356
```

INTERFACE:

```
subroutine finalize_rdhm356(findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use rdhm356_forcingMod, only : rdhm356_struc_precip, &
                               rdhm356_struc_temper, &
                               const_wind
```

DESCRIPTION:

Routine to cleanup dmip II forcing related memory allocations.

54.3 Fortran: Module Interface rdhm356_forcingMod (Source File: rdhm356_forcingMod.F9

This module contains variables and data structures that are used for the implementation of the precipitation and temperature data from the National Center for Environmental Prediction (NCEP) dmip II (dmip2) Doppler Radar+gage product. The dmip II is a national level product, on an hourly interval, and supplements mainly the which base forcing precipitation (e.g., NLDAS) is being used.
The implementation in LIS has the derived data type `rdhm356_struc` that includes the variables to specify the runtime options, and the weights and neighbor information for spatial interpolation
They are desribed below:

ncol Number of columns (along the east west dimension) for the input data

nrow Number of rows (along the north south dimension) for the input data

rdhm356dir Directory containing the input data

rdhm356time The nearest, hourly instance of the incoming data (as a real time).

mi Number of points in the input grid

rlat1 Array containing the latitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

rlon1 Array containing the longitudes of the input grid for each corresponding grid point in LIS (to be used for bilinear interpolation)

n111,n121,n211,n221 Arrays containing the neighbor information of the input grid for each grid point in LIS, for bilinear interpolation.

w111,w121,w211,w221 Arrays containing the weights of the input grid for each grid point in LIS, for bilinear interpolation.

n112,n122,n212,n222 Arrays containing the neighbor information of the input grid for each grid point in LIS, for conservative interpolation.

w112,w122,w212,w222 Arrays containing the weights of the input grid for each grid point in LIS, for conservative interpolation.

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init_rdhm356      !defines the native resolution of
                                !the input data
```

PUBLIC TYPES:

```
public :: rdhm356_struc_precip, rdhm356_struc_temper, const_wind
```

54.3.1 init_rdhm356 (Source File: rdhm356_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
25May2006: Kristi Arsenault; Data and code implementation
20Jul2006: Brian Cosgrove; Conversion to DMIP2
03May2010: Soni Yatheendradas; Precip and Temper. input grids now can have
           different extents and different from the run-domain extent, as
           per the new input grids posted onto the DMIP2 website for Sierra
           Nevada (hardcoding input grid extents for now, will make flexible
           later by making a call to read_xmrg2 and read_xmrgtemp2 from here)

17Dec2013: Shugong Wang; (1) replace old stlye XMRG reader with LIS_XMRG_Reader
                         (2)
```

INTERFACE:

```
subroutine init_rdhm356(findex)
```

USES:

```
use LIS_coreMod, only: LIS_rc
use LIS_timeMgrMod, only : LIS_date2time, LIS_update_timestep
use LIS_XMRG_READER
implicit none
integer,  intent(in) :: findex
```

DESCRIPTION:

Defines the native resolution of the input forcing for dmip2 data. The grid description arrays are based on the decoding schemes used by NCEP and followed in the LIS interpolation schemes V
The routines invoked are:

readcrd_rdhm356 (54.3.7)
reads the runtime options specified for dmip2 data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

LIS_date2time (3.36.24)
converts date to the real time format - time of grid change

54.3.2 getrdhm356 (Source File: rdhm356_get_forcing.F90)

REVISION HISTORY:

```
25 May 2006: Kristi Arsenault;  Data and Code implementation
18 Dec 2013: Shugong Wang; implementation for RDHM356
```

INTERFACE:

```
subroutine get_rdhm356(n,findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod, only : LIS_tick, LIS_get_nstep
use LIS_logMod, only : LIS_logunit
use rdhm356_forcingMod, only : rdhm356_struc_precip, &
                               rdhm356_struc_temper


implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates hourly rdhm356 forcing. At the beginning of a simulation, the code reads the most recent past data (nearest the hour interval), and the nearest future data. These two datasets are used to temporally interpolate the data to the current model timestep.
The arguments are:

n index of the nest

timenow Current LIS Time

rdhm356_file_time2 End boundary time of dmip II file

file_name forcing file filename - passed back to getstg

The routines invoked are:

LIS_tick (3.36.27)
determines the RDHM data times

bilinear_interp_input (5.1.2)
Computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.24)
Computes the neighbor, weights for conservative interpolation

rdhm356_precip_file (54.1.8)
Puts together appropriate file name for 1-hour intervals

rhdm356_read_precip (54.3.5)
Interpolates rdhm356 data to LIS grid

54.3.3 interp_rdhm356_precip (Source File: rdhm356_interp_precip.F90)

REVISION HISTORY:

```
03 May 2010: Soni Yatheendradas; Precip and Temper. input grids now can have different
                                 extents/directories and different from the run-domain
                                 extent, as per the new input grids posted onto the
                                 DMIP2 website for Sierra Nevada
18 Dec 2013: Shugong Wang; implementation for RDHM356
```

INTERFACE:

```
subroutine interp_rdhm356_precip (n,findex, ksec1, nrdhm356, &
                                  ppt_field, lb, lis_gds, &
                                  nc, nr, varfield )
```

USES:

```
use rdhm356_forcingMod, only : rdhm356_struc_precip
use LIS_coreMod, only: LIS_rc

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
integer, intent(in)    :: findex
integer                :: nc      ! Number of columns (in the E-W dimension) in the LIS grid
integer                :: nr      ! Number of rows (in the N-S dimension) in the LIS grid
integer                :: nrdhm356   ! Number of points in original dmip II grid
integer                :: ksec1(100)
real                   :: lis_gds(50)
real                   :: ppt_field(nrdhm356)
logical*1              :: lb(nrdhm356)
real, dimension(nc,nr) :: varfield
```

DESCRIPTION:

This subroutine interpolates a given STG2 field to the LIS grid. The arguments are:

n index of the nest

ksec1 grib decoding array

nrdhm356 number of elements in the input grid

ppt_field input data array to be interpolated

lb input bitmap

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

bilinear_interp (5.1.1)
Spatially interpolate the forcing data using bilinear interpolation, or

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative (budget bil.) interpolation

54.3.4 interp_rdhm356_temper (Source File: rdhm356_interp_temper.F90)

REVISION HISTORY:

```
03 May 2010: Soni Yatheendradas; Precip and Temper. input grids now can have different
                                 extents/directories and different from the run-domain
                                 extent, as per the new input grids posted onto the
                                 DMIP2 website for Sierra Nevada
18 Dec 2013: Shugong Wang; implementation for RDHM356
```

INTERFACE:

```
subroutine interp_rdhm356_temper (n,findex, ksec1, nrdhm356, &
                                  ppt_field, lb, lis_gds, &
                                   nc, nr, varfield )
```

USES:

```
use rdhm356_forcingMod, only : rdhm356_struc_temper
use LIS_coreMod, only: LIS_rc

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
integer, intent(in)    :: findex
integer                :: nc      ! Number of columns (in the E-W dimension) in the LIS grid
integer                :: nr      ! Number of rows (in the N-S dimension) in the LIS grid
integer                :: nrdhm356   ! Number of points in original dmip II grid
integer                :: ksec1(100)
real                   :: lis_gds(50)
real                   :: ppt_field(nrdhm356)
logical*1              :: lb(nrdhm356)
real, dimension(nc,nr) :: varfield
```

DESCRIPTION:

This subroutine interpolates a given STG2 field to the LIS grid. The arguments are:

n index of the nest

ksec1 grib decoding array

nrdhm356 number of elements in the input grid

ppt_field input data array to be interpolated

lb input bitmap

lis_gds array description of the LIS grid

nc number of columns (in the east-west dimension) in the LIS grid

nr number of rows (in the north-south dimension) in the LIS grid

varfield output interpolated field

The routines invoked are:

bilinear_interp (5.1.1)
Spatially interpolate the forcing data using bilinear interpolation, or

conserv_interp (5.1.23)
spatially interpolate the forcing data using conservative (budget bil.) interpolation

---

54.3.5 rdhm356_read_precip (Source File: rdhm356_read_precip.F90)

REVISION HISTORY:

```
25 May 2006: Kristi Arsenault;  Data and code implementation
18 Dec 2013: Shugong Wang; implementation for RDHM 356
```

INTERFACE:

```
subroutine rdhm356_read_precip(n, fname, findex, order, ferror_rdhm356)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_logMod,         only : LIS_logunit, LIS_endrun
use LIS_metforcingMod,  only : LIS_forc
use rdhm356_forcingMod, only : rdhm356_struc_precip
use LIS_XMRG_READER

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
character(len=*)   :: fname
integer, intent(in) :: findex
integer, intent(in) :: order
integer             :: ferror_rdhm356
```

DESCRIPTION:

For the given time, reads RDHM 356 precipitation and interpolates to a designated user-domain. NOTE:: These subroutines use the LIS XMRG Reader for opening and reading the xmrg files.
The arguments are:

n index of the nest

fname name of the hourly xmrg file

ferror_rdhm356 flag to indicate success of the call (=0 indicates success)

filehr current file hour

The routines invoked are:

interp_rdhm356_precip (54.3.3)
spatially interpolates the rdhm356 data

54.3.6 rdhm356_read_temper (Source File: rdhm356_read_temper.F90)

REVISION HISTORY:

```
25 May 2006: Kristi Arsenault;  Data and code implementation
19 Dec 2013: Shugong Wang; Impmementation for RDHM 356
```

INTERFACE:

```
subroutine rdhm356_read_temper( n, findex, order, fname, ferror_rdhm356 )
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_logMod,         only : LIS_logunit
use LIS_metforcingMod,  only : LIS_forc
use rdhm356_forcingMod, only : rdhm356_struc_temper
use LIS_XMRG_READER

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
character(len=*)   :: fname
integer, intent(in) :: findex
integer, intent(in) :: order
integer             :: ferror_rdhm356
```

DESCRIPTION:

For the given time, reads RDHM 356 temperature and interpolates to a designated user-domain. NOTE:: These subroutines use the LIS XMRG Reader for opening and reading the xmrg files.
The arguments are:

n index of the nest

fname name of the hourly xmrg file

ferror_rdhm356 flag to indicate success of the call (=0 indicates success)

filehr current file hour

The routines invoked are:

interp_rdhm356_temper (54.3.4)
spatially interpolates the rdhm356 data

54.3.7 readcrd_rdhm356 (Source File: rdhm356_readcrd.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
05 Jun 2006; Kristi Arsenault, Code and data implementation
03 May 2010: Soni Yatheendradas; Precip and Temper. input grids now can
             have different extents/directories and different from the
             run-domain extent, as per the new input grids posted onto
             the DMIP2 website for Sierra Nevada
06 Man 2014: Shugong Wang, modified for RDHM 3.5.6
```

INTERFACE:

```
subroutine readcrd_rdhm356()
```

USES:

```
use ESMF
use rdhm356_forcingMod, only : rdhm356_struc_precip, &
                               rdhm356_struc_temper, &
                               const_wind
use LIS_coreMod, only : LIS_config, LIS_rc
use LIS_logMod, only : LIS_logunit
```

DESCRIPTION:

This routine reads the options specific to RDHM 3.5.6 forcing from the LIS configuration file. The routines invoked are:

ESMF_ConfigGetAttribute read the specified attribute

ESMF_ConfigFindLabel find the specified label to read the attribute

---

54.3.8 timeinterp_rdhm356 (Source File: timeinterp_rdhm356.F90)

REVISION HISTORY:

```
25May2006: Kristi Arsenault; Initial implementation
10 Oct 2006:   Sujay Kumar: Switched to using ESMF_State for storing
                            forcing data.
03 May 2010: Soni Yatheendradas; Precip and Temper. input grids now
             can have different extents/directories and different from
             the run-domain extent, as per the new input grids posted
             onto the DMIP2 website for Sierra Nevada
19 Dec 2013; Shugong Wang; RDHM356
```

INTERFACE:

```
subroutine timeinterp_rdhm356(n, findex)
```

USES:

```
use ESMF
use LIS_coreMod, only: LIS_rc, LIS_domain
use LIS_FORC_AttributesMod
use LIS_metforcingMod, only : LIS_FORC_Base_State, LIS_forc
use LIS_logMod,         only : LIS_verify, LIS_endrun
use rdhm356_forcingMod, only : rdhm356_struc_precip, &
```

```
                                    rdhm356_struc_temper, &
                                    const_wind

     implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep.
The arguments are:

n index of the nest

54.3.9 xmrg_read_data_c (Source File: xmrg_read_data_c.c)

INTERFACE:

```
 int FTN(xmrg_read_data_c)(char *in_file_name, float *data1d, float *nodata_default)
```

DESCRIPTION:

This function reads the data records of XMRG/XMRG-like files. It automatically determine XMRG and XMRG-like, big endian and little endian, and short integer and real number. Functions invoked by xmrg_read_header_c are

reverse_byte_order (54.3.11)

reverse_byte_order_short (54.3.12)

gzopen

gzseek

gzread

gzclose

54.3.10 xmrg_read_header_c (Source File: xmrg_read_header_c.c)

INTERFACE:

```
 int FTN(xmrg_read_header_c)(char  *in_file_name,
                            float *XOR,
                            float *YOR,
                            int   *MAXX,
                            int   *MAXY,
                            float *CELL_SIZE,
                            float *SCALE_FACTOR,
                            float *NODATA_VALUE,
                            int   *NUM_BYTE,
                            int   *REVERSE_BYTE,
                            int   *REC2_LEN)
```

DESCRIPTION:

This routine reads the first and the second records of XMRG/XMRG-like file. It retrieves the coordinate of the south west corner, numbers of column and rows of XMRG/XMRG-like data. If specified, cell size, scale factor and nodata value also will be retrieved.
IO functions of LIBZ, such as gzopen, gzread, gzseek and gzclose are used instead one of standard IO library. LIBZ functions can deal with compressed (gzipped) and uncompressed formats.
Functions invoked by xmrg_read_header_c_ are

reverse_byte_order (54.3.11)

gzopen

gzseek

gzread

gzclose

54.3.11 reverse_byte_order (Source File: xmrg_reverse_byte.c)

INTERFACE:

```
void reverse_byte_order(int *in_array,int arraysize)
```

DESCRIPTION:

This routine reverses byte order for 4-byte integer. Real number has to be casted into (int *) type when calling reverse_byte_order. This code is adapted from `http://www.nws.noaa.gov/oh/hrl/dmip/2/src/read_xmrg2.c`

54.3.12 reverse_byte_order_short (Source File: xmrg_reverse_byte.c)

INTERFACE:

```
void reverse_byte_order_short(short *in_array,int arraysize)
```

DESCRIPTION:

This routine reverses byte order for 2-byte integer. This code is adpated from `http://www.nws.noaa.gov/oh/hrl/dmip/2/src/read_xmrg2.c`

Part XI
Data Assimilation in LIS

55 Data Assimilation Algorithms in LIS

This section contains the description of the implementations of data assimilation algorithms in LIS. These algorithms are used in conjunction with the specific interface implementations of the land surface model and the observations, which are described in sections XIV and XII, respectively.

---

56 Direct Insertion

56.1 Fortran: Module Interface directInsertion_Mod (Source File: directInsertion_Mod.F90)

This module contains interfaces and subroutines that control the incorporation of a data set using direct insertion in a land surface model.

REVISION HISTORY:

```
 27Feb05    Sujay Kumar;   Initial Specification
 21Jun06    Sujay Kumar:   Updated implementation with ESMF
                           structures
```

USES:

```
  implicit none
  PRIVATE
 ------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
 ------------------------------------------------------------------------------
  public :: di_init  ! Initialization for Direct Insertion
  public :: di_setup
  public :: di_increments ! Routine to compute DI increments
  public :: di_update     ! Routine to apply DI increments
  public :: di_diagnostics ! write DI related diagnostics
  public :: di_final ! Finalization for Direct Insertion
```

56.1.1 di_init (Source File: directInsertion_Mod.F90)

INTERFACE:

```
subroutine di_init
```

DESCRIPTION:

This method performs the required initializations for the direct insertion method.

56.1.2 di_increments (Source File: directInsertion_Mod.F90)

INTERFACE:

```
subroutine di_increments(n,k)
```

DESCRIPTION:

This method computes the analysis increments from direct insertion. The direct insertion method simply retrieves the LSM state, prompts the land surface model to transform the OBS state to an LSM state. The tranformed LSM state is then used to overwrite the current LSM state.
The methods invoked are:

lsmdagetstatevar (3.50.22)
obtain the specified LSM state prognostic variables

Lsmdaobstransform (3.50.26)
transform the observation state into an LSM space

lsmdamapobstolsm (3.50.32)
map the observation state to generate the analysis increments

USES:

```
use ESMF
use LIS_coreMod,only          : LIS_rc
use LIS_lsmMod, only          : LIS_LSM_State, LIS_LSM_Incr_State
use LIS_DAobservationsMod, only : LIS_OBS_State
use LIS_logMod, only          : LIS_logunit, LIS_verify
```

56.1.3 di_update (Source File: directInsertion_Mod.F90)

INTERFACE:

```
subroutine di_update(n,k)
```

USES:

```
use ESMF
use LIS_coreMod, only         : LIS_rc
use LIS_lsmMod, only          : LIS_LSM_State, LIS_LSM_Incr_State
```

ARGUMENTS:

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

DESCRIPTION:

This method applies the analysis increments computed by the direct insertion method and updates the model prognostic variables.
The methods invoked are:

lsmdasetstatevar (3.50.24)
set the specified LSM state prognostic variables

56.1.4 di_diagnostics (Source File: directInsertion_Mod.F90)

INTERFACE:

```
subroutine di_diagnostics(n,k)
```

DESCRIPTION:

This subroutine generates the direct insertion related diagnostics

---

56.1.5 di_final (Source File: directInsertion_Mod.F90)

INTERFACE:

```
subroutine di_final
```

DESCRIPTION:

This method performs the finalization for all direct insertion related structures and subroutines.

---

57 GMAO Ensemble Kalman Filter

57.1 Fortran: Module Interface enkf_Mod (Source File: enkf_Mod.F90)

This module contains interfaces and subroutines that control the incorporation of a data set using the ensemble kalman filter (EnKF) method, into a land surface model.
The EnKF algorithm is based on the work of Rolf Reichle at the NASA Global Modeling and Assimilation Office (GMAO) at the NASA GSFC.
NOTES: Data assimilation is currently only supported for land surface models (and not across different surface model types)

REVISION HISTORY:

```
27 Feb 2005: Sujay Kumar; Initial Specification
```

USES:

```
  use ESMF
  use enkf_types
  use enkf_general
  use my_matrix_functions
  use LIS_coreMod
  use LIS_logMod
  use LIS_DAobservationsMod
  use LIS_lsmMod
  use LIS_fileIOMod
  use LIS_historyMod
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
```

```
  use netcdf
#endif

  implicit none

  PRIVATE
 ------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
 ------------------------------------------------------------------------------
  public :: enkf_init  ! Initialization for EnKF
  public :: enkf_setup
  public :: enkf_increments ! compute analysis increments
  public :: enkf_update ! apply the analysis increments
  public :: enkf_diagnostics ! write EnKF related diagnostics
  public :: enkf_final ! Finalization for EnKF
 ------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
 ------------------------------------------------------------------------------
  public :: enkf_struc ! data structure containing EnKF diagnostics
```

57.1.1 enkf_init (Source File: enkf_Mod.F90)

INTERFACE:

```
subroutine enkf_init()
```

USES:

DESCRIPTION:

This method performs the required initializations for the GMAO EnKF method. The method reads the runtime settings from the LIS configuration file.

57.1.2 enkf_setup (Source File: enkf_Mod.F90)

INTERFACE:

```
subroutine enkf_setup(k)
```

USES:

DESCRIPTION:

This method performs the required initializations for the GMAO EnKF method. The method reads the runtime settings from the LIS configuration file.

57.1.3 enkf_increments (Source File: enkf_Mod.F90)

INTERFACE:

```
subroutine enkf_increments(n,k)
```

USES:

ARGUMENTS:

```
integer, intent(IN)    :: n
integer, intent(IN)    :: k
```

DESCRIPTION:

This routine computes the analysis increments for EnKF. The state variable increments are computed and stored int the Increment State.
The arguments and variables are:

n index of the nest

data_status flag indicating if the observations are new

N_state Number of state prognostic variables

N_ens Number of ensembles

N_obs_size total number of observations

N_selected_obs number of selected observations

st_id,en_id indices of the selected observations

stvar state prognostic variables

Obs_pert observation perturbations

Obs_cov observations error covariances

Observations observations object used in the GMAO routines

The methods invoked are:

generateObservations 57.1.11
obtain the observations

lsmdagetobspred (3.50.28)
obtain model's estimate of the observations

getObsPert (57.1.10)
obtain the observation perturbations

generateObsparam (57.1.12)
generate the 'obsparam' (metadata for observations)

lsmdagetstatevar 3.50.22 obtain the specified LSM prognostic variables.

assemble_obs_cov (57.1.7) assembles the observation error covariance

getSelectedObsNumber(3.42.14)
obtain the number of selected observations for each modeling point.

enkf_analysis (57.2.1)
apply the EnKF filter to obtain the prognostic variable increments.

row_variance (57.4.2)
computes the row variance HPH'

lsmdasetstatevar (3.50.24)
assigns the specified LSM state prognostic variables

scaleLSMstatVar (3.50.38)
scales the state variables for computational stability

descaleLSMstatVar (3.50.40)
descales the state variables to the original state

lsmdaqcstate 3.50.34 QC the updated LSM state

57.1.4 enkf_update (Source File: enkf_Mod.F90)

INTERFACE:

```
subroutine enkf_update(n,k)
```

USES:

```
implicit none

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

DESCRIPTION:

This routine updates the model prognostics using the analysis increments computed earlier.

57.1.5 enkf_diagnostics (Source File: enkf_Mod.F90)

INTERFACE:

```
subroutine enkf_diagnostics(n,k)
```

USES:

ARGUMENTS:

```
integer, intent(IN)    :: n
integer, intent(IN)    :: k
```

DESCRIPTION:

This subroutine generates the EnKF related diagnostics and outputs it to a file. This includes a text output of selected ensemble members, their mean, standard deviation, observations, and normalized innovations. The frequency of diagnostic outputs can be specified in the LIS configuration file.
The routines invoked are:

LIS_create_output_directory (3.14.6)
call to create output directory for DA statistics

getLSMvarnames (57.1.9)
retrieve the names of the LSM prognostic variables

pruneVarname (57.1.6)
trims the variable name, eliminating white spaces

LIS_create_stats_filename (3.14.11)
creates the filename for statistics

LIS_create_innov_filename (3.14.12)
creates the name of the innovations file

lsmdagetstatevar (3.50.22)
retrieve the lsm state variables

getLSMdata (57.1.13)
unpack the LSM state and retrive the data

57.1.6 pruneVarname (Source File: enkf_Mod.F90)

INTERFACE:

```
subroutine pruneVarname(varname)
```

ARGUMENTS:

```
character(len=*), intent(INOUT) :: varname
```

DESCRIPTION:

This routine generates a filename based on the names of the state prognostic variables

varname name of the variable filename

57.1.7 assemble_obs_cov (Source File: enkf_Mod.F90)

INTERFACE:

```
subroutine assemble_obs_cov(nob, N_obs, obs_param, &
     Observations, Obs_cov)

  implicit none
```

ARGUMENTS:

```
integer, intent(in)                              :: nob, N_obs
type(obs_param_type), dimension(nob), intent(in) :: obs_param
type(obs_type), dimension(N_obs), intent(in)     :: Observations
real, intent(out), dimension(N_obs,N_obs)        :: Obs_cov
```

DESCRIPTION:

assemble measurements error covariance (reichle, 27 Jul 2005)

nob number of observation types

N_obs number of observations

obs_param observation settings

Observations Observations data type

Obs_cov observation error covariance

57.1.8 enkf_final (Source File: enkf_Mod.F90)

INTERFACE:

```
subroutine enkf_final
```

DESCRIPTION:

This method performs the finalization for all EnKF related structures and subroutines.

57.1.9 getLSMvarnames (Source File: enkf_Mod.F90)

INTERFACE:

```
subroutine getLSMvarnames(LIS_LSM_State, dim1, varname)
```

USES:

ARGUMENTS:

```
type(ESMF_State)      :: LIS_LSM_State
integer               :: dim1
character(len=*)      :: varname(dim1)
```

DESCRIPTION:

This routine retrieves the names of the state prognostic variables from the LSM state.

57.1.10 getObsPert (Source File: enkf_Mod.F90)

INTERFACE:

```
subroutine getObsPert(OBS_Pert_State, gsize, dim1, dim2, pert)
```

USES:

ARGUMENTS:

```
type(ESMF_State)      :: OBS_Pert_State
integer               :: gsize
integer               :: dim1, dim2
real                  :: pert(dim1,dim2)
```

DESCRIPTION:

This routine retrieves the perturbations to be applied to the observations from the Observation perturbation state
The arguments are:

OBS_Pert_State Observation perturbation state

dim1,dim2 dimensions of the perturbations array

pert perturbation array

57.1.11 generateObservations (Source File: enkf_Mod.F90)

INTERFACE:

```
subroutine generateObservations(n, Nobjs, N_obs_size, LIS_OBS_State, &
     LIS_OBS_Pert_State, Observations )
```

USES:

ARGUMENTS:

```
integer,     intent(IN)        :: n
integer,     intent(IN)        :: Nobjs
integer,     intent(IN)        :: N_obs_size
type(ESMF_State)               :: LIS_OBS_State
type(ESMF_State)               :: LIS_OBS_Pert_State
type(obs_type)                 :: Observations(N_obs_size)
```

DESCRIPTION:

This subroutine unpacks the Observation state and packages them into the 'obs_type' datastructure.

n index of the nest

Nobjs Number of observation types

OBS_State ESMF State containing observations

Observations Observations datastructure being returned.

57.1.12 generateObsparam (Source File: enkf_Mod.F90)

INTERFACE:

```
subroutine generateObsparam(Nobjs, OBS_Pert_State, obs_param)
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer                       :: Nobjs
type(ESMF_State)              :: OBS_Pert_State
type(obs_param_type)          :: obs_param(Nobjs)
```

DESCRIPTION:

This routine obtains the metadata information associated with the observations and puts them into the obs_param datastructure
The arguments are:

Nobjs Number of observation types

OBS_Pert_State ESMF state containing observation perturbations

obs_param obs_param datastructure being returned

57.1.13 getLSMData (Source File: enkf_Mod.F90)

INTERFACE:

```
subroutine getLSMData(LIS_LSM_State, dim1, dim2, lsmdata)
```

USES:

ARGUMENTS:

```
type(ESMF_State)      :: LIS_LSM_State
integer               :: dim1, dim2
real                  :: lsmdata(dim1, dim2)
```

DESCRIPTION:

This routine retrives the LSM state prognostic variables from the LSM state.
The arguments are:

LSM_State LSM State containing prognostic variables

dim1,dim2 dimensions of the prognostic variable array

lsmdata prognostic variable array

57.2 Fortran: Module Interface enkf_general (Source File: enkf_general.F90)

this file contains a collection of general Ensemble Kalman filter subroutines and compact support subroutines

REVISION HISTORY:

```
reichle, 20 Apr 2001
reichle, 18 Mar 2004 - optional arguments
reichle, 27 Jan 2005 - eliminated use of module select_kinds
reichle, 19 Jul 2005 - merged compact_support.f90 and enkf_general.f90
reichle,  1 Aug 2005 - eliminated tile_coord
reichle, 18 Oct 2005 - return increments instead of updated State
```

57.2.1 enkf_analysis (Source File: enkf_general.F90)

INTERFACE:

```
subroutine enkf_analysis( gid, &
     N_state, N_obs, N_ens, &
     Observations, Obs_pred, Obs_err, Obs_cov, &
     State_incr, &
     State_lon, State_lat, xcompact, ycompact )
```

USES:

```
use enkf_types
use my_matrix_functions
use LIS_logMod, only : LIS_logunit, LIS_endrun

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: gid
integer, intent(in) :: N_state, N_obs, N_ens
type(obs_type), intent(in), dimension(N_obs) :: Observations
real, intent(in), dimension(N_obs,N_ens) :: Obs_pred
real, intent(in), dimension(N_obs,N_ens) :: Obs_err
real, intent(in), dimension(N_obs,N_obs) :: Obs_cov
real, intent(inout), dimension(N_state,N_ens) :: State_incr

! optional inputs
real, dimension(N_state), intent(in), optional :: State_lon, State_lat
real, intent(in), optional :: xcompact       ! [deg] longitude
real, intent(in), optional :: ycompact       ! [deg] latitude
```

DESCRIPTION:

perform EnKF update
IMPORTANT: on input, State_incr must contain State_minus(1:N_state,1:N_ens) on output, State_incr contains the increments
if optional inputs State_lon, State_lat, xcompact, and ycompact are present, Hadamard product is applied to HPHt and PHt

---

57.2.2 hadamard_for_repr_matrix (Source File: enkf_general.F90)

INTERFACE:

```
subroutine hadamard_for_repr_matrix( N_obs, Observations,             &
     xcompact, ycompact,                                              &
     Repr_Matrix )
```

USES:

```
use enkf_types                     ! for obs_type

implicit none
```

ARGUMENTS:

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

type(obs_type), intent(in), dimension(N_obs) :: Observations

real, intent(in) :: xcompact       ! [deg] longitude
real, intent(in) :: ycompact       ! [deg] latitude

real, dimension(N_obs,N_obs), intent(inout) :: Repr_matrix
```

57.2.3 gaspari_cohn (Source File: enkf_general.F90)

INTERFACE:

```
function gaspari_cohn( d )
```

DESCRIPTION:

evaluate 5th-order polynomial from Gaspari and Cohn, 1999, Eq (4.10)
On input, d = separation distance relative to the distance at which all correlations vanish. In the isotropic case, Gaspari and Cohn, 1999, Eq. (4.10)

```
d = sqrt(dx**2 + dy**2) / (2*c) = |z| / (2*c)
```

or in the anisotropic case

```
d = sqrt( (dx/xcompact)**2 + (dy/ycompact)**2 )
```

```
*** Use |z|/c = 2*d. All correlations vanish for d > 1. ***
```

57.2.4 get_gaspari_cohn (Source File: enkf_general.F90)

INTERFACE:

```
function get_gaspari_cohn( dx, dy, xcompact, ycompact )
```

DESCRIPTION:

for a given lat/lon distance, compute the anisotropic compact support (Gaspari and Cohn) weights
input coordinates must be in degrees latitude/longitude
dx = longitude separation of two points
dy = latitude separation of two points
xcompact = longitude scale of compact support
ycompact = latitude scale of compact support
for the isotropic Gaspari and Cohn function, the relative distance is in Gaspari and Cohn, 1999, notation (Eq (4.10))
d = sqrt(dx**2 + dy**2) / 2*c
get_gaspari_cohn() uses a generalized anisotropic Gaspari and Cohn approach (essentially coordinate stretching, see handwritten notes for details)
d = sqrt((dx/xcompact)**2 + (dy/ycompact)**2)
All correlations vanish outside of an ellipse with semi-axes xcompact and ycompact, ie Gaspari and Cohn weights vanish for d > 1 (note the factor 2!)
When the anisotropic case is reduced back to the isotropic case, (ie if xcompact==ycompact) then c = xcompact/2 = ycompact/2.

57.3 Fortran: Module Interface enkf_types (Source File: enkf_types.F90)

This module contains the definition of types for the GMAO EnKF

REVISION HISTORY:

```
reichle, 19 Jul 2005
```

57.3.1 ludcmp (Source File: my_lu_decomp.F90)

LU decomposition after Numerical Recipes
Given a matrix a[1..n][1..n], this routine replaces it by the LU decomposition of a rowwise permutation of itself. a and n are inputs. a is output, arranged as in equation (2.3.14) (see book). indx[1..n] is an output vector that records the row permutation effected by the partial pivoting; This routine is used in combination with lubksb to solve linear equations or invert a matrix.

REVISION HISTORY:

```
edited 20 Apr 01, reichle
- eliminated d (for computation of determinant)
- eliminated np (can use automatic arrays nowadays...)
edited 03 Jun 02, reichle

- eliminated parameter NMAX, using dynamic allocation
```

INTERFACE:

```
SUBROUTINE ludcmp(a,n,indx)
```

57.3.2 lubksb (Source File: my_lu_decomp.F90)

LU backsubstitution after Numerical Recipes
Solves the set of n linear equations A X = B. Here a[1..n][1..n] is input, not as the matrix A but rather as its LU decomposition, determined by the routine ludcmp. indx[1..n] is input as the permutation vector returned by ludcmp. b[1..n] is input as the right-hand side vector B, and returns the solution vector X. a,n, and indx ar not modified by this routine and can be left in place for successive calls with different right-hand sides b. This routine takes into account the possibility that b will begin with many zero elements, so it is efficient for use in matrix inversion.

REVISION HISTORY:

```
edited 20 Apr 01, reichle
```

INTERFACE:

```
SUBROUTINE lubksb(a,n,indx,b)
```

57.4 Fortran: Module Interface my_matrix_functions (Source File: my_matrix_functions.F90

this file contains a collection of matrix operation subroutines

57.4.1 (Source File: my_matrix_functions.F90)

reichle, 1 May 01 reichle, 18 Apr 06 - renamed subroutine sort()

57.4.2 row_variance (Source File: my_matrix_functions.F90)

INTERFACE:

```
subroutine row_variance( M, N, A, var )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: M, N

  real, intent(in), dimension(M,N) :: A

  real, intent(out), dimension(M)  :: var
```

DESCRIPTION:

compute variance of each row of an M-by-N matrix A

57.4.3 row_third_moment (Source File: my_matrix_functions.F90)

INTERFACE:

```
subroutine row_third_moment( M, N, A, third_moment )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: M, N
  real, intent(in), dimension(M,N) :: A
  real, intent(out), dimension(M)  :: third_moment
```

DESCRIPTION:

compute third moment of each row of an M-by-N matrix A

$$third_moment = 1/N * sum_{i=1}^{N}(x_i - mean(x)) * *3 \tag{1}$$

57.4.4 five_number_summary (Source File: my_matrix_functions.F90)

INTERFACE:

```
subroutine five_number_summary( M, N, A, five_numbers )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: M, N
  real, intent(in), dimension(M,N)  :: A
  real, intent(out), dimension(M,5)  :: five_numbers
```

DESCRIPTION:

get five number summary (median, lower and upper quartiles, min and max) of each row of an M-by-N data matrix A
inputs: M : number of rows of data = number of different data types N : number of columns of data = number of ensemble members A : M-by-N matrix, left unchanged by this function
outputs: five_numbers : M-by-5 matrix containing statistical summary: column 1: min column 2: lower quartile column 3: median column 4: upper quartile column 5: max
Type: f90 Author: Rolf Reichle Date: 2 May 2001

57.4.5 adjust_mean (Source File: my_matrix_functions.F90)

INTERFACE:

```
subroutine adjust_mean( N_row, N_col, A, M )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: N_row, N_col
  real, intent(inout), dimension(N_row,N_col)  :: A
  real, intent(in), optional, dimension(N_row) :: M
```

DESCRIPTION:

adjust N_row by N_col matrix A such that mean over columns for each row is given by the corresponding element in vector M of length N_row
vector of mean values M is optional input, if not present zero mean is assumed

---

57.4.6 adjust_std (Source File: my_matrix_functions.F90)

INTERFACE:

```
subroutine adjust_std( N_row, N_col, A, std )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: N_row, N_col
  real, intent(inout), dimension(N_row,N_col)  :: A
  real, intent(in), optional :: std
```

DESCRIPTION:

adjust N_row by N_col matrix A such that (sample) standard deviation of all elements is exactly equal to std
std is optional input, if not present std=1 is assumed

57.4.7 matrix_std (Source File: my_matrix_functions.F90)

INTERFACE:

```
subroutine matrix_std( N_row, N_col, A, std )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: N_row, N_col
  real, intent(inout), dimension(N_row,N_col)  :: A
  real, intent(out) :: std
```

DESCRIPTION:

compute std of all elements of N_row by N_col matrix A

57.4.8 nr_sort (Source File: nr_sort.F90)

QUICKSORT algorithm after Numerical Recipes
Sorts an array arr[1..n] into ascending numerical order using the Quicksort algorithm. n is input; arr is replaced on output by its sorted rearrangement.

INTERFACE:

```
SUBROUTINE nr_sort(n,arr)
```

57.5 Fortran: Module Interface enks_general (Source File: enks_general.F90)

this file contains a collection of general Ensemble Kalman filter subroutines and compact support subroutines

REVISION HISTORY:

```
reichle, 20 Apr 2001
reichle, 18 Mar 2004 - optional arguments
reichle, 27 Jan 2005 - eliminated use of module select_kinds
reichle, 19 Jul 2005 - merged compact_support.f90 and enks_general.f90
reichle,  1 Aug 2005 - eliminated tile_coord
reichle, 18 Oct 2005 - return increments instead of updated State
```

57.5.1 enks_analysis (Source File: enks_general.F90)

INTERFACE:

```
subroutine enks_analysis( gid, &
     N_state, N_obs, N_ens, &
     Observations, Obs_pred, Obs_err, Obs_cov, &
     State_incr, &
     State_lon, State_lat, xcompact, ycompact )
```

USES:

```
use enks_types
use my_matrix_functions
use LIS_logMod, only : LIS_logunit, LIS_endrun

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: gid
integer, intent(in) :: N_state, N_obs, N_ens
type(obs_type), intent(in), dimension(N_obs) :: Observations
real, intent(in), dimension(N_obs,N_ens) :: Obs_pred
real, intent(in), dimension(N_obs,N_ens) :: Obs_err
real, intent(in), dimension(N_obs,N_obs) :: Obs_cov
real, intent(inout), dimension(N_state,N_ens) :: State_incr

! optional inputs
real, dimension(N_state), intent(in), optional :: State_lon, State_lat
real, intent(in), optional :: xcompact       ! [deg] longitude
real, intent(in), optional :: ycompact       ! [deg] latitude
```

DESCRIPTION:

perform Enks update
IMPORTANT: on input, State_incr must contain State_minus(1:N_state,1:N_ens) on output, State_incr contains the increments
if optional inputs State_lon, State_lat, xcompact, and ycompact are present, Hadamard product is applied to HPHt and PHt

---

57.5.2 hadamard_for_repr_matrix (Source File: enks_general.F90)

INTERFACE:

```
subroutine hadamard_for_repr_matrix( N_obs, Observations,             &
     xcompact, ycompact,                                              &
     Repr_Matrix )
```

USES:

```
use enks_types                    ! for obs_type

implicit none
```

ARGUMENTS:

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

type(obs_type), intent(in), dimension(N_obs) :: Observations

real, intent(in) :: xcompact       ! [deg] longitude
real, intent(in) :: ycompact       ! [deg] latitude

real, dimension(N_obs,N_obs), intent(inout) :: Repr_matrix
```

57.5.3 gaspari_cohn (Source File: enks_general.F90)

INTERFACE:

```
function gaspari_cohn( d )
```

DESCRIPTION:

evaluate 5th-order polynomial from Gaspari and Cohn, 1999, Eq (4.10)
On input, d = separation distance relative to the distance at which all correlations vanish. In the isotropic case, Gaspari and Cohn, 1999, Eq. (4.10)

```
d = sqrt(dx**2 + dy**2) / (2*c) = |z| / (2*c)
```

or in the anisotropic case

```
d = sqrt( (dx/xcompact)**2 + (dy/ycompact)**2 )
```

```
*** Use |z|/c = 2*d. All correlations vanish for d > 1. ***
```

57.5.4 get_gaspari_cohn (Source File: enks_general.F90)

INTERFACE:

```
function get_gaspari_cohn( dx, dy, xcompact, ycompact )
```

DESCRIPTION:

for a given lat/lon distance, compute the anisotropic compact support (Gaspari and Cohn) weights
input coordinates must be in degrees latitude/longitude
dx = longitude separation of two points
dy = latitude separation of two points
xcompact = longitude scale of compact support
ycompact = latitude scale of compact support
for the isotropic Gaspari and Cohn function, the relative distance is in Gaspari and Cohn, 1999, notation (Eq (4.10))
d = sqrt(dx**2 + dy**2) / 2*c
get_gaspari_cohn() uses a generalized anisotropic Gaspari and Cohn approach (essentially coordinate stretching, see handwritten notes for details)
d = sqrt((dx/xcompact)**2 + (dy/ycompact)**2)
All correlations vanish outside of an ellipse with semi-axes xcompact and ycompact, ie Gaspari and Cohn weights vanish for d > 1 (note the factor 2!)
When the anisotropic case is reduced back to the isotropic case, (ie if xcompact==ycompact) then c = xcompact/2 = ycompact/2.

57.5.5 enks_ludcmp (Source File: enks_lu_decomp.F90)

LU decomposition after Numerical Recipes
Given a matrix a[1..n][1..n], this routine replaces it by the LU decomposition of a rowwise permutation of itself. a and n are inputs. a is output, arranged as in equation (2.3.14) (see book). indx[1..n] is an output vector that records the row permutation effected by the partial pivoting; This routine is used in combination with lubksb to solve linear equations or invert a matrix.

REVISION HISTORY:

```
edited 20 Apr 01, reichle
- eliminated d (for computation of determinant)
- eliminated np (can use automatic arrays nowadays...)
edited 03 Jun 02, reichle

- eliminated parameter NMAX, using dynamic allocation
```

INTERFACE:

```
SUBROUTINE enks_ludcmp(a,n,indx)
```

57.5.6 enks_lubksb (Source File: enks_lu_decomp.F90)

LU backsubstitution after Numerical Recipes
Solves the set of n linear equations A X = B. Here a[1..n][1..n] is input, not as the matrix A but rather as its LU decomposition, determined by the routine enks_ludcmp. indx[1..n] is input as the permutation vector returned by enks_ludcmp. b[1..n] is input as the right-hand side vector B, and returns the solution vector X. a,n, and indx ar not modified by this routine and can be left in place for successive calls with different

right-hand sides b. This routine takes into account the possibility that b will begin with many zero elements, so it is efficient for use in matrix inversion.

REVISION HISTORY:

```
edited 20 Apr 01, reichle
```

INTERFACE:

```
SUBROUTINE enks_lubksb(a,n,indx,b)
```

57.6 Fortran: Module Interface enks_matrix_functions (Source File: enks_matrix_functions.F

this file contains a collection of matrix operation subroutines

57.6.1 (Source File: enks_matrix_functions.F90)

reichle, 1 May 01 reichle, 18 Apr 06 - renamed subroutine sort()

57.6.2 row_variance (Source File: enks_matrix_functions.F90)

INTERFACE:

```
subroutine row_variance( M, N, A, var )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: M, N

  real, intent(in), dimension(M,N) :: A

  real, intent(out), dimension(M)  :: var
```

DESCRIPTION:

compute variance of each row of an M-by-N matrix A

57.6.3 row_third_moment (Source File: enks_matrix_functions.F90)

INTERFACE:

```
subroutine row_third_moment( M, N, A, third_moment )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: M, N
  real, intent(in), dimension(M,N) :: A
  real, intent(out), dimension(M)  :: third_moment
```

DESCRIPTION:

compute third moment of each row of an M-by-N matrix A

$$third_moment = 1/N * sum_{i=1}^{N}(x_i - mean(x)) * *3 \tag{2}$$

57.6.4 five_number_summary (Source File: enks_matrix_functions.F90)

INTERFACE:

```
subroutine five_number_summary( M, N, A, five_numbers )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: M, N
  real, intent(in), dimension(M,N)  :: A
  real, intent(out), dimension(M,5)  :: five_numbers
```

DESCRIPTION:

get five number summary (median, lower and upper quartiles, min and max) of each row of an M-by-N data matrix A
inputs: M : number of rows of data = number of different data types N : number of columns of data = number of ensemble members A : M-by-N matrix, left unchanged by this function
outputs: five_numbers : M-by-5 matrix containing statistical summary: column 1: min column 2: lower quartile column 3: median column 4: upper quartile column 5: max
Type: f90 Author: Rolf Reichle Date: 2 May 2001

57.6.5 adjust_mean (Source File: enks_matrix_functions.F90)

INTERFACE:

```
subroutine adjust_mean( N_row, N_col, A, M )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: N_row, N_col
  real, intent(inout), dimension(N_row,N_col)  :: A
  real, intent(in), optional, dimension(N_row) :: M
```

DESCRIPTION:

adjust N_row by N_col matrix A such that mean over columns for each row is given by the corresponding element in vector M of length N_row
vector of mean values M is optional input, if not present zero mean is assumed

---

57.6.6 adjust_std (Source File: enks_matrix_functions.F90)

INTERFACE:

```
subroutine adjust_std( N_row, N_col, A, std )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: N_row, N_col
  real, intent(inout), dimension(N_row,N_col)  :: A
  real, intent(in), optional :: std
```

DESCRIPTION:

adjust N_row by N_col matrix A such that (sample) standard deviation of all elements is exactly equal to std
std is optional input, if not present std=1 is assumed

57.6.7 matrix_std (Source File: enks_matrix_functions.F90)

INTERFACE:

```
subroutine matrix_std( N_row, N_col, A, std )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: N_row, N_col
  real, intent(inout), dimension(N_row,N_col)  :: A
  real, intent(out) :: std
```

DESCRIPTION:

compute std of all elements of N_row by N_col matrix A

57.6.8 nr_sort (Source File: enks_matrix_functions.F90)

QUICKSORT algorithm after Numerical Recipes
Sorts an array arr[1..n] into ascending numerical order using the Quicksort algorithm. n is input; arr is replaced on output by its sorted rearrangement.

INTERFACE:

```
SUBROUTINE nr_sort(n,arr)
```

57.7 Fortran: Module Interface enks_types (Source File: enks_types.F90)

This module contains the definition of types for the GMAO Enks

REVISION HISTORY:

```
reichle, 19 Jul 2005
```

57.8 Fortran: Module Interface enksgrace_Mod (Source File: enksgrace_Mod.F90)

This module contains interfaces and subroutines that control the incorporation of a data set using the ensemble kalman filter (EnKF) method, into a land surface model.
The EnKF algorithm is based on the work of Rolf Reichle at the NASA Global Modeling and Assimilation Office (GMAO) at the NASA GSFC.

REVISION HISTORY:

```
27 Feb 2005: Sujay Kumar; Initial Specification
```

USES:

```
use ESMF
use enks_types

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: enksgrace_init  ! Initialization for EnKF
public :: enksgrace_setup
public :: enksgrace_increments ! compute analysis increments
public :: enksgrace_update ! apply the analysis increments
public :: enksgrace_diagnostics ! write EnKF related diagnostics
public :: enksgrace_final ! Finalization for EnKF
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: enksgrace_struc ! data structure containing EnKF diagnostics
```

57.8.1 enksgrace_init (Source File: enksgrace_Mod.F90)

INTERFACE:

```
subroutine enksgrace_init()
```

USES:

```
use LIS_coreMod,         only : LIS_rc,LIS_config
```

DESCRIPTION:

This method performs the required initializations for the GMAO EnKF method. The method reads the runtime settings from the LIS configuration file.

57.8.2 enksgrace_setup (Source File: enksgrace_Mod.F90)

INTERFACE:

```
subroutine enksgrace_setup(k)
```

USES:

```
    use LIS_coreMod,         only : LIS_rc,LIS_config
    use LIS_DAobservationsMod, only : LIS_OBS_State
    use LIS_logMod,          only : LIS_logunit, LIS_endrun
BZ    use smootherDA_runMod, only : smootherDA_increments_mode
```

DESCRIPTION:

This method performs the required initializations for the GMAO EnKF method. The method reads the runtime settings from the LIS configuration file.

57.8.3 enksgrace_increments (Source File: enksgrace_Mod.F90)

INTERFACE:

```
subroutine enksgrace_increments(n,k)
```

USES:

```
    use LIS_coreMod,         only : LIS_rc, LIS_domain, LIS_surface,&
         LIS_masterproc
    use LIS_lsmMod,          only : LIS_LSM_State, LIS_LSM_Pert_State,&
         LIS_LSM_Incr_State
    use LIS_DAobservationsMod, only : LIS_OBS_State, LIS_OBS_Pert_State
    use LIS_logMod,          only : LIS_logunit, LIS_verify
    use LIS_fileIOMod,       only : LIS_create_output_directory, &
         LIS_create_gain_filename
    use enks_general,        only : enks_analysis
    use enks_matrix_functions, only : row_variance
```

ARGUMENTS:

```
    integer, intent(IN)    :: n
    integer, intent(IN)    :: k
```

DESCRIPTION:

This routine computes the analysis increments for Enks. The state variable increments are computed and stored int the Increment State.
The arguments and variables are:

n index of the nest

data_status flag indicating if the observations are new

N_state Number of state prognostic variables

N_ens Number of ensembles

N_obs_size total number of observations

N_selected_obs number of selected observations

st_id,en_id indices of the selected observations

stvar state prognostic variables

Obs_pert observation perturbations

Obs_cov observations error covariances

Observations observations object used in the GMAO routines

The methods invoked are:

generateObservations 57.8.11
obtain the observations

lsmdagetobspred (3.50.28)
obtain model's estimate of the observations

getObsPert (57.8.10)
obtain the observation perturbations

generateObsparam (57.8.12)
generate the 'obsparam' (metadata for observations)

lsmdagetstatevar 3.50.22
obtain the specified LSM prognostic variables.

assemble_obs_cov (57.8.7) assembles the observation error covariance

getSelectedObsNumber(3.42.14)
obtain the number of selected observations for each modeling point.

enks_analysis (57.5.1)
apply the Enks filter to obtain the prognostic variable increments.

row_variance (57.6.2)
computes the row variance HPH'

lsmdascalestatvar (3.50.38)
scales the state variables for computational stability

lsmdadescalestatevar (3.50.40)
descales the state variables to the original state

57.8.4 enksgrace_update (Source File: enksgrace_Mod.F90)

INTERFACE:

```
subroutine enksgrace_update(n,k)
```

USES:

```
use LIS_coreMod,         only : LIS_rc, LIS_config
use LIS_lsmMod,          only : LIS_LSM_State, LIS_LSM_Incr_State
  use LIS_timeMgrMod, only : LIS_calendar, LIS_clock
use LIS_DAobservationsMod, only : LIS_OBS_State
use LIS_logMod,          only : LIS_logunit, LIS_verify
implicit none

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

DESCRIPTION:

This routine updates the model prognostics using the analysis increments computed earlier.

57.8.5 enksgrace_diagnostics (Source File: enksgrace_Mod.F90)

INTERFACE:

```
subroutine enksgrace_diagnostics(n,k)
```

USES:

```
use LIS_coreMod,     only : LIS_rc, LIS_domain, LIS_masterproc
use LIS_lsmMod,      only : LIS_LSM_State
use LIS_DAobservationsMod, only : LIS_OBS_State
use LIS_fileIOMod, only   :  LIS_create_output_directory, &
     LIS_create_daspread_filename,&
     LIS_create_innov_filename, &
     LIS_create_gain_filename
use LIS_historyMod,  only : LIS_write_dastats, LIS_write_innov, &
     LIS_writevar_restart
use LIS_logMod, only      : LIS_logunit
```

ARGUMENTS:

```
integer, intent(IN)    :: n
integer, intent(IN)    :: k
```

DESCRIPTION:

This subroutine generates the Enks related diagnostics and outputs it to a file. This includes a text output of selected ensemble members, their mean, standard deviation, observations, and normalized innovations. The frequency of diagnostic outputs can be specified in the LIS configuration file.
The routines invoked are:

LIS_create_output_directory (3.14.6)
call to create output directory for DA statistics

getLSMvarnames (57.8.9)
retrieve the names of the LSM prognostic variables

pruneVarname (57.8.6)
trims the variable name, eliminating white spaces

LIS_create_stats_filename (3.14.11)
creates the filename for statistics

LIS_create_innov_filename (3.14.12)
creates the name of the innovations file

lsmdagetstatevar (3.50.22)
retrieve the lsm state variables

getLSMdata (57.8.13)
unpack the LSM state and retrive the data

57.8.6 pruneVarname (Source File: enksgrace_Mod.F90)

INTERFACE:

```
subroutine pruneVarname(varname)
```

ARGUMENTS:

```
character(len=*), intent(INOUT) :: varname
```

DESCRIPTION:

This routine generates a filename based on the names of the state prognostic variables

varname name of the variable filename

57.8.7 assemble_obs_cov (Source File: enksgrace_Mod.F90)

INTERFACE:

```
subroutine assemble_obs_cov(nob, N_obs, obs_param, &
     Observations, Obs_cov)

implicit none
```

ARGUMENTS:

```
integer, intent(in)                              :: nob, N_obs
type(obs_param_type), dimension(nob), intent(in) :: obs_param
type(obs_type), dimension(N_obs), intent(in)     :: Observations
real, intent(out), dimension(N_obs,N_obs)        :: Obs_cov
```

DESCRIPTION:

assemble measurements error covariance (reichle, 27 Jul 2005)

nob number of observation types

N_obs number of observations

obs_param observation settings

Observations Observations data type

Obs_cov observation error covariance

57.8.8 enksgrace_final (Source File: enksgrace_Mod.F90)

INTERFACE:

```
subroutine enksgrace_final
```

DESCRIPTION:

This method performs the finalization for all Enks related structures and subroutines.

57.8.9 getLSMvarnames (Source File: enksgrace_Mod.F90)

INTERFACE:

```
subroutine getLSMvarnames(LIS_LSM_State, dim1, varname)
```

USES:

```
use LIS_logMod,  only : LIS_verify
```

ARGUMENTS:

```
type(ESMF_State)      :: LIS_LSM_State
integer               :: dim1
character(len=*)      :: varname(dim1)
```

DESCRIPTION:

This routine retrieves the names of the state prognostic variables from the LSM state.

57.8.10 getObsPert (Source File: enksgrace_Mod.F90)

INTERFACE:

```
subroutine getObsPert(OBS_Pert_State, dim1, dim2, pert)
```

USES:

```
use LIS_logMod,  only : LIS_verify
```

ARGUMENTS:

```
type(ESMF_State)      :: OBS_Pert_State
integer               :: dim1, dim2
real                  :: pert(dim1,dim2)
```

DESCRIPTION:

This routine retrieves the perturbations to be applied to the observations from the Observation perturbation state
The arguments are:

OBS_Pert_State Observation perturbation state

dim1,dim2 dimensions of the perturbations array

pert perturbation array

57.8.11 generateObservations (Source File: enksgrace_Mod.F90)

INTERFACE:

```
subroutine generateObservations(n, Nobjs, N_obs_size, LIS_OBS_State, &
     LIS_OBS_Pert_State, Observations )
```

USES:

```
use LIS_logMod,  only : LIS_verify
use LIS_coreMod, only        : LIS_domain
```

ARGUMENTS:

```
integer,     intent(IN)        :: n
integer,     intent(IN)        :: Nobjs
integer,     intent(IN)        :: N_obs_size
type(ESMF_State)               :: LIS_OBS_State
type(ESMF_State)               :: LIS_OBS_Pert_State
type(obs_type)                 :: Observations(N_obs_size)
```

DESCRIPTION:

This subroutine unpacks the Observation state and packages them into the 'obs_type' datastructure.

n index of the nest

Nobjs Number of observation types

OBS_State ESMF State containing observations

Observations Observations datastructure being returned.

57.8.12 generateObsparam (Source File: enksgrace_Mod.F90)

INTERFACE:

```
subroutine generateObsparam(Nobjs, OBS_Pert_State, obs_param)
```

USES:

```
use LIS_logMod,  only : LIS_verify
implicit none
```

ARGUMENTS:

```
integer                       :: Nobjs
type(ESMF_State)              :: OBS_Pert_State
type(obs_param_type)          :: obs_param(Nobjs)
```

DESCRIPTION:

This routine obtains the metadata information associated with the observations and puts them into the obs_param datastructure
The arguments are:

Nobjs Number of observation types

OBS_Pert_State ESMF state containing observation perturbations

obs_param obs_param datastructure being returned

57.8.13 getLSMData (Source File: enksgrace_Mod.F90)

INTERFACE:

```
subroutine getLSMData(LIS_LSM_State, dim1, dim2, lsmdata)
```

USES:

```
use LIS_logMod,  only : LIS_verify
```

ARGUMENTS:

```
type(ESMF_State)      :: LIS_LSM_State
integer               :: dim1, dim2
real                  :: lsmdata(dim1, dim2)
```

DESCRIPTION:

This routine retrives the LSM state prognostic variables from the LSM state.
The arguments are:

LSM_State LSM State containing prognostic variables

dim1,dim2 dimensions of the prognostic variable array

lsmdata prognostic variable array

58 Perturbation Algorithms in LIS

This section contains the description of the implementations of perturbation algorithms in LIS. The algorithms are used to perturb forcing, observation and prognostic varibles for data assimilation.

---

59 GMAO Perturbation Algorithm

59.1 Fortran: Module Interface gmaopert_Mod (Source File: gmaopert_Mod.F90)

This module contains routines to peturb either the forcing, prognostic variables, or observations based on the algorithmic implementations of Rolf Reichle (NASA GMAO).

REVISION HISTORY:

```
08Jul05    Sujay Kumar; Initial Specification
24Nov10    Sujay Kumar; Added support for time varying
                        perturbations
```

59.1.1 gmaopert_init (Source File: gmaopert_Mod.F90)

Initializes memory structures for specified perturbation options.

INTERFACE:

```
subroutine gmaoPert_init(index)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

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

59.1.2 gmaopert_setup (Source File: gmaopert_Mod.F90)

INTERFACE:

```
subroutine gmaoPert_setup(index, k, Base_State, Pert_State)
```

USES:

```
use landpert_routines, only : assemble_pert_param,&
     get_init_pert_rseed, get_pert
use LIS_coreMod, only : LIS_rc, LIS_domain,LIS_surface
use LIS_logMod,  only  : LIS_verify
use nr_ran2_gasdev, only : NRANDSEED
```

DESCRIPTION:

Initializes memory structures for specified perturbation options.

59.1.3 gmaoperturb (Source File: gmaopert_Mod.F90)

INTERFACE:

```
subroutine gmaoperturb(id, n, k, Base_State, Pert_State)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain,LIS_surface
use LIS_logMod,  only  : LIS_verify
use landpert_routines, only : get_pert

implicit none
```

ARGUMENTS:

```
integer, intent(in)       :: id
integer, intent(in)       :: n
integer, intent(in)       :: k
type(ESMF_State)          :: Base_State
type(ESMF_State)          :: Pert_State
```

DESCRIPTION:

This routine computes the perturbations (as increments) and packages them into an ESMF state.
The pert_time_type specifies whether the perturbation standard deviation is constant or time varying. A value of 0 indicates constant standard deviation. For pert_time_type = 1, a linear model of perturbation standard deviation (as a linear function of perturbations state in the form y=mx+b, where x is the perturbations state, m is the coefficient of standard deviation and b is the base standard deviation. m and b are specified through the perturbations attributes file.

59.1.4 gmaopert_writerestart (Source File: gmaopert_Mod.F90)

INTERFACE:

```
subroutine gmaopert_writerestart(n)
```

USES:

```
use LIS_coreMod,     only : LIS_rc, LIS_domain, LIS_surface,LIS_masterproc
use LIS_historyMod,  only : LIS_writevar_restart
use LIS_fileIOMod
use LIS_logMod
```

ARGUMENTS:

```
implicit none

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

DESCRIPTION:

Write one single file that includes the restart info for forcing, states and observations

---

59.1.5 gmaopert_readrestart (Source File: gmaopert_Mod.F90)

INTERFACE:

```
subroutine gmaopert_readrestart()
```

USES:

```
use LIS_coreMod,      only : LIS_rc, LIS_domain,LIS_surface
use LIS_logMod,       only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_historyMod,  only : LIS_readvar_restart

implicit none
```

ARGUMENTS:

DESCRIPTION:

This routine reads from a perturbations restart file

59.2 Fortran: Module Interface landpert_module.F90 (Source File: landpert_module.F90)

This module contains routines that can be used to perturb forcing fields such as precipitation and radiation or model prognostic variables such as soil moisture or soil temperature
MUST initialize random seed and Pert_ntrmdt by calling get_pert() with initialize=.true. at the start of the driver program (otherwise set initialize=.false.)

REVISION HISTORY:

```
24Jan05 Rolf Reichle  Initial Specification
07Jun05 Rolf Reichle  more init options
07Jul05 Sujay Kumar   Initial Specification
```

59.2.1 allocate_pert_param (Source File: landpert_module.F90)

INTERFACE:

```
subroutine allocate_pert_param(N_pert, N_x, N_y, pert_param)

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: N_pert, N_x, N_y
  type(pert_param_type), dimension(:), pointer :: pert_param
```

DESCRIPTION:

allocates memory for the perturbation data structures.

59.3 Fortran: Module Interface landpert_routines.F90 (Source File: landpert_routines.F90)

This module contains routines that can be used to perturb forcing fields such as precipitation and radiation or model prognostic variables such as soil moisture or soil temperature
MUST initialize random seed and Pert_ntrmdt by calling get_pert() with initialize=.true. at the start of the driver program (otherwise set initialize=.false.)

REVISION HISTORY:

```
24Jan05 Rolf Reichle  Initial Specification
07Jun05 Rolf Reichle  more init options
07Jul05 Sujay Kumar   Specification in LIS
```

59.3.1 assemble_pert_param (Source File: landpert_routines.F90)

INTERFACE:

```
subroutine assemble_pert_param( Pert_State, N_x, N_y,              &
     N_pert, pert_param )
```

DESCRIPTION:

sample subroutine that demonstrates how pert_param can be assembled for forcing perturbations
return N_force_pert, allocate and assemble structure pert_param
make sure order of fields is compatible with your driver
forcing field 1 = precip forcing field 2 = shortwave forcing field 3 = longwave forcing field 4 = air temperature

REVISION HISTORY:

```
reichle, 11 Feb 2005
reichle,  8 Jun 2005
```

USES:

```
use LIS_coreMod
use LIS_logMod
use ESMF

implicit none
```

ARGUMENTS:

```
type(ESMF_State), intent(in) :: Pert_State
integer, intent(in) :: N_x, N_y
integer, intent(in) :: N_pert
type(pert_param_type), dimension(:), pointer :: pert_param ! out
```

59.3.2 get_sqrt_corr_matrix (Source File: landpert_routines.F90)

INTERFACE:

```
subroutine get_sqrt_corr_matrix( N, A, S )
```

USES:

```
use nr_jacobi

implicit none
```

ARGUMENTS:

```
integer, intent(in)                  :: N
real,    intent(in),  dimension(N,N) :: A
real,    intent(out), dimension(N,N) :: S
```

DESCRIPTION:

get sqrt S of real, symmetric (correlation) matrix A
NOTE: there is no check that A is indeed symmetric!
A = S*transpose(S)
reichle, 7 Jun 2005

59.3.3 get_init_Pert_rseed (Source File: landpert_routines.F90)

INTERFACE:

```
subroutine get_init_Pert_rseed( N_ens, N_domain, ens_id, domain_id, &
     init_Pert_rseed )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: N_ens, N_domain
  integer, dimension(N_ens),    intent(in) :: ens_id
  integer, dimension(N_domain), intent(in) :: domain_id
  integer, dimension(N_ens,N_domain), intent(out) :: init_Pert_rseed
```

DESCRIPTION:

get initial random seed "init_Pert_rseed" for initializing Pert_rseed within get_pert()
A different random seed is necessary for each ensemble member. In parallel applications, the random seed must also be different for each subdomain.
This subroutine is meant as a sample for how the initial Pert_rseed can be set. In this example, ens_id and domain_id are meant to be nonnegative small integers.

59.3.4 get_pert (Source File: landpert_routines.F90)

INTERFACE:

```
subroutine get_pert(                                &
     N_pert, N_ens, N_x, N_y,                       &
     dx, dy, dtstep,                                &
     std,                                           &
     pert_param,                                    &
     Pert_rseed,                                    &
     Pert_ntrmdt,                                   &
     Pert,                                          &
     initialize_rseed,                              &
     initialize_ntrmdt               )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: N_pert   ! # different perturbations

  integer, intent(in) :: N_ens  ! # ensemble members

  integer, intent(in) :: N_x    ! # grid cells in longitudinal direction
  integer, intent(in) :: N_y    ! # grid cells in latitudinal direction

  real, intent(in) :: dx   ! grid cell size in longitudinal direction [deg]
  real, intent(in) :: dy   ! grid cell size in latitudinal direction [deg]

  real, intent(in) :: dtstep        ! perturbation time step in seconds
```

```
real, intent(in) :: std(N_pert, N_x, N_y)         ! temporally varying standard deviation
! Parameter structure for perturbations (see type definition for details).

type(pert_param_type), dimension(:), pointer :: pert_param

! Pert_ntrmdt are intermediate perturbation fields
! that need to be remembered between calls to this subroutine.
! In essence, they store N_pert mutually uncorrelated
! perturbation fields of standard-normal distribution.
! Pert_rseed is the random seed for the generation of
! Pert_ntrmdt and is treated similarly to a prognostic variable.
! Each ensemble member has its own random seed.

integer, dimension(NRANDSEED,N_ens), intent(inout) :: Pert_rseed

real, dimension(N_pert,N_x,N_y,N_ens), intent(inout) :: Pert_ntrmdt

! Pert are N_pert cross-correlated perturbation
! fields that are rotated and scaled versions of Pert_ntrmdt
! so that Pert has the the mean values, standard deviations and
! cross-correlations specified in pert_param.
! The distribution is lognormal for multiplicative perturbations.
! Pert should be used as follows for field F (eg. large-scale
! precip, convective precip, lw radiation, ...)
! Fprime = F+Pert   for additive perturbations
! Fprime = F*Pert   for multiplicative perturbations
! Note that this subroutine does NOT ensure physically meaningful
! perturbed fields.  This is best done outside this subroutine
! after the perturbations have been applied.

real, dimension(N_pert,N_x,N_y,N_ens), intent(out) :: Pert

! If initialize_rseed==.true., set initial random seed vector.
! If initialize_rseed==.true., the first row of Pert_rseed must be
! filled with a different negative integer for each ensemble member.
! Note that in parallel applications initial Pert_rseed must also differ
! for each subdomain.  See sample subroutine get_init_Pert_rseed().
! If initialize_ntrmdt==.true., generate initial Pert_ntrmdt (must be
! allocated!!!).
! Note that when get_pert() is used for generating independent
! perturbations to forcings and prognostic variables, only one
! common Pert_rseed should be used, so one of the "ntrmdt" fields
! must be initialized without initializing "rseed" again.

! If initialize_*==.false. or absent, Pert_rseed and
! Pert_ntrmdt must be what was obtained as output from the last call
! to get_pert().  There is only one exception: if there are no temporal
! correlations, it is not necessary to remember Pert_ntrmdt.

logical, intent(in), optional :: initialize_rseed
logical, intent(in), optional :: initialize_ntrmdt
```

DESCRIPTION:

get perturbations
reichle, 11 Feb 2005
N_pert is the number of *perturbation* fields and is not necessarily equal to the number of forcing fields. E.g. generate N_pert=3 perturbations for precip, shortwave radiation, and longwave radiation. These can then be applied to N_force forcing fields, possibly various precip fields (incl large-scale & convective precip and snow) and to radiation fields.

59.3.5 propagate_pert (Source File: landpert_routines.F90)

INTERFACE:

```
subroutine propagate_pert(                      &
     N_pert, N_ens, loc_grid, dtstep,           &
     Pert_rseed,                                &
     pert_param,                                &
     Pert_ntrmdt,                               &
     initialize              )
```

USES:

```
use LIS_logMod, only : LIS_logunit, LIS_endrun

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: N_ens, N_pert

type(grid_def_type), intent(in) :: loc_grid   ! local grid definition

type(pert_param_type), dimension(N_pert), intent(in) :: pert_param

real, intent(in) :: dtstep  ! time step of generation of error fields [s]

integer, dimension(NRANDSEED,N_ens), intent(inout) :: Pert_rseed

real, dimension(N_pert,loc_grid%N_x,loc_grid%N_y,N_ens), &
     intent(inout) :: Pert_ntrmdt

logical, intent(in) :: initialize   ! switch
```

DESCRIPTION:

generate zero-mean, unit-variance (!!) time series of N_pert 2d perturbation fields
can also be used just to get a set of 2d random fields (set dtstep to arbitrary number and "initialize" to .true.)
on input, Pert_ntrmdt must contain the corresponding perturbations from the previous time step
accounts for temporal correlation with AR(1) approach (if pert_param%tcorr==0 then error is white in time)
adapted from off-line EnKF, subroutine propagate_err() from NCAT_59124_tskin in enkf_catchment.f90
reichle, 14 Feb 2005

59.3.6 truncate_std_normal (Source File: landpert_routines.F90)

INTERFACE:

```
subroutine truncate_std_normal( N_x, N_y, std_normal_max, grid_data )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: N_x, N_y

  real, intent(in) :: std_normal_max

  real, dimension(N_x,N_y), intent(inout) :: grid_data
```

DESCRIPTION:

truncate a realization of standard normal variables (scale back freak outliers)

59.3.7 adjust_mean (Source File: landpert_routines.F90)

INTERFACE:

```
subroutine adjust_mean( N_row, N_col, A, M )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: N_row, N_col
  real, intent(inout), dimension(N_row,N_col)  :: A
  real, intent(in), optional, dimension(N_row) :: M
```

DESCRIPTION:

adjust N_row by N_col matrix A such that mean over columns for each row is given by the corresponding element in vector M of length N_row
vector of mean values M is optional input, if not present zero mean is assumed

---

59.4 Fortran: Module Interface nr_jacobi.F90 (Source File: nr_jacobi.F90)

This module performs the Jacobi transformation. Adopted from f77 Numerical Recipes.

REVISION HISTORY:

```
07Jun05 Rolf Reichle  Initial Specification
07Jul05 Sujay Kumar   Specification in LIS
```

59.4.1 jacobi (Source File: nr_jacobi.F90)

INTERFACE:

```
subroutine jacobi(amat,n,d,v,nrot)
```

ARGUMENTS:

```
integer,                  intent(in)               :: n
real,    dimension(n,n),  intent(in)               :: amat
real,    dimension(n,n),  intent(out)              :: v
real,    dimension(n),    intent(out)              :: d
integer,                  intent(out), optional    :: nrot
```

DESCRIPTION:

Computes all eigenvalues and eigenvectors of a real symmetric matrix amat, which is of size n by n. The vector d returns the eigenvalues of amat in its first n elements. v is a matrix with the same dimensions as amat, whose columns contain, on output, the normalized eigenvectors of amat. nrot returns the number of Jacobi rotations that were required.
The Jacobi method is absolutely foolproof for all real symmetric matrices. For matrices of order greater than about 10, say, the algorithm is slower, by a significant constant factor, than the QR method. However, the Jacobi algorithm is much simpler than the more efficient methods. We thus recommend it for matrices of moderate order, where expense is not a major consideration.
Eigenvector decomposition:
Amat = V*diag(D)*transpose(V)

59.5 Fortran: Module Interface nr_ran2_gasdev.F90 (Source File: nr_ran2_gasdev.F90)

adapted Numerical Recipes random number generator ran2() and gasdev()
use ran2() instead of rand() and/or ran1() for longer period (1e18)
gasdev() uses ran2() instead of ran1()
eliminate all save attributes, convert functions into subroutines, pass seed state vector into subroutines, enclose in module

REVISION HISTORY:

```
16Feb05 Rolf Reichle  Initial Specification
18Feb05 Rolf Reichle  make nr_ran2() public, change call statement
07Jul05 Sujay Kumar   Specification in LIS
```

59.6 Fortran: Module Interface random_fields.F90 (Source File: random_fields.F90)

This module contains a random field generator in 2d to generate a pair of random fields in 2d with zero mean
subroutines rfg2d_fft() and sqrt_gauss_spectrum are translated from C++ code rfg2d.C written for MIT EnKF work by reichle (see janus: reichle/nasa/EnKF)
covariance is specified through its spectrum, so far only Gaussian
IMPORTANT: read comments for function rfg2d_fft()
written for NSIPP - EnKF Type: f90 Author: Rolf Reichle Date: 2 Nov 2001

REVISION HISTORY:

```
18Feb05 Rolf Reichle  Initial Specification
                      updated for use with module nr\_ran2\_gasdev
                      deleted use of module select\_kinds
07Jul05 Sujay Kumar   Specification in LIS
```

59.6.1 sqrt_gauss_spectrum_2d (Source File: random_fields.F90)

INTERFACE:

```
subroutine sqrt_gauss_spectrum_2d( &
     variance, N_x, N_y, dkx, dky, lambda_x, lambda_y, &
     sqrt_spectrum_2d )

  implicit none
```

ARGUMENTS:

```
  real, intent(in) :: variance, dkx, dky, lambda_x, lambda_y
  integer, intent(in) :: N_x, N_y
  real, intent(out), dimension(N_x,N_y) :: sqrt_spectrum_2d
```

DESCRIPTION:

get SQUARE ROOT of 2d Gaussian spectrum (incl volume element)
2d Gaussian spectrum:

$$S(kx, ky) = variance * lambda_x * lambda_y/(2 * pi) * exp(-(kx^2 * lambda_x^2 + ky^2 * lambda_y^2)/2)$$

return: sqrt(S*dkx*dky)
that is return the SQUARE ROOT of the spectrum multiplied with the square root of the volume element d2k=dkx*dky of the ifft integral
spectrum is returned in "wrap-around" order compatible with CXML and matlab FFT algorithms
inputs: variance : variance desired for complex field, if pair of real fields is used each field must eventually be multiplied with sqrt(2) N_x : number of nodes in x direction N_y : number of nodes in y direction dkx : wave number spacing in x direction dky : wave number spacing in y direction lambda_x : decorrelation length in x direction lambda_y : decorrelation length in y direction

60 Bias Estimation Algorithms in LIS

This section contains the description of the implementations of bias estimation algorithms in LIS.

60.1 Fortran: Module Interface gmaobias_estimationMod (Source File: gmaobias_estimationMod.F90)

This module contains interfaces and subroutines that control the computation of dynamic bias estimation routines using the approach of Bosilovich et al. (2007)
The implementation of the algorithm is based on the work of Rolf Reichle at the NASA Global Modeling and Assimilation Office (GMAO) at the NASA GSFC.

REVISION HISTORY:

```
26 Nov 2007, Sujay Kumar  : Initial Specification in LIS
```

60.1.1 gmaobiasestimation_calc (Source File: gmaobias_estimationMod.F90)

INTERFACE:

```
subroutine gmaobiasestimation_calc(n,k)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_lsmMod,  only : LIS_LSM_State
use LIS_logMod,  only : LIS_verify, LIS_logunit

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

60.1.2 gmaobiasestimation_update (Source File: gmaobias_estimationMod.F90)

INTERFACE:

```
subroutine gmaobiasestimation_update(n,k)
```

USES:

```
use ESMF
use LIS_coreMod,     only : LIS_rc
use LIS_lsmMod,      only : LIS_LSM_Incr_State
use LIS_logMod,      only : LIS_verify

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

60.1.3 gmaobiasestimation_restart (Source File: gmaobias_estimationMod.F90)

INTERFACE:

```
subroutine gmaobiasestimation_write_restart(n,k)
```

USES:

```
use LIS_historyMod, only    : LIS_writevar_reduced_tilespace
use LIS_coreMod,    only    : LIS_rc, LIS_masterproc
use LIS_logMod,     only    : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,  only    : LIS_create_output_directory
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine generates the bias restart file based on the specified output frequency.

60.1.4 gmaobias_restart_filename (Source File: gmaobias_estimationMod.F90)

INTERFACE:

```
subroutine gmaobias_restart_filename(filename)
```

USES:

```
use LIS_coreMod,   only : LIS_rc
```

ARGUMENTS:

```
character(len=*)       :: filename
```

DESCRIPTION:

This subroutine generates a time stamped name of the bias restart file

---

60.1.5 bias_update (Source File: gmaobias_estimationMod.F90)

REVISION HISTORY:

```
19 Oct 2005, Rolf Reichle : Initial Specification
26 Nov 2007, Sujay Kumar  : Included in LIS
19 Aug 2008, Rolf Reichle : Added call to get_bias_const_param()
```

INTERFACE:

```
subroutine bias_update(n, k, N_state, N_size, progn_incr)
```

USES:

```
use LIS_coreMod,         only      : LIS_rc

implicit none
```

ARGUMENTS:

```
integer,           intent(in)    :: n
integer,           intent(in)    :: k
integer,           intent(in)    :: N_state
integer,           intent(in)    :: N_size
real,              intent(in)    :: progn_incr(N_state, N_size)
```

DESCRIPTION:

60.1.6 bias_update_helper (Source File: gmaobias_estimationMod.F90)

REVISION HISTORY:

```
17 Oct 2005 Rolf Reichle:  Initial Specification
26 Nov 2006 Sujay Kumar :  Included in LIS
```

INTERFACE:

```
subroutine bias_update_helper( dtstep, Nparam, const_param, tconst, &
    increment, bias_param )

implicit none
```

ARGUMENTS:

```
real, intent(in) :: dtstep, tconst, increment
integer, intent(in) :: Nparam
real, dimension(Nparam), intent(in) :: const_param
real, dimension(Nparam), intent(inout) :: bias_param
```

DESCRIPTION:

update bias parameters for a single tile and a single field of cat_progn

---

60.1.7 get_ens_avg (Source File: gmaobias_estimationMod.F90)

INTERFACE:

```
subroutine get_ens_avg(N_tile, N_size, N_state, N_ens, &
    State_incr, State_incr_vec)
```

DESCRIPTION:

60.1.8 bias_corr (Source File: gmaobias_estimationMod.F90)

REVISION HISTORY:

```
19 Oct 2005, Rolf Reichle : Initial Specification
26 Nov 2007, Sujay Kumar  : Included in LIS
19 Aug 2008, Rolf Reichle : Added call to get_bias_const_param()
```

INTERFACE:

```
subroutine bias_corr(n, k, N_tile, N_state, N_size, N_ens, progn_field)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

DESCRIPTION:

apply bias correction to prognostic variables, relaxes bias parameters

---

60.1.9 bias_corr_helper (Source File: gmaobias_estimationMod.F90)

REVISION HISTORY:

```
17 Oct 2005 Rolf Reichle:  Initial Specification
26 Nov 2006 Sujay Kumar :  Included in LIS
```

INTERFACE:

```
subroutine bias_corr_helper( dtstep, N_ens, Nparam,   &
     const_param, trelax, progn_field, bias_param )

 implicit none
```

DESCRIPTION:

diagnose/apply bias flux and relax bias parameters for a single tile and a single field of progn

---

60.1.10 bias_get_const_param (Source File: gmaobias_estimationMod.F90)

REVISION HISTORY:

```
18 Aug 2008 Rolf Reichle:  Initial Specification
19 Aug 2008 Rolf Reichle:  Included in LIS
```

INTERFACE:

```
subroutine bias_get_const_param(const_param)
```

USES:

```
use LIS_coreMod,         only      : LIS_rc
use LIS_constantsMod,    only      : LIS_CONST_PI

implicit none
```

ARGUMENTS:

```
real,           intent(out)   :: const_param(N_bias_param_max)
```

DESCRIPTION:

60.1.11 bias_time_of_day_index (Source File: gmaobias_estimationMod.F90)

REVISION HISTORY:

```
18 Aug 2008 Rolf Reichle:  Initial Specification
19 Aug 2008 Rolf Reichle:  Included in LIS
```

INTERFACE:

```
integer function bias_time_of_day_index( Nparam )
```

USES:

```
use LIS_coreMod,         only      : LIS_rc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

60.1.12 bias_options_helper (Source File: gmaobias_estimationMod.F90)

REVISION HISTORY:

```
18 Aug 2008 Rolf Reichle:  Initial Specification
19 Aug 2008 Rolf Reichle:  Included in LIS
```

INTERFACE:

```
subroutine bias_options_helper( Nparam, const_param_in, &
     ind_start, ind_end, const_param_out )
```

USES:

```
implicit none
```

ARGUMENTS:

```
integer,                    intent(in)  :: Nparam
real,    dimension(Nparam), intent(in)  :: const_param_in

integer,                    intent(out) :: ind_start, ind_end
real,    dimension(Nparam), intent(out) :: const_param_out
```

DESCRIPTION:

Part XII
Observation data handling for assimilation in LIS

61 Observation data handling

This section contains the description of observational data handling for assimilation in LIS.

---

62 ANSA Snow Cover Fraction

62.1 Fortran: Module Interface ANSASCFsnow_Mod (Source File: ANSASCFsnow_Mod.F90)

This module contains interfaces and subroutines to handle ANSA (AFWA NASA snow algorithm) SCF retrievals. The SCF data is primarily based on AMSR-E, with observation gaps filled in using previous overpasses. For more details, please see the ANSA reference:
Foster et al. "A blended global snow product using visible, passive microwave and scatterometer satellite data", International Journal of Remote Sensing, 2010.

REVISION HISTORY:

```
1 Jun 09   Sujay Kumar;   Initial Specification
August 2013/May 2014 Yuqiong Liu; adapted for ANSA SCF assimilation using either EnKF or DI
```

```
module ANSASCFsnow_Mod
```

USES:

```
use ESMF
```

62.1.1 ANSASCFsnow_setup (Source File: ANSASCFsnow_Mod.F90)

INTERFACE:

```
subroutine ANSASCFsnow_setup(k, OBS_State, OBS_Pert_State)
```

USES:

```
use LIS_coreMod
use LIS_timeMgrMod
use LIS_historyMod
use LIS_perturbMod
use LIS_logMod

implicit none
```

ARGUMENTS:

```
integer                 ::  k
type(ESMF_State)        ::  OBS_State(LIS_rc%nnest)
type(ESMF_State)        ::  OBS_Pert_State(LIS_rc%nnest)
```

DESCRIPTION:

This routine completes the runtime initializations and creation of data strctures required for ANSA SCF data.
The arguments are:

OBS_State observation state

OBS_Pert_State observation perturbations state

62.1.2 getNSO_ANSASCFsnow (Source File: getNSO_ANSASCFsnow.F90)

Returns the number of selected observations (NSO) corresponding to the grid point specified

REVISION HISTORY:

```
1 Jun 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine getNSO_ANSASCFsnow(n, gid,st_id,en_id)

  implicit none
```

DESCRIPTION:

Returns the number of selected Tskin observations (NSO) corresponding to the selected modeling point
The arguments are:

n index of the nest

gid index of the modeling point

st_id starting index of the selected observations

en_id ending index of the selected observations

62.1.3 read_ANSASCFsnow (Source File: read_ANSASCFsnow.F90)

REVISION HISTORY:

```
1 Jun 2009: Sujay Kumar; Initial Specification
Aug 2013 Yuqiong Liu; adpated for ANSA SCF assimilation using EnKF or DI
```

INTERFACE:

```
subroutine read_ANSASCFsnow(n, OBS_State,OBS_Pert_State)
```

USES:

```
#if(defined USE_HDF5)
  use hdf5
#endif

  use ESMF
```

```
use LIS_mpiMod
use LIS_coreMod
use LIS_timeMgrMod
use LIS_logMod
 use LIS_pluginIndices, only : LIS_ANSASCFsnowobsId
use ANSASCFsnow_Mod, only : ANSASCFsnow_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: OBS_State
type(ESMF_State)    :: OBS_Pert_State
```

DESCRIPTION:

reads ANSA snow observations and packages it into an ESMF State with certain predefined attributes. The routine reads the snow data at 0z, performs spatial interpolation to the LIS grid and keeps it in memory. At 14:00 localtime for each grid point, the code then packages the interpolated observations into the ESMF state. Note that the data files are in HDF5 format and the routine employs the HDF5-subsetting tools to extract and subset data.
The arguments are:

n index of the nest

OBS_State observations state

62.1.4 ANSAscf_filename (Source File: read_ANSASCFsnow.F90)

INTERFACE:

```
subroutine ANSAscf_filename(name, scf_fn_conv, ndir, yr, mo,da)

   use LIS_coreMod, only: LIS_rc
   use LIS_logMod, only: LIS_logunit

   implicit none
```

ARGUMENTS:

```
character (len=*) :: name
integer           :: yr, mo, da
character (len=*) :: ndir
character (len=*) :: scf_fn_conv
integer           :: len1, i1, i2
character (len=50) :: str1, str2, str0
```

DESCRIPTION:

This subroutine creates a timestamped ANSA filename
The arguments are:

name name of the ANSA SCF filename

scf_fn_conv ANSA SCF data file name convention

ndir name of the ANSA SCF data directory

yr current year

mo current month

da current day

62.1.5 write_ANSASCFsnowobs (Source File: write_ANSASCFsnow.F90)

REVISION HISTORY:

```
1 Jun 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine write_ANSASCFsnowobs(n, OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_logMod,     only : LIS_verify, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,  only : LIS_create_output_directory
use LIS_historyMod, only : LIS_writevar_gridded

implicit none
```

ARGUMENTS:

```
integer,     intent(in)  :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

writes the transformed (interpolated/upscaled/reprojected) ANSA observations to a file

---

62.1.6 ANSASCFsnow_obsname (Source File: write_ANSASCFsnow.F90)

INTERFACE:

```
subroutine ANSASCFsnow_obsname(obsname)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

```
character(len=*)      :: obsname
```

DESCRIPTION:

62.2 Fortran: Module Interface ECV_sm_Mod (Source File: ECV_sm_Mod.F90)

This module handles the observation plugin for the soil moisture Essential Climate Variable (ECV) product. Soil moisture is recognized as an Essential Climate Variable (ECV) by NASA, European Space Agency (ESA), and other agencies and institutions.
The Water Cycle Multi-mission Observation Strategy (WACMOS) and Climate Change Initiative (CCI) Soil Moisture projects have released a 32-year harmonized product, from 1978-2010, and referred to as the ECV soil moisture dataset. The product includes both active and passive microwave sensor retrievals. The active dataset provided by the University of Vienna (TU Wien) uses ERS-1, ERS-2 and METOP-A C-band scatterometers. The passive microwave dataset is developed by VU University of Amsterdam and NASA, and includes NASA-based measurements from Nimbus 7 SSMR, DMSP SSM/I, TRMM TMI and Aqua AMSR-E sensors.
The merged dataset uses a fixed ranking system based on error derivations from triple collocation. So if AMSR has an higher error than ASCAT, then ASCAT is used (and visa versa, see for example Liu et al.,RSE 2012 and Liu et al., HESS 2011). In the near future, a dynamic weighting function is planned to be used within the merging routine (considered research in progress).
Further information about the product can be obtained at: http://www.esa-soilmoisture-cci.org/
REFERENCES: Liu, Y. Y., W. A. Dorigo, et al. (2012). "Trend-preserving blending of passive and active microwave soil moisture retrievals." Remote Sensing of Environment 123: 280-297.
Wagner, W., Dorigo, W., de Jeu, R., Fernandez, D., Benveniste, J., Haas, E., and Ertl, M.: Fusion of active and passive microwave observations to create an Essential Climate Variable data record on soil moisture. ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., I-7, 315-321, doi:10.5194/isprsannals-I-7-315-2012, 2012.

REVISION HISTORY:

```
    01 Oct 2012: Sujay Kumar, Initial Specification

 module ECV_sm_Mod
```

USES:

```
   use ESMF
   use map_utils

   implicit none

   PRIVATE

  ------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
  ------------------------------------------------------------------------------
   public :: ECV_sm_setup
  ------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
  ------------------------------------------------------------------------------
   public :: ECV_sm_struc
```

62.2.1 ECV_sm_setup (Source File: ECV_sm_Mod.F90)

INTERFACE:

```
subroutine ECV_sm_setup(k, OBS_State, OBS_Pert_State)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_timeMgrMod
use LIS_historyMod
use LIS_dataAssimMod
use LIS_perturbMod
use LIS_logmod

implicit none
```

ARGUMENTS:

```
integer                 ::  k
type(ESMF_State)        ::  OBS_State(LIS_rc%nnest)
type(ESMF_State)        ::  OBS_Pert_State(LIS_rc%nnest)
```

DESCRIPTION:

This routine completes the runtime initializations and creation of data strctures required for handling ECV soil moisture data
The arguments are:

OBS_State observation state

OBS_Pert_State observation perturbations state

62.2.2 getNSO_ECVsm.F90 (Source File: getNSO_ECVsm.F90)

Returns the number of selected observations (NSO) corresponding to the grid point specified

REVISION HISTORY:

```
21Jun2006: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine getNSO_ECVsm(n, gid,st_id,en_id)

implicit none
```

DESCRIPTION:

Returns the number of selected soil moisture observations (NSO) corresponding to the selected modeling point
The arguments are:

n index of the nest

gid index of the modeling point

st_id starting index of the selected observations

en_id ending index of the selected observations

62.2.3 read_ECVsm (Source File: read_ECVsm.F90)

REVISION HISTORY:

```
01 Oct 2012: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine read_ECVsm(n, OBS_State, OBS_Pert_State)
```

USES:

```
use ESMF
use LIS_mpiMod
use LIS_coreMod
use LIS_logMod
use LIS_timeMgrMod
use LIS_dataAssimMod
use map_utils
use LIS_pluginIndices
use ECV_sm_Mod, only : ECV_sm_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: OBS_State
type(ESMF_State)    :: OBS_Pert_State
```

DESCRIPTION:

This subroutine provides the data reader for the ECV soil moisture retrieval product.
The arguments are:

n index of the nest

OBS_State observations state

62.2.4 read_ECV_data (Source File: read_ECVsm.F90)

INTERFACE:

```
subroutine read_ECV_data(n, fname, smobs_ip)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
  use LIS_coreMod,  only : LIS_rc
  use LIS_logMod,   only : LIS_verify
  use map_utils,    only : latlon_to_ij
  use ECV_sm_Mod, only : ECV_sm_struc

  implicit none
```

INPUT PARAMETERS:

```
integer                        :: n
character (len=*)              :: fname
real                           :: smobs_ip(LIS_rc%lnc(n)*LIS_rc%lnr(n))
```

OUTPUT PARAMETERS:

DESCRIPTION:

This subroutine reads the ECV NETCDF file and applies the data quality flags to filter the data.The data quality flags provided with the data is applied by excluding data masked for dense vegetation, temperature below zero and lack of convergence of the algorithm.
The arguments are:

n index of the nest

fname name of the ECV AMSR-E file

smobs_ip soil moisture data processed to the LIS domain

FILES USED:

REVISION HISTORY:

62.2.5 create_ECVsm_filename (Source File: read_ECVsm.F90)

INTERFACE:

```
subroutine create_ECVsm_filename(ndir, yr, mo,da, filename)
```

USES:

```
implicit none
```

ARGUMENTS:

```
character(len=*)  :: filename
integer           :: yr, mo, da
character (len=*) :: ndir
```

DESCRIPTION:

This subroutine creates the ECV filename based on the time and date
The arguments are:

ndir name of the ECV soil moisture directory

yr current year

mo current month

da current day

filename Generated ECV filename

62.2.6 write_ECVsmobs (Source File: write_ECVsmobs.F90)

REVISION HISTORY:

```
25Jan2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine write_ECVsmobs(n, OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_logMod,     only : LIS_verify, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,  only : LIS_create_output_directory
use LIS_historyMod, only : LIS_writevar_gridded

implicit none
```

ARGUMENTS:

```
integer,     intent(in)  :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

writes the transformed (interpolated/upscaled/reprojected) ECV observations to a file

---

62.2.7 ECV_smobsname (Source File: write_ECVsmobs.F90)

INTERFACE:

```
subroutine ECV_smobsname(obsname)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

```
character(len=*)      :: obsname
```

DESCRIPTION:

62.3 Fortran: Module Interface GRACEobs_module (Source File: GRACEobs_module.F90)

This module contains interfaces and subroutines to handle synthetic soil moisture observations (such as the one generated from a previous LIS-Noah simulation)

REVISION HISTORY:

```
27Feb05    Sujay Kumar;   Initial Specification
11Aug11    Ben Zaitchik;  Modified for GRACE
```

```
module GRACEobs_module
```

USES:

```
use ESMF
```

62.3.1 GRACEobs_setup (Source File: GRACEobs_module.F90)

INTERFACE:

```
subroutine GRACEobs_setup(k, OBS_State, OBS_Pert_State)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_config, LIS_vecGrid, LIS_ensOnGrid
use LIS_timeMgrMod, only : LIS_clock, LIS_calendar
use LIS_logMod
use LIS_perturbMod

implicit none
```

ARGUMENTS:

```
integer                 ::  k
type(ESMF_State)        ::  OBS_State(LIS_rc%nnest)
type(ESMF_State)        ::  OBS_Pert_State(LIS_rc%nnest)
```

DESCRIPTION:

This routine completes the runtime initializations and creation of data strctures required for soil moisture assimilation
The arguments are:

OBS_State observation state

OBS_Pert_State observation perturbations state

62.3.2 getNSO_GRACE.F90 (Source File: getNSO_GRACE.F90)

Returns the number of selected observations (NSO) corresponding to the grid point specified

REVISION HISTORY:

```
21Jun2006: Sujay Kumar; Initial Specification
11Aug2011: Ben Zaitchik; Modified for GRACE
```

INTERFACE:

```
subroutine getNSO_GRACE(n, gid,st_id,en_id)

implicit none
```

DESCRIPTION:

Returns the number of selected GRACE observations (NSO) corresponding to the selected modeling point
The arguments are:

n index of the nest

gid index of the modeling point

st_id starting index of the selected observations

en_id ending index of the selected observations

62.3.3 read_GRACEobs (Source File: read_GRACEobs.F90)

REVISION HISTORY:

```
21Jun2006: Sujay Kumar; Initial Specification
11Aug2011: Ben Zaitchik; Modified for GRACE
```

INTERFACE:

```
subroutine read_GRACEobs(n, OBS_State, OBS_Pert_State)
```

USES:

```
use ESMF
use LIS_mpiMod
use LIS_coreMod
use LIS_logMod
use LIS_timeMgrMod
use GRACEobs_module
use LIS_fileIOMod
 use smootherDA_runMod, only: smootherDA_increments_mode

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: OBS_State
type(ESMF_State)    :: OBS_Pert_State
```

DESCRIPTION:

reads the GRACE TWS observations produced using a LIS open loop run and GRACE TELLUS gridded files, and packages it into an ESMF State with certain predefined attributes
The arguments are:

n index of the nest

OBS_State observations state

62.3.4 write_GRACEobs (Source File: write_GRACEobs.F90)

REVISION HISTORY:

```
25Jan2008: Sujay Kumar; Initial Specification
28Sep2011: Ben Zaitchik; modified for GRACE
```

INTERFACE:

```
subroutine write_GRACEobs(n, OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_logMod,     only : LIS_verify, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,  only : LIS_create_output_directory
use LIS_historyMod, only : LIS_writevar_gridded

implicit none
```

ARGUMENTS:

```
integer,     intent(in)  :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

writes the transformed (interpolated/upscaled/reprojected) GRACE observations to a file

---

62.3.5 GRACE_twsobsname (Source File: write_GRACEobs.F90)

INTERFACE:

```
subroutine GRACE_twsobsname(obsname)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

```
character(len=*)      :: obsname
```

DESCRIPTION:

62.4 Fortran: Module Interface LPRM_AMSREsm_Mod (Source File: LPRM_AMSREsm_M

This module contains interfaces and subroutines to handle Land Parameter Retrieval Model (LPRM) AMSR-E soil moisture retrievals
Ref: Owe et al. 2008; Multi-sensor historical climatology of satellite- derived global land surface moisture.

REVISION HISTORY:

```
 17 Jun 10    Sujay Kumar; Updated for use with LPRM retrieval version 5.
 20 Sep 12    Sujay Kumar; Updated to the NETCDF version from GES-DISC.
```

```
module LPRM_AMSREsm_Mod
```

USES:

```
use ESMF
use map_utils

implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: LPRM_AMSREsm_setup
```

PUBLIC TYPES:

```
public :: LPRM_AMSREsm_struc
```

62.4.1 LPRM_AMSREsm_setup (Source File: LPRM_AMSREsm_Mod.F90)

INTERFACE:

```
subroutine LPRM_AMSREsm_setup(k, OBS_State, OBS_Pert_State)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_timeMgrMod
use LIS_historyMod
use LIS_dataAssimMod
use LIS_perturbMod
use LIS_logmod

implicit none
```

ARGUMENTS:

```
integer                 ::  k
type(ESMF_State)        ::  OBS_State(LIS_rc%nnest)
type(ESMF_State)        ::  OBS_Pert_State(LIS_rc%nnest)
```

DESCRIPTION:

This routine completes the runtime initializations and creation of data strctures required for handling LPRM AMSR-E soil moisture data.
The arguments are:

OBS_State observation state

OBS_Pert_State observation perturbations state

62.4.2 getNSO_LPRM_AMSREsm.F90 (Source File: getNSO_LPRM_AMSREsm.F90)

Returns the number of selected observations (NSO) corresponding to the grid point specified

REVISION HISTORY:

```
21Jun2006: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine getNSO_LPRM_AMSREsm(n, gid,st_id,en_id)

  implicit none
```

DESCRIPTION:

Returns the number of selected soil moisture observations (NSO) corresponding to the selected modeling point
The arguments are:

n index of the nest

gid index of the modeling point

st_id starting index of the selected observations

en_id ending index of the selected observations

62.4.3 read_LPRM_AMSREsm (Source File: read_LPRM_AMSREsm.F90)

REVISION HISTORY:

```
17 Jun 2010: Sujay Kumar; Updated for use with LPRM AMSRE Version 5.
20 Sep 2012: Sujay Kumar; Updated to the NETCDF version of the data.
```

INTERFACE:

```
subroutine read_LPRM_AMSREsm(n, OBS_State, OBS_Pert_State)
```

USES:

```
  use ESMF
  use LIS_mpiMod
  use LIS_coreMod
  use LIS_logMod
  use LIS_timeMgrMod
  use LIS_dataAssimMod
  use map_utils
  use LIS_pluginIndices
  use LPRM_AMSREsm_Mod, only : LPRM_AMSREsm_struc

  implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: OBS_State
type(ESMF_State)    :: OBS_Pert_State
```

DESCRIPTION:

reads the AMSRE soil moisture observations from NETCDF files and applies the spatial masking for dense vegetation, rain and RFI. The data is then rescaled to the land surface model's climatology using rescaling algorithms.
The arguments are:

n index of the nest

OBS_State observations state

62.4.4 read_LPRM_data (Source File: read_LPRM_AMSREsm.F90)

INTERFACE:

```
subroutine read_LPRM_data(n, fname, smobs_ip)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
  use LIS_coreMod,  only : LIS_rc
  use LIS_logMod,   only : LIS_verify
  use map_utils,    only : latlon_to_ij
  use LPRM_AMSREsm_Mod, only : LPRM_AMSREsm_struc

  implicit none
```

INPUT PARAMETERS:

```
integer                        :: n
character (len=*)              :: fname
real                           :: smobs_ip(LIS_rc%lnc(n)*LIS_rc%lnr(n))
```

OUTPUT PARAMETERS:

DESCRIPTION:

This subroutine reads the LPRM NETCDF file and applies the data quality flags to filter the data. The retrievals are rejected when land surface temperature is below freezing, if rain is present, if RFI is present, if residual error is above 0.5 or if optical depth is above 0.8. Finally the routine combines both the C-band and X-band retrievals.
The arguments are:

n index of the nest

fname name of the LPRM AMSR-E file

smobs_ip soil moisture data processed to the LIS domain

FILES USED:

REVISION HISTORY:

62.4.5 create_LPRM_AMSREsm_filename (Source File: read_LPRM_AMSREsm.F90)

INTERFACE:

```
subroutine create_LPRM_AMSREsm_filename(ndir, path, yr, mo,da, filename)
```

USES:

```
implicit none
```

ARGUMENTS:

```
character(len=*)  :: filename
character(len=*)  :: path
integer           :: yr, mo, da
character (len=*) :: ndir
```

DESCRIPTION:

This subroutine creates the LPRM AMSRE cmd based on the time and date
The arguments are:

ndir name of the LPRM AMSRE soil moisture directory

path name of the sensor path (A-ascending, D-descending)

yr current year

mo current month

da current day

filename Generated LPRM filename

62.4.6 write_LPRM_AMSREsmobs (Source File: write_LPRM_AMSREsmobs.F90)

REVISION HISTORY:

```
25Jan2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine write_LPRM_AMSREsmobs(n, OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_logMod,     only : LIS_verify, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,  only : LIS_create_output_directory
use LIS_historyMod, only : LIS_writevar_gridded

implicit none
```

ARGUMENTS:

```
integer,     intent(in)  :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

writes the transformed (interpolated/upscaled/reprojected) LPRM AMSRE observations to a file

---

62.4.7 LPRM_AMSRE_smobsname (Source File: write_LPRM_AMSREsmobs.F90)

INTERFACE:

```
subroutine LPRM_AMSRE_smobsname(obsname)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

```
character(len=*)      :: obsname
```

DESCRIPTION:

---

63 MODIS Snow Cover Area

63.1 Fortran: Module Interface MODISscaobs_module (Source File: MODISscaobs_module.F90)

This module contains interfaces and subroutines to handle modis snow cover area retrievals. This implementation handles the 5km standard modis product and the 25km gap filled product

REVISION HISTORY:

```
 31 Mar 08    Jiarui Dong; Initial Specification
 10 Sep 08    Sujay Kumar; Adopted in LIS 6.0
 17 Apr 09    Sujay Kumar; Added support for the gap-filled product
```

```
module MODISscaobs_module
```

USES:

```
use ESMF
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: MODISscaobs_setup
```

PUBLIC TYPES:

```
PUBLIC :: MODISsca_obs_obj
```

63.1.1 MODISscaobs_setup (Source File: MODISscaobs_module.F90)

INTERFACE:

```
subroutine MODISscaobs_setup(k, OBS_State, OBS_Pert_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_config,LIS_vecGrid, LIS_ensOnGrid
use LIS_logMod,  only : LIS_logunit, LIS_verify
use LIS_perturbMod

implicit none
```

ARGUMENTS:

```
integer                 ::  k
type(ESMF_State)        ::  OBS_State(LIS_rc%nnest)
type(ESMF_State)        ::  OBS_Pert_State(LIS_rc%nnest)
```

DESCRIPTION:

This routine completes the runtime initializations and creation of data strctures required for snow cover area observations to be used in data assimilation
The arguments are:

k index of the data assimilation instance

OBS_State observation state

OBS_Pert_State observation perturbations state

63.1.2 getNSO_MODISsca.F90 (Source File: getNSO_MODISsca.F90)

Returns the number of selected observations (NSO) corresponding to the grid point specified

REVISION HISTORY:

```
21Jun2006: Sujay Kumar; Initial Specification
31Mar08    Jiarui Dong; adapted from getNSO_AMSREsm
```

INTERFACE:

```
subroutine getNSO_MODISsca(n, gid,st_id,en_id)

  implicit none
```

DESCRIPTION:

Returns the number of selected swe observations (NSO) corresponding to the selected modeling point
The arguments are:

n index of the nest

gid index of the modeling point

st_id starting index of the selected observations

en_id ending index of the selected observations

63.1.3 read_MODISscaobs (Source File: read_MODISscaobs.F90)

REVISION HISTORY:

```
31 Mar 08    Jiarui Dong; Initial Specification
10 Sep 08    Sujay Kumar; Adopted in LIS 6.0, included upscaling algorithm
17 Apr 09    Sujay Kumar; Added support for gap filled product
```

INTERFACE:

```
subroutine read_MODISscaobs(n, OBS_State, OBS_Pert_State)
```

USES:

```
  use ESMF
  use LIS_historyMod, only : LIS_readvar_gridded
  use LIS_coreMod,  only : LIS_rc, LIS_domain
  use LIS_logMod,     only : LIS_logunit, LIS_verify
  use MODISscaobs_module, only : MODISsca_obs_obj
  use LIS_mpiMod

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: n
  type(ESMF_State)    :: OBS_State
  type(ESMF_State)    :: OBS_Pert_State
```

DESCRIPTION:

reads the MODIS snow cover area observations from HDF-EOS files and packages it into an ESMF State with certain predefined attributes
The arguments are:

n index of the nest

OBS_State observations state

63.1.4 MODISscaGF_filename (Source File: read_MODISscaobs.F90)

REVISION HISTORY:

```
17 Apr 09    Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine MODISscaGF_filename(name, ndir)
```

USES:

```
use LIS_coreMod,only : LIS_rc
use LIS_logMod, only : LIS_logunit

implicit none
```

ARGUMENTS:

```
character(len=*)  :: name
character (len=*) :: ndir
```

DESCRIPTION:

This routine creates the name of the MODIS sca observation based on the current model date.
The arguments are:

name Name of the MODIS sca observation file

ndir Name of the MODIS sca root directory

63.1.5 MODISsca_filename (Source File: read_MODISscaobs.F90)

REVISION HISTORY:

```
10 Sep 08    Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine MODISsca_filename(name, ndir)
```

USES:

```
use LIS_coreMod,only : LIS_rc
use LIS_logMod, only : LIS_logunit

implicit none
```

ARGUMENTS:

```
character(len=*)  :: name
character (len=*) :: ndir
```

DESCRIPTION:

This routine creates the name of the MODIS sca observation based on the current model date.
The arguments are:

name Name of the MODIS sca observation file

ndir Name of the MODIS sca root directory

63.1.6 getMODISsca (Source File: read_MODISscaobs.F90)

INTERFACE:

```
subroutine getMODISsca(n,name,tmp_obsl)
```

USES:

```
use LIS_coreMod,only : LIS_rc,LIS_domain
use LIS_logMod, only : LIS_logunit, LIS_verify
use MODISscaobs_module, only : MODISsca_obs_obj

implicit none
```

DESCRIPTION:

This routine retrievs the MODIS sca observations. The data is read from a file in HDF-EOS format followed by the application of the confidence index and cloud cover flags. Finally the data is upscaled or downscaled to the LIS domain

---

63.1.7 calculate_sca (Source File: read_MODISscaobs.F90)

INTERFACE:

```
subroutine calculate_sca(n, dim1, dim2, ps,ci,pc,go)
```

USES:

```
use MODISscaobs_module, only: MODISsca_obs_obj
use LIS_coreMod, only : LIS_rc

implicit none
!ARGURMENTS:
integer,  intent(IN) :: n
```

```
integer,  intent(IN) :: dim1
integer,  intent(IN) :: dim2
integer*1            :: ps(dim1*dim2)
integer*1            :: ci(dim1*dim2)
integer*1            :: pc(dim1*dim2)
real                 :: go(LIS_rc%lnc(n)*LIS_rc%lnr(n))
```

DESCRIPTION:

This routine processes the raw sca observations by eliminating the cloud covered points and points of low CI. This routine also upscales/downscales the data MODIS sca data to the LIS resolution

---

63.1.8 write_MODISsca (Source File: write_MODISsca.F90)

REVISION HISTORY:

```
17 Apr 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine write_MODISsca(n, OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_logMod,     only : LIS_verify, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,  only : LIS_create_output_directory
use LIS_historyMod, only : LIS_writevar_gridded

implicit none
```

ARGUMENTS:

```
integer,     intent(in)  :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

writes the transformed (interpolated/upscaled/reprojected) NASA AMSRE observations to a file

---

63.1.9 MODIS_scaobsname (Source File: write_MODISsca.F90)

INTERFACE:

```
subroutine MODIS_scaobsname(obsname)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

```
character(len=*)      :: obsname
```

DESCRIPTION:

---

64 NASA AMSR-E Soil Moisture

64.1 Fortran: Module Interface NASA_AMSREsm_Mod (Source File: NASA_AMSREsm_M

This module contains interfaces and subroutines to handle NASA AMSR-E soil moisture retrievals. This plugin handles the gridded Aqua Daily L3 surface soil moisture data. For more details, please see: http://nsidc.org/data/ae_land3.html

REVISION HISTORY:

```
 06Nov07    Bailing Li; Initial Specification
 17Jun10    Sujay Kumar; Updated for use with NASA retrieval version 6.
```

```
module NASA_AMSREsm_Mod
```

USES:

```
use ESMF

implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: NASA_AMSREsm_setup
```

PUBLIC TYPES:

```
public :: NASA_AMSREsm_struc
```

64.1.1 NASA_AMSREsm_setup (Source File: NASA_AMSREsm_Mod.F90)

INTERFACE:

```
subroutine NASA_AMSREsm_setup(k, OBS_State, OBS_Pert_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_config, LIS_vecGrid, LIS_ensOnGrid
use LIS_timeMgrMod, only : LIS_clock, LIS_calendar, LIS_registerAlarm
use LIS_historyMod, only : LIS_readvar_gridded
use LIS_perturbMod
use LIS_logmod, only : LIS_logunit, LIS_verify, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber

implicit none
```

ARGUMENTS:

```
integer                 ::  k
type(ESMF_State)        ::  OBS_State(LIS_rc%nnest)
type(ESMF_State)        ::  OBS_Pert_State(LIS_rc%nnest)
```

DESCRIPTION:

This routine completes the runtime initializations and creation of data strctures required for soil moisture assimilation
The arguments are:

OBS_State observation state

OBS_Pert_State observation perturbations state

64.1.2 NASA_AMSREsm_computeInterpWeights (Source File: NASA_AMSREsm_Mod.F90)

INTERFACE:

```
subroutine NASA_AMSREsm_computeInterpWeights()
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod, only : LIS_logunit

implicit none
```

DESCRIPTION:

This subroutine sets up the interpolation weights to transform the AMSRE data in EASE grid to the grid used in the LIS simulation. The code employs a neighbor-based interpolation

---

64.1.3 getNSO_NASA_AMSREsm.F90 (Source File: getNSO_NASA_AMSREsm.F90)

Returns the number of selected observations (NSO) corresponding to the grid point specified

REVISION HISTORY:

```
21Jun2006: Sujay Kumar; Initial Specification
06Nov2007: Bailing Li; adapted from getNSO_syntheticsm1
```

INTERFACE:

```
subroutine getNSO_NASA_AMSREsm(n, gid,st_id,en_id)

  implicit none
```

DESCRIPTION:

Returns the number of selected soil moisture observations (NSO) corresponding to the selected modeling point
The arguments are:

n index of the nest

gid index of the modeling point

st_id starting index of the selected observations

en_id ending index of the selected observations

64.1.4 read_NASA_AMSREsm (Source File: read_NASA_AMSREsm.F90)

REVISION HISTORY:

```
06Nov2007: Bailing Li; Initial Specification
17Jun2010: Sujay Kumar; Updated for use with NASA AMSRE Version 6,
                        added LSM-based QC, generic handling of obs scaling
```

INTERFACE:

```
subroutine read_NASA_AMSREsm(n, OBS_State, OBS_Pert_State)
```

USES:

```
  use ESMF
  use LIS_mpiMod
  use LIS_coreMod,    only : LIS_rc, LIS_domain, &
       LIS_masterproc, LIS_npes, LIS_masterproc, LIS_localPet
  use LIS_timeMgrMod, only : LIS_isAlarmRinging
  use LIS_logMod,     only : LIS_logunit, LIS_verify
  use LIS_timeMgrMod, only : LIS_calendar, LIS_clock
  use LIS_logMod,     only : LIS_logunit, LIS_endrun, &
       LIS_getNextUnitNumber, LIS_releaseUnitNumber, LIS_verify
  use NASA_AMSREsm_Mod, only : NASA_AMSREsm_struc
  use LIS_pluginIndices

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: n
  type(ESMF_State)    :: OBS_State
  type(ESMF_State)    :: OBS_Pert_State
```

DESCRIPTION:

reads the amsr-e soil moisture observations from HDF-EOS files and packages it into an ESMF State with certain predefined attributes
The arguments are:

n index of the nest

OBS_State observations state

64.1.5 read_AMSREsm (Source File: read_NASA_AMSREsm.F90)

INTERFACE:

```
subroutine read_AMSREsm(n,name)
```

USES:

```
  use LIS_coreMod,      only : LIS_rc,LIS_domain
  use LIS_logmod,       only : LIS_logunit
  use NASA_AMSREsm_Mod, only : NASA_AMSREsm_struc
  implicit none

#if (defined USE_HDF4)
#include "hdf.f90"
#endif
```

ARGUMENTS:

```
  integer, intent(in)  :: n
  character(len=*)     :: name
```

DESCRIPTION:

64.1.6 maskObs_basedonQC (Source File: read_NASA_AMSREsm.F90)

INTERFACE:

```
subroutine maskObs_basedonQC(npts, smc, qc)

  implicit none
```

ARGUMENTS:

```
  integer        :: npts
  real           :: smc(npts,2)
  integer*2      :: qc(npts,2)
```

DESCRIPTION:

This subroutine masks the observations based on the QC flags. data points with ice, dense vegetation and RFI are masked out.
http://nsidc.org/data/docs/daac/ae_land3_l3_soil_moisture.gd.html

---

64.1.7 maskObs_basedonTime (Source File: read_NASA_AMSREsm.F90)

INTERFACE:

```
subroutine maskObs_basedonTime(npts, smc, smtime, rsmc,fnd)
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use LIS_logMod,   only : LIS_logunit
use LIS_timeMgrMod, only : LIS_get_julhr

implicit none
```

ARGUMENTS:

```
integer        :: npts
real           :: smc(npts,2)
real           :: smtime(npts,2)
real           :: rsmc(npts)
integer        :: fnd
```

DESCRIPTION:

This subroutine finds the observations in the AMSRE data that matches the current LIS timestep

---

64.1.8 NASA_AMSREsm_filename (Source File: read_NASA_AMSREsm.F90)

INTERFACE:

```
subroutine NASA_AMSREsm_filename(name, ndir, yr, mo,da)
```

USES:

```
use LIS_coreMod,only : LIS_rc
use LIS_logMod, only : LIS_logunit

implicit none
```

ARGUMENTS:

```
character*200      :: name
integer           :: yr, mo, da, hr,mn
character (len=*) :: ndir
```

DESCRIPTION:

This subroutine creates the NASA AMSRE filename based on the time and date
The arguments are:

name name of the NASA AMSRE soil moisture filename

ndir name of the NASA AMSRE soil moisture directory

yr current year

mo current month

da current day

64.1.9 write_NASA_AMSREsmobs (Source File: write_NASA_AMSREsmobs.F90)

REVISION HISTORY:

```
25Jan2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine write_NASA_AMSREsmobs(n, OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_logMod,     only : LIS_verify, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,  only : LIS_create_output_directory
use LIS_historyMod, only : LIS_writevar_gridded

implicit none
```

ARGUMENTS:

```
integer,     intent(in)  :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

writes the transformed (interpolated/upscaled/reprojected) NASA AMSRE observations to a file

---

64.1.10 NASA_AMSRE_smobsname (Source File: write_NASA_AMSREsmobs.F90)

INTERFACE:

```
subroutine NASA_AMSRE_smobsname(obsname)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

```
character(len=*)      :: obsname
```

DESCRIPTION:

---

65 PMW SWE or snow depth Observations

65.1 Fortran: Module Interface PMW_snow_Mod (Source File: PMW_snow_Mod.F90)

This module contains interfaces and subroutines to handle PMW-based swe or snow depth retrievals (e.g., AMSR-E, ANSA, SSMI, SMMR). Currently supported data products include:
1) LATLON grid in HDF5 file format (e.g., ANSA, the original dataset or bias corrected dataset; see references below)
2) EASE grid (NL & SL) in HDF-EOS file format (e.g., AMSR-E Leve 3 SWE)
3) EASE grid (NL & SL) in HDF4 file format (e.g., SMMR and SSMI)
NOTE: As of 4/23/2014, only option 1) is rigorously tested and confirmed to work as expected. Options 2) and 3) compile successfully but have not been tested.
Note the data file names must have one of the following two conventions:
- *YYYYMMDD*
- *YYYYDOY*
In addition, data files should be stored in corresponding year directory as follows: datadir/YYYY/YYYMMDD
ANSA snow depth data is primarily based on PMW, with observation gaps filled in using previous overpasses.
For more details, please see the ANSA reference:
Foster et al. "A blended global snow product using visible, passive microwave and scatterometer satellite data", International Journal of Remote Sensing, 2010.
For PMW snow depth bias correction, please refer to:
Liu, Y., C.D. Peters-Lidard, S. Kumar, J. Foster, M. Shaw, Y. Tian, and G. Fall, 2013: Assimilating satellite-based snow depth and snow cover products for improving snow predictions in Alaska. Advances in Water Resources, 54, 208-227.

REVISION HISTORY:

```
1 Jun 09   Sujay Kumar;  Initial Specification
12 Apr 2012 Yuqiong Liu; adapted to assimilate PMW-based snow depth or
                         SWE data
```

```
module PMW_snow_Mod
```

USES:

```
use ESMF
```

65.1.1 PMW_snow_setup (Source File: PMW_snow_Mod.F90)

INTERFACE:

```
subroutine PMW_snow_setup(k, OBS_State, OBS_Pert_State)
```

USES:

```
use LIS_coreMod
use LIS_timeMgrMod
use LIS_historyMod
use LIS_perturbMod
use LIS_logMod

implicit none
```

ARGUMENTS:

```
integer                 ::  k
type(ESMF_State)        ::  OBS_State(LIS_rc%nnest)
type(ESMF_State)        ::  OBS_Pert_State(LIS_rc%nnest)
```

DESCRIPTION:

This routine completes the runtime initializations and creation of data strctures required for PMW snow data.
The arguments are:

OBS_State observation state

OBS_Pert_State observation perturbations state

65.1.2 readPMWSnowDataConfig (Source File: PMW_snow_Mod.F90)

INTERFACE:

```
subroutine readPMWSnowDataConfig()
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod, only : LIS_logunit, LIS_verify

implicit none

integer       :: n, i
integer       :: status
```

DESCRIPTION:

This subroutine reads the configuration for PMW snow assimilation from the lis.config file

---

65.1.3 computeInterpWeights_latlon (Source File: PMW_snow_Mod.F90)

INTERFACE:

```
subroutine computeInterpWeights_latlon()
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_config, LIS_domain
use LIS_logMod, only : LIS_logunit

implicit none

integer    :: n, i
real       :: dx, dy
```

DESCRIPTION:

This subroutine sets up the interpolation weights to transform the PMW snow data in latlon grid to the grid used in the LIS simulation. The code employs a bilinear-based interpolation

---

65.1.4 computeInterpWeights_ease (Source File: PMW_snow_Mod.F90)

INTERFACE:

```
subroutine computeInterpWeights_ease()
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_config, LIS_domain
use LIS_logMod, only : LIS_logunit
use map_utils

implicit none
```

DESCRIPTION:

This subroutine sets up the interpolation weights to transform the AMSRE data in EASE grid to the grid used in the LIS simulation. The code employs a neighbor-based interpolation

---

65.1.5 getNSO_PMW_snow (Source File: getNSO_PMW_snow.F90)

Returns the number of selected observations (NSO) corresponding to the grid point specified

REVISION HISTORY:

```
1 Jun 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine getNSO_PMW_snow(n, gid,st_id,en_id)

implicit none
```

DESCRIPTION:

Returns the number of selected Tskin observations (NSO) corresponding to the selected modeling point The arguments are:

n index of the nest

gid index of the modeling point

st_id starting index of the selected observations

en_id ending index of the selected observations

65.1.6 read_PMW_snow (Source File: read_PMW_snow.F90)

REVISION HISTORY:

```
1 Jun 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_PMW_snow(n, OBS_State,OBS_Pert_State)
```

USES:

```
use ESMF
use LIS_mpiMod
use LIS_coreMod
use LIS_timeMgrMod
use LIS_logMod
use LIS_pluginIndices, only : LIS_PMWsnowobsId
use PMW_snow_Mod, only : PMW_snow_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: OBS_State
type(ESMF_State)    :: OBS_Pert_State
```

DESCRIPTION:

reads PMW-based snow observations and packages it into an ESMF State with certain predefined attributes. The routine reads the snow data at 0z, performs spatial interpolation to the LIS grid and keeps it in memory. At given localtime for each grid point, the code then packages the interpolated observations into the ESMF state.
The arguments are:

n index of the nest

OBS_State observations state

65.1.7 read_PMWSnow_HDF4 (Source File: read_PMW_snow.F90)

INTERFACE:

```
subroutine read_PMWSnow_HDF4(n,name)
```

USES:

```
use LIS_coreMod,      only : LIS_rc,LIS_domain
use LIS_logmod,       only : LIS_logunit
use PMW_snow_Mod, only : PMW_snow_struc
implicit none

#if (defined USE_HDF4)
#include "hdf.f90"
#endif
```

ARGUMENTS:

```
integer, intent(in)  :: n
character(len=*)     :: name
```

DESCRIPTION:

This routine extracts the SWE or snow depth retrievals and the associated quality control flags from the HDF4 files.

65.1.8 read_PMWSnow_HDFEOS (Source File: read_PMW_snow.F90)

INTERFACE:

```
subroutine read_PMWSnow_HDFEOS(n,name)
```

USES:

```
  use LIS_coreMod,      only : LIS_rc,LIS_domain
  use LIS_logmod,       only : LIS_logunit
  use PMW_snow_Mod, only : PMW_snow_struc
  implicit none

#if (defined USE_HDF4)
#include "hdf.f90"
#endif
```

ARGUMENTS:

```
  integer, intent(in)  :: n
  character(len=*)     :: name
```

DESCRIPTION:

This routine extracts the SWE or snow depth retrievals and the associated quality control flags from the HDF-EOS files.

65.1.9 read_PMWSnow_HDF5 (Source File: read_PMW_snow.F90)

INTERFACE:

```
subroutine read_PMWSnow_HDF5(n,name)
```

USES:

```
#if (defined USE_HDF5)
  use hdf5
#endif
  use LIS_coreMod,      only : LIS_rc,LIS_domain
  use LIS_logMod,       only : LIS_logunit, LIS_verify
  use PMW_snow_Mod,     only : PMW_snow_struc
  implicit none
```

ARGUMENTS:

```
  integer, intent(in)  :: n
  character(len=*)     :: name
```

DESCRIPTION:

This routine extracts the SWE or snow depth retrievals and the associated quality control flags from the LATLON HDF5 files. dim

---

65.1.10 PMW_snow_filename (Source File: read_PMW_snow.F90)

INTERFACE:

```
subroutine PMW_snow_filename(name, data_fn_conv, ndir, yr, mo,da)

  use LIS_coreMod, only: LIS_rc
  use LIS_logMod, only: LIS_logunit

  implicit none
```

ARGUMENTS:

```
  character*80      :: name
  integer           :: yr, mo, da
  character (len=*) :: ndir
  character (len=*) :: data_fn_conv
  character (len=50) :: str1, str2, str0
  integer           :: len1, i1, i2
```

DESCRIPTION:

This subroutine creates a timestamped PMW_snow data filename
The arguments are:

name PMW snow data file name

data_fn_conv PMW snow data file name convention

ndir PMW snow data directory

yr current year

mo current month

da current day

65.1.11 qc_PMWSnow (Source File: read_PMW_snow.F90)

INTERFACE:

```
subroutine qc_PMWSnow(n,snowobs, qc_flag, npts, snowobs_qc)
```

USES:

```
  use LIS_coreMod,      only : LIS_rc
  use PMW_snow_Mod, only : PMW_snow_struc

  implicit none
```

ARGUMENTS:

```
  integer, intent(in)  :: n
  integer, intent(in)  :: npts
  real, intent(in), dimension(npts)         :: snowobs, qc_flag
  real, intent(inout), dimension(npts)      :: snowobs_qc
  integer              :: i,j
```

DESCRIPTION:

This routine applies various qc conditions to the PMW snow data

---

65.1.12 write_PMW_snowobs (Source File: write_PMW_snowobs.F90)

REVISION HISTORY:

```
1 Jun 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine write_PMW_snowobs(n, OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_logMod,     only : LIS_verify, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,  only : LIS_create_output_directory
use LIS_historyMod, only : LIS_writevar_gridded

implicit none
```

ARGUMENTS:

```
integer,     intent(in)  :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

writes the transformed (interpolated/upscaled/reprojected) PMW observations to a file

---

65.1.13 PMW_snow_obsname (Source File: write_PMW_snowobs.F90)

INTERFACE:

```
subroutine PMW_snow_obsname(obsname)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

```
character(len=*)      :: obsname
```

DESCRIPTION:

65.2 Fortran: Module Interface RT_SMOPSsm_Mod (Source File: RT_SMOPSsm_Mod.F90)

This module contains interfaces and subroutines to handle

REVISION HISTORY:

```
 8 May 2013    Sujay Kumar; initial specification
```

```
module RT_SMOPSsm_Mod
```

USES:

```
use ESMF
use map_utils

implicit none

PRIVATE

-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: RT_SMOPSsm_setup
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: RT_SMOPSsm_struc
```

65.2.1 RT_SMOPSsm_setup (Source File: RT_SMOPSsm_Mod.F90)

INTERFACE:

```
subroutine RT_SMOPSsm_setup(k, OBS_State, OBS_Pert_State)
```

USES:

```
  use ESMF
  use LIS_coreMod
  use LIS_timeMgrMod
  use LIS_historyMod
  use LIS_dataAssimMod
  use LIS_perturbMod
  use LIS_logmod

  implicit none
```

ARGUMENTS:

```
  integer                 ::  k
  type(ESMF_State)        ::  OBS_State(LIS_rc%nnest)
  type(ESMF_State)        ::  OBS_Pert_State(LIS_rc%nnest)
```

DESCRIPTION:

This routine completes the runtime initializations and creation of data strctures required for handling RT SMOPS soil moisture data.
The arguments are:

OBS_State observation state

OBS_Pert_State observation perturbations state

65.2.2 getNSO_RT_SMOPSsm.F90 (Source File: getNSO_RT_SMOPSsm.F90)

Returns the number of selected observations (NSO) corresponding to the grid point specified

REVISION HISTORY:

```
21Jun2006: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine getNSO_RT_SMOPSsm(n, gid,st_id,en_id)

  implicit none
```

DESCRIPTION:

Returns the number of selected soil moisture observations (NSO) corresponding to the selected modeling point
The arguments are:

n index of the nest

gid index of the modeling point

st_id starting index of the selected observations

en_id ending index of the selected observations

65.2.3 read_RT_SMOPSsm (Source File: read_RT_SMOPSsm.F90)

REVISION HISTORY:

```
17 Jun 2010: Sujay Kumar; Updated for use with LPRM AMSRE Version 5.
20 Sep 2012: Sujay Kumar; Updated to the NETCDF version of the data.
```

INTERFACE:

```
subroutine read_RT_SMOPSsm(n, OBS_State, OBS_Pert_State)
```

USES:

```
  use ESMF
  use LIS_mpiMod
  use LIS_coreMod
  use LIS_logMod
  use LIS_timeMgrMod
```

```
use LIS_dataAssimMod
use map_utils
use LIS_pluginIndices
use RT_SMOPSsm_Mod, only : RT_SMOPSsm_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: OBS_State
type(ESMF_State)    :: OBS_Pert_State
```

DESCRIPTION:

reads the AMSRE soil moisture observations from NETCDF files and applies the spatial masking for dense vegetation, rain and RFI. The data is then rescaled to the land surface model's climatology using rescaling algorithms.
The arguments are:

n index of the nest

OBS_State observations state

65.2.4 read_RTSMOPS_data (Source File: read_RT_SMOPSsm.F90)

INTERFACE:

```
subroutine read_RTSMOPS_data(n, fname, smobs_ip, smtime_ip)
```

USES:

```
#if(defined USE_GRIBAPI)
  use grib_api
#endif
  use LIS_coreMod,  only : LIS_rc
  use LIS_logMod
  use LIS_timeMgrMod
  use RT_SMOPSsm_Mod, only : RT_SMOPSsm_struc

  implicit none
```

INPUT PARAMETERS:

```
integer                       :: n
character (len=*)             :: fname
real                          :: smobs_ip(LIS_rc%lnc(n)*LIS_rc%lnr(n))
real                          :: smtime_ip(LIS_rc%lnc(n)*LIS_rc%lnr(n))
```

OUTPUT PARAMETERS:

DESCRIPTION:

This subroutine reads the RTSMOPS grib2 file and applies the data quality flags to filter the data. The retrievals are rejected when the estimated error is above a predefined threshold (the recommeded value is 5 The information from gribtab is:

```
{ 2, 0, 7, 1, 3, 210, "BLENDEDSM", "NOAA Blended Liquid Volumetric Soil Moisture (Non-Frozen)", "m3m-3"},
{ 2, 0, 7, 1, 3, 211, "AMSRESM", "NOAA AMSR-E Liquid Volumetric Soil Moisture (Non-Frozen)", "m3m-3"},
{ 2, 0, 7, 1, 3, 212, "SMOSSM", "SMOS Liquid Volumetric Soil Moisture (Non-Frozen)", "m3m-3"},
{ 2, 0, 7, 1, 3, 213, "ASCATSM", "ASCAT Liquid Volumetric Soil Moisture (Non-Frozen)", "m3m-3"},
{ 2, 0, 7, 1, 3, 214, "WINDSATSM", "NOAA WindSat Liquid Volumetric Soil Moisture (Non-Frozen)", "m3m-3"},
{ 2, 0, 7, 1, 3, 215, "RESSM1", "Reserved Liquid Volumetric Soil Moisture 1 (Non-Frozen)", "m3m-3"},
{ 2, 0, 7, 1, 3, 216, "RESSM2", "Reserved Liquid Volumetric Soil Moisture 2 (Non-Frozen)", "m3m-3"},
{ 2, 0, 7, 1, 3, 217, "BLENDEDSMHR", "Observation Hour for BLENDEDSM", "-"},
{ 2, 0, 7, 1, 3, 218, "BLENDEDSMMN", "Observation Minute for BLENDEDSM", "-"},
{ 2, 0, 7, 1, 3, 219, "AMSRESMHR", "Observation Hour for AMSRESM", "-"},
{ 2, 0, 7, 1, 3, 220, "AMSRESMMN", "Observation Minute for AMSRESM", "-"},
{ 2, 0, 7, 1, 3, 221, "SMOSSMHR", "Observation Hour for SMOSSM", "-"},
{ 2, 0, 7, 1, 3, 222, "SMOSSMMN", "Observation Minute for SMOSSM", "-"},
{ 2, 0, 7, 1, 3, 223, "ASCATSMHR", "Observation Hour for ASCATSM", "-"},
{ 2, 0, 7, 1, 3, 224, "ASCATSMMN", "Observation Minute for ASCATSM", "-"},
{ 2, 0, 7, 1, 3, 225, "WINDSATSMHR", "Observation Hour for WINDSATSM", "-"},
{ 2, 0, 7, 1, 3, 226, "WINDSATSMMN", "Observation Minute for WINDSATSM", "-"},
{ 2, 0, 7, 1, 3, 227, "RESSM1HR", "Observation Hour for RESSM1", "-"},
{ 2, 0, 7, 1, 3, 228, "RESSM1MN", "Observation Minute for RESSM1", "-"},
{ 2, 0, 7, 1, 3, 229, "RESSM2HR", "Observation Hour for RESSM2", "-"},
{ 2, 0, 7, 1, 3, 230, "RESSM2MN", "Observation Minute for RESSM2", "-"},
{ 2, 0, 7, 1, 3, 231, "BLENDEDSMQA", "NOAA Blended Liquid Volumetric Soil Moisture QA", "-"},
{ 2, 0, 7, 1, 3, 232, "AMSRESMQA", "NOAA AMSR-E Liquid Volumetric Soil Moisture QA", "-"},
{ 2, 0, 7, 1, 3, 233, "SMOSSMQA", "SMOS Liquid Volumetric Soil Moisture QA", "-"},
{ 2, 0, 7, 1, 3, 234, "ASCATSMQA", "ASCAT Liquid Volumetric Soil Moisture QA", "-"},
{ 2, 0, 7, 1, 3, 235, "WINDSATSMQA", "NOAA WindSat Liquid Volumetric Soil Moisture QA", "-"},
{ 2, 0, 7, 1, 3, 236, "RESSM1QA", "Reserved Liquid Volumetric Soil Moisture 1 QA", "-"},
{ 2, 0, 7, 1, 3, 237, "RESSM2QA", "Reserved Liquid Volumetric Soil Moisture 2 QA", "-"},
```

The arguments are:

n index of the nest

fname name of the RTSMOPS AMSR-E file

smobs_ip soil moisture data processed to the LIS domain

65.2.5 create_RT_SMOPSsm_filename (Source File: read_RT_SMOPSsm.F90)

INTERFACE:

```
subroutine create_RT_SMOPSsm_filename(ndir, useRT, yr, mo,da, hr,filename)
```

USES:

```
implicit none
```

ARGUMENTS:

```
character(len=*)  :: filename
integer           :: useRT
integer           :: yr, mo, da,hr
character (len=*) :: ndir
```

DESCRIPTION:

This subroutine creates the RT SMOPS filename based on the time and date
The arguments are:

ndir name of the RT SMOPS soil moisture data directory

yr current year

mo current month

da current day

filename Generated RT SMOPS filename

65.2.6 write_RT_SMOPSsmobs (Source File: write_RT_SMOPSsmobs.F90)

REVISION HISTORY:

```
25Jan2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine write_RT_SMOPSsmobs(n, OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_logMod,     only : LIS_verify, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,  only : LIS_create_output_directory
use LIS_historyMod, only : LIS_writevar_gridded

implicit none
```

ARGUMENTS:

```
integer,     intent(in)  :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

writes the transformed (interpolated/upscaled/reprojected) LPRM AMSRE observations to a file

---

65.2.7 RT_SMOPS_smobsname (Source File: write_RT_SMOPSsmobs.F90)

INTERFACE:

```
subroutine RT_SMOPS_smobsname(obsname)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

```
character(len=*)      :: obsname
```

DESCRIPTION:

---

66 WindSat Soil Moisture

66.1 Fortran: Module Interface WindSatsm_Mod (Source File: WindSatsm_Mod.F90)

This module contains interfaces and subroutines to handle WindSat soil moisture retrievals. This plugin processes the retrievals based on 10, 18.7 and 37GHz channels. For more details, please see:
Li et al, "WindSat global soil moisture retrieval and validation", IEEE Transactions on Geoscience and Remote Sensing, 2009

REVISION HISTORY:

```
 22 Dec 09    Sujay Kumar;   Initial Specification
```

```
module WindSatsm_Mod
```

USES:

```
use ESMF
```

66.1.1 WindSatsm_setup (Source File: WindSatsm_Mod.F90)

INTERFACE:

```
subroutine WindSatsm_setup(k, OBS_State, OBS_Pert_State)
```

USES:

```
  use ESMF
  use LIS_coreMod
  use LIS_timeMgrMod
```

```
use LIS_historyMod
use LIS_dataAssimMod
use LIS_perturbMod
use LIS_logmod

implicit none
```

ARGUMENTS:

```
integer                 ::  k
type(ESMF_State)        ::  OBS_State(LIS_rc%nnest)
type(ESMF_State)        ::  OBS_Pert_State(LIS_rc%nnest)
```

DESCRIPTION:

This routine completes the runtime initializations and creation of data strctures required for WindSat soil moisture data.
Please note that currently only neighbor interpolation is supported
The arguments are:

OBS_State observation state

OBS_Pert_State observation perturbations state

66.1.2 getNSO_WindSatsm (Source File: getNSO_WindSatsm.F90)

Returns the number of selected observations (NSO) corresponding to the grid point specified

REVISION HISTORY:

```
22 Dec 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine getNSO_WindSatsm(n, gid,st_id,en_id)

  implicit none
```

DESCRIPTION:

Returns the number of selected Tskin observations (NSO) corresponding to the selected modeling point
The arguments are:

n index of the nest

gid index of the modeling point

st_id starting index of the selected observations

en_id ending index of the selected observations

66.1.3 read_WindSatsm (Source File: read_WindSatsm.F90)

REVISION HISTORY:

```
22 Dec 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_WindSatsm(n, OBS_State, OBS_Pert_State)
```

USES:

```
use ESMF
use LIS_mpiMod
use LIS_coreMod
use LIS_logMod
use LIS_timeMgrMod
use LIS_dataAssimMod
use map_utils
use LIS_pluginIndices
use WindSatsm_Mod, only : WindSatsm_struc


implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: OBS_State
type(ESMF_State)    :: OBS_Pert_State
```

DESCRIPTION:

reads the WindSat soil moisture observations and packages it into an ESMF State with certain predefined attributes. The routine also interpolates the soil moisture data to the LIS resolution
Current limitations
* only neighbor search is supported
* the assimilation interval is expected to be 3 hours
The arguments are:

n index of the nest

OBS_State observations state

66.1.4 WindSatsm_filename (Source File: read_WindSatsm.F90)

INTERFACE:

```
subroutine WindSatsm_filename(smname, tmname, tsname, clsname, ndir, yr, mo,da)

implicit none
```

ARGUMENTS:

```
character(len=*)     :: smname
character(len=*)     :: tmname
character(len=*)     :: tsname
character(len=*)     :: clsname

integer           :: yr, mo, da
character (len=*) :: ndir
```

DESCRIPTION:

This subroutine creates the WindSat filenames based the time and date
The arguments are:

smname name of the WindSat soil moisture filename

tmname name of the WindSat time information filename

ndir name of the WindSat soil moisture directory

yr current year

mo current month

da current day

66.1.5 write_WindSatsmobs (Source File: write_WindSatsmobs.F90)

REVISION HISTORY:

```
22 Dec 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine write_WindSatsmobs(n, OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_logMod,     only : LIS_verify, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,  only : LIS_create_output_directory
use LIS_historyMod, only : LIS_writevar_gridded

implicit none
```

ARGUMENTS:

```
integer,     intent(in)  :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

writes the transformed (interpolated/upscaled/reprojected) WindSat observations to a file

---

66.1.6 WindSat_smobsname (Source File: write_WindSatsmobs.F90)

INTERFACE:

```
subroutine WindSat_smobsname(obsname)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

```
character(len=*)      :: obsname
```

DESCRIPTION:

Part XIII
Parameter Estimation in LIS

67 Parameter Estimation in LIS

68 DREAM

68.1 Fortran: Module Interface DEMCzAlgorithm (Source File: DEMCzAlgorithm.F90)

This module contains routines that define the operations of the DEMCz algorithm. It is a parallel implementation of the ter Braak (2006) algorithm

REVISION HISTORY:

```
 08 July 2010; Sujay Kumar, Ken Harrison; Initial Specification
TODO:
 1. Initialization capability from a GA file
 2. Modify the output so that it can be used to initialize the LSM

use ESMF
use DEMCz_varctl
use LIS_coreMod
use LIS_optUEMod
use LIS_logMod

implicit none

PRIVATE

------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
------------------------------------------------------------------------------
 public :: DEMCz_init
 public :: DEMCz_setup
 public :: DEMCz_run
 public :: DEMCz_checkConvergence
 public :: DEMCz_getdecSpaceValues
 public :: DEMCz_readrestart
 public :: DEMCz_getNparam
------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
------------------------------------------------------------------------------
```

68.1.1 DEMCz_init (Source File: DEMCzAlgorithm.F90)

INTERFACE:

```
subroutine DEMCz_init()
```

USES:

DESCRIPTION:

This routine performs the initialization steps for DEMCz. It initializes the required memory structures, and creates an initial random population, for each grid point.

68.1.2 DEMCz_setup (Source File: DEMCzAlgorithm.F90)

INTERFACE:

```
subroutine DEMCz_setup()
```

USES:

DESCRIPTION:

This subroutine performs the second part of the DEMCz initialization, The routine obtains the decision space object (from the models used in optimization instance) and assigns the decision space variables to the DEMCz data structures

---

68.1.3 DEMCz_checkConvergence (Source File: DEMCzAlgorithm.F90)

INTERFACE:

```
subroutine DEMCz_checkConvergence(check)
```

ARGUMENTS:

```
  logical, intent(INOUT) :: check
```

DESCRIPTION:

This routine checks to see if the convergence criteria for DEMCz is reached. In this case, the routine simply checks to see if the specified number of generations is reached.

68.1.4 DEMCz_run (Source File: DEMCzAlgorithm.F90)

INTERFACE:

```
subroutine DEMCz_run()
```

USES:

DESCRIPTION:

68.1.5 DEMCz_setPreLnZ (Source File: DEMCzAlgorithm.F90)

INTERFACE:

```
subroutine  DEMCz_recordLnZ_X0()
```

DESCRIPTION:

68.1.6 DEMCz_drawCandidates (Source File: DEMCzAlgorithm.F90)

INTERFACE:

```
subroutine DEMCz_drawCandidates()
```

USES:

```
use LIS_historyMod,      only : LIS_readvar_restart
```

DESCRIPTION:

68.1.7 DEMCz_acceptReject (Source File: DEMCzAlgorithm.F90)

INTERFACE:

```
subroutine DEMCz_acceptReject()
```

DESCRIPTION:

68.1.8 writeDEMCzrestart (Source File: DEMCzAlgorithm.F90)

INTERFACE:

```
subroutine writeDEMCzrestart
```

USES:

```
use LIS_fileIOMod,  only : LIS_create_output_directory
use LIS_historyMod, only : LIS_writevar_restart
```

DESCRIPTION:

This routine writes the checkpoint data for a DEMCz restart

68.1.9 DEMCz_readrestart (Source File: DEMCzAlgorithm.F90)

INTERFACE:

```
subroutine DEMCz_readrestart
```

USES:

```
use LIS_historyMod,      only : LIS_readvar_restart
```

DESCRIPTION:

This routine reads the checkpoint data for a DEMCz restart

68.1.10 DEMCz_getNparam (Source File: DEMCzAlgorithm.F90)

INTERFACE:

```
subroutine DEMCz_getNparam(nparam)
```

DESCRIPTION:

This method returns the number of decision space variables

68.1.11 DEMCz_getdecSpaceValues (Source File: DEMCzAlgorithm.F90)

INTERFACE:

```
subroutine DEMCz_getdecSpaceValues(n)
```

USES:

DESCRIPTION:

This routine returns the array of decision space variables from the GA data structures

---

68.2 Fortran: Module Interface DEMCz_varctl (Source File: DEMCz_varctl.F90)

Module for specifying variables used to control the Genetic Algorithm implementation.

REVISION HISTORY:

```
04 Feb 2008; Sujay Kumar; Initial Specification
```

---

69 GeneticAlgorithm

69.1 Fortran: Module Interface GA_varctl (Source File: GA_varctl.F90)

Module for specifying variables used to control the Genetic Algorithm implementation.

REVISION HISTORY:

```
04 Feb 2008; Sujay Kumar; Initial Specification
```

69.2 Fortran: Module Interface GeneticAlgorithm (Source File: GeneticAlgorithm.F90)

This module contains routines that define the operations of a Genetic Algorithm (GA)

REVISION HISTORY:

```
 04 Feb 2008; Sujay Kumar; Initial Specification
use ESMF
use GA_varctl

implicit none

PRIVATE

-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
 public :: GAOpt_init
 public :: GAOpt_setup
 public :: GAOpt_run
 public :: GAOpt_checkConvergence
 public :: GAOpt_getdecSpaceValues
 public :: GAOpt_getNparam
 public :: GAOpt_readrestart
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
```

69.2.1 GAOpt_init (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine GAOpt_init()
```

USES:

```
use LIS_coreMod,         only : LIS_rc, LIS_config, LIS_vecTile
use LIS_optUEMod,        only : LIS_decisionSpace
use LIS_logMod,          only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun, LIS_verify
```

DESCRIPTION:

This routine performs the initialization steps for the GA. It initializes the required memory structures, and creates an initial random population, for each grid point.

69.2.2 GAOpt_setup (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine GAOpt_setup()
```

USES:

```
use LIS_coreMod,         only : LIS_rc, LIS_masterproc

use LIS_optUEMod,        only : LIS_decisionSpace
use LIS_logMod,          only : LIS_logunit, LIS_verify, LIS_endrun

implicit none
```

DESCRIPTION:

This subroutine performs the second part of the GA initialization, The routine obtains the decision space object (from the models used in optimization instance) and assigns the decision space variables to the GA data structures

---

69.2.3 GAOpt_checkConvergence (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine GAOpt_checkConvergence(check)
```

ARGUMENTS:

```
logical, intent(INOUT) :: check
```

DESCRIPTION:

This routine checks to see if the convergence criteria for GA is reached. In this case, the routine simply checks to see if the specified number of generations is reached.

69.2.4 GAOpt_run (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine GAOpt_run()
```

USES:

```
use LIS_coreMod,   only : LIS_rc
use LIS_logMod,    only : LIS_logunit
```

DESCRIPTION:

This routine performs the run steps for the GA, which includes the following steps

Evaluate Fitness : Evaluates the fitness of potential solutions

Selection : Selects better solutions for "mating"

Crossover : Performs recombination between parent solutions to produce new potential solutions

Mutation : Performs the mutation step

69.2.5 GAOpt_getdecSpaceValues (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine GAOpt_getdecSpaceValues(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_optUEMod,   only : LIS_decisionSpace
use LIS_logMod,     only : LIS_verify
```

DESCRIPTION:

This routine returns the array of decision space variables from the GA data structures

---

69.2.6 GAOpt_getNparam (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine GAOpt_getNparam(nparam)
```

DESCRIPTION:

This method returns the number of decision space variables

69.2.7 GA_recordFitness (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine GA_recordFitness()
```

USES:

```
use LIS_coreMod,         only : LIS_rc, LIS_masterproc
use LIS_optUEMod,        only : LIS_ObjectiveFunc, LIS_feasibleSpace
use LIS_logMod,          only : LIS_flush, LIS_logunit, LIS_verify, LIS_endrun
```

DESCRIPTION:

This method computes the fitness values for each organism, selects the best solution and ensures its selection in the subsequent generation (if elitism is enabled).

69.2.8 GA_selection (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine GA_selection(t)
```

DESCRIPTION:

This routine chooses pool of prospective parents via binary selection

---

69.2.9 GA_crossover (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine GA_crossover(t)
```

DESCRIPTION:

This method performs the crossover or recombination operation, using either a two point crossover or a uniform crossover method.

69.2.10 GA_mutate (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine GA_mutate(t)

  implicit none
  integer         :: t
```

DESCRIPTION:

This subroutine performs mutations on the children generation. Random jump mutation or random creep mutation is performed based on the mutation probability.

69.2.11 GA_newGeneration (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine GA_newGeneration
```

USES:

```
  use LIS_coreMod,  only : LIS_rc
  use LIS_optUEMod, only : LIS_decisionSpace
  use LIS_PE_HandlerMod, only : LIS_setPEDecisionSpace


  implicit none
```

DESCRIPTION:

This method performs the computations to update the decision space variables based on the new generation organisms.

69.2.12 select (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine select(t, mate, ipick)
```

DESCRIPTION:

This subroutine selects the better of two possible parents for mating.

69.2.13 encode (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine encode(j,k,array,iarray)
```

ARGUMENTS:

```
integer  :: j
integer  :: k
real     :: array(ga_ctl%nparam, ga_ctl%npopsize)
integer  :: iarray(ga_ctl%ngenes, ga_ctl%npopsize)
```

DESCRIPTION:

This routine converts a real number to a binary string

69.2.14 decode (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine decode(i,array,iarray)
```

ARGUMENTS:

```
integer  :: i
real     :: array(ga_ctl%nparam, ga_ctl%npopsize)
integer  :: iarray(ga_ctl%ngenes, ga_ctl%npopsize)
```

DESCRIPTION:

This routine converts a binary string to a real number.

69.2.15 writeGArestart (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine writeGArestart
```

USES:

```
use LIS_optUEMod,   only : LIS_decisionSpace
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_fileIOMod,  only : LIS_create_output_directory
use LIS_logMod,     only : LIS_logunit, LIS_verify
use LIS_historyMod, only : LIS_writevar_restart
```

DESCRIPTION:

This routine writes the checkpoint data for a GA restart

69.2.16 GAopt_readrestart (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine GAopt_readrestart
```

USES:

```
use LIS_optUEMod,        only : LIS_decisionSpace
use LIS_coreMod,         only : LIS_rc
use LIS_logMod,          only : LIS_logunit, LIS_verify
use LIS_historyMod,      only : LIS_readvar_restart
```

ARGUMENTS:

DESCRIPTION:

This routine reads the checkpoint data for a GA restart

69.2.17 writeGAoutput (Source File: GeneticAlgorithm.F90)

INTERFACE:

```
subroutine writeGAoutput
```

USES:

```
use LIS_optUEMod
use LIS_coreMod
use LIS_fileIOMod
use LIS_logMod
use LIS_historyMod
```

DESCRIPTION:

This routine writes a gridded output file, including GA fitness and GA best set of parameters

---

69.3 Fortran: Module Interface MCSIMAlgorithm (Source File: MCSIMAlgorithm.F90)

This module contains routines that define the operations of the MCSIM algorithm.

REVISION HISTORY:

```
08 July 2010; Sujay Kumar, Ken Harrison; Initial Specification
```

USES:

```
use ESMF
use MCSIM_varctl
use PobjFunc_Mod
use LIS_coreMod
use LIS_optUEMod
use LIS_logMod

implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: MCSIM_init
public :: MCSIM_setup
public :: MCSIM_run
public :: MCSIM_checkConvergence
public :: MCSIM_getdecSpaceValues
public :: MCSIM_getNparam
public :: MCSIM_readrestart
```

PUBLIC TYPES:

69.3.1 MCSIM_init (Source File: MCSIMAlgorithm.F90)

INTERFACE:

```
subroutine MCSIM_init()
```

USES:

DESCRIPTION:

This routine performs the initialization steps for MCSIM. It initializes the required memory structures, and creates an initial random population, for each grid point.

69.3.2 MCSIM_setup (Source File: MCSIMAlgorithm.F90)

INTERFACE:

```
subroutine MCSIM_setup()
```

USES:

DESCRIPTION:

This subroutine performs the second part of the MCMC initialization, This routine is currently empty.

69.3.3 MCSIM_checkConvergence (Source File: MCSIMAlgorithm.F90)

INTERFACE:

```
subroutine MCSIM_checkConvergence(check)
```

ARGUMENTS:

```
logical, intent(INOUT) :: check
```

DESCRIPTION:

This routine checks to see if the convergence criteria for GA is reached. In this case, the routine simply checks to see if the specified number of generations is reached.

69.3.4 MCSIM_run (Source File: MCSIMAlgorithm.F90)

INTERFACE:

```
subroutine MCSIM_run()
```

USES:

```
use LIS_coreMod,   only : LIS_rc
use LIS_optUEMod,  only : LIS_ObjectiveFunc, LIS_feasibleSpace
use LIS_logMod,    only : LIS_logunit, LIS_verify
```

DESCRIPTION:

This routine performs the run steps for the GA, which includes the following steps

Evaluate Fitness : Evaluates the fitness of potential solutions

Selection : Selects better solutions for "mating"

Crossover : Performs recombination between parent solutions to produce new potential solutions

Mutation : Performs the mutation step

69.3.5 writeMCSIMrestart (Source File: MCSIMAlgorithm.F90)

INTERFACE:

```
subroutine writeMCSIMrestart
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_fileIOMod,  only : LIS_create_output_directory
use LIS_logMod,     only : LIS_logunit, LIS_verify
use LIS_historyMod, only : LIS_writevar_restart
```

DESCRIPTION:

This routine writes the checkpoint data for a MCMC restart. Actually, the restart file is not necessary but instead serves as an output capability in the same format as the other optUE algorithms

---

69.3.6 MCSIM_readrestart (Source File: MCSIMAlgorithm.F90)

INTERFACE:

```
subroutine MCSIM_readrestart
```

USES:

```
use LIS_historyMod,      only : LIS_readvar_restart
```

DESCRIPTION:

This routine reads the checkpoint data for a MCSIM restart

69.3.7 MCSIM_getNparam (Source File: MCSIMAlgorithm.F90)

INTERFACE:

```
subroutine MCSIM_getNparam(nparam)
```

DESCRIPTION:

This method returns the number of decision space variables

69.3.8 MCSIM_getdecSpaceValues (Source File: MCSIMAlgorithm.F90)

INTERFACE:

```
subroutine MCSIM_getdecSpaceValues(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc
```

DESCRIPTION:

This routine returns the array of decision space variables from the GA data structures

---

69.4 Fortran: Module Interface MCSIM_varctl (Source File: MCSIM_varctl.F90)

Module for specifying variables used to control the Genetic Algorithm implementation.

REVISION HISTORY:

```
04 Feb 2008; Sujay Kumar; Initial Specification
```

69.5 Fortran: Module Interface LLobjFunc_Mod (Source File: LLobjFunc_Mod.F90)

This module provides the objects and methods to compute a Least Square (LL) metric for use in parameter estimation.

REVISION HISTORY:

```
15 Jul 2009: Ken Harrison and Soni Yatheendradas; Initial implementation
```

USES:

```
use ESMF

implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public ::  initializeLLObjFunc
```

69.5.1 intializeLLObjFunc (Source File: LLobjFunc_Mod.F90)

INTERFACE:

```
subroutine initializeLLObjFunc()
```

USES:

```
use LIS_coreMod,         only : LIS_vecTile
use LIS_optUEMod,        only : LIS_ObjectiveFunc
use LIS_logMod,          only : LIS_verify
```

DESCRIPTION:

This method initializes the objects to be used in the least square computations

---

69.5.2 computeLLestimate (Source File: computeLLestimate.F90)

REVISION HISTORY:

```
15 Jul 2009: Ken Harrison and Soni Yatheendradas; Initial implementation
```

INTERFACE:

```
subroutine computeLLestimate()
```

USES:

```
use ESMF
use LIS_coreMod,         only : LIS_rc
use LIS_optUEMod, only : LIS_ObjectiveFunc
use LIS_logMod,          only : LIS_logUnit, LIS_verify
```

DESCRIPTION:

This routine computes the objective function values to be used in the optimization algorithm. The routine is required to specify both a maximization and a minimization criteria

---

69.5.3 updateLLestimate (Source File: updateLLestimate.F90)

REVISION HISTORY:

```
15 Jul 2009: Ken Harrison and Soni Yatheendradas; Initial implementation
```

INTERFACE:

```
subroutine updateLLestimate()
```

USES:

```
use ESMF
use LIS_coreMod,         only : LIS_rc, LIS_domain
use LIS_optUEMod,        only : LIS_ObjectiveFunc, LIS_DecisionSpace
use LIS_logMod,          only : LIS_verify, LIS_endrun, LIS_logunit
use LIS_PE_HandlerMod,       only : LIS_PEOBS_State, LIS_PEOBSPred_State
```

DESCRIPTION:

This method updates the running sum of the squared error. Here the squared error is computed using the observations used for parameter estimation and the model simulated values (obspred).

69.6 Fortran: Module Interface LSobjFunc_Mod (Source File: LSobjFunc_Mod.F90)

This module provides the objects and methods to compute a Least Square (LS) metric for use in parameter estimation.

REVISION HISTORY:

```
15 Jul 2009: Sujay Kumar; Initial implementation
```

USES:

```
use ESMF

implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public ::  initializeLSObjFunc
```

PUBLIC TYPES:

```
public :: ls_ctl
```

69.6.1 intializeLSObjFunc (Source File: LSobjFunc_Mod.F90)

INTERFACE:

```
subroutine initializeLSObjFunc()
```

USES:

```
use LIS_coreMod,         only : LIS_vecTile, LIS_config
use LIS_optUEMod,        only : LIS_ObjectiveFunc
use LIS_logMod,          only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun, LIS_verify
use LIS_PE_HandlerMod,   only : LIS_PEOBS_State
```

DESCRIPTION:

This method initializes the objects to be used in the least square computations

---

69.6.2 computeLSestimate (Source File: computeLSestimate.F90)

REVISION HISTORY:

```
15 Jul 2009: Sujay Kumar; Initial implementation
```

INTERFACE:

```
subroutine computeLSestimate()
```

USES:

```
use ESMF
use LIS_mpiMod
use LIS_coreMod,         only : LIS_rc, LIS_masterproc, &
     LIS_localPet, LIS_npes, LIS_domain
use LIS_optUEMod,        only : LIS_ObjectiveFunc,LIS_feasibleSpace
use LIS_logMod,          only : LIS_logUnit, LIS_verify
use LSObjFunc_Mod,       only : ls_ctl
```

DESCRIPTION:

This routine computes the objective function values to be used in the optimization algorithm. The routine is required to specify both a maximization and a minimization criteria

---

69.6.3 updateLSestimate (Source File: updateLSestimate.F90)

REVISION HISTORY:

```
15 Jul 2009: Sujay Kumar; Initial implementation
```

INTERFACE:

```
subroutine updateLSestimate()
```

USES:

```
use ESMF
use LIS_mpiMod
use LIS_coreMod,         only : LIS_rc, LIS_masterproc, &
     LIS_localPet, LIS_npes, LIS_domain
use LIS_optUEMod,        only : LIS_ObjectiveFunc
use LIS_logMod,          only : LIS_verify
use LSObjFunc_Mod,       only : ls_ctl
use LIS_PE_HandlerMod,   only : LIS_PEOBS_State, LIS_PEOBSPred_State
```

DESCRIPTION:

This method updates the running sum of the squared error. Here the squared error is computed using the observations used for parameter estimation and the model simulated values (obspred).

69.7 Fortran: Module Interface PobjFunc_Mod (Source File: PobjFunc_Mod.F90)

This module provides the objects and methods to compute a "(P)osterior probability" metric useful for Bayesian algorithms (and uncertainty propagation). As described below,while related to posterior probability, it is not equated to posterior probability. Posterior probability is defined as `p(theta|y)`, where theta is the vector of parameters for which uncertainty estimation is sought (think parameters requiring calibration) and y is data (e.g., remote sensing data). Its evaluation is dictated by Bayes' rule of probability:

```
p(theta|y)=p(y|theta)*p(theta)
           -------------------
                 p(y)
```

where `p(y|theta)` is the likelihood of data y given theta; p(theta) is the initial, or prior, probability of theta; and p(y) is the probability of observing y.
The metric used is the natural log of the numerator of Bayes' rule, and thus ignores the term, p(y), in the denominator. p(y) can be thought of as a normalizing constant to ensure that the posterior probability integrated over all theta is equal to one. The difficulty in carrying out the integration to compute p(y) (=int over theta of `p(y|theta)*p(theta)`) is what motivates the development of the Bayesian algorithms that use this metric. All such algorithms involve computing the ratio of the posterior probability (density) of two values of theta. In this ratio-ing, the problematic denominator conveniently cancels out. Therefore this metric only concerns the numerator of Bayes' rule.
We work in log space as the the evaluation of the likelihood term –`p(y|theta)`– is subject to truncation error (involving much product-ing of very low values, any one of which may be truncated to zero). Moreover, the operations, specifically, the aformentioned ratio-ing can be carried out conveniently in log space.

REVISION HISTORY:

```
7 Sep 2010: Ken Harrison
```

USES:

```
use ESMF

implicit none

PRIVATE
integer, parameter :: dist_uniformID     = 0
integer, parameter :: dist_normalID      = -1
integer, parameter :: dist_lognormalID   = -2
integer, parameter :: dist_betaID        = -3
integer, parameter :: dist_triangularID  = -4
integer, parameter :: dist_discreteID    = -5
integer, parameter :: dist_mvnormalID    = -6

real, parameter :: mindensity=-1e+20
real, parameter :: PI=3.14159265359
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
public ::  initializePObjFunc
public ::  dist_sample
public ::  dist_lnprob

-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
```

69.7.1 intializePObjFunc (Source File: PobjFunc_Mod.F90)

INTERFACE:

```
subroutine initializePObjFunc()
```

USES:

```
use LIS_coreMod,         only : LIS_vecTile, LIS_config
use LIS_optUEMod,        only : LIS_ObjectiveFunc, LIS_decisionSpace
use LIS_logMod,          only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun, LIS_verify
```

DESCRIPTION:

This method initializes the objects to be used in the posterior probability metric for use by parameter and uncertainty estimation routines.

69.7.2 computePestimate (Source File: computePestimate.F90)

REVISION HISTORY:

```
10 Sep 2009: Ken Harrison; Initial implementation
```

INTERFACE:

```
subroutine computePestimate()
```

USES:

```
use ESMF
use LIS_coreMod,         only : LIS_rc
use LIS_optUEMod, only : LIS_ObjectiveFunc, LIS_decisionSpace
use LIS_logMod,          only : LIS_logUnit, LIS_verify
use PobjFunc_Mod ! , only : dist_lnprob, dist_ctl
```

DESCRIPTION:

This routine stores the posterior probabilities for use by the optimization and uncertainty estimation algorithms. The routine is required to specify both a maximization and a minimization criteria.

69.7.3 updatePestimate (Source File: updatePestimate.F90)

REVISION HISTORY:

```
10 Sep 2010: Ken Harrison; Initial implementation
```

INTERFACE:

```
subroutine updatePestimate()
```

USES:

```
use ESMF
use LIS_coreMod,         only : LIS_rc, LIS_domain
use LIS_optUEMod,        only : LIS_ObjectiveFunc, LIS_DecisionSpace
use LIS_logMod,          only : LIS_verify, LIS_endrun, LIS_logunit
use LIS_PE_HandlerMod,       only : LIS_PEOBS_State, LIS_PEOBSPred_State
```

DESCRIPTION:

The updating of the posterior probability metric involves only the likelihood (LL) term of Bayes' rule. The likelihood is the means by which the observations y are compared to the model simulated values (obspred).

69.8 Fortran: Module Interface AMSRE_SR_em_obsMod (Source File: AMSRE_SR_em_obsMod.F90)

REVISION HISTORY:

```
    11 Jul 11    Ken Harrison;   Initial Specification
```

```
module AMSRE_SR_em_obsMod
```

USES:

```
   use ESMF
   use map_utils
```

69.8.1 AMSRE_SR_em_obs_setup (Source File: AMSRE_SR_em_obsMod.F90)

INTERFACE:

```
   subroutine AMSRE_SR_em_obs_setup(Obs_State)
```

USES:

```
     use LIS_coreMod, only : LIS_rc, LIS_config, &
          LIS_vecGrid, LIS_domain, LIS_masterproc
     use LIS_timeMgrMod, only : LIS_calendar
     use LIS_logMod, only : LIS_logunit, LIS_verify, &
          LIS_getNextUnitNumber, LIS_releaseUnitNumber
     use map_utils, only: map_init, map_set, latlon_to_ij

     implicit none
```

ARGUMENTS:

```
     type(ESMF_State)       ::  Obs_State(LIS_rc%nnest)
```

DESCRIPTION:

The arguments are:

Obj_Space observation/Objective space object

69.8.2 read_AMSRE_SR_em_obsdata (Source File: read_AMSRE_SRdata.F90)

REVISION HISTORY:

```
31 Jan 2012: Ken Harrison; Initial Specification
```

INTERFACE:

```
subroutine read_AMSRE_SR_em_obsdata(Obj_Space)
```

USES:

```
use ESMF
use LIS_mpiMod
use LIS_coreMod,  only : LIS_rc, LIS_domain, LIS_npes, LIS_localPet
use LIS_timeMgrMod, only : LIS_calendar
use LIS_logMod,     only : LIS_logunit, LIS_verify, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber
use LIS_fileIOMod,      only : LIS_readData
use AMSRE_SR_em_obsMod, only : SRemobs
use map_utils

implicit none
```

ARGUMENTS:

```
type(ESMF_State)    :: Obj_Space
```

DESCRIPTION:

The arguments are:

n index of the nest

Obj_State observations state

69.8.3 reset_AMSRE_SR_em_obs_data (Source File: reset_AMSRE_SRObsdata.F90)

REVISION HISTORY:

```
09 Jul 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine reset_AMSRE_SR_em_obs_data(Obj_Space)
#if 0
```

USES:

```
use ESMF
use LIS_coreMod,  only : LIS_rc, LIS_masterproc
use LIS_logMod,     only : LIS_logunit, LIS_verify, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber
use LIS_fileIOMod,      only : LIS_create_output_directory
use LIS_historyMod,     only : LIS_writevar_gridded

implicit none
```

ARGUMENTS:

```
type(ESMF_State)    :: Obj_Space
```

DESCRIPTION:

write the Walnut Gulch PBMR soil moisture data to disk
The arguments are:

n index of the nest

Obj_State observations state

69.8.4 write_AMSRE_SR_em_obs_data (Source File: write_AMSRE_SR_em_obs_data.F90)

REVISION HISTORY:

```
09 Jul 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine write_AMSRE_SR_em_obs_data(Obj_Space)
#if 0
```

USES:

```
use ESMF
use LIS_coreMod,  only : LIS_rc, LIS_masterproc
use LIS_logMod,     only : LIS_logunit, LIS_verify, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber
use LIS_fileIOMod,      only : LIS_create_output_directory
use LIS_historyMod,     only : LIS_writevar_gridded

implicit none
```

ARGUMENTS:

```
type(ESMF_State)    :: Obj_Space
```

DESCRIPTION:

write the Walnut Gulch PBMR soil moisture data to disk
The arguments are:

n index of the nest

Obj_State observations state

69.8.5 AMSRE_SR_obsname (Source File: write_AMSRE_SR_em_obs_data.F90)

INTERFACE:

```
subroutine AMSRE_SR_obsname(obsname)
```

USES:

```
use LIS_coreMod, only : LIS_rc
```

ARGUMENTS:

```
character(len=*)     :: obsname
```

DESCRIPTION:

This method generates a timestamped filename for the processed PBMR observations.

69.9 Fortran: Module Interface Empty_obsMod (Source File: Empty_obsMod.F90)

REVISION HISTORY:

```
 09 Jul 09    Sujay Kumar;   Initial Specification
```

```
module Empty_obsMod
```

USES:

```
use ESMF
```

69.9.1 Empty_obs_setup (Source File: Empty_obsMod.F90)

INTERFACE:

```
subroutine Empty_obs_setup(Obs_State)
```

USES:

```
  use LIS_coreMod, only : LIS_rc, LIS_config, LIS_vecGrid
  use LIS_timeMgrMod, only : LIS_calendar
  use LIS_logMod, only : LIS_logunit, LIS_verify, &
       LIS_getNextUnitNumber, LIS_releaseUnitNumber

  implicit none
```

ARGUMENTS:

```
  type(ESMF_State)       ::  Obs_State(LIS_rc%nnest)
```

DESCRIPTION:

The arguments are:

Obj_Space observation/Objective space object

69.10 Fortran: Module Interface LPRM_AMSREsm_obsMod (Source File: LPRM_AMSREs

REVISION HISTORY:

```
11 Jul 11    Ken Harrison;   Initial Specification
```

```
module LPRM_AMSREsm_obsMod
```

USES:

```
use ESMF
use map_utils
```

69.10.1 LPRM_AMSREsm_obs_setup (Source File: LPRM_AMSREsm_obsMod.F90)

INTERFACE:

```
subroutine LPRM_AMSREsm_obs_setup(Obs_State)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_config, LIS_vecGrid, LIS_domain
use LIS_timeMgrMod
use LIS_logMod, only : LIS_logunit, LIS_verify, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber
use map_utils, only: map_init, map_set, latlon_to_ij

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       ::  Obs_State(LIS_rc%nnest)
```

DESCRIPTION:

The arguments are:

Obj_Space observation/Objective space object

69.10.2 read_LPRM_AMSREsm_obsdata (Source File: read_LPRM_AMSREsmdata.F90)

REVISION HISTORY:

```
31 Jan 2012: Ken Harrison; Initial Specification
```

INTERFACE:

```
subroutine read_LPRM_AMSREsm_obsdata(Obj_Space)
```

USES:

```
use ESMF
use LIS_mpiMod
use LIS_coreMod,  only : LIS_rc, LIS_domain, LIS_npes, LIS_localPet
use LIS_timeMgrMod
use LIS_logMod,     only : LIS_logunit, LIS_verify, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber
use LIS_fileIOMod,      only : LIS_readData
use LPRM_AMSREsm_obsMod, only : LPRM_AMSREsm_obs_struc
use map_utils

implicit none
```

ARGUMENTS:

```
type(ESMF_State)    :: Obj_Space
```

DESCRIPTION:

The arguments are:

n index of the nest

Obj_State observations state

69.10.3 read_LPRM_obs_data (Source File: read_LPRM_AMSREsmdata.F90)

INTERFACE:

```
subroutine read_LPRM_obs_data(n, fname, smobs_ip)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
  use LIS_coreMod,  only : LIS_rc
  use LIS_logMod,   only : LIS_verify
  use map_utils,    only : latlon_to_ij
  use LPRM_AMSREsm_obsMod, only : LPRM_AMSREsm_obs_struc

  implicit none
```

INPUT PARAMETERS:

```
  integer                       :: n
  character (len=*)             :: fname
  real                          :: smobs_ip(LIS_rc%lnc(n)*LIS_rc%lnr(n))
```

OUTPUT PARAMETERS:

DESCRIPTION:

This subroutine reads the LPRM NETCDF file and applies the data quality flags to filter the data. The retrievals are rejected when land surface temperature is below freezing, if rain is present, if RFI is present, if residual error is above 0.5 or if optical depth is above 0.8. Finally the routine combines both the C-band and X-band retrievals.
The arguments are:

n index of the nest

fname name of the LPRM AMSR-E file

smobs_ip soil moisture data processed to the LIS domain

FILES USED:

REVISION HISTORY:

69.10.4 create_LPRM_AMSREsm_obs_filename (Source File: read_LPRM_AMSREsmdata.F90)

INTERFACE:

```
subroutine create_LPRM_AMSREsm_obs_filename(ndir, path, yr, mo,da, filename)
```

USES:

```
implicit none
```

ARGUMENTS:

```
character(len=*)  :: filename
character(len=*)  :: path
integer           :: yr, mo, da
character (len=*) :: ndir
```

DESCRIPTION:

This subroutine creates the LPRM AMSRE cmd based on the time and date
The arguments are:

ndir name of the LPRM AMSRE soil moisture directory

path name of the sensor path (A-ascending, D-descending)

yr current year

mo current month

da current day

filename Generated LPRM filename

69.10.5 reset_LPRM_AMSREsm_obs_data (Source File: reset_LPRM_AMSREsmObsdata.F90)

REVISION HISTORY:

```
19 Feb 2013: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine reset_LPRM_AMSREsm_obs_data(Obj_Space)
```

USES:

```
use ESMF
use LPRM_AMSREsm_obsMod, only : LPRM_AMSREsm_obs_struc
use LIS_coreMod, only : LIS_rc
use LIS_logMod, only : LIS_logunit
use LIS_timeMgrMod

implicit none
```

ARGUMENTS:

```
type(ESMF_State)    :: Obj_Space
```

DESCRIPTION:

The arguments are:

n index of the nest

Obj_State observations state

69.10.6 write_LPRM_AMSREsm_obs_data (Source File: write_LPRM_AMSREsm_obs_data.F90)

REVISION HISTORY:

```
09 Jul 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine write_LPRM_AMSREsm_obs_data(Obj_Space)
#if 0
```

USES:

```
use ESMF
use LIS_coreMod,  only : LIS_rc, LIS_masterproc
use LIS_logMod,     only : LIS_logunit, LIS_verify, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber
use LIS_fileIOMod,      only : LIS_create_output_directory
use LIS_historyMod,     only : LIS_writevar_gridded

implicit none
```

ARGUMENTS:

```
type(ESMF_State)    :: Obj_Space
```

DESCRIPTION:

write the Walnut Gulch PBMR soil moisture data to disk
The arguments are:

n index of the nest

Obj_State observations state

69.10.7 read_templateObs (Source File: read_templateObs.F90)

REVISION HISTORY:

```
09 Jul 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_templateObs(Obj_Space)
```

USES:

```
use ESMF
use LIS_logMod,    only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)    :: Obj_Space
```

DESCRIPTION:

This is an empty, template for reading observations for optUE
The arguments are:

n index of the nest

Obj_State observations state

69.10.8 reset_templateObs (Source File: reset_templateObs.F90)

REVISION HISTORY:

```
09 Jul 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine reset_templateObs(Obj_Space)
```

USES:

```
use ESMF

implicit none
```

ARGUMENTS:

```
type(ESMF_State)    :: Obj_Space
```

DESCRIPTION:

Empty template for reseting the optUE-observation data structures
The arguments are:

n index of the nest

Obj_State observations state

69.11 Fortran: Module Interface templateObs_module (Source File: templateObs_module.F90)

This is a template for the definition of observations for optUE

REVISION HISTORY:

```
 09 Jul 09    Sujay Kumar;   Initial Specification
```

```
module templateObs_module
```

USES:

```
use ESMF
```

69.11.1 templateObs_setup (Source File: templateObs_module.F90)

INTERFACE:

```
subroutine templateObs_setup(Obs_State)
```

USES:

```
use LIS_coreMod,   only : LIS_rc
use LIS_logMod,    only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       ::  Obs_State(LIS_rc%nnest)
```

DESCRIPTION:

This is a empty, template for the definition of observation (for optUE) data structures
The arguments are:

Obj_Space observation/Objective space object

69.11.2 write_templateObs (Source File: write_templateObs.F90)

REVISION HISTORY:

```
09 Jul 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine write_templateObs(Obj_Space)
```

USES:

```
use ESMF

implicit none
```

ARGUMENTS:

```
type(ESMF_State)    :: Obj_Space
```

DESCRIPTION:

empty template for writing the optUE observations
The arguments are:

n index of the nest

Obj_State observations state

Part XIV
Land Surface Models in LIS

70 Land Surface Models in LIS

71 Community Land Model version 2.0

This section describes the interface implementations for the CLM land surface model. CLM (Community Land Model) is a 1-D land surface model, written in Fortran 90, developed by a grass-roots collaboration of scientists who have an interest in making a general land model available for public use. LIS currently uses CLM version 2.0. CLM version 2.0 was released in May 2002. The source code for CLM 2.0 is freely available from the National Center for Atmospheric Research (NCAR) [1]. The CLM is used as the land model for the Community Climate System Model (CCSM) (http://www.ccsm.ucar.edu/), which includes the Community Atmosphere Model (CAM) (http://www.cgd.ucar.edu/cms/). CLM is executed with all forcing, parameters, dimensioning, output routines, and coupling performed by an external driver of the user's design (in this case done by LIS). CLM requires pre-processed data such as the land surface type, soil and vegetation parameters, model initialization, and atmospheric boundary conditions as input. The model applies finite-difference spatial discretization methods and a fully implicit time-integration scheme to numerically integrate the governing equations. The model subroutines apply the governing equations of the physical processes of the soil-vegetation-snowpack medium, including the surface energy balance equation, Richards' [11] equation for soil hydraulics, the diffusion equation for soil heat transfer, the energy-mass balance equation for the snowpack, and the Collatz et al. [4] formulation for the conductance of canopy transpiration.

71.0.3 canhtset (Source File: canhtset.F90)

REVISION HISTORY:

```
15 Nov 2002: Jon Gottschalck; Initial code
```

INTERFACE:

```
subroutine canhtset (n)
```

USES:

```
use clm2_lsmMod          ! CLM tile variables
use clm2_varpar     , only : maxpatch
use LIS_coreMod, only  : LIS_rc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine reads in canopy height information into CLM. Expects a file with the top and bottom canopy heights for each veg type.

71.0.4 clm2_dynsetup (Source File: clm2_dynsetup.F90)

REVISION HISTORY:

```
20 Jan 2003  Sujay Kumar Initial Specification
```

INTERFACE:

```
subroutine clm2_dynsetup(n)

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine sets up the time-dependent variables in CLM The routine invokes methods to read the LAI data

---

71.0.5 clm2_finalize (Source File: clm2_finalize.F90)

REVISION HISTORY:

```
  26 Oct 2005: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine clm2_finalize()
```

USES:

```
  use LIS_coreMod, only : LIS_rc
  use clm2_lsmMod
```

DESCRIPTION:

This routine cleans up the allocated memory structures in CLM

71.0.6 clm2_lairead.F90: (Source File: clm2_lairead.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 data
   01 Oct 2002: Jon Gottschalck; Modified to add MODIS LAI data
```

INTERFACE:

```
subroutine clm2_lairead (n)
```

USES:

```
  use LIS_coreMod, only : LIS_rc, LIS_surface
  use LIS_vegDataMod, only : LIS_lai, LIS_sai
  use clm2_lsmMod
```

71.1 Fortran: Module Interface clm2_lsmMod (Source File: clm2_lsmMod.F90)

This module provides the definition of derived data type used to control the operation of CLM. It also provides the entry method for the initialization of CLM-specific variables. The derived data type `clm_struc` includes the variables that specify the runtime options and other control variables as described below:

clm_rfile name of the CLM restart file

clm_vfile name of the CLM vegetation parameter lookup table

clm_chtfile name of the CLM canopy heights lookup table

count variable to keep track of the number of timesteps before an output

clmopen variable to keep track of opened files

numout number of output times

clm_ism initial soil moisture for a cold start run

clm_it initial soil temperature for a cold start run

clm_iscv initial snow mass for a cold start run

outInterval output writing interval

rstInterval restart writing interval

clm CLM specific variables

USES:

```
use clm2type

implicit none

PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
public :: clm2_lsm_ini
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
public :: clm2_struc
```

71.1.1 clm2_lsm_ini (Source File: clm2_lsmMod.F90)

INTERFACE:

```
subroutine clm2_lsm_ini()
```

USES:

```
use clm2_varpar
use pft_varcon
use clm2_shr_orb_mod,  only : clm2_shr_orb_params
use LIS_timeMgrMod,   only : LIS_clock, LIS_calendar, &
     LIS_update_timestep, LIS_registerAlarm
use LIS_logMod, only : LIS_logunit, LIS_verify
use LIS_coreMod, only : LIS_rc
use LIS_precisionMod
```

DESCRIPTION:

This routine creates the datatypes and allocates memory for CLM-specific variables. It also invokes the routine to read the runtime specific options for CLM from the configuration file.
The routines invoked are:

readclm2crd (71.3.1) reads the runtime options for CLM

clm_varder_init (71.1.2) set initial values to CLM variables

iniTimeConst (71.2.31) initialize time invariant parameters

71.1.2 clm2_lsm_init (Source File: clm2_lsmMod.F90)

INTERFACE:

```
subroutine clm2_lsm_init(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_precisionMod
use clm2_varpar,   only : nlevsoi, nlevsno, numrad
```

DESCRIPTION:

Initializes clm variables
The arguments are:

n index of the nest

71.1.3 clm2_main (Source File: clm2_main.F90)

INTERFACE:

```
subroutine clm2_main (n)
```

USES:

```
use LIS_precisionMod
use LIS_timeMgrMod, only : LIS_isAlarmRinging
use clm2_lsmMod
use clm2_varpar, only : nlevsoi
use clm2_varcon     , only : doalb, eccen, obliqr, lambm0, mvelpp, &
```

```
      denh2o, denice, hvap, hsub, hfus, istwet
  use LIS_coreMod, only : LIS_rc
  use LIS_timeMgrMod  , only : LIS_get_curr_calday, LIS_get_nstep
#if (defined RTM)
  use RtmMod        , only : Rtmriverflux
#endif

#if (defined COUP_CSM)
  use clm_csmMod    , only : csm_dosndrcv, csm_recv, csm_send, csm_flxave, &
                             dorecv, dosend, csmstop_now
#endif
  use clm2_shr_sys_mod   , only : clm2_shr_sys_flush
  use LIS_histDataMod
  use LIS_constantsMod,  only : LIS_CONST_RHOFW

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This is the entry point for calling the CLM physics. This routine calls various physics routines for CLM that performs the land surface computations, to solve for water and energy equations. For a more detailed documentation of CLM, please contact the NCAR repository
Calling sequence:

```
  -> loop over patch points calling for each patch point:
     -> Hydrology1          canopy interception and precip on ground
     -> Biogeophysics1      leaf temperature and surface fluxes
     -> Biogeophysics_Lake  lake temperature and surface fluxes
     -> Biogeophysics2      soil/snow and ground temp and update surface fluxes
     -> Hydrology2          surface and soil hydrology
     -> Hydrology_Lake      lake hydrology
     -> Biogeochemistry     surface biogeochemical fluxes (LSM)
     -> EcosystemDyn:       ecosystem dynamics: phenology, vegetation,
                            soil carbon
     -> SurfaceAlbedo:      albedos for next time step
       -> SnowAlbedo:       snow albedos: direct beam
       -> SnowAlbedo:       snow albedos: diffuse
       -> SoilAlbedo:       soil/lake albedos
       -> TwoStream:        absorbed, reflected, transmitted
                            solar fluxes (vis dir)
       -> TwoStream:        absorbed, reflected, transmitted
                            solar fluxes (vis dif)
       -> TwoStream:        absorbed, reflected, transmitted
                            solar fluxes (nir dir)
       -> TwoStream:        absorbed, reflected, transmitted
                            solar fluxes (nir dif)
     -> BalanceCheck        check for errors in energy and water balances
```

The arguments are:

n index of the nest

71.1.4 clm2_readrestart (Source File: clm2_readrestart.F90)

REVISION HISTORY:

```
20 Jan 2003; Sujay Kumar Initial Specification
```

INTERFACE:

```
subroutine clm2_readrestart()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_logMod, only : LIS_logunit, LIS_endrun
use LIS_historyMod, only : LIS_readvar_restart
use clm2_varpar,     only : nlevsno, nlevsoi
use clm2_lsmMod
```

DESCRIPTION:

This program reads restart files for CLM. This includes all relevant water/energy storages and tile information.
The routines invoked are:

drv_readvar_restart (3.17.6) reads a variable from the restart file

---

71.1.5 clm2_setup (Source File: clm2_setup.F90)

REVISION HISTORY:

```
20 Jan 2003  Sujay Kumar Initial Specification
```

INTERFACE:

```
subroutine clm2_setup()
```

USES:

```
use LIS_coreMod, only: LIS_rc
use LIS_timeMgrMod, only : LIS_get_nstep
use LIS_logMod, only :LIS_logunit
use clm2_lsmMod
use clm2_varcon, only: eccen, obliqr, lambm0 , mvelpp
```

DESCRIPTION:

This routine is the entry point to set up the parameters required for CLM. The method invokes the routine to call time variant parameters.
The routines invoked are:

canhtset (71.0.3) reads the canopy height information

iniTimeVar (71.2.32) initialize time variant parameters

71.1.6 clm2_writerestart (Source File: clm2_writerestart.F90)

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 clm2_writerestart(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_timeMgrMod, only : LIS_isAlarmRinging
use LIS_logMod, only : LIS_logunit
use LIS_fileIOMod, only : LIS_create_output_directory, &
                          LIS_create_restart_filename
use clm2_lsmMod, only : clm2_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This program writes restart files for CLM. This includes all relevant water/energy storage and tile information The routines invoked are:

LIS_create_output_directory (3.14.6) creates a timestamped directory for the restart files

LIS_create_restart_filename (3.14.2) generates a timestamped restart filename

clm_dump_restart (71.1.7) writes the CLM variables into the restart file reads a variable from the restart file

---

71.1.7 clm2_dump_restart (Source File: clm2_writerestart.F90)

INTERFACE:

```
subroutine clm2_dump_restart(n, ftn)
```

USES:

```
use clm2_lsmMod
use clm2_varpar,   only : nlevsoi, nlevsno
use LIS_coreMod,  only : LIS_rc, LIS_masterproc
use LIS_historyMod, only : LIS_writevar_restart

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine gathers the necessary restart variables and performs the actual write statements to create the restart files.
The arguments are:

n index of the nest

ftn unit number for the restart file

The routines invoked are:

LIS_writevar_restart (3.17.3) writes a variable to the restart file

---

71.2 Fortran: Module Interface clm2type (Source File: clm2type.F90)

The code in this file provides a description of the data structure containing the CLM 1-d variables. The variables specified in the data structure include:

nstep time step number

kpatch tile index

itypveg vegetation type

itypwat water type

itypprc precipitation type

isoicol color class for soil albedos

snl number of snow layers

frac_veg_nosno fraction of vegetation not covered by snow

frac_veg_nosno_alb fraction of vegetation not covered by snow

imelt flag for meltiing (=1), freezing (=2), not=0(new)

lakpoi flag for lakpoint (true=lake point)

do_capsnow flag to indicate snow capping (true=do sno caping)

present whether PFT is present in the current patch

lat latitude of the patch

lon longitude of the patch

dtime model timestep

zi interface level below a "z" level (m)

dz layer depth (m)

z layer thickness (m)

bsw Clapp and Hornberger "b"

watsat volumetric soil water at saturation (porosity)

hksat hydraulic conductivity at saturation (mm H2O /s)

sucsat minimum soil suction (mm)

csol heat capacity, soil solids (J/m**3/Kelvin)

tkmg thermal conductivity, soil minerals [W/m-K] (new)

tkdry thermal conductivity, dry soil (W/m/Kelvin)

tksatu thermal conductivity, saturated soil [W/m-K] (new)

rootfr fraction of roots in each soil layer

rootr effective fraction of roots in each layer

begwb water mass at the beginning of the time step

endwb water mass at the end of the time step

forc_t atmospheric temperature (K)

forc_u atmospheric wind speed in east direction (m/s)

forc_v atmospheric wind speed in north direction (m/s)

forc_q atmospheric specific humidity (kg/kg)

forc_hgt atmospheric reference height (m)

forc_hgt_u observational height of wind [m]

forc_hgt_v observational height of temperature [m]

forc_hgt_q observational height of humidity [m] (new)

forc_pbot atmospheric pressure (Pa)

forc_th atmospheric potential temperature (Kelvin)

forc_vp atmospheric vapor pressure (Pa)

forc_rho density (kg/m**3)

forc_lwrad downward infrared (longwave) radiation (W/m**2)

forc_solad direct beam radiation (vis=forc_sols , nir=forc_soll)

forc_solai diffuse radiation (vis=forc_solsd, nir=forc_solld)

forc_ch heat/moisture exchange coefficient

forc_rain rain rate [mm/s]

forc_snow snow rate [mm/s]

rssun sunlit stomatal resistance (s/m)

rssha shaded stomatal resistance (s/m)

psnsun sunlit leaf photosynthesis (umol CO2 /m**2/ s)

psnsha shaded leaf photosynthesis (umol CO2 /m**2/ s)

laisun sunlit leaf area

laisha shaded leaf area

sabg solar radiation absorbed by ground (W/m**2)

sabv solar radiation absorbed by vegetation (W/m**2)

fsa solar radiation absorbed (total) (W/m**2)

taux wind stress: e-w (kg/m/s**2)

tauy wind stress: n-s (kg/m/s**2)

eflx_lwrad_out emitted infrared (longwave) radiation (W/m**2)

eflx_lwrad_net net infrared (longwave) rad (W/m**2) [+ = to atm]

eflx_sh_tot total sensible heat flux (W/m**2) [+ to atm]

eflx_sh_veg sensible heat flux from leaves (W/m**2) [+ to atm]

eflx_sh_grnd sensible heat flux from ground (W/m**2) [+ to atm]

eflx_lh_tot total latent heat flux (W/m8*2) [+ to atm]

eflx_soil_grnd soil heat flux (W/m**2) [+ = into soil]

t_veg vegetation temperature (Kelvin)

t_grnd ground temperature (Kelvin)

t_rad radiative temperature (Kelvin)

t_ref2m 2 m height surface air temperature (Kelvin)

t_soisno soil temperature (Kelvin)

qflx_infl infiltration (mm H2O /s)

qflx_surf surface runoff (mm H2O /s)

qflx_drain sub-surface runoff (mm H2O /s)

qflx_top_soil net water input into soil from top (mm/s)

qflx_evap_soi soil evaporation (mm H2O/s) (+ = to atm)

qflx_evap_veg vegetation evaporation (mm H2O/s) (+ = to atm)

qflx_tran_veg vegetation transpiration (mm H2O/s) (+ = to atm)

qflx_snomelt snow melt (mm H2O /s)

qflx_evap_tot qflx_evap_soi + qflx_evap_veg + qflx_tran_veg

qflx_rain_grnd rain on ground after interception (mm H2O/s) [+]

qflx_evap_grnd ground surface evaporation rate (mm H2O/s) [+]

qflx_dew_grnd ground surface dew formation (mm H2O /s) [+]

qflx_sub_snow sublimation rate from snow pack (mm H2O /s) [+]

qflx_dew_snow surface dew added to snow pack (mm H2O /s) [+]

qflx_snowcap excess precipitation due to snow capping (mm H2O /s) [+]

qflx_qrgwl qflx_surf at glaciers, wetlands, lakes

h2osno snow water (mm H2O)

h2ocan canopy water (mm H2O)

h2osoi_liq liquid water (kg/m2)

h2osoi_ice ice lens (kg/m2)

h2osoi_vol volumetric soil water (0¡=h2osoi_vol¡=watsat) [m3/m3]

snowdp snow height (m)

snowage non dimensional snow age [-]

h2osno_old snow mass for previous time step (kg/m2) (new)

frac_sno fraction of ground covered by snow (0 to 1)

frac_iceold fraction of ice relative to the total water (new)

eff_porosity effective porosity = porosity - vol_ice

parsun average absorbed PAR for sunlit leaves (W/m**2)

albgrd ground albedo (direct)

albgri ground albedo (diffuse)

fabd flux absorbed by veg per unit direct flux

fabi flux absorbed by veg per unit diffuse flux

ftdd down direct flux below veg per unit dir flx

ftid down diffuse flux below veg per unit dir flx

ftii down diffuse flux below veg per unit dif flx

fsun sunlit fraction of canopy

surfalb instantaneous all-wave surface albedo

snoalb instantaneous all-wave snow albedo

hbot canopy bottom (m)

htop canopy top (m)

tlai one-sided leaf area index, no burying by snow

tsai one-sided stem area index, no burying by snow

elai one-sided leaf area index with burying by snow

esai one-sided stem area index with burying by snow

fwet fraction of canopy that is wet (0 to 1)

fdry fraction of foliage that is green and dry [-] (new)

annpsn annual photosynthesis (umol CO2 /m**2)

annpsnpot annual potential photosynthesis (same units)

wf soil water as frac. of whc for top 0.5 m

z0mr ratio of momentum roughness length to canopy top height [-]

z0m momentum roughness length [m]

displar ratio of displacement height to canopy top height [-]

displa displacement height [m]

dleaf leaf dimension [m]

xl pft_varcon leaf/stem orientation index

rhol pft_varcon leaf reflectance : 1=vis, 2=nir

rhos pft_varcon stem reflectance : 1=vis, 2=nir

taul pft_varcon leaf transmittance: 1=vis, 2=nir

taus pft_varcon stem transmittance: 1=vis, 2=nir

qe25 quantum efficiency at 25c (umol co2 / umol photon)

vcmx25 maximum rate of carboxylation at 25c (umol co2/m**2/s)

mp slope for conductance-to-photosynthesis relationship

c3psn photosynthetic pathway: 0. = c4, 1. = c3

totfsa solar absorbed solar radiation [W/m2]

toteflx_lwrad_net net longwave radiation [W/m2]

toteflx_lh_tot total latent heat flux [W/m2]

toteflx_sh_tot total sensible heat flux [W/m2]

toteflx_soil_grnd ground heat flux [W/m2]

toqflx_snomelt snowmelt heat flux [W/m2]

totrain accumulation of rain [mm]

totsnow accumulation of snow [mm]

totqflx_evap total evaporation [mm]

totqflx_surf surface runoff [mm]

totqflx_drain subsurface runoff [mm]

totqflx_ecanop interception evaporation [W/m2]

totqflx_tran_veg Total vegetation transpiration

totqflx_evap_grnd Total ground surface evaporation

totqflx_sub_snow Total sublimation rate from snow pack

acond aerodyamic conductance

soilmtc_prev Total column soil moisture for the prev.timestep

h2osno_prev Total column snow water equivalent for the prev.timestep

USES:

```
use LIS_precisionMod, only : r8
```

71.2.1 clm2_setwrfesmfexport (Source File: clm2_setwrfesmfexport.F90)

LIS CLM data writer: Writes clm output in binary format

REVISION HISTORY:

```
02 Dec 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
subroutine clm2_setwrfesmfexport(n)
```

USES:

```
#if 0
  use LIS_coreMod,     only : LIS_rc, LIS_domain
  use LIS_precisionMod
  use lisWRFGridCompMod, only : liswrf_export
  use LIS_historyMod, only : LIS_tile2grid
  use clm2_lsmMod, only : clm2_struc
  use clm2_varcon, only : denh2o, denice, hvap, hsub, hfus, istwet
  use clm2_varpar, only : nlevsoi
#endif
  implicit none
  integer :: n
#if 0
  integer :: i,j,k,t,m
  real :: temp(LIS_rc%ntiles(n))
  real :: asurft(LIS_rc%ntiles(n))
  real :: snowt(LIS_rc%ntiles(n))
  real :: soilmr(LIS_rc%ntiles(n))
  real :: soilmtc(LIS_rc%ntiles(n))
  real :: soilmoist(LIS_rc%ntiles(n),1:nlevsoi)
  real :: q1(LIS_rc%ntiles(n))
  real :: RCH
#endif
```

71.2.2 clm2_wrf_esmff2t (Source File: clm2_wrf_esmff2t.F90)

INTERFACE:

```
subroutine clm2_wrf_esmff2t(n)
```

USES:

```
use ESMF
use LIS_precisionMod
use clm2_lsmMod
use clm2_varcon  , only : rair, cpair, po2, pco2, tcrit, tfrz
use LIS_coreMod, only : LIS_rc
use LIS_metforcingMod, only : LIS_FORC_State
use LIS_logMod,         only : LIS_verify

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the CLM model tiles, when used in the mode coupled to WRF.
The arguments are:

n index of the nest

71.2.3 clm2_setwrfexport.F90 (Source File: clm2_setwrfexport.F90)

Defines the export states from CLM to WRF in coupled mode

REVISION HISTORY:

```
02 Dec 2003; Sujay Kumar, Initial Version
17 Nov 2008; Sujay Kumar, Modified for the ESMF coupled version
```

INTERFACE:

```
subroutine clm2_setwrfexport(n, expState)
```

USES:

```
use ESMF
use LIS_precisionMod
use LIS_coreMod,    only : LIS_rc, LIS_surface
use LIS_historyMod, only : LIS_patch2tile
use LIS_logMod,     only : LIS_verify
use LISWRFGridCompMod, only : LISWRF_export
use clm2_lsmMod
use clm2_varcon, only : denh2o, denice, hvap, hsub, hfus, istwet
use clm2_varpar, only : nlevsoi
```

71.2.4 clm2_wrf_f2t (Source File: clm2_wrf_f2t.F90)

REVISION HISTORY:

```
15 Oct 2002: Sujay Kumar; Initial Code
```

INTERFACE:

```
subroutine clm2_wrf_f2t(n)
```

USES:

```
use ESMF
use LIS_precisionMod
use LIS_coreMod ,      only : LIS_rc
use LIS_metforcingMod, only : LIS_FORC_State
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use clm2_lsmMod
use clm2_varcon  , only : rair, cpair, po2, pco2, tcrit, tfrz

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the Clm model tiles in a coupled mode to WRF
The arguments are:

n index of the nest

71.2.5 clm2_descale_lst (Source File: clm2_descale_lst.F90)

REVISION HISTORY:

```
1 Apr 2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine clm2_descale_lst(n, LSM_State)
```

USES:

```
use ESMF

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

descales the land surface temperature related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

71.2.6 clm2_getlst (Source File: clm2_getlst.F90)

REVISION HISTORY:

```
1 Apr 2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine clm2_getlst(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use clm2_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the land surface temperature related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

71.2.7 clm2_getsynlstpred (Source File: clm2_getsynlstpred.F90)

REVISION HISTORY:

```
1 Apr 2007: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clm2_getsynlstpred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_surface
use clm2_lsmMod
```

71.2.8 clm2_qclst (Source File: clm2_qclst.F90)

REVISION HISTORY:

```
1 Apr 2007: Sujay Kumar: Initial Specification
```

INTERFACE:

```
subroutine clm2_qclst(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use clm2_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Qc's the LST related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

71.2.9 clm2_scale_lst (Source File: clm2_scale_lst.F90)

REVISION HISTORY:

```
1 Apr 2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine clm2_scale_lst(n, LSM_State)
```

USES:

```
use ESMF

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

scales the land surface temperature related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

71.2.10 clm2_setlst (Source File: clm2_setlst.F90)

REVISION HISTORY:

```
1 Apr 2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine clm2_setlst(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use clm2_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

sets the land surface temperature related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

71.2.11 clm2_descale_snow (Source File: clm2_descale_snow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clm2_descale_snow(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clm2_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

71.2.12 clm2_getsnowvars (Source File: clm2_getsnowvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
06Oct2008: Gabrielle De Lannoy: state variables for scf/a EnKF in CLM2.0
04Nov2008: Gabrielle De Lannoy: reduce state var to h2osno and snowdp
```

INTERFACE:

```
subroutine clm2_getsnowvars(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clm2_lsmMod
use clm2_varcon,    only : denh2o

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
NOW: swe (h2osno) and snod (snowdp) are diagnostics, also dz and z (snow layer thickness, level) are 'diagnostic'! ==¿ NOTE: possibly snow layer temp could be included as prognostic (cfr. KA)
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

71.2.13 clm2_getsnwdpred (Source File: clm2_getsnwdpred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clm2_getsnwdpred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_surface
use clm2_lsmMod
use clm2_varcon,    only : denice, denh2o
```

71.2.14 clm2_getswepred (Source File: clm2_getswepred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clm2_getswepred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_surface
use clm2_lsmMod
use clm2_varcon,    only : denice, denh2o
```

71.2.15 clm2_map_sca (Source File: clm2_map_sca.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar; Updated for the ESMF design
03May2007: K. Arseanult; Added MODIS SCA (MOD10A1) Obs Option
16Feb2009: K. Arsenault; Updated for MODIS SCF Obs
```

INTERFACE:

```
subroutine clm2_map_sca( n, OBS_State, LSM_Incr_State )
```

USES:

```
use ESMF
use LIS_coreMod,  only : LIS_rc
use LIS_logMod,   only : LIS_logunit, LIS_verify
use clm2_varcon,   only  : bdsno, tfrz
use clm2_lsmMod,   only : clm2_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
type(ESMF_State)         :: LSM_Incr_State
```

DESCRIPTION:

This subroutine directly maps the observation state to the corresponding variables in the LSM state for SCF data assimilation.
The arguments are:

n index of the nest

OBS_State ESMF State for observations

LSM_State ESMF State for LSM state variables

71.2.16 di_method_rh04 (Source File: clm2_map_sca.F90)

REVISION HISTORY:

```
16Feb2009: K. Arsenault -- Updated for MODIS SCF Obs
```

INTERFACE:

```
subroutine di_method_rh04 ( num_tiles, obs_scf, lsm_swe, lsm_snod )
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use LIS_logMod,   only : LIS_logunit
use clm2_varcon, only    : bdsno, tfrz

implicit none
```

DESCRIPTION:

This routine is based on the simple direction insertion method of Rodell and Houser (2004).
The arguments are:

num_tiles ?

obs_scf ?

lsm_swe ?

lsm_snod ?

71.2.17 clm2_qc_snowobs (Source File: clm2_qc_snowobs.F90)

REVISION HISTORY:

```
25Feb2008: Sujay Kumar: Initial Specification
```

INTERFACE:

```
subroutine clm2_qc_snowobs(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use LIS_constantsMod, only : LIS_CONST_TKFRZ

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

This subroutine performs any model-based QC of the observation prior to data assimilation. Here the snow observations are flagged when LSM indicates that (1) rain is falling (2) ground is fully or partially covered with snow.
The arguments are:

n index of the nest

OBS_State ESMF state container for observations

71.2.18 clm2_qcsnow (Source File: clm2_qcsnow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
06Oct2008: Gabrielle De Lannoy: limit the updated state to realistic values
04Nov2008: Gabrielle De Lannoy: now only for 2 state vars (total snow water, depth)
```

INTERFACE:

```
subroutine clm2_qcsnow(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clm2_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation - limit outcome to min/max in attribute-file
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

71.2.19 clm2_scale_snow (Source File: clm2_scale_snow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clm2_scale_snow(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clm2_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

71.2.20 clm2_setsnowvars (Source File: clm2_setsnowvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
06Oct2008: Gabrielle De Lannoy: update prognostic
           and directly derived diagnostic vars for sca/f EnKFiltering
04Nov2008: Gabrielle De Lannoy: only for total snow water
```

INTERFACE:

```
subroutine clm2_setsnowvars(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clm2_lsmMod
use clm2_varpar,    only : nlevsno
use clm2_varcon,    only : denh2o, tfrz, bdsno

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

This routine assigns the soil moisture prognostic variables to clm's model space.

71.2.21 clm2_transform_obssca (Source File: clm2_transform_obssca.F90)

REVISION HISTORY:

```
25Jun2006: Sujay Kumar: Initial Specification
03May2007: K. Arseanult -- Added MODIS SCA (MOD10A1) Obs Option
19Feb2009: K. Arsenault -- Updated CLM2 DI Code for SCF; based on G. de Lannoy's code.
```

INTERFACE:

```
subroutine clm2_transform_obssca(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod,  only : LIS_rc
use LIS_logMod,   only : LIS_verify
```

71.2.22 clm2_updatesnowvars (Source File: clm2_updatesnowvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
06Oct2008: Gabrielle De Lannoy: Update the state vars for scf/a DA in CLM
04Nov2008: Gabrielle De Lannoy: only do update for 2 total state vars in CLM
```

INTERFACE:

```
subroutine clm2_updatesnowvars(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clm2_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

LSM_Incr_State ESMF State container for LSM state increments

71.2.23 clm2_descale_soilm (Source File: clm2_descale_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clm2_descale_soilm(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clm2_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

71.2.24 clm2_getsoilm (Source File: clm2_getsoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clm2_getsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clm2_lsmMod
use clm2_varcon,    only : denh2o

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

71.2.25 clm2_getsynsmpred (Source File: clm2_getsynsmpred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clm2_getsynsmpred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_surface
use clm2_lsmMod
use clm2_varcon,    only : denice, denh2o
```

71.2.26 clm2_qcsoilm (Source File: clm2_qcsoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clm2_qcsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clm2_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

71.2.27 clm2_scale_soilm (Source File: clm2_scale_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clm2_scale_soilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clm2_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

71.2.28 clm2_setsoilm (Source File: clm2_setsoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clm2_setsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clm2_lsmMod
use clm2_varcon,    only : denh2o

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

This routine assigns the soil moisture prognostic variables to clm's model space.

71.2.29 clm2_updatesoilm (Source File: clm2_updatesoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clm2_updatesoilm(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clm2_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

This routine assigns the soil moisture prognostic variables to clm's model space.

71.2.30 clm2_write_soilm (Source File: clm2_write_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clm2_write_soilm(ftn, n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc
use LIS_historyMod, only : LIS_writevar_restart
use clm2_lsmMod,     only : clm2_struc
use clm2_varcon,     only : denh2o, denice
use clm2_varpar,     only : nlevsoi

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
integer, intent(in)    :: ftn
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

71.2.31 iniTimeConst (Source File: iniTimeConst.F90)

INTERFACE:

```
subroutine iniTimeConst (n)
```

USES:

```
use LIS_precisionMod
use clm2_lsmMod
use clm2_varpar   , only : nlevsoi, nlevlak, numrad
use clm2_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 clm2_varsur   , only : zlak, dzlak, zsoi, dzsoi, zisoi
use clm2_shr_const_mod, only : SHR_CONST_PI
use LIS_coreMod,   only : LIS_rc, LIS_surface,LIS_domain
use LIS_soilsMod, only  : LIS_soils
use LIS_logMod, only    : LIS_logunit, LIS_endrun

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine initializes the time invariant CLM variables
The arguments are:

n index of the nest

71.2.32 iniTimeVar (Source File: iniTimeVar.F90)

INTERFACE:

```
subroutine iniTimeVar (n, eccen, obliqr, lambm0 , mvelpp)
```

USES:

```
use LIS_precisionMod
use LIS_coreMod, only : LIS_rc
use clm2_lsmMod
use LIS_timeMgrMod, only : LIS_get_curr_calday
use clm2_varpar,   only : nlevsno, nlevsoi
use clm2_varcon  , only : bdsno, istice, istwet, &
     istsoil, denice, denh2o, tfrz, spval, doalb
use clm2_shr_sys_mod , only : clm2_shr_sys_abort
use LIS_logMod, only : LIS_logunit

implicit none
```

ARGUMENTS:

```
integer , intent(in) :: n
real(r8), intent(in) :: eccen
real(r8), intent(in) :: obliqr
real(r8), intent(in) :: lambm0
real(r8), intent(in) :: mvelpp
```

DESCRIPTION:

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.
The arguments are:

n index of the nest

eccen Earth's orbital eccentricity

obliqr Earth's obliquity in radians

lambm0 Mean longitude of perihelion at the vernal equinox (radians)

mvelpp Earth's moving vernal equinox long. of perihelion + pi (radians)

71.3 Fortran: Module Interface clm2_atmdrvMod (Source File: clm2_atmdrvMod.F90)

This module defines the variables and routines to enable the interaction with an atmospheric component

USES:

71.3.1 clm2_atmdrv (Source File: clm2_atmdrvMod.F90)

INTERFACE:

```
subroutine clm2_atmdrv(n)
```

USES:

```
use ESMF
use clm2_lsmMod
use clm2_varcon  , only : rair, cpair, po2, pco2, tcrit, tfrz
use LIS_precisionMod
use LIS_coreMod, only : LIS_rc, LIS_surface
use LIS_FORC_AttributesMod
use LIS_metforcingMod, only : LIS_FORC_State
use LIS_logMod,         only : LIS_verify, LIS_endrun

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the CLM2 model tiles.
The arguments are:

n index of the nest

!ROUTINE : readclm2crd

REVISION HISTORY:

```
14 Oct 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readclm2crd()
```

USES:

```
use ESMF
use LIS_logMod, only : LIS_logunit
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod,  only : LIS_verify
use LIS_timeMgrMod, only : LIS_parseTimeString
use clm2_lsmMod, only : clm2_struc
```

DESCRIPTION:

This routine reads the options specific to CLM from the LIS configuration file.

```
!
! Fill the buffer, first in X then in Y
!
      do J = y1, y2
```

```
        do I = x1, x2
          buffer( I,J ) = in( I, J )
        enddo
      enddo
      return
```

```
!$omp  parallel do&
!$omp& default(shared)&
!$omp& private(i,j,k)
      do K = z1, z2
        do J = y1, y2
          do I = x1, x2
            buffer( I,J,K ) = in( I, J, K )
          enddo
        enddo
      enddo
      return
```

```
!
! Fill the buffer, first in X then in Y
!
   do L = n1, n2

!$omp  parallel do private(i, j, k)

      do k = z1, z2
         do j = y1, y2
            do i = x1, x2
               buffer(i,j,k,L) = in(i,k,L,j)
            enddo
         enddo
      enddo
   enddo
```

```
!
! Fill the buffer, first in X then in Y
!
      do J = y1, y2
        do I = x1, x2
          out( I,J ) = buffer( I,J )
        enddo
      enddo
      return
```

71.3.2 bufferunpack3d — Pack a ghost region into a buffer

INTERFACE:

```
subroutine bufferunpack3d( buffer, x1, x2, y1, y2,                &
                           z1, z2, xfrom, xto, yfrom,             &
                           yto, zfrom, zto, Out)
```

USES:

```
use LIS_precisionMod
implicit none
```

INPUT PARAMETERS:

```
integer , intent( in )  :: x1,x2                     ! X limits
integer , intent( in )  :: y1,y2                     ! Y limits
integer , intent( in )  :: z1,z2                     ! Z limits
real(r8), intent( in )  :: buffer(x1:x2,y1:y2,z1:z2) ! Packed buffer
integer , intent( in )  :: xfrom,xto                 ! X dims
integer , intent( in )  :: yfrom,yto                 ! Y dim
integer , intent( in )  :: zfrom,zto                 ! Z dim
```

INPUT/OUTPUT PARAMETERS:

```
real(r8), intent( inout )  :: out(xfrom:xto,yfrom:yto,zfrom:zto) ! Out
```

DESCRIPTION:

This routine puts a 3-D ghost region at the end of a buffer, first in X then in Y then in Z.

LOCAL VARIABLES:

```
integer  I, J, K
```

REVISION HISTORY:

```
99.09.30   Sawyer     Creation
99.10.18   Sawyer     FVCCM3 format (no capitalization)
00.07.08   Sawyer     Use precision module
01.02.02   Sawyer     Removed SGI directives. OpenMP only; free format
```

```
!$omp  parallel do&
!$omp& default(shared)&
!$omp& private(i,j,k)
      do K = z1, z2
        do J = y1, y2
          do I = x1, x2
            out( I,J,K ) = buffer( I, J, K )
          enddo
        enddo
      enddo
      return
```

71.3.3 bufferunpack4d — Pack a ghost region into a buffer

INTERFACE:

```
subroutine bufferunpack4d( buffer, x1, x2, y1, y2,                &
                           z1, z2, n1, n2, xfrom,                 &
                           xto, yfrom, yto, zfrom, zto,           &
                           Nfrom, Nto, Out )
```

USES:

```
use LIS_precisionMod
implicit none
```

INPUT PARAMETERS:

```
integer , intent( in )  :: x1,x2                     ! X limits
integer , intent( in )  :: y1,y2                     ! Y limits
integer , intent( in )  :: z1,z2                     ! Z limits
integer , intent( in )  :: n1,n2                     ! N limits
real(r8), intent( in )  :: buffer(x1:x2,y1:y2,z1:z2,n1:n2)
integer , intent( in )  :: xfrom,xto                 ! X dims
integer , intent( in )  :: yfrom,yto                 ! Y dim
integer , intent( in )  :: zfrom,zto                 ! Z dim
integer , intent( in )  :: Nfrom,Nto                 ! N dim
```

INPUT/OUTPUT PARAMETERS:

```
real(r8),intent(inout)  :: out(xfrom:xto,yfrom:yto,               &
                               zfrom:zto,nfrom:nto)
```

DESCRIPTION:

This routine puts a 4-D ghost region at the end of a buffer, packed in the order of the indices.

LOCAL VARIABLES:

```
integer  I, J, K, L
```

REVISION HISTORY:

```
99.09.30   Sawyer     Creation
99.10.18   Sawyer     FVCCM3 format (no capitalization)
00.07.08   Sawyer     Use precision module
01.02.02   Sawyer     Removed SGI directives. OpenMP only; free format
```

```
!
! Fill the buffer, first in X then in Y
!
      do L = n1, n2
!$omp  parallel do&
!$omp& default(shared)&
!$omp& private(i,j,k)
        do K = z1, z2
```

```
      do J = y1, y2
        do I = x1, x2
          out( I,J,K,L ) = buffer( I,J,K,L )
        enddo
      enddo
    enddo
  enddo
  return
```

71.4 Fortran: Module Interface debugutilitiesmodule (Source File: debugutilitiesmodule.F90)

USES:

```
      IMPLICIT NONE

#define  MAX_STACK_LEVEL 20
#define  MAX_STRING_LEN  40
```

PUBLIC MEMBER FUNCTIONS:

```
      PUBLIC     DumAssert, DumEnter, DumLeave
```

DESCRIPTION:

This module provides the basic utilities to support debugging

DumAssert	Make an assertion
DumEnter	Tracing: enter a subroutine
DumLeave	Tracing: leave a subroutine

The DumAssert makes an assertion (i.e., claims that a boolean argument is true) for a given line of code in a given source file. DumEnter and DumLeave to be used as a pair and placed at the beginning and end of routines to be traced.
It is not intended for the user to make use of these routines directly but rather in conjunction with the CPP macros defined in the "Debug.h" file in the INCLUDE directory. The CPP macros define the calls to the three above-mention routines if the -DDEBUG_ON option is set on the compile line. The line #include "Debug.h" statement in any routine which makes use of these facilities. In production compilations where DEBUG_ON is not set, "Debug.h" defines blank lines and thus does not affect code performance. The CPP definition of DEBUG_LEVEL in the compile line, e.g., -DDEBUG_LEVEL=2, denotes the level of debugging performed. A higher level performs all the debugging at the lower levels and then some.
Note that, unlike other include statements, "Debug.h" must be included *before* the IMPLICIT NONE statement (since "Debug.h" contains a USE DebugModule statement.
Compile options used: `MPI_VER`, `DEBUG_LEVEL`

LOCAL VARIABLES:

```
      CHARACTER(len=MAX_STRING_LEN) :: TraceStack( MAX_STACK_LEVEL )
      INTEGER    :: StackLevel = 0
```

REVISION HISTORY:

```
    97.09.30   Sawyer     Creation
    98.03.09   Sawyer     Added documentation for walkthrough
    01.02.12   Sawyer     Converted to free format
```

BUGS:

71.4.1 DumAssert — Raise Assertion

INTERFACE:

```
      SUBROUTINE DumAssert ( Condition, FileName, Linenumber )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      LOGICAL, INTENT(IN)       :: Condition      ! Condition asserted
      CHARACTER(*), INTENT(IN)  :: FileName       ! Source file
      INTEGER, INTENT( IN )     :: LineNumber     ! Source line
```

DESCRIPTION:

Condition is claimed by the calling Routine in Filename at Linenumber to be true. If it is, do nothing. If not, print as much information as possible.

Debug Level	Action
0	Return immediately
1	Print assertion failed
2	Print assertion failed and trace stack

LOCAL VARIABLES:

```
      INTEGER I, MyID, Ierror
```

SYSTEM ROUTINES:

REVISION HISTORY:

```
     97.09.30   Sawyer     Creation


#if !defined(DEBUG_LEVEL)
#define DEBUG_LEVEL 1
#endif

#if ( DEBUG_LEVEL > 0 )
      IF (.NOT. Condition) THEN
        print *, 'Assertion failed:',                                   &
     &            ' source file: ', FileName,                           &
     &            ' source line: ', LineNumber
!
!  Check if trace available
!
#if ( DEBUG_LEVEL > 1 )
```

```
        PRINT *, "Printing Trace: "
        DO I = 1, StackLevel
          PRINT *, "Level ", StackLevel,
     &             " Called ", TraceStack( StackLevel )
        ENDDO
#endif
      ENDIF
#endif
      RETURN
```

71.4.2 DumEnter — Tracing: Enter a Subroutine

INTERFACE:

```
      SUBROUTINE DumEnter ( RoutineName )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      CHARACTER(*), INTENT(IN)     :: RoutineName    ! Source file
```

DESCRIPTION:

This routine marks the beginning of a region to be traced, usually a subroutine.

Debug Level	Action
0	Return immediately
1	Perform bookkeeping
2	Perform bookkeeping, print trace

LOCAL VARIABLES:

```
      INTEGER MyID, Ierror
```

REVISION HISTORY:

```
     97.09.30   Sawyer     Creation
```

```
#if !defined(DEBUG_LEVEL)
#define DEBUG_LEVEL 1
#endif

#if ( DEBUG_LEVEL > 0 )
      StackLevel = StackLevel + 1
      IF ( StackLevel .GT. MAX_STACK_LEVEL ) THEN
        PRINT *, "StackLevel overflow: ", StackLevel, " Stopping"
        STOP
      ENDIF
      TraceStack( StackLevel ) = RoutineName
#if ( DEBUG_LEVEL > 1 )
      PRINT *, "Level ", StackLevel, " Entering ", RoutineName
```

```
#endif
#endif
      RETURN
```

71.4.3 DumLeave — Tracing: Leave a Subroutine

INTERFACE:

```
      SUBROUTINE DumLeave ( RoutineName )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      CHARACTER(*), INTENT(IN)     :: RoutineName    ! Source file
```

DESCRIPTION:

Tracing facility: leave a subroutine, remove the history trail. Depending on the debugging level, do nothing (0), update the stack only (1), or update stack and print trace message (2) to stdout. The CALL to Leave should be placed just before every egress of the subroutine (hopefully the exit point is unique).

Debug Level	Action
0	Return immediately
1	Perform bookkeeping, consistency check
2	Bookkeeping, consistency, print trace

LOCAL VARIABLES:

```
      INTEGER MyID, Ierror
```

REVISION HISTORY:

```
     97.09.30   Sawyer     Creation
```

```
#if !defined(DEBUG_LEVEL)
#define DEBUG_LEVEL 1
#endif

#if ( DEBUG_LEVEL > 0 )
!
!  Make sure that the Enter and Leave correspond
!
      IF ( TraceStack(StackLevel) .NE. RoutineName ) THEN
        PRINT *, "Expected: ", TraceStack(StackLevel),                  &
     &           "Got: ", RoutineName, " STOPPING "
        STOP
      ENDIF
#if ( DEBUG_LEVEL > 1 )
      PRINT *, "Level ", StackLevel, " Leaving ", RoutineName
#endif
      IF ( StackLevel .LE. 0 ) THEN
```

```
        PRINT *, "StackLevel underflow: ", StackLevel, " Stopping"
        STOP
      ENDIF
      TraceStack( StackLevel ) = ""
      StackLevel = StackLevel - 1
#endif
      RETURN
```

71.5 Fortran: Module Interface decompmodule (Source File: decompmodule.F90)

USES:

```
 #include "debug.h"

       IMPLICIT NONE
```

DESCRIPTION:

This module provides the DecompType and its create and destroy routines.

DecompType	Type to describe a decomposition
DecompFree	Destroy a decomposition
DecompCopy	Copy decomposition to newly created one
DecompPermute	Permute decomposition
DecompRegular1D	Create a 1-D decomposition
DecompRegular2D	Create a 2-D decomposition
DecompRegular3D	Create a 3-D decomposition
DecompCreateIrr	Create an irregular 1-D decomposition
DecompCreateTags	Create a decomposition from Pe and Tags
DecompGlobalToLocal	Map a global index to a local one
DecompLocalToGlobal	Map a local index to a global one

The decomposition type contains the sizes of the global array, the number of entries on each PE, and for each PE a list of "runs", i.e., the starting and finishing global indices or "tags" whose inclusive array section resides on that PE. Clearly this method of decomposition is only efficient if there are long runs, i.e., long array sections which are mapped to one PE. A random decomposition will cause poor results.
The decomposition is thus very efficient for 1-D, 2-D or 3-D block distributions (particularly for 1-D distributions, where there is one "run" per processor). Problems may occur for an irregular decomposition (which is by definition 1-D). If there is little correspondence between the global indices of the entries and the actual decomposition (e.g., the tags are assigned randomly), then there will be many runs, most containing only one tag, and the resulting instance of DecompType will be very large. Fortunately, most applications assign tags to entries in some sort of contiguous fashion, which is then quite appropriate for this data structure.
All numbering of multi-dimensional arrays is ROW-MAJOR, that is, first in the X direction and then in the Y (and then, if appropriate, in Z). This is true for both the 2-D and 3-D data sets as also the Cartesian description of the PEs.
There is one glaring feature of DecompType. It is supposed to be a 'one-size-fits-all' description of the decomposition (with the exception of the random indexing mentioned above). Unfortunately, to describe 2-D and 3-D regions, it is necessary to carry additional dimension information in order have complete information for the mapping. This means that 2-D and 3-D decompositions inherently carry more information than a 1-D decomposition. Thus it *is* possible to use a decomposition created with the Regular2D or Regular3D

routines to describe the corresponding decomposition when the 2-D or 3-D array is viewed as a 1-D array, but it is clearly *not* possible to use a decomposition created with Regular1D to describe the decomposition of a 2-D or 3-D array — the appropriate information just is not there.

REVISION HISTORY:

```
97.07.22   Sawyer     Creation
97.09.01   Sawyer     Release date
97.11.06   Sawyer     Addition of row and column communicators
97.01.24   Sawyer     Added support for non-MPI derived types solution
97.01.29   Sawyer     Minor revisions for production service
98.01.30   Sawyer     Added DecompCopy
98.02.04   Sawyer     Removed Comm, CommRow and CommCol from DecompType
98.03.13   Sawyer     Removed DecompTypeOld, brushed up for walkthrough
98.03.19   Sawyer     Minor corrections after walkthrough
98.05.02   Sawyer     Added DecompPermute
98.05.11   Sawyer     Removed Permutation from all but DecompPermute
99.01.19   Sawyer     Minor cleaning
00.07.07   Sawyer     Removed DimSizes; decomp is now 1D only
00.11.12   Sawyer     Added DecompCreateTags and DecompInfo
01.02.03   Sawyer     Updated for free format; corrected DecompCreateTags
01.03.20   Sawyer     Added DecompRegular3DOrder
```

PUBLIC TYPES:

```
      PUBLIC DecompType, DecompCreate, DecompFree
      PUBLIC DecompCopy, DecompPermute
      PUBLIC DecompGlobalToLocal, DecompLocalToGlobal, DecompInfo
  Currently not yet tested:
      PUBLIC DecompRegular3DOrder

      INTERFACE     DecompCreate
        MODULE PROCEDURE DecompRegular1D
        MODULE PROCEDURE DecompRegular2D
        MODULE PROCEDURE DecompRegular3D
        MODULE PROCEDURE DecompCreateIrr
        MODULE PROCEDURE DecompCreateTags
      END INTERFACE

  Decomposition info

       TYPE Lists
         INTEGER, POINTER     :: StartTags(:) ! Start of tag run
         INTEGER, POINTER     :: EndTags(:)   ! Start of tag run
       END TYPE Lists

       TYPE DecompType
         INTEGER              :: GlobalSize   ! Size in each dimension
         INTEGER, POINTER     :: NumEntries(:)! Number of entries per PE
         TYPE(Lists), POINTER :: Head(:)      ! Array of pointers
       END TYPE DecompType
```

71.5.1 DecompFree — Free a decomposition

INTERFACE:

```
      SUBROUTINE DecompFree ( Decomp )
```

USES:

```
      IMPLICIT NONE
```

INPUT/OUTPUT PARAMETERS:

```
      TYPE(DecompType), INTENT( INOUT ):: Decomp  ! Decomp information
```

DESCRIPTION:

Free the decomposition – deallocate the data structures.

SYSTEM ROUTINES:

```
      ASSOCIATED, DEALLOCATE
```

REVISION HISTORY:

```
     98.01.30   Sawyer     Creation

!
! !LOCAL VARIABLES:
      INTEGER  :: I, NPEs
!
      CPP_ENTER_PROCEDURE( "DECOMPFREE" )

      IF ( ASSOCIATED( Decomp%NumEntries ) )                            &
     &               DEALLOCATE( Decomp%NumEntries )
      IF ( ASSOCIATED( Decomp%Head ) ) THEN
        NPEs = SIZE( Decomp%Head )
        DO I = 1, NPEs
!
! Copy the number of entries on each PE
!
          IF ( ASSOCIATED( Decomp%Head(I)%StartTags ) )                 &
     &               DEALLOCATE( Decomp%Head(I)%StartTags )
          IF ( ASSOCIATED( Decomp%Head(I)%EndTags ) )                   &
     &               DEALLOCATE( Decomp%Head(I)%EndTags )
        ENDDO
        DEALLOCATE( Decomp%Head )
      ENDIF

      CPP_LEAVE_PROCEDURE( "DECOMPFREE" )
      RETURN
```

71.5.2 DecompCopy — Copy one decomposition to another

INTERFACE:

```
      SUBROUTINE DecompCopy ( DecompIn, DecompOut )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      TYPE(DecompType), INTENT( IN )   :: DecompIn  ! Decomp information
```

OUTPUT PARAMETERS:

```
      TYPE(DecompType), INTENT( OUT )  :: DecompOut ! Decomp information
```

DESCRIPTION:

Creates an output decomposition and copies the DecompIn input values

SYSTEM ROUTINES:

```
      ALLOCATE
```

REVISION HISTORY:

```
     98.01.30   Sawyer     Creation
```

```
!
! !LOCAL VARIABLES:
      INTEGER  :: I, J, NDims, NPEs, NRuns
!
      CPP_ENTER_PROCEDURE( "DECOMPCOPY" )
!
! Copy the size of the global array
!
      DecompOut%GlobalSize = DecompIn%GlobalSize

!
! Allocate the number of entries and list head arrays
!
      NPEs = SIZE( DecompIn%NumEntries )
      CPP_ASSERT_F90( SIZE( DecompIn%Head ) .EQ. NPEs )
      ALLOCATE( DecompOut%NumEntries( NPEs ) )
      ALLOCATE( DecompOut%Head( NPEs ) )

      DO I = 1, NPEs
!
! Copy the number of entries on each PE
!
        DecompOut%NumEntries( I ) = DecompIn%NumEntries( I )
        NRuns = SIZE( DecompIn%Head( I )%StartTags )
        CPP_ASSERT_F90( SIZE( DecompIn%Head( I )%EndTags ) .EQ. NRuns )
!
! Allocate and copy the array of runs
!
        ALLOCATE( DecompOut%Head(I)%StartTags( NRuns ) )
        ALLOCATE( DecompOut%Head(I)%EndTags( NRuns ) )
        DO J = 1, NRuns
          DecompOut%Head(I)%StartTags(J) = DecompIn%Head(I)%StartTags(J)
          DecompOut%Head(I)%EndTags(J) = DecompIn%Head(I)%EndTags(J)
        ENDDO
      ENDDO

      CPP_LEAVE_PROCEDURE( "DECOMPCOPY" )
      RETURN
```

71.5.3 DecompPermute — Permute one decomposition to another

INTERFACE:

```
SUBROUTINE DecompPermute ( Permutation, Decomp )
```

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
INTEGER :: Permutation( : )                  ! Permutation
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(DecompType), INTENT( INOUT ) :: Decomp  ! Decomp information
```

DESCRIPTION:

Permutes the PE assignment of a given decomposition. Confusion will always arise about whether this is a forward or backward transformation. Picture it this way: draw the array and slice it up as indicated by the distribution. The resulting boxes are of course indexed by natural numbering 1, 2, 3, 4, ... (these are the virtual one-based PEs). Now write the true PE numbering (one-based) as you would like it. The resulting array is Perm.

SYSTEM ROUTINES:

```
ALLOCATE
```

REVISION HISTORY:

```
     98.05.02   Sawyer     Creation

!
! !LOCAL VARIABLES:
      INTEGER, POINTER     :: NumEntries(:)! Number of entries
      TYPE(Lists), POINTER :: Head(:)      ! Array of pointers
      INTEGER              :: I, J, NPEs, NRuns, TruePE
!
      CPP_ENTER_PROCEDURE( "DECOMPPERMUTE" )
!
! Allocate the number of entries and list head arrays
!
      NPEs = SIZE( Decomp%NumEntries )
      ALLOCATE( NumEntries( NPEs ) )
      DO I = 1, NPEs
        TruePE = Permutation( I )
        NumEntries( TruePE ) = Decomp%NumEntries( I )
      ENDDO
!
! Deallocate old NumEntries and put the new pointer in its place
!
      DEALLOCATE( Decomp%NumEntries )
      Decomp%NumEntries => NumEntries
```

```
      NULLIFY( NumEntries )

!
! Allocate and set the permuted Lists called with pointer Head
!
      ALLOCATE( Head( NPEs ) )
      DO I = 1, NPEs
        TruePE = Permutation( I )
        NRuns = SIZE( Decomp%Head(I)%StartTags )
        CPP_ASSERT_F90( SIZE( Decomp%Head(I)%EndTags ) .EQ. NRuns )
!
! Allocate and permute the array of runs
!
        ALLOCATE( Head(TruePE)%StartTags(NRuns) )
        ALLOCATE( Head(TruePE)%EndTags(NRuns) )
        DO J = 1, NRuns
          Head(TruePE)%StartTags(J) = Decomp%Head(I)%StartTags(J)
          Head(TruePE)%EndTags(J)   = Decomp%Head(I)%EndTags(J)
        ENDDO
      ENDDO
!
! Deallocate the arrays of starting and ending tags
!
      DO I = 1, NPEs
        DEALLOCATE( Decomp%Head(I)%StartTags )
        DEALLOCATE( Decomp%Head(I)%EndTags )
      ENDDO
!
! Deallocate the heads to the Lists
!
      DEALLOCATE( Decomp%Head )

!
! Link the new head to that in the decomposition
!
      Decomp%Head => Head

      NULLIFY( Head )

      CPP_LEAVE_PROCEDURE( "DECOMPPERMUTE" )
      RETURN
```

71.5.4 DecompRegular1D — Create a decomposition for a 1-D grid

INTERFACE:

```
      SUBROUTINE DecompRegular1D ( NPEs, Dist, Decomp )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      INTEGER, INTENT( IN )            :: NPEs     ! Number of PEs
      INTEGER, INTENT( IN )            :: Dist(:)  ! Distribution in X
```

OUTPUT PARAMETERS:

```
      TYPE(DecompType), INTENT( OUT )  :: Decomp   ! Decomp information
```

DESCRIPTION:

Creates a variable block decomposition for a regular 1-D grid (this is also known as a "block-general" distribution). The decomposition is given through the Dist distribution which contains the number of entries on each PE.

SYSTEM ROUTINES:

```
      ALLOCATE
```

REVISION HISTORY:

```
     98.01.19   Sawyer     Creation
     98.01.22   Sawyer     Corrections, TESTED
     98.05.11   Sawyer     Removed Perm from arglist -- see DecompPermute
     00.07.07   Sawyer     Removed use of DimSizes(:) array
```

```
!
! !LOCAL VARIABLES:
      INTEGER  :: I, Counter
!
      CPP_ENTER_PROCEDURE( "DECOMPREGULAR1D" )
!
      CPP_ASSERT_F90( NPEs .EQ. SIZE( Dist ) )
!
! The head contains NPEs pointers to the tag lists.
!
      Decomp%GlobalSize = SUM(Dist)
      ALLOCATE( Decomp%NumEntries( NPEs ) )
      ALLOCATE( Decomp%Head( NPEs ) )
      Counter = 0
      DO I = 1, NPEs
        Decomp%NumEntries(I) = Dist(I)
!
! Since this is a regular distribution there is only one run of tags per PE.
!
        NULLIFY( Decomp%Head(I)%StartTags )
        NULLIFY( Decomp%Head(I)%EndTags )
        ALLOCATE( Decomp%Head(I)%StartTags(1) )
        ALLOCATE( Decomp%Head(I)%EndTags(1) )
!
! The starting and ending tags are immediately determined from
! the decomposition arrays
!
        Decomp%Head(I)%StartTags(1) = Counter+1
        Counter = Counter + Dist(I)
        Decomp%Head(I)%EndTags(1) = Counter
      ENDDO

      CPP_LEAVE_PROCEDURE( "DECOMPREGULAR1D" )
      RETURN
```

71.5.5 DecompRegular2D — Create a decomposition for a 2-D grid

INTERFACE:

```
SUBROUTINE DecompRegular2D( NPEsX, NPEsY, Xdist, Ydist, Decomp )
```

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
INTEGER, INTENT( IN )            :: NPEsX    ! Number of PEs in X
INTEGER, INTENT( IN )            :: NPEsY    ! Number of PEs in Y
INTEGER, INTENT( IN )            :: Xdist(:) ! Distribution in X
INTEGER, INTENT( IN )            :: Ydist(:) ! Distribution in Y
```

OUTPUT PARAMETERS:

```
TYPE(DecompType), INTENT( OUT )  :: Decomp  ! Decomp information
```

DESCRIPTION:

Creates a variable block-block decomposition for a regular 2-D grid. The decomposition is given through the Xdist and Ydist distributions, which contain the number of entries on each PE in that dimension. This routine thus defines a rectangular "checkerboard" distribution.

SYSTEM ROUTINES:

```
ALLOCATE
```

REVISION HISTORY:

```
98.01.19   Sawyer     Creation
98.01.22   Sawyer     Corrections, TESTED
98.05.11   Sawyer     Removed Perm from arglist -- see DecompPermute
00.07.07   Sawyer     Removed use of DimSizes(:) array
```

BUGS:

```
This routine makes the assumption that the sum of the
distribution in each dimension adds up to the total
number of entries in that dimension.  It will cause
problems if the actual local arrays are over- or
under-allocated.  For example, if the local array is
allocated statically for the maximum size of the
array on any processor, problems will occur on those
PEs which have less than the maximum.
```

```
!
! !LOCAL VARIABLES:
      INTEGER :: TruePE, I, J, K, Counter1, Counter2, SizeX, SizeY
!
      CPP_ENTER_PROCEDURE( "DECOMPREGULAR2D" )
!
! Some sanity checks
!
      CPP_ASSERT_F90( NPEsX .EQ. SIZE( Xdist ) )
      CPP_ASSERT_F90( NPEsY .EQ. SIZE( Ydist ) )
```

```
!
! The head contains NPEs pointers to the tag lists.
!
      SizeX = SUM(Xdist)
      SizeY = SUM(Ydist)
      Decomp%GlobalSize = SizeX * SizeY
      ALLOCATE( Decomp%NumEntries( NPEsX*NPEsY ) )
      ALLOCATE( Decomp%Head( NPEsX*NPEsY ) )
      Counter1 = 0
      DO J = 1, NPEsY
        DO I = 1, NPEsX
!
! WARNING!!!!  The definition of the PE is Row-major ordering
!
          TruePE = ( J-1 ) * NPEsX + I

!
! The number of entries is the product of the local X, Y, Z allotment
!
          Decomp%NumEntries(TruePE) = Xdist(I)*Ydist(J)
!
! For each Y there is a separate run
!
          NULLIFY( Decomp%Head(TruePE)%StartTags )
          NULLIFY( Decomp%Head(TruePE)%EndTags )
          ALLOCATE( Decomp%Head(TruePE)%StartTags(Ydist(J)) )
          ALLOCATE( Decomp%Head(TruePE)%EndTags(Ydist(J)) )
          Counter2 = Counter1
          DO K = 1, Ydist(J)
!
! Since this is a regular distribution the definition of
! tags is dictated by Xdist(I), and appears Ydist(J) times
!
!
            Decomp%Head(TruePE)%StartTags(K) = Counter2 + 1
            Decomp%Head(TruePE)%EndTags(K) = Counter2 + Xdist(I)
            Counter2 = Counter2 + SizeX
          ENDDO
          Counter1 = Counter1 + Xdist(I)
        ENDDO
!
! Align the counter such that it points to the start of the next
! block.  (Ydist(J)-1) since already one layer has been added in.
! Implicit assumption that SizeX = SUM( Xdist )
!
        Counter1 = Counter1 + SizeX*(Ydist(J)-1)
      ENDDO

      CPP_LEAVE_PROCEDURE( "DECOMPREGULAR2D" )
      RETURN
```

71.5.6 DecompRegular3D — Create a decomposition for a 3-D grid

INTERFACE:

```
      SUBROUTINE DecompRegular3D ( NPEsX, NPEsY, NPEsZ,                  &
     &                             Xdist, Ydist, Zdist, Decomp )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      INTEGER, INTENT( IN )            :: NPEsX    ! Number of PEs in X
      INTEGER, INTENT( IN )            :: NPEsY    ! Number of PEs in Y
      INTEGER, INTENT( IN )            :: NPEsZ    ! Number of PEs in Z
      INTEGER, INTENT( IN )            :: Xdist(:) ! Distribution in X
      INTEGER, INTENT( IN )            :: Ydist(:) ! Distribution in Y
      INTEGER, INTENT( IN )            :: Zdist(:) ! Distribution in Z
```

OUTPUT PARAMETERS:

```
      TYPE(DecompType), INTENT( OUT )  :: Decomp  ! Decomp information
```

DESCRIPTION:

Creates a decomposition for a regular 3-D grid. The decomposition is given through the Xdist, Ydist, and Zdist distributions, which contain the number of entries on each PE in that dimension. This routine thus defines a parallelopiped (SOMA-block) distribution.

SYSTEM ROUTINES:

```
      ALLOCATE
```

REVISION HISTORY:

```
     98.01.19   Sawyer     Creation
     98.05.11   Sawyer     Removed Perm from arglist -- see DecompPermute
     00.07.07   Sawyer     Removed use of Sizes(:) array
```

BUGS:

```
      This routine makes the assumption that the sum of the
      distribution in each dimension adds up to the total
      number of entries in that dimension.  It will cause
      problems if the actual local arrays are over- or
      under-allocated.  For example, if the local array is
      allocated statically for the maximum size of the
      array on any processor, problems will occur on those
      PEs which have less than the maximum.
      Currently untested (98.05.11)

!
! !LOCAL VARIABLES:
      INTEGER  :: TruePE, Counter1, Counter2, Counter3
      INTEGER  :: I, J, K, L, M, N, SizeX, SizeY, SizeZ
!
      CPP_ENTER_PROCEDURE( "DECOMPREGULAR3D" )
!
```

```
! Some sanity checks
!
!
      CPP_ASSERT_F90( NPEsX .EQ. SIZE( Xdist ) )
      CPP_ASSERT_F90( NPEsY .EQ. SIZE( Ydist ) )
      CPP_ASSERT_F90( NPEsZ .EQ. SIZE( Zdist ) )
      CPP_ASSERT_F90( .NOT. ASSOCIATED( Decomp%Head ) )
!
! The head contains NPEs pointers to the tag lists.
!
      SizeX = SUM(Xdist)
      SizeY = SUM(Ydist)
      SizeZ = SUM(Zdist)
      Decomp%GlobalSize = SizeX * SizeY * SizeZ
      ALLOCATE( Decomp%NumEntries( NPEsX*NPEsY*NPEsZ ) )
      ALLOCATE( Decomp%Head( NPEsX*NPEsY*NPEsZ ) )
      Counter1 = 0
      DO K = 1, NPEsZ
        DO J = 1, NPEsY
          DO I = 1, NPEsX
!
! WARNING!!!!  The definition of the PE is Row-major ordering
!
            TruePE = (K-1)*NPEsX*NPEsY + (J-1)*NPEsX + I
            NULLIFY( Decomp%Head(TruePE)%StartTags )
            NULLIFY( Decomp%Head(TruePE)%EndTags )
!
! The number of entries is the product of the local X, Y, Z allotment
!
            Decomp%NumEntries(TruePE) = Xdist(I)*Ydist(J)*Zdist(K)
!
! For each Z there are Y separate runs
!
            ALLOCATE( Decomp%Head(TruePE)%StartTags(Ydist(J)*Zdist(K)) )
            ALLOCATE( Decomp%Head(TruePE)%EndTags(Ydist(J)*Zdist(K)) )
            Counter2 = Counter1
            L = 0
            DO N = 1, Zdist(K)
              Counter3 = Counter2
              DO M = 1, Ydist(J)
!
!     Since this is a regular distribution the definition of
!     tags is dictated by Xdist(I), and appears Ydist(J) times
!
!
                L = L + 1
                Decomp%Head(TruePE)%StartTags(L) = Counter3 + 1
                Decomp%Head(TruePE)%EndTags(L) = Counter3 + Xdist(I)
                Counter3 = Counter3 + SizeX
              ENDDO
              Counter2 = Counter2 + SizeX*SizeY
            ENDDO
            Counter1 = Counter1 + Xdist(I)
          ENDDO
!
```

```
! Align the counter such that it points to the start of the next
! block.  (Ydist(J)-1) since already one X layer has been added in.
! Implicit assumption that SizeX = SUM( Xdist )
!
          Counter1 = Counter1 + SizeX*(Ydist(J)-1)
        ENDDO
!
! Align the counter such that it points to the start of the next
! block.  (Zdist(K)-1) since already one X-Y layer has been added in.
! Implicit assumption that SizeY = SUM( Ydist )
!
        Counter1 = Counter1 + SizeX*SizeY*(Zdist(K)-1)
      ENDDO

      CPP_LEAVE_PROCEDURE( "DECOMPREGULAR3D" )
      RETURN
```

71.5.7 DecompRegular3Dorder — Create a decomposition for a 3-D grid

INTERFACE:

```
      SUBROUTINE DecompRegular3Dorder( Order, NPEsX, NPEsY, NPEsZ,       &
     &                                 Xdist, Ydist, Zdist, Decomp )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      CHARACTER(3), INTENT( IN )       :: Order    ! Dim. ordering
      INTEGER, INTENT( IN )            :: NPEsX    ! Number of PEs in X
      INTEGER, INTENT( IN )            :: NPEsY    ! Number of PEs in Y
      INTEGER, INTENT( IN )            :: NPEsZ    ! Number of PEs in Z
      INTEGER, INTENT( IN )            :: Xdist(:) ! Distribution in X
      INTEGER, INTENT( IN )            :: Ydist(:) ! Distribution in Y
      INTEGER, INTENT( IN )            :: Zdist(:) ! Distribution in Z
```

OUTPUT PARAMETERS:

```
      TYPE(DecompType), INTENT( OUT )  :: Decomp  ! Decomp information
```

DESCRIPTION:

Creates a variable block-block-block decomposition for a regular 3-D grid, where the ordering of the PEs can be explicitly given (see next paragraph). The decomposition is given through the Xdist, Ydist, and Zdist distributions, which contain the number of entries on each PE in that dimension. This routine thus defines a parallelopiped (SOMA-block) distribution.
With the string argument Order, the order of counting in the 3d PE space can be specified. There are six possible values: "xyz", "xzy", "yxz", "yzx", "zxy", and "zyx".
The same as DecompRegular3Dorder could also be achieved by using DecompRegular3D and then permuting the PE ownership with DecompPermute.

SYSTEM ROUTINES:

```
       ALLOCATE
```

REVISION HISTORY:

```
     01.03.20   Sawyer     Creation from DecompRegular3Dzy, added ordering
```

BUGS:

```
     Not yet tested

! !LOCAL VARIABLES:
      INTEGER  :: TruePE, Counter1, Counter2, Counter3
      INTEGER  :: I, J, K, L, M, N, SizeX, SizeY, SizeZ
      INTEGER  :: Imult, Jmult, Kmult
!
      CPP_ENTER_PROCEDURE( "DECOMPREGULAR3DORDER" )
!
! Some sanity checks
!
!
      CPP_ASSERT_F90( NPEsX .EQ. SIZE( Xdist ) )
      CPP_ASSERT_F90( NPEsY .EQ. SIZE( Ydist ) )
      CPP_ASSERT_F90( NPEsZ .EQ. SIZE( Zdist ) )
      CPP_ASSERT_F90( .NOT. ASSOCIATED( Decomp%Head ) )

      IF ( Order=="xyz" ) THEN
! Looks like:      TruePE = (K-1)*NPEsX*NPEsY + (J-1)*NPEsX + (I-1) + 1
        Imult = 1
        Jmult = NPEsX
        Kmult = NPEsX*NPEsY
      ELSE IF ( Order=="xzy" ) THEN
! Looks like:      TruePE = (J-1)*NPEsX*NPEsZ + (K-1)*NPEsX + (I-1) + 1
        Imult = 1
        Jmult = NPEsX*NPEsZ
        Kmult = NPEsX
      ELSE IF ( Order=="yxz" ) THEN
! Looks like:      TruePE = (K-1)*NPEsY*NPEsX + (I-1)*NPEsY + (J-1) + 1
        Imult = NPEsY
        Jmult = 1
        Kmult = NPEsX*NPEsY
      ELSE IF ( Order=="yzx" ) THEN
! Looks like:      TruePE = (I-1)*NPEsY*NPEsZ + (K-1)*NPEsY + (J-1) + 1
        Imult = NPEsY*NPEsZ
        Jmult = 1
        Kmult = NPEsY
      ELSE IF ( Order=="zxy" ) THEN
! Looks like:      TruePE = (J-1)*NPEsX*NPEsZ + (I-1)*NPEsZ + (K-1) + 1
        Imult = NPEsZ
        Jmult = NPEsX*NPEsZ
        Kmult = 1
      ELSE IF ( Order=="zyx" ) THEN
! Looks like:      TruePE = (I-1)*NPEsY*NPEsZ + (J-1)*NPEsZ + (K-1) + 1
        Imult = NPEsY*NPEsZ
        Jmult = NPEsZ
        Kmult = 1
      ELSE
! Looks like:      TruePE = (K-1)*NPEsX*NPEsY + (J-1)*NPEsX + (I-1) + 1
```

```
        print *, "Warning: DecompCreate3Dorder", Order, "not supported"
        print *, "         Continuing with XYZ ordering"
        Imult = 1
        Jmult = NPEsX
        Kmult = NPEsX*NPEsY
      ENDIF

!
! The head contains NPEs pointers to the tag lists.
!
      SizeX = SUM(Xdist)
      SizeY = SUM(Ydist)
      SizeZ = SUM(Zdist)
      Decomp%GlobalSize = SizeX * SizeY * SizeZ
      ALLOCATE( Decomp%NumEntries( NPEsX*NPEsY*NPEsZ ) )
      ALLOCATE( Decomp%Head( NPEsX*NPEsY*NPEsZ ) )
      Counter1 = 0

      DO K = 1, NPEsZ
        DO J = 1, NPEsY
          DO I = 1, NPEsX
!
! WARNING!!!!  The definition of the PE is Row-major ordering
!

            TruePE = (I-1)*Imult + (J-1)*Jmult + (K-1)*Kmult + 1
!
! The number of entries is the product of the local X, Y, Z allotment
!
            Decomp%NumEntries(TruePE) = Xdist(I)*Ydist(J)*Zdist(K)
!
! For each Z there are Y separate runs
!
            ALLOCATE( Decomp%Head(TruePE)%StartTags(Ydist(J)*Zdist(K)) )
            ALLOCATE( Decomp%Head(TruePE)%EndTags(Ydist(J)*Zdist(K)) )
            Counter2 = Counter1
            L = 0
            DO N = 1, Zdist(K)
              Counter3 = Counter2
              DO M = 1, Ydist(J)
!
!     Since this is a regular distribution the definition of
!     tags is dictated by Xdist(I), and appears Ydist(J) times
!
!
                L = L + 1
                Decomp%Head(TruePE)%StartTags(L) = Counter3 + 1
                Decomp%Head(TruePE)%EndTags(L) = Counter3 + Xdist(I)
                Counter3 = Counter3 + SizeX
              ENDDO
              Counter2 = Counter2 + SizeX*SizeY
            ENDDO
            Counter1 = Counter1 + Xdist(I)
          ENDDO
!
```

```
! Align the counter such that it points to the start of the next
! block.  (Ydist(J)-1) since already one X layer has been added in.
! Implicit assumption that SizeX = SUM( Xdist )
!
          Counter1 = Counter1 + SizeX*(Ydist(J)-1)
        ENDDO
!
! Align the counter such that it points to the start of the next
! block.  (Zdist(K)-1) since already one X-Y layer has been added in.
! Implicit assumption that SizeY = SUM( Ydist )
!
        Counter1 = Counter1 + SizeX*SizeY*(Zdist(K)-1)
      ENDDO

      CPP_LEAVE_PROCEDURE( "DECOMPREGULAR3DORDER" )
      RETURN
```

71.5.8 DecompCreateIrr — Decomposition for an irregular mesh

INTERFACE:

```
SUBROUTINE DecompCreateIrr( NPEs, Pe, TotalPts, Decomp )
```

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
INTEGER, INTENT( IN )            :: NPEs     ! Number of PEs
INTEGER, INTENT( IN )            :: Pe(:)    ! Processor location
INTEGER, INTENT( IN )            :: TotalPts ! Number of points
```

OUTPUT PARAMETERS:

```
TYPE(DecompType), INTENT( OUT )  :: Decomp  ! Decomp information
```

DESCRIPTION:

Creates a decomposition for a irregular 1-D mesh. The decomposition is given through the number of points and an array containing the PE which each point is mapped to. This mapping naturally assumes that the local numbering is incrementally increasing as points are mapped to PEs. This assumption is sufficient for most applications, but another irregular mapping routine is available for more complex cases.

SYSTEM ROUTINES:

```
ALLOCATE
```

REVISION HISTORY:

```
98.01.19   Sawyer     Creation, with requirements from Jay Larson
98.11.02   Sawyer     Rewritten to requirements for Andrea Molod
00.07.07   Sawyer     Removed use of DimSizes(:) array
00.11.12   Sawyer     Changed argument order for overloading
```

```
!
! !LOCAL VARIABLES:
      INTEGER  :: I, PEhold
      INTEGER  :: Counter( NPEs )
!
      CPP_ENTER_PROCEDURE( "DECOMPCREATEIRR" )
!
      CPP_ASSERT_F90( TotalPts .LE. SIZE( PE ) )
      CPP_ASSERT_F90( .NOT. ASSOCIATED( Decomp%Head ) )

!
! The head contains NPEs pointers to the tag lists.
!
      Decomp%GlobalSize = TotalPts
      ALLOCATE( Decomp%NumEntries( NPEs ) )
      ALLOCATE( Decomp%Head( NPEs ) )
!
! Perform over all points in the mapping
!
      PEhold= -1
      Counter = 0
      Decomp%NumEntries = 0
      DO I=1, TotalPts
        CPP_ASSERT_F90( ( PE( I ) .LT. NPEs .AND. PE( I ) .GE. 0 ) )
        IF ( PE( I ) .NE. PEhold ) THEN
          PEhold = PE( I )
          Counter( PEhold+1 ) = Counter( PEhold+1 ) + 1
        ENDIF
        Decomp%NumEntries(PEHold+1) = Decomp%NumEntries(PEHold+1) + 1
      ENDDO
      DO I=1, NPEs
!
! Now the amount of space to allocate is known.  It is acceptable
! to in allocated an array of size 0 (F90 Handbook, Section 6.5.1)
!
        ALLOCATE( Decomp%Head(I)%StartTags(Counter(I)) )
        ALLOCATE( Decomp%Head(I)%EndTags(Counter(I)) )
      ENDDO
!
! Perform over all points in the mapping
!
      PEhold= -1
      Counter = 0
      DO I=1, TotalPts
        IF ( PE( I ) .NE. PEhold ) THEN
!
! If not first entry, close up shop on previous run
!
          IF ( I .GT. 1 ) THEN
            Decomp%Head(PEhold+1)%EndTags(Counter(PEhold+1)) = I-1
          ENDIF
          PEhold = PE( I )
          Counter( PEhold+1 ) = Counter( PEhold+1 ) + 1
          Decomp%Head(PEhold+1)%StartTags(Counter(PEhold+1)) = I
        ENDIF
```

```
      ENDDO
!
! Clean up shop for the final run
!
      IF ( TotalPts .GT. 0 ) THEN
        Decomp%Head(PEhold+1)%EndTags(Counter(PEhold+1)) = TotalPts
      ENDIF

      CPP_LEAVE_PROCEDURE( "DECOMPCREATEIRR" )
      RETURN
```

71.5.9 DecompCreateTags — Decomposition from Pe and Tags

INTERFACE:

```
      SUBROUTINE DecompCreateTags(Npes, Pe, TotalPts, Tags, Decomp )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      INTEGER, INTENT( IN )            :: NPEs     ! Number of PEs
      INTEGER, INTENT( IN )            :: Pe(:)    ! Processor location
      INTEGER, INTENT( IN )            :: TotalPts ! Number of points
      INTEGER, INTENT( IN )            :: Tags(:)  ! Global index
```

OUTPUT PARAMETERS:

```
      TYPE(DecompType), INTENT( OUT )  :: Decomp   ! Decomp information
```

DESCRIPTION:

Creates a decomposition for a irregular mesh from the Pe ownership and the Tags. This is a simple extension of DecompCreateIrr (previously DecompIrregular1D) but is much more dangerous, since the user can define the Tags (global indices) arbitrarily.

SYSTEM ROUTINES:

```
      ALLOCATE
```

REVISION HISTORY:

```
     00.11.12   Sawyer     Creation from DecompCreateIrr

!
! !LOCAL VARIABLES:
      INTEGER  :: I, PEhold, LastTag
      INTEGER  :: Counter( NPEs )
!
      CPP_ENTER_PROCEDURE( "DECOMPCREATETAGS" )
!
      CPP_ASSERT_F90( TotalPts .LE. SIZE( PE ) )
      CPP_ASSERT_F90( TotalPts .LE. SIZE( Tags ) )
```

```
      CPP_ASSERT_F90( .NOT. ASSOCIATED( Decomp%Head ) )

!
! The head contains NPEs pointers to the tag lists.
!
      Decomp%GlobalSize = TotalPts
      ALLOCATE( Decomp%NumEntries( NPEs ) )
      ALLOCATE( Decomp%Head( NPEs ) )
!
! Perform over all points in the mapping
!
      PEhold  = -1
      LastTag = -999999999
      Counter = 0
      Decomp%NumEntries = 0
      DO I=1, TotalPts
        CPP_ASSERT_F90( PE( I ) .LT. NPEs .AND. PE( I ) .GE. 0 )
        IF ( LastTag==0 .OR. Tags(I)/=LastTag+1 .OR. PE(I)/=PEhold ) THEN
          PEhold = PE( I )
          Counter( PEhold+1 ) = Counter( PEhold+1 ) + 1
        ENDIF
        Decomp%NumEntries(PEHold+1) = Decomp%NumEntries(PEHold+1) + 1
        LastTag = Tags(I)
      ENDDO

      DO I=1, NPEs
!
! Now the amount of space to allocate is known.  It is acceptable
! to in allocated an array of size 0 (F90 Handbook, Section 6.5.1)
!
        ALLOCATE( Decomp%Head(I)%StartTags(Counter(I)) )
        ALLOCATE( Decomp%Head(I)%EndTags(Counter(I)) )
      ENDDO

!
! Perform over all points in the domain
!
      PEhold  = -1
      LastTag = -999999999
      Counter = 0
      DO I=1, TotalPts
        IF ( LastTag==0 .OR. Tags(I)/=LastTag+1 .OR. PE(I)/=PEhold ) THEN
!
! If not first entry, close up shop on previous run
!
          IF ( I .GT. 1 ) THEN
            Decomp%Head(PEhold+1)%EndTags(Counter(PEhold+1)) = LastTag
          ENDIF
          PEhold = PE( I )
          Counter( PEhold+1 ) = Counter( PEhold+1 ) + 1
          Decomp%Head(PEhold+1)%StartTags(Counter(PEhold+1)) = Tags(I)
        ENDIF
        LastTag = Tags(I)
      ENDDO
!
```

```
! Clean up shop for the final run
!
      IF ( TotalPts .GT. 0 ) THEN
        Decomp%Head(PEhold+1)%EndTags(Counter(PEhold+1)) =Tags(TotalPts)
      ENDIF

      CPP_LEAVE_PROCEDURE( "DECOMPCREATETAGS" )
      RETURN
```

71.5.10 DecompGlobalToLocal — Map global index to local and PE

INTERFACE:

```
      SUBROUTINE DecompGlobalToLocal ( Decomp, Global, Local, Pe )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      TYPE(DecompType), INTENT( IN )   :: Decomp  ! Decomp information
      INTEGER, INTENT( IN )            :: Global  ! Global index
```

OUTPUT PARAMETERS:

```
      INTEGER, INTENT( OUT )  :: Local            ! Local index
      INTEGER, INTENT( OUT )  :: Pe               ! Pe location
```

DESCRIPTION:

Given a decomposition and a global index, this routine returns the local index and PE location of that global tag. If the global index is not found, Local = 0, Pe = -1 is returned.
Note that this routine is not efficient by any stretch of the imagination — only one index can be converted at a time. In addition, a search procedure must be performed, whose efficiency is inversely proportional to the size of the decomposition (in particular, to the number of "runs"). Conceptually this mapping should be used only once in the program for initialization, and subsequently all calculations should take place using local indices.

SYSTEM ROUTINES:

```
      SIZE
```

REVISION HISTORY:

```
     98.03.20   Sawyer     Creation
     01.03.17   Sawyer     Test for Global==0 (undefined element)

!
! !LOCAL VARIABLES:
      INTEGER  :: I, J
      LOGICAL  :: Found
!
      CPP_ENTER_PROCEDURE( "DECOMPGLOBALTOLOCAL" )
```

```
!
! Search over all the PEs
!
      Pe = -1
      Local = 0
      Found = .FALSE.
      DO WHILE ( .NOT. Found .AND. Global .NE. 0 )
!
! Copy the number of entries on each PE
!
        Pe = Pe + 1
        CPP_ASSERT_F90( ( SIZE(Decomp%Head(Pe+1)%StartTags) .EQ. SIZE(Decomp%Head(Pe+1)%EndTags) ) )
!
! Search through the local data segment
!
        Local = 1
        J = 1
        DO WHILE ( .NOT. Found .AND.                                        &
     &             J .LE. SIZE( Decomp%Head(Pe+1)%StartTags ) )
          IF ( Global .GE. Decomp%Head(Pe+1)%StartTags(J) .AND.             &
     &         Global .LE. Decomp%Head(Pe+1)%EndTags(J) ) THEN
            Local = Local +  Global - Decomp%Head(Pe+1)%StartTags(J)
            Found = .TRUE.
          ELSE
            Local = Local + Decomp%Head(Pe+1)%EndTags(J) -                  &
     &                      Decomp%Head(Pe+1)%StartTags(J) + 1
          ENDIF
          J = J+1
        ENDDO
!
! Emergency brake
!
        IF ( Pe .EQ. (SIZE( Decomp%Head )-1) .AND. .NOT. Found ) THEN
          Found = .TRUE.
          Local = 0
          Pe    = -1
        ENDIF
      ENDDO

      CPP_ASSERT_F90( Local .LE. Decomp%NumEntries(Pe+1) )

      CPP_LEAVE_PROCEDURE( "DECOMPGLOBALTOLOCAL" )
      RETURN
!
```

71.5.11 DecompLocalToGlobal — Map global index to local and PE

INTERFACE:

```
      SUBROUTINE DecompLocalToGlobal ( Decomp, Local, Pe, Global )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      TYPE(DecompType), INTENT( IN )   :: Decomp  ! Decomp information
      INTEGER, INTENT( IN )            :: Local   ! Local index
      INTEGER, INTENT( IN )            :: Pe      ! Pe location
```

OUTPUT PARAMETERS:

```
      INTEGER, INTENT( OUT )           :: Global  ! Global index
```

DESCRIPTION:

Given a decomposition and a local-pe index pair, this routine returns the 2-D global index. If the local index is not found, 0 is returned.
Note that this routine is not efficient by any stretch of the imagination — only one index can be converted at a time. In addition, a search procedure must be performed, whose efficiency is inversely proportional to the size of the decomposition (in particular, to the number of "runs"). Conceptually this mapping should be used only once in the program for initialization, and subsequently all calculations should take place using local indices.

SYSTEM ROUTINES:

```
      SIZE
```

REVISION HISTORY:

```
     98.03.20   Sawyer     Creation

!
! !LOCAL VARIABLES:
      INTEGER  :: J, Counter
      LOGICAL  :: Found
!
      CPP_ENTER_PROCEDURE( "DECOMPLOCALTOGLOBAL" )
      CPP_ASSERT_F90( Pe .GE. 0 )
      CPP_ASSERT_F90( Pe .LT. SIZE(Decomp%Head) )
      CPP_ASSERT_F90( Local .GT. 0 )
      CPP_ASSERT_F90( Local .LE. Decomp%NumEntries(Pe+1) )

      Counter = 0
      Found   = .FALSE.
      J = 0
      DO WHILE ( .NOT. Found )
        J = J+1
        Counter = Counter + Decomp%Head(Pe+1)%EndTags(J) -               &
     &                      Decomp%Head(Pe+1)%StartTags(J) + 1
        IF ( Local .LE.  Counter ) THEN
          Found = .TRUE.
!
! The following calculation is not immediately obvious.  Think about it
!
          Global = Local - Counter + Decomp%Head(Pe+1)%EndTags(J)
          Found = .TRUE.
        ELSEIF ( J .GE. SIZE( Decomp%Head(Pe+1)%StartTags ) ) THEN
!
! Emergency brake
```

```
!
          Found = .TRUE.
          Global = 0
        ENDIF
      ENDDO

      CPP_LEAVE_PROCEDURE( "DECOMPLOCALTOGLOBAL" )
      RETURN
!
```

71.5.12 DecompInfo — Information about decomposition

INTERFACE:

```
       SUBROUTINE DecompInfo( Decomp, Npes, TotalPts )
```

USES:

```
       IMPLICIT NONE
```

INPUT PARAMETERS:

```
       TYPE(DecompType), INTENT( IN ):: Decomp   ! Decomp information
```

OUTPUT PARAMETERS:

```
       INTEGER, INTENT( OUT )        :: Npes     ! Npes in decomposition
       INTEGER, INTENT( OUT )        :: TotalPts ! Total points in domain
```

DESCRIPTION:

Return information about the decomposition: the number of PEs over which the domain is decomposed, and the size of the domain.

REVISION HISTORY:

```
     00.11.12   Sawyer     Creation

!
!
      CPP_ENTER_PROCEDURE( "DECOMPINFO" )

      Npes = SIZE( Decomp%Head )
      TotalPts = Decomp%GlobalSize

      CPP_LEAVE_PROCEDURE( "DECOMPINFO" )
      RETURN
```

71.6 Fortran: Module Interface ghostmodule (Source File: ghostmodule.F90)

USES:

```
      USE decompmodule, ONLY : DecompType
#include "debug.h"
#include "pilgrim.h"
      IMPLICIT NONE
```

DESCRIPTION:

This module provides the basic support for "ghost regions". In reality the ghost region just subset of the global domain described by a decomposition (pro memoria: a decomposition describes a partition of a global index space over a number of PEs; this is inherently non-overlapping).
It contains the following public types and routines.

GhostType	Type to describe ghosted local vector
GhostFree	Destroy a ghost definition
GhostCreate	Copy ghost definition to newly created one
GhostInfo	Returns some information about the region

GhostCreate is overloaded to support different types of domains:

GhostRegular1D	Define a subset of a 1D domain
GhostRegular2D	Define a subset of a 2D domain
GhostRegular3D	Define a subset of a 3D domain
GhostIrregular	Define a subset of an irregular domain

Generally one will use the GhostCreate routine which corresponds to the underlying decomposition; e.g., if the decomposition was defined with DecompRegular3D you would probably use GhostRegular3D to define the ghost region. But since decompositions and ghost regions are generic, i.e., one-size-fits-all, this is not a requirement. Be very careful if you use non-corresponding routines!
The ghost type contains a decomposition which describes the *non-overlapping* distribution of the global domain (this is a replicated data structure with complete information about all data structures on all PEs). Its other components are a list of the global indices of the on the boundary (not replicated), and a description of the mapping of the ghosted local region to global indices.
This module is communication-free and is a foundation for ParUtilitiesModule. Since GhostType is local to the PE, the modules routines can and should be called with non-replicated data structures. Before boundary communication takes place, the communication pattern derived from the ghost regions must be created (see ParUtilitiesModule).

REVISION HISTORY:

```
00.11.10   Sawyer     Creation
01.02.07   Sawyer     Improvements; added Border to GhostType
01.02.12   Sawyer     Converted to free format
```

PUBLIC TYPES:

```
      PUBLIC GhostType
      PUBLIC GhostFree
      PUBLIC GhostCreate
      PUBLIC GhostInfo
```

```
      INTERFACE     GhostCreate
        MODULE PROCEDURE GhostIrregular
        MODULE PROCEDURE GhostRegular1D
        MODULE PROCEDURE GhostRegular2D
        MODULE PROCEDURE GhostRegular3D
      END INTERFACE

  Decomposition info

       TYPE GhostType
         TYPE(DecompType) :: Decomp ! Decomposition of global partition
         TYPE(DecompType) :: Local  ! Decomposition of local region
         TYPE(DecompType) :: Border ! Decomposition of local segment
       END TYPE GhostType
```

71.6.1 GhostFree — Free a ghosted region

INTERFACE:

```
      SUBROUTINE GhostFree ( Ghost )
```

USES:

```
      USE decompmodule, ONLY : DecompFree
      IMPLICIT NONE
```

INPUT/OUTPUT PARAMETERS:

```
      TYPE(GhostType), INTENT( INOUT ):: Ghost  ! Ghost information
```

DESCRIPTION:

Free the ghost decomposition – deallocate the data structures.

SYSTEM ROUTINES:

```
      ASSOCIATED, DEALLOCATE
```

REVISION HISTORY:

```
     00.11.12   Sawyer     Creation

!
!
      CPP_ENTER_PROCEDURE( "GHOSTFREE" )

      CALL DecompFree( Ghost%Border )
      CALL DecompFree( Ghost%Local  )
      CALL DecompFree( Ghost%Decomp )

      CPP_LEAVE_PROCEDURE( "GHOSTFREE" )
      RETURN
```

71.6.2 GhostCopy — Copy one decomposition to another

INTERFACE:

```
SUBROUTINE GhostCopy ( GhostIn, GhostOut )
```

USES:

```
USE decompmodule, ONLY : DecompCopy
IMPLICIT NONE
```

INPUT PARAMETERS:

```
TYPE(GhostType), INTENT( IN )   :: GhostIn  ! Ghost information
```

OUTPUT PARAMETERS:

```
TYPE(GhostType), INTENT( OUT )  :: GhostOut ! Ghost information
```

DESCRIPTION:

Creates an output ghost definition and copies GhostIn to it

SYSTEM ROUTINES:

```
ALLOCATE
```

REVISION HISTORY:

```
     00.11.12   Sawyer     Creation

! !LOCAL VARIABLES:
      INTEGER  :: I, Nsize

      CPP_ENTER_PROCEDURE( "GHOSTCOPY" )

      CALL DecompCopy( GhostIn%Decomp, GhostOut%Decomp )
      CALL DecompCopy( GhostIn%Local,  GhostOut%Local  )
      CALL DecompCopy( GhostIn%Border, GhostOut%Border )

      CPP_LEAVE_PROCEDURE( "GHOSTCOPY" )
      RETURN
```

71.6.3 GhostIrregular — Create a ghost definition for 1-D grid

INTERFACE:

```
SUBROUTINE GhostIrregular( Decomp, Id, LocalSize, Tags, Ghost )
```

USES:

```
 USE decompmodule, ONLY : DecompCreate, DecompCopy,                &
&                         DecompGlobalToLocal, DecompInfo
 IMPLICIT NONE
```

INPUT PARAMETERS:

```
      TYPE(DecompType), INTENT( IN ) :: Decomp     ! Decomp information
      INTEGER, INTENT( IN )          :: Id         ! Local PE identifer
      INTEGER, INTENT( IN )          :: LocalSize  ! Size of local segment
      INTEGER, INTENT( IN )          :: Tags(:)    ! Global tags
```

OUTPUT PARAMETERS:

```
      TYPE(GhostType), INTENT( OUT ) :: Ghost  ! Ghost definition
```

DESCRIPTION:

Creates a ghost definition for a ghosted array given by the PEs and Tags of the local points. Note that none of the array bounds can be outside the global domain!

SYSTEM ROUTINES:

```
      ALLOCATE, DEALLOCATE
```

REVISION HISTORY:

```
     00.11.12   Sawyer     Creation
```

BUGS:

```
     None of the array bounds can be outside of the global domain!
     This is significant if the local region is on the edge of the
     domain, and, in other words, the ghost region cannot cover
     empty space.  This limitation may be relaxed in the future.

! !LOCAL VARIABLES:
      INTEGER :: I, NPEs, GlobalSize, Local, Count, Ipe
      INTEGER, ALLOCATABLE :: Pe(:), Other(:)
!
!
      CPP_ENTER_PROCEDURE( "GHOSTIRREGULAR" )
!
! Allocate the basic data structures
!
      CALL DecompInfo( Decomp, Npes, GlobalSize )

      ALLOCATE( Pe( LocalSize ) )
      ALLOCATE( Other( LocalSize ) )

!
! Use decompmodule to create global and local portions of Ghost
! The local version is only on the local processor "0"

      Other = Id
      CALL DecompCreate( Npes, Other, LocalSize, Tags, Ghost%Local )

!
! Perform over all points local segment
!
      Count = 0
      DO I= 1, LocalSize
        CALL DecompGlobalToLocal( Decomp, Tags(I), Local, Ipe )
        CPP_ASSERT_F90( (Local .GT. 0) .AND. (ipe .GE. 0) )
        IF ( Ipe .ne. id ) THEN
```

```
          Count = Count + 1
          Other( Count ) = Tags(I)
          Pe( Count )    = Ipe
        ENDIF
      ENDDO

!
! Define the border regions.  Presumably Count << LocalSize
!
      CALL DecompCreate( Npes, Pe, Count, Other, Ghost%Border )

!
! Copy the decomposition too
!
      CALL DecompCopy( Decomp, Ghost%Decomp )

! Clean up

      DEALLOCATE( Pe )
      DEALLOCATE( Other )

      CPP_LEAVE_PROCEDURE( "GHOSTIRREGULAR" )
      RETURN
```

71.6.4 GhostRegular1D — Create a ghost definition for 1-D grid

INTERFACE:

```
       SUBROUTINE GhostRegular1D( Decomp, Id, Xglobal, Xfrom, Xto, Xwrap, &
      &                           Ghost )
```

USES:

```
       USE decompmodule, ONLY : DecompCreate, DecompCopy,                &
      &                         DecompGlobalToLocal, DecompInfo
       IMPLICIT NONE
```

INPUT PARAMETERS:

```
       TYPE(DecompType), INTENT( IN )   :: Decomp ! Decomp information
       INTEGER, INTENT( IN )            :: Id     ! Local PE identifer
       INTEGER, INTENT( IN )            :: Xglobal! Total in X
       INTEGER, INTENT( IN )            :: Xfrom  ! Low index in X
       INTEGER, INTENT( IN )            :: Xto    ! High index in X
       LOGICAL, INTENT( IN )            :: Xwrap  ! Wrap in X?
```

OUTPUT PARAMETERS:

```
       TYPE(GhostType), INTENT( OUT )   :: Ghost  ! Ghost definition
```

DESCRIPTION:

Creates a ghost definition for a regular 1-D array with the array bounds Xfrom:Xto.
If the array bounds are outside of the global domain they may be wrapped around back into the global domain (variable Xwrap). If the region is not wrapped, it is advisable that the ghost region end at the

boundary (which usually requires special case treatment depending on the PE number). If it does not end at the boundary, undefined points are introduced.

SYSTEM ROUTINES:

```
       ALLOCATE, DEALLOCATE
```

REVISION HISTORY:

```
     00.11.12   Sawyer     Creation
```

BUGS:

```
     There are certain limitations to ghost regions which can be
     avoided by clean programming practices.  If the ghosted region
     wraps back onto core regions of the same PE, problems can arise.
     The simple case -- a ghosted region on 1 PE -- is supported in
     most cases.  However, if it wraps back onto the local PE
     in such a way that more than one ghost points is mapped to
     one core domain global index, then the code may fail.  Note
     that this is rarely the case if the ghost regions are small
     and enough processors are used to avoid wrapping back on the
     local one.


! !LOCAL VARIABLES:
      INTEGER :: I, L, NPEs, GlobalSize, LocalSize, Count, Local, Ipe
      INTEGER :: Global
      INTEGER, ALLOCATABLE :: Pe(:), Tags(:), Other(:)
!
!
      CPP_ENTER_PROCEDURE( "GHOSTREGULAR1D" )

!
! Allocate the basic data structures
!
      CALL DecompInfo( Decomp, NPEs, GlobalSize )
      CPP_ASSERT_F90( GlobalSize .EQ. Xglobal )

      LocalSize = Xto - Xfrom + 1
      CPP_ASSERT_F90( LocalSize .GE. 0 )

      ALLOCATE( Pe( LocalSize ) )
      ALLOCATE( Tags( LocalSize ) )
      ALLOCATE( Other( LocalSize ) )

!
! Perform over all points local segment
!
      Count = 0
      L = 0
      DO I = Xfrom, Xto
        L = L + 1
        Global = MODULO(I-1,Xglobal)+1  ! Wrap around condition
        IF (Xwrap .OR. Global==I) THEN
          Tags(L) = Global                 ! Global Tags
          CALL DecompGlobalToLocal( Decomp, Global, Local, Ipe )
```

```
          IF ( Ipe .ne. Id .AND. Ipe .GE. 0 ) THEN
            Count = Count + 1
            Other( Count ) = Global        ! Local Tags
            Pe( Count )    = Ipe
          ENDIF
!
! Special case: the domain wraps-around onto the same PE.  This is
! very tricky:  the ghost points are distinguished from their true
! local core domain counterparts by a minus sign.  This makes the
! address space in both Ghost%Border and Ghost%Local unique
!
          IF ( Ipe .eq. Id .AND. I .ne. Global ) THEN
            Count = Count + 1
            Other( Count ) = -Global       ! Local Tags
            Pe( Count )    = Ipe
            Tags(L) = -Global              ! Global Tags (mark ghost region!)
          ENDIF
        ELSE
          Tags(L) = 0
        ENDIF
      ENDDO

!
! Perform over all points local segment
!
      CALL DecompCreate( Npes, Pe, Count, Other, Ghost%Border )

!
! Use decompmodule to create global and local portions of Ghost
! The local version is only on the local PE
!
      Other = Id
      CALL DecompCreate( Npes, Other, LocalSize, Tags, Ghost%Local )

!
! Copy the decomposition too
!
      CALL DecompCopy( Decomp, Ghost%Decomp )

! Clean up

      DEALLOCATE( Other    )
      DEALLOCATE( Tags     )
      DEALLOCATE( Pe       )

      CPP_LEAVE_PROCEDURE( "GHOSTREGULAR1D" )
      RETURN
```

71.6.5 GhostRegular2D — Create a ghost definition for 2-D grid

INTERFACE:

```
 SUBROUTINE GhostRegular2D( Decomp, Id, Xglobal, Xfrom, Xto, Xwrap, &
&                           Yglobal, Yfrom, Yto, Ywrap, Ghost )
```

USES:

```
 USE decompmodule, ONLY : DecompCreate, DecompCopy,                 &
&                         DecompGlobalToLocal, DecompInfo
 IMPLICIT NONE
```

INPUT PARAMETERS:

```
 TYPE(DecompType), INTENT( IN )   :: Decomp ! Decomp information
 INTEGER, INTENT( IN )            :: Id     ! Local PE identifer
 INTEGER, INTENT( IN )            :: Xglobal! Total in X
 INTEGER, INTENT( IN )            :: Xfrom  ! Low index in X
 INTEGER, INTENT( IN )            :: Xto    ! High index in X
 LOGICAL, INTENT( IN )            :: Xwrap  ! Wrap in X?
 INTEGER, INTENT( IN )            :: Yglobal! Total in X
 INTEGER, INTENT( IN )            :: Yfrom  ! Distribution in X
 INTEGER, INTENT( IN )            :: Yto    ! Distribution in Y
 LOGICAL, INTENT( IN )            :: Ywrap  ! Wrap in Y?
```

OUTPUT PARAMETERS:

```
 TYPE(GhostType), INTENT( OUT )   :: Ghost  ! Ghost definition
```

DESCRIPTION:

Creates a ghost definition for a regular 2-D array with the array bounds Xfrom:Xto,Yfrom:Yto.
If the array bounds are outside of the global domain they may be wrapped around back into the global domain (Xwrap, Ywrap). If the region is not wrapped, it is advisable that the ghost region end at the boundary (which usually requires special case treatment depending on the PE number). If it does not end at the boundary, undefined points are introduced.

SYSTEM ROUTINES:

```
 ALLOCATE, DEALLOCATE
```

REVISION HISTORY:

```
00.11.12   Sawyer     Creation
```

BUGS:

```
There are certain limitations to ghost regions which can be
avoided by clean programming practices.  If the ghosted region
wraps back onto core regions of the same PE, problems can arise.
The simple case -- a ghosted region on 1 PE -- is supported in
most cases.  However, if it wraps back onto the local PE
in such a way that more than one ghost points is mapped to
one core domain global index, then the code may fail.  Note
that this is rarely the case if the ghost regions are small
and enough processors are used to avoid wrapping back on the
local one.

WARNING:  If the domain wraps around in both X and Y there is a
the code should be run with at least 2 PEs so that in one of the
two dimensions there is no wrap-around onto the same PE.
```

```
! !LOCAL VARIABLES:
      INTEGER :: I, J, L, Ipe, Npes, GlobalSize, LocalSize
      INTEGER :: Global, Count, Local, Xtrue, Ytrue
      INTEGER, ALLOCATABLE :: Pe(:), Tags(:), Other(:)
!
!
      CPP_ENTER_PROCEDURE( "GHOSTREGULAR2D" )

!
! Allocate the basic data structures
!
      CALL DecompInfo( Decomp, Npes, GlobalSize )
      CPP_ASSERT_F90( GlobalSize .EQ. Xglobal*Yglobal )

      LocalSize = (Xto - Xfrom + 1)*(Yto - Yfrom + 1)
      CPP_ASSERT_F90( LocalSize .GE. 0 )
      ALLOCATE( Pe( LocalSize ) )
      ALLOCATE( Tags( LocalSize ) )
      ALLOCATE( Other( LocalSize ) )
!
! Perform over all points local segment
!
      Count = 0
      L = 0
      DO J= Yfrom, Yto
        Ytrue = MODULO(J-1,Yglobal) + 1
        DO I= Xfrom, Xto
          Xtrue = MODULO(I-1,Xglobal) + 1
          L = L + 1
          Global = (Ytrue-1)*Xglobal + Xtrue
          IF ( (Xwrap.OR.(Xtrue==I)) .AND. (Ywrap.OR.(Ytrue==J)) ) THEN
            Tags( L ) = Global
            CALL DecompGlobalToLocal( Decomp, Global, Local, Ipe )
            IF ( Ipe .ne. Id .AND. Ipe .GE. 0 ) THEN
              Count = Count + 1
              Other( Count ) = Global     ! Local Tags
              Pe( Count )    = Ipe
            ENDIF
!
! Special case: the domain wraps-around onto the same PE.  This is
! very tricky:  the ghost points are distinguished from their true
! local core domain counterparts by a minus sign.  This makes the
! address space in both Ghost%Border and Ghost%Local unique
!
            IF ( Ipe.EQ.Id .AND. ( I.NE.Xtrue .OR. J.NE.Ytrue ) ) THEN
              Count = Count + 1
              Other( Count ) = -Global       ! Local Tags
              Pe( Count )    = Ipe
              Tags(L) = -Global              ! Global Tags (mark ghost region!)
            ENDIF
          ELSE
            Tags(L) = 0
          ENDIF
        ENDDO
      ENDDO
```

```
!
! Perform over all points local segment
!
      CALL DecompCreate( Npes, Pe, Count, Other, Ghost%Border )

!
! Use decompmodule to create global and local portions of Ghost
! The local version is only on the local PE
!
      Other = Id
      CALL DecompCreate( Npes, Other, LocalSize, Tags, Ghost%Local )

!
! Copy the decomposition too
!
      CALL DecompCopy( Decomp, Ghost%Decomp )

! Clean up

      DEALLOCATE( Other )
      DEALLOCATE( Tags  )
      DEALLOCATE( Pe    )

      CPP_LEAVE_PROCEDURE( "GHOSTREGULAR2D" )
      RETURN
```

71.6.6 GhostRegular3D — Create a ghost definition for 3-D grid

INTERFACE:

```
       SUBROUTINE GhostRegular3D( Decomp, Id, Xglobal, Xfrom, Xto, Xwrap, &
      &                           Yglobal, Yfrom, Yto, Ywrap,             &
      &                           Zglobal, Zfrom, Zto, Zwrap, Ghost )
```

USES:

```
       USE decompmodule, ONLY : DecompCreate, DecompCopy,                 &
      &                         DecompGlobalToLocal, DecompInfo
       IMPLICIT NONE
```

INPUT PARAMETERS:

```
       TYPE(DecompType), INTENT( IN )   :: Decomp ! Decomp information
       INTEGER, INTENT( IN )            :: Id     ! Local PE identifer
       INTEGER, INTENT( IN )            :: Xglobal! Total in X
       INTEGER, INTENT( IN )            :: Xfrom  ! Low index in X
       INTEGER, INTENT( IN )            :: Xto    ! High index in X
       LOGICAL, INTENT( IN )            :: Xwrap  ! Wrap in X?
       INTEGER, INTENT( IN )            :: Yglobal! Total in Y
       INTEGER, INTENT( IN )            :: Yfrom  ! Distribution in Y
       INTEGER, INTENT( IN )            :: Yto    ! Distribution in Y
       LOGICAL, INTENT( IN )            :: Ywrap  ! Wrap in Y?
       INTEGER, INTENT( IN )            :: Zglobal! Total in Z
```

```
      INTEGER, INTENT( IN )            :: Zfrom  ! Distribution in Z
      INTEGER, INTENT( IN )            :: Zto    ! Distribution in Z
      LOGICAL, INTENT( IN )            :: Zwrap  ! Wrap in Z?
```

OUTPUT PARAMETERS:

```
      TYPE(GhostType), INTENT( OUT )   :: Ghost  ! Ghost definition
```

DESCRIPTION:

Creates a ghost definition for a regular 3-D array with the array bounds Xfrom:Xto,Yfrom:Yto,Zfrom:Zto. If the array bounds are outside of the global domain they may be wrapped around back into the global domain (Xwrap, Ywrap). If the region is not wrapped, it is advisable that the ghost region end at the boundary (which usually requires special case treatment depending on the PE number). If it does not end at the boundary, undefined points are introduced.

SYSTEM ROUTINES:

```
      ALLOCATE, DEALLOCATE
```

REVISION HISTORY:

```
     00.11.12   Sawyer     Creation
```

BUGS:

```
     There are certain limitations to ghost regions which can be
     avoided by clean programming practices.  If the ghosted region
     wraps back onto core regions of the same PE, problems can arise.
     The simple case -- a ghosted region on 1 PE -- is supported in
     most cases.  However, if it wraps back onto the local PE
     in such a way that more than one ghost points is mapped to
     one core domain global index, then the code may fail.  Note
     that this is rarely the case if the ghost regions are small
     and enough processors are used to avoid wrapping back on the
     local one.

     WARNING:  If the domain wraps around in two of the three dims
     the code should be run with at least 2 PEs so that in one of the
     two dimensions there is no wrap-around onto the same PE.  If it
     wraps around in all three dimensions it should be run on at least
     4 PEs.  Note these are extremely rare toriodal cases.

! !LOCAL VARIABLES:
      INTEGER :: I, J, K, L, Ipe, Npes, GlobalSize, LocalSize
      INTEGER :: Global, Count, Local, Xtrue, Ytrue, Ztrue
      LOGICAL :: IsX, IsY, IsZ
      INTEGER, ALLOCATABLE :: Pe(:), Tags(:), Other(:)
!
!
      CPP_ENTER_PROCEDURE( "GHOSTREGULAR3D" )

!
! Allocate the basic data structures
!
      CALL DecompInfo( Decomp, Npes, GlobalSize )
      CPP_ASSERT_F90( GlobalSize .EQ. Xglobal*Yglobal*Zglobal )
```

```
      LocalSize = (Xto-Xfrom+1) * (Yto-Yfrom+1) * (Zto-Zfrom+1)

      CPP_ASSERT_F90( LocalSize .GE. 0 )
      ALLOCATE( Pe( LocalSize ) )
      ALLOCATE( Tags( LocalSize ) )
      ALLOCATE( Other( LocalSize ) )
!
! Perform over all points local segment
!
      Count = 0
      L = 0
      DO K = Zfrom, Zto
        Ztrue = MODULO(K-1,Zglobal) + 1
        DO J = Yfrom, Yto
          Ytrue = MODULO(J-1,Yglobal) + 1
          DO I = Xfrom, Xto
            Xtrue = MODULO(I-1,Xglobal) + 1
            L = L + 1
            Global = ((Ztrue-1)*Yglobal+(Ytrue-1))*Xglobal+Xtrue
!
! Check to see if this is an defined global index
!
            CALL DecompGlobalToLocal( Decomp, Global, Local, Ipe )
            CPP_ASSERT_F90( (Local .GT. 0) .AND. (Ipe .GE. 0) )
!
! The wrapping case: mark as undefined

            IsX = Xtrue/=I
            IsY = Ytrue/=J
            IsZ = Ztrue/=K
            IF ( (.NOT.Xwrap.AND.IsX) .OR. (.NOT.Ywrap.AND.IsY)          &
     &          .OR. (.NOT.Zwrap.AND.IsZ) ) THEN
              Count = Count + 1
              Other( Count ) = 0           ! Local Tags
              Pe( Count )    = Ipe
              Tags( L )      = 0
            ELSE IF ( Ipe .ne. Id ) THEN
!
! Boundary case:  Global is in a ghost region not belonging
! to this PE.  Mark it in the border data structure (Arrays Other and Pe)
!
              Count = Count + 1
              Other( Count ) = Global      ! Local Tags
              Pe( Count )    = Ipe
              Tags( L )      = Global
            ELSE IF ( Ipe==Id .AND. (IsX.OR.IsY.OR.IsZ) ) THEN
!
! Special case: the domain wraps-around onto the same PE.  This is
! very tricky:  the ghost points are distinguished from their true
! local core domain counterparts by a minus sign.  This makes the
! address space in both Ghost%Border and Ghost%Local unique
!
              Count = Count + 1
              Other( Count ) = -Global     ! Local Tags
              Pe( Count )    = Ipe
```

```
            Tags(L)        = -Global     ! Global Tags (mark ghost region!)
          ELSE
            Tags( L ) = Global
          ENDIF
        ENDDO
      ENDDO
    ENDDO
    CPP_ASSERT_F90( LocalSize==L )
!
! Perform over all points local segment
!
    CALL DecompCreate( Npes, Pe, Count, Other, Ghost%Border )

!
! Use decompmodule to create global and local portions of Ghost
! The local version is only on the local PE
!
    Other = Id
    CALL DecompCreate( Npes, Other, LocalSize, Tags, Ghost%Local )

!
! Copy the decomposition too
!
    CALL DecompCopy( Decomp, Ghost%Decomp )

! Clean up

    DEALLOCATE( Other )
    DEALLOCATE( Tags  )
    DEALLOCATE( Pe    )

    CPP_LEAVE_PROCEDURE( "GHOSTREGULAR3D" )
    RETURN
```

71.6.7 GhostInfo — Information about ghosted decompostion

INTERFACE:

```
     SUBROUTINE GhostInfo( Ghost, Npes,                                &
    &                      GlobalSize, LocalSize, BorderSize )
```

USES:

```
     USE decompmodule, ONLY : DecompInfo
     IMPLICIT NONE
```

INPUT PARAMETERS:

```
     TYPE(GhostType), INTENT( IN ):: Ghost  ! Ghost information
```

INPUT PARAMETERS:

```
      INTEGER, INTENT( OUT )   :: Npes       ! Number of Pes
      INTEGER, INTENT( OUT )   :: GlobalSize ! Size of global domain
      INTEGER, INTENT( OUT )   :: LocalSize  ! Size of ghosted local region
      INTEGER, INTENT( OUT )   :: BorderSize ! Size of border
```

DESCRIPTION:

Return information about the ghosted region

SYSTEM ROUTINES:

REVISION HISTORY:

```
     00.11.12   Sawyer     Creation

!
!
      CPP_ENTER_PROCEDURE( "GHOSTINFO" )

      CALL DecompInfo( Ghost%Decomp, Npes, GlobalSize )
      CALL DecompInfo( Ghost%Local,  Npes, LocalSize )
      CALL DecompInfo( Ghost%Border, Npes, BorderSize )

      CPP_LEAVE_PROCEDURE( "GHOSTINFO" )
      RETURN
```

71.7 Fortran: Module Interface parutilitiesmodule (Source File: parutilitiesmodule.F90)

USES:

```
      USE precision
#include "debug.h"
      IMPLICIT NONE
#include "mpif.h"
#include "pilgrim.h"
```

PUBLIC DATA MEMBERS:

```
#if defined(USE_ARENAS)
      COMMON /ARENA/  buf01, buf02, buf03
      POINTER(buf01,volume)
      INTEGER(i4), DIMENSION(MAX_PE,MAX_PE) :: volume
      POINTER(buf02,databuf)
      REAL(r8), DIMENSION(MAX_BUF) :: databuf
      POINTER(buf03,intbuf)
      INTEGER(i4), DIMENSION(MAX_BUF)  :: intbuf
#endif
      PUBLIC     CommGlobal, GID, Gsize
      PUBLIC     SUMOP, MAXOP, MINOP
```

```
      INTEGER,SAVE :: CommGlobal   ! Global communicator (before ParSplit)
      INTEGER,SAVE :: GSize        ! Size of communicator CommGlobal
      INTEGER,SAVE :: GID          ! My rank in communicator CommGlobal

#define CPP_SUM_OP 101
#define CPP_MAX_OP 102
#define CPP_MIN_OP 103
#define CPP_BCST_OP 104
#if defined( USE_ARENAS )
      INTEGER,SAVE :: SUMOP = CPP_SUM_OP
      INTEGER,SAVE :: MAXOP = CPP_MAX_OP
      INTEGER,SAVE :: MINOP = CPP_MIN_OP
      INTEGER,SAVE :: BCSTOP = CPP_BCST_OP
#else
      INTEGER,SAVE :: SUMOP = MPI_SUM
      INTEGER,SAVE :: MAXOP = MPI_MAX
      INTEGER,SAVE :: MINOP = MPI_MIN
      INTEGER,SAVE :: BCSTOP = CPP_BCST_OP
#endif

      INTEGER,SAVE :: numcpu, numcps( MAX_PE ), blocksize, packetsize
```

PUBLIC MEMBER FUNCTIONS:

```
      PUBLIC ParPatternType

       PRIVATE BlockDescriptor
       TYPE BlockDescriptor
         INTEGER, POINTER     :: Displacements(:)   ! Offsets in local segment
         INTEGER, POINTER     :: BlockSizes(:)      ! Block sizes to transfer
       END TYPE BlockDescriptor

      TYPE ParPatternType
        INTEGER ::     Comm                  ! Communicator
        INTEGER ::     Iam                   ! My rank in communicator
        INTEGER ::     Size                  ! Size of communicator
#if defined( USE_ARENAS )
        TYPE(BlockDescriptor), POINTER :: SendDesc(:) ! Array of descriptors
        TYPE(BlockDescriptor), POINTER :: RecvDesc(:) ! Array of descriptors
#else
        INTEGER, POINTER :: SendDesc( : )    ! Send descriptors
        INTEGER, POINTER :: RecvDesc( : )    ! Receive descriptors
#endif
      END TYPE ParPatternType

      PUBLIC     ParInit, ParSplit, ParFree, ParExit
      PUBLIC     ParScatter, ParGather
      PUBLIC     ParBeginTransfer, ParEndTransfer
      PUBLIC     ParExchangeVector, ParCollective
      PUBLIC     ParPatternCreate, ParPatternFree

      INTERFACE     ParPatternCreate
        MODULE PROCEDURE ParPatternGhost
        MODULE PROCEDURE ParPatternDecompToDecomp
 !!        MODULE PROCEDURE ParPatternDecompToGhost
 !!        MODULE PROCEDURE ParPatternGhostToDecomp
```

```
!!        MODULE PROCEDURE ParPatternGhostToGhost
     END INTERFACE

     INTERFACE     ParScatter
       MODULE PROCEDURE ParScatterReal
       MODULE PROCEDURE ParScatterInt
     END INTERFACE

     INTERFACE     ParGather
       MODULE PROCEDURE ParGatherReal
       MODULE PROCEDURE ParGatherInt
     END INTERFACE

     INTERFACE     ParBeginTransfer
       MODULE PROCEDURE ParBeginTransferReal
       MODULE PROCEDURE ParBeginTransferPattern
        MODULE PROCEDURE ParBeginTransferInt
     END INTERFACE

     INTERFACE     ParEndTransfer
       MODULE PROCEDURE ParEndTransferReal
       MODULE PROCEDURE ParEndTransferPattern
        MODULE PROCEDURE ParEndTransferInt
     END INTERFACE

     INTERFACE     ParExchangeVector
       MODULE PROCEDURE ParExchangeVectorReal
       MODULE PROCEDURE ParExchangeVectorInt
     END INTERFACE

     INTERFACE     ParCollective
       MODULE PROCEDURE ParCollectiveBarrier
       MODULE PROCEDURE ParCollective0D
       MODULE PROCEDURE ParCollective1D
       MODULE PROCEDURE ParCollective2D
       MODULE PROCEDURE ParCollective3D
       MODULE PROCEDURE ParCollective0DInt
       MODULE PROCEDURE ParCollective1DInt
     END INTERFACE
```

DESCRIPTION:

This module provides the basic utilities to support parallelism on a distributed or shared memory multiprocessor.

ParInit	Initialize the parallel system
ParExit	Exit from the parallel system
ParSplit	Create a Compute grid of PEs
ParFree	Free a split communicator
ParScatter	Scatter global slice to local slices
ParGather	Gather local slices to one global
ParBeginTransfer	Initiate an all-to-all packet transfer
ParEndTransfer	Complete an all-to-all packet transfer
ParExchangeVector	Complete an all-to-all packet transfer
ParCollective	Collective operation across communicator

Other utilities can be added to this module as needs evolve.
Conceptually the intention is to aggregate as many of the MPI communication calls as possible into a well-maintained module. This will help avoid the occurrence of MPI spaghetti code.
This module is tailored to GEOS DAS and implements the design of Lucchesi/Mirin/Sawyer/Larson.

REVISION HISTORY:

```
97.02.01   Sawyer     Creation
97.07.22   Sawyer     Removal of DecompType related subroutines
97.08.13   Sawyer     Added ParScatter/Gather for Integers
97.09.26   Sawyer     Additions of Sparse communication primitives
97.12.01   Sawyer     Changed all MPI_SSEND to MPI_ISEND
97.12.23   Lucchesi   Added member variables IsIONode and InterComm
98.01.06   Sawyer     Additions from RL for I/O Nodes
98.02.02   Sawyer     Added the Cartesian data members
98.02.05   Sawyer     Removed the use of intercommunicators
98.02.23   Sawyer     Added ghosting utilities
98.02.25   Sawyer     Modified interface of BeginTransfer
98.03.03   Sawyer     Added Global ID number to public data members
98.03.25   Sawyer     Added documentation for walkthrough
98.04.16   Sawyer     Removed all use of MPI_CART (CommRow redefined)
98.07.23   Sawyer     Added ParGhost, ParPoleDot; ParBegin/EndGhost out
98.09.15   Sawyer     Added ParMerge, ParPoleGhost
98.09.17   Sawyer     Added ParSum, removed ParPoleDot
99.01.18   Sawyer     Minor cleaning
99.03.04   Sawyer     Revised SHMEM concept for Transfer
99.04.22   Sawyer     Removed COMMON for handles -- they are
                      always used in same program unit.
99.05.21   Sawyer     Reintroduced barriers in Scatter/Gather
99.06.03   Sawyer     USE_SHMEM revisions
99.12.10   Sawyer     ParInit now sets GID, Gsize
99.12.13   Sawyer     Version slimmed down for FVCCM release
00.06.14   Sawyer     Precision module now used
00.07.07   Sawyer     Removed 2D scatter/gather; simplified API
00.07.30   Sawyer     Full implementation with shared memory
00.08.09   Sawyer     Replaced ParSum with ParCollective
00.08.28   Sawyer     Moved LLNL 2D data to LLNL2DModule; new MLP impl
01.02.04   Sawyer     Added PatternType and related routines
01.02.12   Sawyer     Converted to free format
```

BUGS:

```
There are several MPI_Barriers at locations in the code.
These avoid potential race conditions which probably only occur
```

```
     if the number of real processors is less than the number of
     message passing processes.  Remove these barriers at your own risk
```

71.7.1 ParInit — Initialize the parallel execution

INTERFACE:

```
      SUBROUTINE ParInit (  )
```

USES:

```
      IMPLICIT NONE
```

DESCRIPTION:

Initializes the system. In MPI mode, call MPI_INIT if not done already. In USE_ARENAS mode, initialize the shared memory buffer.
This routine is the very *first* thing which is executed!

SYSTEM ROUTINES:

```
      MPI_INITIALIZED, MPI_INIT
```

REVISION HISTORY:

```
     97.03.20   Sawyer     Creation
     97.04.16   Sawyer     Cleaned up for walk-through
     97.07.03   Sawyer     Reformulated documentation
     00.07.23   Sawyer     Added shared memory arena implementation


! !LOCAL VARIABLES:
      INTEGER Ierror
      LOGICAL Flag
!
#if defined(USE_ARENAS)
#include <ulocks.h>
      INTEGER(i4) :: ipe, fork, getpid, master
      INTEGER(i8) :: nvars, extent(100), pnt(100)
      character*80 evalue

! Get the memory for the global variables
      extent(1) = MAX_PE * MAX_PE * 4
      extent(2) = MAX_BUF * 8
      extent(3) = MAX_BUF * 4
      nvars = 3
      call getmem(nvars,extent,pnt)
      buf01=pnt(1)
      buf02=pnt(2)
      buf03=pnt(3)

! Get the number of processes
      call getenv('N_MPI',evalue)
      read(evalue,*) Gsize
```

```
! Get the max number of threads per process
      call getenv('N_SMP',evalue)
      read(evalue,*) numcpu
      do ipe=1,Gsize
        numcps(ipe)=numcpu
      enddo
!!!      print *, "Starting MLP PILGRIM with", Gsize, "processes and",
!!!     &         numcpu, "threads per process"

! Calculate maximum blocksize and packetsize
      blocksize = MAX_BUF / Gsize
      packetsize = blocksize / MAX_PAX

! Destroy and recreate the environment
      master=getpid()
      call mp_destroy
      gid = 0
      do while ( getpid() .eq. master .and. gid < Gsize-1 )
        ierror=fork()
        gid = gid+1
      enddo
      if ( getpid() .eq. master ) gid = 0
!!!      call mp_set_threads(numcps(gid+1))
!
! Jim has more code here to pin the processes to the nodes
!
#else
!
!     Check if MPI is initialized.  If not, initialize.  No mpi_call
!
      CALL MPI_INITIALIZED( Flag, Ierror )
      CPP_ASSERT_F90( Ierror == 0 )
      IF ( .not. Flag ) then
        CALL MPI_INIT( ierror )
        CPP_ASSERT_F90( Ierror == 0 )
      ENDIF
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, Gsize, Ierror )
      CALL MPI_COMM_RANK( MPI_COMM_WORLD, GID, Ierror )
      CALL MPI_COMM_DUP( MPI_COMM_WORLD, CommGlobal, Ierror )
#endif
      RETURN
```

71.7.2 ParExit — Finalize the parallel execution

INTERFACE:

```
SUBROUTINE ParExit ( )
```

USES:

```
IMPLICIT NONE
```

DESCRIPTION:

All PEs, compute nodes and IO nodes alike meet here to terminate themselves. If someone does not check in, everything will hang here.
This routine is the very *last* thing which is executed!

LOCAL VARIABLES:

```
      INTEGER Ierror
```

SYSTEM ROUTINES:

```
      MPI_BARRIER, MPI_FINALIZE
```

REVISION HISTORY:

```
     97.03.20   Sawyer     Creation
     97.04.16   Sawyer     Cleaned up for walk-through
     97.07.03   Sawyer     Reformulated documentation
     00.07.23   Sawyer     Added shared memory arena implementation
```

```
#if !defined( USE_ARENAS )
      CALL MPI_BARRIER( MPI_COMM_WORLD, Ierror )
      CALL MPI_FINALIZE( Ierror )
#endif
      RETURN
```

71.7.3 ParSplit — Split into group for I/O and computation

INTERFACE:

```
      SUBROUTINE ParSplit( InComm, Color, InID, Comm, MyID, Nprocs )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      INTEGER, INTENT( IN )     :: InComm    ! Communicator to split
      INTEGER, INTENT( IN )     :: Color     ! Group label
      INTEGER, INTENT( IN )     :: InID      ! Input ID
```

OUTPUT PARAMETERS:

```
      INTEGER, INTENT( OUT )    :: Comm      ! Split communicator
      INTEGER, INTENT( OUT )    :: MyID      ! Group label
      INTEGER, INTENT( OUT )    :: Nprocs    ! Number of PEs in my group
```

DESCRIPTION:

This routine splits the PEs into groups. This is currently only supported in MPI mode. Read the chapter on MPI_COMM_SPLIT thoroughly.

SYSTEM ROUTINES:

```
      MPI_COMM_SPLIT, MPI_COMM_SIZE, MPI_COMM_RANK
```

REVISION HISTORY:

```
     97.03.20   Sawyer     Creation
     97.04.16   Sawyer     Cleaned up for walk-through
     97.07.03   Sawyer     Reformulated documentation
     97.12.01   Sawyer     Xnodes and Ynodes are explicit arguments
     97.12.23   Lucchesi   Added call to MPI_INTERCOMM_CREATE
     98.01.06   Sawyer     Additions from RL for I/O Nodes
     98.02.02   Sawyer     Added the Cartesian information
     98.02.05   Sawyer     Removed the use of intercommunicators
     98.04.16   Sawyer     Removed all use of MPI_CART (CommRow redefined)
     99.01.10   Sawyer     CommRow now defined for all rows
     00.07.09   Sawyer     Removed 2D computational mesh
     00.08.08   Sawyer     Redefined as wrapper to mpi_comm_split

! !LOCAL VARIABLES:
      INTEGER  Ierror

      CPP_ENTER_PROCEDURE( "PARSPLIT" )
#if !defined( USE_ARENAS )
!
!     Split the communicators
!
      CALL MPI_COMM_SPLIT( InComm, Color, InID, Comm, Ierror )
      IF ( Comm .ne. MPI_COMM_NULL ) THEN
        CALL MPI_COMM_RANK( Comm, MyID, Ierror )
        CALL MPI_COMM_SIZE( Comm, Nprocs, Ierror )
      ELSE
!
!     This PE does not participate: mark with impossible values
!
        MyID = -1
        Nprocs = -1
      ENDIF
#endif

      CPP_LEAVE_PROCEDURE( "PARSPLIT" )
      RETURN
```

71.7.4 ParFree — Free a communicator

INTERFACE:

```
      SUBROUTINE ParFree( InComm )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      INTEGER InComm
```

DESCRIPTION:

This routine frees a communicator created with ParSplit

REVISION HISTORY:

```
     97.09.11   Sawyer     Creation, to complement ParSplit
     00.07.24   Sawyer     Revamped ParMerge into a free communicator
```

LOCAL VARIABLES:

```
      INTEGER  Ierror

     CPP_ENTER_PROCEDURE( "PARFREE" )
!
#if !defined( USE_ARENAS )
     CALL MPI_COMM_FREE( InComm, Ierror )
#endif
     CPP_LEAVE_PROCEDURE( "PARFREE" )
     RETURN
```

71.7.5 ParPatternGhost — Create pattern for given ghosting

INTERFACE:

```
      SUBROUTINE ParPatternGhost( InComm, Ghost, Pattern )
```

USES:

```
      USE decompmodule, ONLY : DecompGlobalToLocal, DecompLocalToGlobal
      USE ghostmodule, ONLY : GhostType, GhostInfo
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      INTEGER,  INTENT( IN )               :: InComm  ! # of PEs
      TYPE(GhostType),  INTENT( IN )       :: Ghost   ! # of PEs
```

OUTPUT PARAMETERS:

```
      TYPE(ParPatternType), INTENT( OUT )  :: Pattern ! Comm Pattern
```

DESCRIPTION:

This routine contructs a communication pattern from the ghost region definition. That is, the resulting communication pattern can be used in ParBegin/EndTransfer with the ghosted arrays as inputs.

SYSTEM ROUTINES:

```
      MPI_TYPE_INDEXED
```

REVISION HISTORY:

```
     01.02.10   Sawyer     Creation
     01.06.02   Sawyer     Renamed ParPatternGhost

! !LOCAL VARIABLES:
      INTEGER  i, j, ipe, pe, Iam, GroupSize, Num, Length, Ptr, Ierror
      INTEGER  Global, End, Local, GlobalSize, LocalSize, BorderSize
      INTEGER, ALLOCATABLE :: InVector(:), OutVector(:)
      INTEGER, ALLOCATABLE :: LenInVector(:), LenOutVector(:)

      CPP_ENTER_PROCEDURE( "PARPATTERNGHOST" )

!
! First request the needed ghost values from other processors.
!
#if !defined( USE_ARENAS )
      CALL MPI_COMM_SIZE( InComm, GroupSize, Ierror )
      CALL MPI_COMM_RANK( InComm, Iam, Ierror )
#endif

      ALLOCATE( Pattern%SendDesc( GroupSize ) )
      ALLOCATE( Pattern%RecvDesc( GroupSize ) )

!
! Temporary variables
!
      ALLOCATE( LenInVector( GroupSize ) )
      ALLOCATE( LenOutVector( GroupSize ) )

      CALL GhostInfo( Ghost,GroupSize,GlobalSize,LocalSize,BorderSize )
      ALLOCATE( InVector( 2*BorderSize ) )
      ALLOCATE( OutVector( 2*LocalSize ) )

!
! A rather complicated loop to define the local ghost region.
! The concept is the following:  go through all the points in the
! border data structure.   It contains global indices of the points
! which have to be copied over from neighboring PEs.  These indices
! are collected into InVector for transmission to those PEs, in
! effect informing them of the local PEs requirements.
!
! A special case is supported:  if the ghost domain wraps around
! onto the domain of the local PE!  This is very tricky, because
! the index space in both Ghost%Border and Ghost%Local MUST be
! unique for DecompGlobalToLocal to work.   Solution:  ghost
! points are marked with the negative value of the needed domain
! value in both Ghost%Border and Ghost%Local.  These are "snapped
! over" to the true global index with the ABS function, so that
! they can be subsequently found in the true local domain.
!
      j = 1
      DO ipe=1, GroupSize
        Num = SIZE(Ghost%Border%Head(ipe)%StartTags)
        Length = 0
        DO i = 1, Num
          Global = Ghost%Border%Head(ipe)%StartTags(i)
          IF ( Global /= 0 ) THEN
```

```
            Length = Length + 1
            End    = Ghost%Border%Head(ipe)%EndTags(i)
            InVector(j) = ABS(Global)
            InVector(j+1) = ABS(End)
            CALL DecompGlobalToLocal( Ghost%Local, Global, Local, Pe )
            OutVector(Length) = Local-1                ! Zero-based address
            OutVector(Length+Num) = End - Global+1     ! Chunk size
            j = j + 2
          ENDIF
        ENDDO
        LenInVector( ipe ) = 2*Length

!
! Set the receive buffer descriptor
!
#if defined(DEBUG_PARPATTERNGHOST)
        print *,"Iam",Iam,"Pe",Ipe-1,"Lens",OutVector(Num+1:Num+Length), &
     &       "Displacements", OutVector(1:Length)
#endif
#if defined( USE_ARENAS )
! This code is currently untested
         ALLOCATE( Pattern%RecvDesc(ipe)%Displacements(Num) )
         ALLOCATE( Pattern%RecvDesc(ipe)%BlockSizes(Num) )
         DO i=1, Num
           Pattern%RecvDesc(ipe)%Displacements(i) = OutVector(i)
           Pattern%RecvDesc(ipe)%BlockSizes(i)    = OutVector(Num+i)
         ENDDO
#else
        CALL MPI_TYPE_INDEXED( Length, OutVector(Num+1), OutVector,      &
     &     CPP_MPI_REAL, Ptr, Ierror )
        CALL MPI_TYPE_COMMIT( Ptr, Ierror )
        Pattern%RecvDesc( ipe ) = Ptr
#endif
      ENDDO

!
! Everybody exchanges the needed information
!
#if defined(DEBUG_PARPATTERNGHOST)
      print *, "iam", iam, "In", LenInVector,                            &
     &           InVector( 1:SUM(LenInVector) )
#endif
      CALL ParExchangeVectorInt( InComm, LenInVector, InVector,          &
     &                                LenOutVector, OutVector )
#if defined(DEBUG_PARPATTERNGHOST)
      print *, "iam", iam, "Out", LenOutVector,                          &
     &           OutVector( 1:SUM(LenOutVector) )
#endif
!
! Now everyone has the segments which need to be sent to the
! immediate neighbors.  Save these in PatternType.
!
      j = 1
      DO ipe = 1, GroupSize
        Num = LenOutVector(ipe) / 2
```

```
        DO i = 1, Num
          CALL DecompGlobalToLocal( Ghost%Local,OutVector(j),Local,pe )
          InVector(i) = Local-1
          InVector(i+Num) = OutVector(j+1) - OutVector(j) + 1
          j = j + 2
        ENDDO
#if defined(DEBUG_PARPATTERNGHOST)
        print *, "Iam", Iam, "To", ipe-1, "InVector",                    &
     &        InVector(1:Num), "block size", InVector(Num+1:2*Num)
#endif
#if defined( USE_ARENAS )
         ALLOCATE( Pattern%SendDesc(ipe)%Displacements(Num) )
         ALLOCATE( Pattern%SendDesc(ipe)%BlockSizes(Num) )
         DO i=1, Num
           Pattern%SendDesc(ipe)%Displacements(i) = InVector(i)
           Pattern%SendDesc(ipe)%BlockSizes(i)    = InVector(Num+i)
         ENDDO
#else
        CALL MPI_TYPE_INDEXED( Num, InVector(Num+1), InVector,           &
     &      CPP_MPI_REAL, Ptr, Ierror )
        CALL MPI_TYPE_COMMIT( Ptr, Ierror )
        Pattern%SendDesc( ipe ) = Ptr
#endif
      ENDDO

!
! Copy the communicator into the pattern
!
      CALL MPI_COMM_DUP( InComm, Pattern%Comm, Ierror )
      CALL MPI_COMM_RANK( InComm, Pattern%Iam, Ierror )
      CALL MPI_COMM_SIZE( InComm, Pattern%Size, Ierror )
!
! Clean up the locally allocate variables
!
      DEALLOCATE( OutVector )
      DEALLOCATE( InVector )
      DEALLOCATE( LenOutVector )
      DEALLOCATE( LenInVector )

      CPP_LEAVE_PROCEDURE( "PARPATTERNGHOST" )
      RETURN
```

71.7.6 ParPatternDecompToDecomp — Create pattern between decomps

INTERFACE:

```
      SUBROUTINE ParPatternDecompToDecomp( InComm, DA, DB, Pattern )
```

USES:

```
      USE decompmodule, ONLY : DecompType, DecompGlobalToLocal, DecompInfo
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
INTEGER,  INTENT( IN )              :: InComm  ! # of PEs
TYPE(DecompType),  INTENT( IN )     :: DA      ! Source Decomp Desc
TYPE(DecompType),  INTENT( IN )     :: DB      ! Target Decomp Desc
```

OUTPUT PARAMETERS:

```
TYPE(ParPatternType), INTENT( OUT )  :: Pattern ! Comm Pattern
```

DESCRIPTION:

This routine contructs a communication pattern for a transformation from one decomposition to another, i.e., a so-called "transpose". The resulting communication pattern can be used in ParBegin/EndTransfer with the decomposed arrays as inputs.

SYSTEM ROUTINES:

BUGS:

```
Under development
```

REVISION HISTORY:

```
01.05.29   Sawyer     Creation from RedistributeCreate
01.07.13   Sawyer     Rewritten to minimize DecompGlobalToLocal
```

```
! !LOCAL VARIABLES:
      LOGICAL NewIpe
      INTEGER I, J, Tag, Local, Pe, LenB, JB, Ipe, Num, Inc, Off
      INTEGER Ptr                                ! Pointer type
      INTEGER GroupSize, Iam, Ierror
      INTEGER OldPe, TotalPtsA, NpesA, TotalPtsB, NpesB

      INTEGER, ALLOCATABLE :: Count(:)           ! # segments for each recv PE
      INTEGER, ALLOCATABLE :: CountOut(:)        ! # segments for each send PE

      INTEGER, ALLOCATABLE :: DisplacementsA(:)  ! Generic displacements
      INTEGER, ALLOCATABLE :: BlockSizesA(:)     ! Generic block sizes
      INTEGER, ALLOCATABLE :: LocalA(:)          ! Generic Local indices

      INTEGER, ALLOCATABLE :: DisplacementsB(:)  ! Displacements for B
      INTEGER, ALLOCATABLE :: BlockSizesB(:)     ! Block sizes for B
      INTEGER, ALLOCATABLE :: LocalB(:)          ! Local indices for B
      INTEGER, ALLOCATABLE :: PeB(:)             ! Processor element numbers

      CPP_ENTER_PROCEDURE( "PARPATTERNDECOMPTODECOMP" )

      CALL DecompInfo( DA, NpesA, TotalPtsA )
      CALL DecompInfo( DB, NpesB, TotalPtsB )

#if defined( USE_ARENAS )
! Not yet supported: provoke an error
      You should not be compiling for shared memory arenas
#else
      CALL MPI_COMM_SIZE( InComm, GroupSize, Ierror )
      CALL MPI_COMM_RANK( InComm, Iam, Ierror )
```

```
      CALL MPI_COMM_DUP( InComm, Pattern%Comm, Ierror )
#endif
      Pattern%Size = GroupSize
      Pattern%Iam  = Iam
!
! Allocate the number of entries and list head arrays
!
      CPP_ASSERT_F90( NpesA .EQ. GroupSize )
      CPP_ASSERT_F90( NpesB .EQ. GroupSize )

!
! Allocate the patterns
!
      ALLOCATE( Pattern%SendDesc( GroupSize ) )
      ALLOCATE( Pattern%RecvDesc( GroupSize ) )

!
! Local allocations
!
      ALLOCATE( DisplacementsA( TotalPtsA ) )   ! Allocate for worst case
      ALLOCATE( BlockSizesA( TotalPtsA ) )      ! Allocate for worst case
      ALLOCATE( LocalA( TotalPtsA ) )           ! Allocate for worst case

      ALLOCATE( DisplacementsB( TotalPtsB ) )   ! Allocate for worst case
      ALLOCATE( BlockSizesB( TotalPtsB ) )      ! Allocate for worst case
      ALLOCATE( LocalB( TotalPtsA ) )           ! Allocate for worst case
      ALLOCATE( PeB( TotalPtsB ) )              ! Allocate for worst case

      ALLOCATE( Count( GroupSize ) )
      ALLOCATE( CountOut( GroupSize ) )

      JB        = 0
      Count     = 0
      LenB      = 0

      NewIpe = .TRUE.
      Num    = 0
      Inc    = 0

!
! Parse through all the tags in the local segment
      DO J = 1, SIZE( DB%Head(iam+1)%StartTags )
        OldPe     = -1         ! Set PE undefined
        DO Tag=DB%Head(iam+1)%StartTags(J), DB%Head(iam+1)%EndTags(J)
!
! Determine the index and PE of this entry on A. This might be inlined later
!
          CALL DecompGlobalToLocal( DA, Tag, Local, Pe )

!
! If ipe-1 is my id, then this is an entry ipe will receive from Pe
!
          IF ( Pe /= OldPe ) THEN
            OldPe   = Pe
            IF ( jb > 0 ) THEN
```

```
            BlockSizesB(jb) = LenB
            LenB = 0
          ENDIF
          jb = jb+1                     ! increment the segment index
          DisplacementsB(jb) = Inc      ! Zero-based offset of local segment
          LocalB(jb) = Local-1          ! The local index (zero-based)
          PeB(jb) = Pe                  ! Note the ID of the sender
          Count(Pe+1) = Count(Pe+1)+1 ! Increment counter of segments
        ENDIF
        LenB = LenB+1                   ! Good -- segment is getting longer
        Inc = Inc+1                     ! Increment local index
      ENDDO
    ENDDO
!
! Clean up
!
    BlockSizesB(jb) = LenB
    print *, iam, "BlockSizes", BlockSizesB(1:jb), DisplacementsB(1:jb), PeB(1:jb), Count

    CPP_ASSERT_F90( JB .LE. GlobalSize )
!
! Now create the pattern from the displacements and block sizes
!
    Inc = 0
    DO ipe = 1, GroupSize
!
! Find the segments which are relevant for the sender ipe
! Make compact arrays BlockSizes and Displacements
!
      DO j = 1, jb
        IF ( PeB(j) == ipe-1 ) THEN
          Inc = Inc + 1
          BlockSizesA(Inc) = BlockSizesB(j)
          DisplacementsA(Inc) = DisplacementsB(j)
          LocalA(Inc)      = LocalB(j)
        ENDIF
      ENDDO
    ENDDO

!
! Create the receiver communication pattern
!
    Off = 0
    DO ipe = 1, GroupSize
      Num = Count(ipe)
      print *, "Receiver Iam", Iam, "Ipe", Ipe-1, "Num", Num, &
               "Displacements", DisplacementsA(Off+1:Off+Num), &
               "BlockSizes", BlockSizesA(Off+1:Off+Num)
#if defined( USE_ARENAS )
      ALLOCATE( Pattern%RecvDesc(ipe)%Displacements(Num) )
      ALLOCATE( Pattern%RecvDesc(ipe)%BlockSizes(Num) )
      DO i=1, Num
        Pattern%RecvDesc(ipe)%Displacements(i) = DisplacementsA(i+Off)
        Pattern%RecvDesc(ipe)%BlockSizes(i)    = BlockSizesA(i+Off)
      ENDDO
```

```
#else
        CALL MPI_TYPE_INDEXED( Num, BlockSizesA(Off+1),DisplacementsA(Off+1), &
     &                         CPP_MPI_REAL, Ptr, Ierror )
        Pattern%RecvDesc( ipe ) = Ptr
#endif
        Off = Off + Num
      ENDDO

!
! Now communicate what the receiver is expecting to the sender
!
      CALL ParExchangeVectorInt( InComm, Count, LocalA,                 &
     &                           CountOut, DisplacementsB  )
      CALL ParExchangeVectorInt( InComm, Count, BlockSizesA,            &
     &                           CountOut, BlockSizesB )

!
! Sender A: BlockSizes and Displacements can now be stored
!
      Off = 0
      DO ipe=1, GroupSize
        Num = CountOut(ipe)
        print *, "Sender Iam", Iam, "Ipe", Ipe-1, "Num", Num,  &
                 "Displacements", DisplacementsB(Off+1:Off+Num), &
                 "BlockSizes", BlockSizesB(Off+1:Off+Num)

#if defined( USE_ARENAS )
        ALLOCATE( Pattern%SendDesc(ipe)%Displacements(Num) )
        ALLOCATE( Pattern%SendDesc(ipe)%BlockSizes(Num) )
        DO i=1, Num
          Pattern%SendDesc(ipe)%Displacements(i) = DisplacementsB(i+Off)
          Pattern%SendDesc(ipe)%BlockSizes(i)    = BlockSizesB(i+Off)
        ENDDO
#else
        CALL MPI_TYPE_INDEXED( Num, BlockSizesB(Off+1),DisplacementsB(Off+1),&
     &                         CPP_MPI_REAL, Ptr, Ierror )
        Pattern%SendDesc( ipe ) = Ptr
#endif
        Off = Off + Num
      ENDDO

      DEALLOCATE( CountOut )
      DEALLOCATE( Count )

      DEALLOCATE( PeB )
      DEALLOCATE( LocalB )
      DEALLOCATE( BlockSizesB )
      DEALLOCATE( DisplacementsB )

      DEALLOCATE( LocalA )
      DEALLOCATE( BlockSizesA )
      DEALLOCATE( DisplacementsA )

      CPP_LEAVE_PROCEDURE( "PARPATTERNDECOMPTODECOMP" )
      RETURN
```

71.7.7 ParPatternDecompToGhost — Create pattern decomp to ghost

INTERFACE:

```
SUBROUTINE ParPatternDecompToGhost( InComm, DA, GB, Pattern )
```

USES:

```
USE decompmodule, ONLY : DecompType, DecompGlobalToLocal,         &
                         DecompInfo
USE ghostmodule, ONLY : GhostType, GhostInfo
IMPLICIT NONE
```

INPUT PARAMETERS:

```
INTEGER,  INTENT( IN )               :: InComm  ! # of PEs
TYPE(DecompType),  INTENT( IN )      :: DA      ! Source Ghost Desc
TYPE(GhostType),  INTENT( IN )       :: GB      ! Target Ghost Desc
```

OUTPUT PARAMETERS:

```
TYPE(ParPatternType), INTENT( OUT )  :: Pattern ! Comm Pattern
```

DESCRIPTION:

This routine contructs a communication pattern for a transformation from decomposition to a ghosted decomposition, i.e., a so-called "transpose". The resulting communication pattern can be used in ParBegin/EndTransfer with the decomposed arrays as inputs.

SYSTEM ROUTINES:

BUGS:

```
Under development
```

REVISION HISTORY:

```
12.07.01   Sawyer     Creation from ParPatternDecompToDecomp

! !LOCAL VARIABLES:
      LOGICAL NewIpe
      INTEGER I, J, Tag, Local, Pe, LenB, JB, Ipe, Num, Inc, Off
      INTEGER Ptr                                ! Pointer type
      INTEGER GroupSize, Iam, Ierror
      INTEGER OldPe, TotalPtsA, NpesA
      INTEGER GlobalSizeB, LocalSizeB, BorderSizeB, NpesB

      INTEGER, ALLOCATABLE :: Count(:)           ! # segments for each recv PE
      INTEGER, ALLOCATABLE :: CountOut(:)        ! # segments for each send PE

      INTEGER, ALLOCATABLE :: DisplacementsA(:)  ! Generic displacements
      INTEGER, ALLOCATABLE :: BlockSizesA(:)     ! Generic block sizes
```

```
      INTEGER, ALLOCATABLE :: LocalA(:)          ! Generic Local indices

      INTEGER, ALLOCATABLE :: DisplacementsB(:)  ! Displacements for B
      INTEGER, ALLOCATABLE :: BlockSizesB(:)     ! Block sizes for B
      INTEGER, ALLOCATABLE :: LocalB(:)          ! Local indices for B
      INTEGER, ALLOCATABLE :: PeB(:)             ! Processor element numbers

      CPP_ENTER_PROCEDURE( "PARPATTERNDECOMPTOGHOST" )

      CALL DecompInfo( DA, NpesA, TotalPtsA )
      CALL GhostInfo( GB, NpesB, GlobalSizeB, LocalSizeB, BorderSizeB )

#if defined( USE_ARENAS )
! Not yet supported: provoke an error
      You should not be compiling for shared memory arenas
#else
      CALL MPI_COMM_SIZE( InComm, GroupSize, Ierror )
      CALL MPI_COMM_RANK( InComm, Iam, Ierror )
      CALL MPI_COMM_DUP( InComm, Pattern%Comm, Ierror )
#endif
      Pattern%Size = GroupSize
      Pattern%Iam  = Iam
!
! Allocate the number of entries and list head arrays
!
      CPP_ASSERT_F90( NpesA .EQ. GroupSize )
      CPP_ASSERT_F90( NpesB .EQ. GroupSize )

!
! Allocate the patterns
!
      ALLOCATE( Pattern%SendDesc( GroupSize ) )
      ALLOCATE( Pattern%RecvDesc( GroupSize ) )

!
! Local allocations
!
      ALLOCATE( DisplacementsA( TotalPtsA ) )   ! Allocate for worst case
      ALLOCATE( BlockSizesA( TotalPtsA ) )      ! Allocate for worst case
      ALLOCATE( LocalA( TotalPtsA ) )           ! Allocate for worst case

      ALLOCATE( DisplacementsB( GlobalSizeB ) ) ! Allocate for worst case
      ALLOCATE( BlockSizesB( GlobalSizeB ) )    ! Allocate for worst case
      ALLOCATE( LocalB( GlobalSizeB ) )         ! Allocate for worst case
      ALLOCATE( PeB( GlobalSizeB ) )            ! Allocate for worst case

      ALLOCATE( Count( GroupSize ) )
      ALLOCATE( CountOut( GroupSize ) )

      JB        = 0
      Count     = 0
      LenB      = 0

      NewIpe = .TRUE.
      Num    = 0
```

```
      Inc    = 0

!
! Parse through all the tags in the local segment
      DO J = 1, SIZE( GB%Local%Head(iam+1)%StartTags )
        OldPe     = -1         ! Set PE undefined
        DO Tag=GB%Local%Head(iam+1)%StartTags(J),                        &
                GB%Local%Head(iam+1)%EndTags(J)
!
! Determine the index and PE of this entry on A. This might be inlined later
!
          CALL DecompGlobalToLocal( DA, Tag, Local, Pe )

!
! If ipe-1 is my id, then this is an entry ipe will receive from Pe
!
          IF ( Pe /= OldPe ) THEN
            OldPe   = Pe
            IF ( jb > 0 ) THEN
              BlockSizesB(jb) = LenB
              LenB = 0
            ENDIF
            jb = jb+1                     ! increment the segment index
            DisplacementsB(jb) = Inc      ! Zero-based offset of local segment
            LocalB(jb) = Local-1          ! Local indices (zero-based)
            PeB(jb) = Pe                  ! Note the ID of the sender
            Count(Pe+1) = Count(Pe+1)+1 ! Increment counter of segments
          ENDIF
          LenB = LenB+1                   ! Good -- segment is getting longer
          Inc = Inc+1                     ! Increment local index
        ENDDO
      ENDDO
!
! Clean up
!
      BlockSizesB(jb) = LenB

      CPP_ASSERT_F90( JB .LE. GlobalSize )
!
! Now create the pattern from the displacements and block sizes
!
      Inc = 0
      DO ipe = 1, GroupSize
!
! Find the segments which are relevant for the sender ipe
! Make compact arrays BlockSizes and Displacements
!
        DO j = 1, jb
          IF ( PeB(j) == ipe-1 ) THEN
            Inc = Inc + 1
            BlockSizesA(Inc) = BlockSizesB(j)
            DisplacementsA(Inc) = DisplacementsB(j)
            LocalA(Inc)      = LocalB(j)
          ENDIF
        ENDDO
```

```
      ENDDO

      Off = 0
      DO ipe = 1, GroupSize
        Num = Count(ipe)
        print *, "Receiver Iam", Iam, "Ipe", Ipe-1, "Num", Num, &
                 "Displacements", DisplacementsA(Off+1:Off+Num), &
                 "BlockSizes", BlockSizesA(Off+1:Off+Num)

!
! Create the receiver communication pattern
!
#if defined( USE_ARENAS )
        ALLOCATE( Pattern%RecvDesc(ipe)%Displacements(Num) )
        ALLOCATE( Pattern%RecvDesc(ipe)%BlockSizes(Num) )
        DO i=1, Num
          Pattern%RecvDesc(ipe)%Displacements(i) = DisplacementsA(i+Off)
          Pattern%RecvDesc(ipe)%BlockSizes(i)    = BlockSizesA(i+Off)
        ENDDO
#else
        CALL MPI_TYPE_INDEXED( Num, BlockSizesA(Off+1),DisplacementsA(Off+1), &
     &                         CPP_MPI_REAL, Ptr, Ierror )
        Pattern%RecvDesc( ipe ) = Ptr
#endif
        Off = Off + Num
      ENDDO

!
! Now communicate what the receiver is expecting to the sender
!
      CALL ParExchangeVectorInt( InComm, Count, LocalA,                 &
     &                           CountOut, DisplacementsB  )
      CALL ParExchangeVectorInt( InComm, Count, BlockSizesA,            &
     &                           CountOut, BlockSizesB )

!
! Sender A: BlockSizes and Displacements can now be stored
!
      Off = 0
      DO ipe=1, GroupSize
        Num = CountOut(ipe)
        print *, "Sender Iam", Iam, "Ipe", Ipe-1, "Num", Num,           &
                 "Displacements", DisplacementsB(Off+1:Off+Num),        &
                 "BlockSizes", BlockSizesB(Off+1:Off+Num)
#if defined( USE_ARENAS )
        ALLOCATE( Pattern%SendDesc(ipe)%Displacements(Num) )
        ALLOCATE( Pattern%SendDesc(ipe)%BlockSizes(Num) )
        DO i=1, Num
          Pattern%SendDesc(ipe)%Displacements(i) = DisplacementsB(i+Off)
          Pattern%SendDesc(ipe)%BlockSizes(i)    = BlockSizesB(i+Off)
        ENDDO
#else
        CALL MPI_TYPE_INDEXED( Num, BlockSizesB(Off+1), DisplacementsB(Off+1),&
                               CPP_MPI_REAL, Ptr, Ierror )
        Pattern%SendDesc( ipe ) = Ptr
```

```
#endif
      ENDDO

      DEALLOCATE( CountOut )
      DEALLOCATE( Count )

      DEALLOCATE( PeB )
      DEALLOCATE( LocalB )
      DEALLOCATE( BlockSizesB )
      DEALLOCATE( DisplacementsB )

      DEALLOCATE( LocalA )
      DEALLOCATE( BlockSizesA )
      DEALLOCATE( DisplacementsA )

      CPP_LEAVE_PROCEDURE( "PARPATTERNDECOMPTOGHOST" )
      RETURN
```

71.7.8 ParPatternFree — Free the communication pattern

INTERFACE:

```
      SUBROUTINE ParPatternFree( InComm, Pattern )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      INTEGER,  INTENT( IN )                 :: InComm  ! # of PEs
```

INPUT/OUTPUT PARAMETERS:

```
      TYPE(ParPatternType), INTENT( INOUT )  :: Pattern ! Comm Pattern
```

DESCRIPTION:

This routine frees a communication pattern.

SYSTEM ROUTINES:

```
      MPI_TYPE_FREE
```

BUGS:

```
      The MPI_TYPE_FREE statement does not seem to work with FFC
```

REVISION HISTORY:

```
     01.02.10   Sawyer     Creation

! !LOCAL VARIABLES:
      INTEGER  ipe, GroupSize, Pointer, Ierror
```

```
      CPP_ENTER_PROCEDURE( "PARPATTERNFREE" )

!
! First request the needed ghost values from other processors.
!
#if defined( USE_ARENAS )
      DO ipe, Pattern%Size
        DEALLOCATE( Pattern%RecvDesc(ipe)%Displacements )
        DEALLOCATE( Pattern%RecvDesc(ipe)%BlockSizes )
        DEALLOCATE( Pattern%SendDesc(ipe)%Displacements )
        DEALLOCATE( Pattern%SendDesc(ipe)%BlockSizes )
      ENDDO
      DEALLOCATE( Pattern%RecvDesc )
      DEALLOCATE( Pattern%SendDesc )
#else
      CALL MPI_COMM_SIZE( InComm, GroupSize, Ierror )
!
! Free all the MPI derived types
!
      DO ipe=1, Pattern%Size
        Pointer = Pattern%SendDesc(ipe)
        CALL MPI_TYPE_FREE( Pointer, Ierror )
        Pointer = Pattern%RecvDesc(ipe)
        CALL MPI_TYPE_FREE( Pointer, Ierror )
      ENDDO
#endif

      DEALLOCATE( Pattern%SendDesc )
      DEALLOCATE( Pattern%RecvDesc )

      CPP_LEAVE_PROCEDURE( "PARPATTERNFREE" )
      RETURN
```

71.7.9 ParScatterReal — Scatter slice to all PEs

INTERFACE:

```
       SUBROUTINE ParScatterReal ( InComm, Root, Slice, Decomp, Local )
```

USES:

```
       USE decompmodule, ONLY:  DecompType, Lists
       IMPLICIT NONE
```

INPUT PARAMETERS:

```
       INTEGER, INTENT( IN )          :: InComm       ! Communicator
       INTEGER, INTENT( IN )          :: Root         ! Root PE
       REAL(CPP_REAL), INTENT( IN )   :: Slice(*)     ! Global Slice
       TYPE(DecompType), INTENT( IN ) :: Decomp       ! Decomp information
```

OUTPUT PARAMETERS:

```
      REAL(CPP_REAL), INTENT( OUT )  :: Local(*)     ! Local Slice
```

DESCRIPTION:

Given a decomposition of the domain, dole out a slice (one-dimensional array) to all the constituent PEs as described by the decomposition Decomp.

SYSTEM ROUTINES:

```
      MPI_ISEND, MPI_RECV, MPI_COMM_RANK
```

REVISION HISTORY:

```
     97.04.14   Sawyer     Creation
     97.04.16   Sawyer     Cleaned up for walk-through
     97.05.01   Sawyer     Use Decomp%Comm for all local info
     97.05.18   Sawyer     DecompType has moved to ParUtilitiesTypes
     97.05.29   Sawyer     Changed 2-D arrays to 1-D
     97.07.03   Sawyer     Reformulated documentation
     97.07.22   Sawyer     DecompType has moved to DecompModule
     97.12.01   Sawyer     Changed MPI_SSEND to MPI_ISEND
     97.12.05   Sawyer     Added InComm and Root as arguments
     97.12.05   Sawyer     Added logic to support intercommunicators
     98.01.24   Sawyer     Removed dependence on MPI derived types TESTED
     98.02.05   Sawyer     Removed the use of intercommunicators
     98.03.30   Sawyer     Stats dimension corrected: Gsize*MPI_STATUS_SIZE
     99.01.19   Sawyer     Dropped assumed-size arrays
     00.07.07   Sawyer     Removed "1D" references
     00.07.23   Sawyer     Implementation with shared memory arenas

! !LOCAL VARIABLES:

      INTEGER Ierror, I, J, K, L, Iam, GroupSize, Reqs( Gsize )
#if !defined( USE_ARENAS )
      INTEGER Status( MPI_STATUS_SIZE ), Stats( Gsize*MPI_STATUS_SIZE )
      REAL(CPP_REAL), ALLOCATABLE    :: SendBuf(:)
#endif
!
      CPP_ENTER_PROCEDURE( "PARSCATTERREAL" )
!
#if defined( USE_ARENAS )
!
! Pull the local process information out of the communicator
!
!!!      Iam = MOD( Comm, MAX_PES )
!!!      L   = Comm / MAX_PES
!!!      GroupSize = MOD( L, MAX_PES ) + 1
!!!      L   = L / MAX_PES
!!!      Color = MOD( L, MAX_PES )

!
! For now, Iam and GroupSize take on the global communicator values
!
      Iam = GID
```

```
      GroupSize = Gsize
      IF ( Iam .EQ. Root ) THEN
        L = 0
        DO I = 1, GroupSize
!
! Pick out the array sections to be sent.
! This is the inverse of the operation in ParGather
!
          DO J = 1, SIZE( Decomp%HEAD(I)%StartTags )
            DO K = Decomp%HEAD(I)%StartTags(J),Decomp%HEAD(I)%EndTags(J)
              L = L+1
              DataBuf(L) = Slice(K)
            ENDDO
          ENDDO
        ENDDO
      ENDIF
!
! Barrier: all PEs participate!
!
      CALL mlp_barrier(gid,gsize)

!
! All receive from the root.
!

! The local array may be larger than that specified in the decomposition
!
      L = 0
      IF ( Iam .GT. 0 ) L = SUM( Decomp%NumEntries(1:Iam) )
      DO I=1, Decomp%NumEntries(Iam+1)
        Local( I ) = DataBuf( L + I )
      ENDDO
!
! The following is needed to ensure that DataBuf can now be reused
!
      CALL mlp_barrier(gid,gsize)
#else
      CALL MPI_COMM_RANK( InComm, Iam, Ierror )
      CALL MPI_COMM_SIZE( InComm, GroupSize, Ierror )

      IF ( Iam .EQ. Root ) THEN
        ALLOCATE( SendBuf( SUM( Decomp%NumEntries ) ) )
        L = 0
        DO I = 1, GroupSize
!
! Pick out the array sections to be sent.
! This is the inverse of the operation in ParGather
!
          DO J = 1, SIZE( Decomp%HEAD(I)%StartTags )
            DO K = Decomp%HEAD(I)%StartTags(J),Decomp%HEAD(I)%EndTags(J)
              L = L+1
              SendBuf(L) = Slice(K)
            ENDDO
          ENDDO
!
```

```
! This is a non-blocking send. SendBuf cannot be immediately deallocated
!
! WARNING: F90-MPI inconsistency: make sure the indexing below always works
!
          CALL MPI_ISEND( SendBuf(L-Decomp%NumEntries(I)+1),             &
     &                    Decomp%NumEntries(I), CPP_MPI_REAL,            &
     &                    I-1, 0, InComm, Reqs(I), Ierror )

        ENDDO
      ENDIF

!
! All receive from the root.
!
! The local array may be larger than that specified in the decomposition
!
      CALL MPI_RECV( Local, Decomp%NumEntries(Iam+1),                    &
     &               CPP_MPI_REAL,                                       &
     &               Root, 0, InComm, Status, Ierror )
!
! Experience shows that we should wait for all the non-blocking
! PEs to check in, EVEN THOUGH THE MPI_RECV HAS COMPLETED !!
!
      IF ( Iam .EQ. Root ) THEN
        CALL MPI_WAITALL( GroupSize, Reqs, Stats, Ierror )
        DEALLOCATE( SendBuf )
      ENDIF

!
! The following may be needed on some platforms to avoid an MPI bug.
!
      CALL MPI_BARRIER( InComm, Ierror )
#endif
      CPP_LEAVE_PROCEDURE( "PARSCATTERREAL" )
      RETURN
```

71.7.10 ParScatterInt — Scatter slice to all PEs

INTERFACE:

```
SUBROUTINE ParScatterInt ( InComm, Root, Slice, Decomp, Local )
```

USES:

```
USE decompmodule, ONLY:  DecompType, Lists
IMPLICIT NONE
```

INPUT PARAMETERS:

```
INTEGER, INTENT( IN )          :: InComm       ! Communicator
INTEGER, INTENT( IN )          :: Root         ! Root PE
INTEGER, INTENT( IN )          :: Slice(*)     ! Global Slice
```

```
      TYPE(DecompType), INTENT( IN ) :: Decomp       ! Decomp information
```

OUTPUT PARAMETERS:

```
      INTEGER, INTENT( OUT )         :: Local(*)     ! Local Slice
```

DESCRIPTION:

Given a decomposition of the domain, dole out a slice (one-dimensional array) to all the constituent PEs as described by the decomposition Decomp.

SYSTEM ROUTINES:

```
      MPI_ISEND, MPI_RECV, MPI_COMM_RANK
```

REVISION HISTORY:

```
     97.04.14   Sawyer     Creation
     97.04.16   Sawyer     Cleaned up for walk-through
     97.05.01   Sawyer     Use Decomp%Comm for all local info
     97.05.18   Sawyer     DecompType has moved to ParUtilitiesTypes
     97.05.29   Sawyer     Changed 2-D arrays to 1-D
     97.07.03   Sawyer     Reformulated documentation
     97.07.22   Sawyer     DecompType has moved to DecompModule
     97.12.01   Sawyer     Changed MPI_SSEND to MPI_ISEND
     97.12.05   Sawyer     Added InComm and Root as arguments
     97.12.05   Sawyer     Added logic to support intercommunicators
     98.01.24   Sawyer     Removed dependence on MPI derived types TESTED
     98.02.05   Sawyer     Removed the use of intercommunicators
     98.03.30   Sawyer     Stats dimension corrected: Gsize*MPI_STATUS_SIZE
     99.01.19   Sawyer     Dropped assumed-size arrays
     00.07.07   Sawyer     Removed "1D" references
     00.07.23   Sawyer     Implementation with shared memory arenas

! !LOCAL VARIABLES:

      INTEGER Ierror, I, J, K, L, Iam, GroupSize, Reqs( Gsize )
#if !defined( USE_ARENAS )
      INTEGER Status( MPI_STATUS_SIZE ), Stats( Gsize*MPI_STATUS_SIZE )
      INTEGER, ALLOCATABLE    :: SendBuf(:)
#endif
!
      CPP_ENTER_PROCEDURE( "PARSCATTERINT" )
!
#if defined( USE_ARENAS )
!
! Pull the local process information out of the communicator
!
!!!      Iam = MOD( Comm, MAX_PES )
!!!      L   = Comm / MAX_PES
!!!      GroupSize = MOD( L, MAX_PES ) + 1
!!!      L   = L / MAX_PES
!!!      Color = MOD( L, MAX_PES )

!
! For now, Iam and GroupSize take on the global communicator values
```

```
!
      Iam = GID
      GroupSize = Gsize
      IF ( Iam .EQ. Root ) THEN
        L = 0
        DO I = 1, GroupSize
!
! Pick out the array sections to be sent.
! This is the inverse of the operation in ParGather
!
          DO J = 1, SIZE( Decomp%HEAD(I)%StartTags )
            DO K = Decomp%HEAD(I)%StartTags(J),Decomp%HEAD(I)%EndTags(J)
              L = L+1
              IntBuf(L) = Slice(K)
            ENDDO
          ENDDO
        ENDDO
      ENDIF
!
! Barrier: all PEs participate!
!
      CALL mlp_barrier(gid,gsize)

!
! All receive from the root.
!

! The local array may be larger than that specified in the decomposition
!
      L = 0
      IF ( Iam .GT. 0 ) L = SUM( Decomp%NumEntries(1:Iam) )
      DO I=1, Decomp%NumEntries(Iam+1)
        Local( I ) = IntBuf( L + I )
      ENDDO
!
! The following is needed to ensure that IntBuf can now be reused
!
      CALL mlp_barrier(gid,gsize)
#else
      CALL MPI_COMM_RANK( InComm, Iam, Ierror )
      CALL MPI_COMM_SIZE( InComm, GroupSize, Ierror )

      IF ( Iam .EQ. Root ) THEN
        ALLOCATE( SendBuf( SUM( Decomp%NumEntries ) ) )
        L = 0
        DO I = 1, GroupSize
!
! Pick out the array sections to be sent.
! This is the inverse of the operation in ParGather
!
          DO J = 1, SIZE( Decomp%HEAD(I)%StartTags )
            DO K = Decomp%HEAD(I)%StartTags(J),Decomp%HEAD(I)%EndTags(J)
              L = L+1
              SendBuf(L) = Slice(K)
            ENDDO
```

```
         ENDDO
!
! This is a non-blocking send. SendBuf cannot be immediately deallocated
!
! WARNING: F90-MPI inconsistency: make sure the indexing below always works
!
         CALL MPI_ISEND( SendBuf(L-Decomp%NumEntries(I)+1),                  &
     &                   Decomp%NumEntries(I), CPP_MPI_INTEGER,              &
     &                   I-1, 0, InComm, Reqs(I), Ierror )

        ENDDO
      ENDIF

!
! All receive from the root.
!
! The local array may be larger than that specified in the decomposition
!
      CALL MPI_RECV( Local, Decomp%NumEntries(Iam+1),                        &
     &               CPP_MPI_INTEGER,                                        &
     &               Root, 0, InComm, Status, Ierror )
!
! Experience shows that we should wait for all the non-blocking
! PEs to check in, EVEN THOUGH THE MPI_RECV HAS COMPLETED !!
!
      IF ( Iam .EQ. Root ) THEN
        CALL MPI_WAITALL( GroupSize, Reqs, Stats, Ierror )
        DEALLOCATE( SendBuf )
      ENDIF

!
! The following may be needed on some platforms to avoid an MPI bug.
!
      CALL MPI_BARRIER( InComm, Ierror )
#endif
      CPP_LEAVE_PROCEDURE( "PARSCATTERINT" )
      RETURN
```

71.7.11 ParGatherReal — Gather Slice from all PEs

INTERFACE:

```
      SUBROUTINE ParGatherReal ( InComm, Root, Local, Decomp, Slice )
```

USES:

```
      USE decompmodule, ONLY:  DecompType, Lists
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      INTEGER, INTENT( IN )          :: InComm       ! Communicator
      INTEGER, INTENT( IN )          :: Root         ! Root PE
      REAL(CPP_REAL), INTENT( IN )   :: Local(*)     ! Local Slice
      TYPE(DecompType), INTENT( IN ) :: Decomp       ! Decomp information
```

OUTPUT PARAMETERS:

```
      REAL(CPP_REAL), INTENT( OUT )  :: Slice(*)     ! Global Slice
```

DESCRIPTION:

Given a decomposition of the domain and a local portion of the total slice on each PE, gather together the portions into a global slice on the root PE

SYSTEM ROUTINES:

```
      MPI_ISEND, MPI_RECV, MPI_COMM_RANK
```

REVISION HISTORY:

```
     97.04.14   Sawyer     Creation
     97.04.16   Sawyer     Cleaned up for walk-through
     97.05.01   Sawyer     Use Decomp%Comm for all local info
     97.05.18   Sawyer     DecompType has moved to ParUtilitiesTypes
     97.05.29   Sawyer     Changed 2-D arrays to 1-D
     97.07.03   Sawyer     Reformulated documentation
     97.07.22   Sawyer     DecompType has moved to DecompModule
     97.12.01   Sawyer     Changed MPI_SSEND to MPI_ISEND
     97.12.05   Sawyer     Added InComm and Root as arguments
     97.12.05   Sawyer     Added logic to support intercommunicators
     98.01.24   Sawyer     Removed dependence on MPI derived types TESTED
     98.01.29   Sawyer     Corrected assertions
     98.02.05   Sawyer     Removed the use of intercommunicators
     98.03.31   Sawyer     Stat dimension corrected: MPI_STATUS_SIZE
     98.04.22   Sawyer     Local no longer assumed shape: Local(*)
     99.01.19   Sawyer     Dropped assumed-size arrays
     00.07.07   Sawyer     Removed "1D" references
     00.07.23   Sawyer     Implementation with shared memory arenas
```

```
! !LOCAL VARIABLES:
      INTEGER Ierror, I, J, K, L, Iam, GroupSize, Req
#if !defined( USE_ARENAS )
      INTEGER Status( MPI_STATUS_SIZE ), Stat( MPI_STATUS_SIZE )
#endif
      REAL(CPP_REAL), ALLOCATABLE    :: RecvBuf(:)
!
      CPP_ENTER_PROCEDURE( "PARGATHERREAL" )
!
#if defined( USE_ARENAS )
!
! Pull the local process information out of the communicator
!
!!!      Iam = MOD( Comm, MAX_PES )
!!!      L   = Comm / MAX_PES
!!!      GroupSize = MOD( L, MAX_PES ) + 1
!!!      L   = L / MAX_PES
```

```
!!!      Color = MOD( L, MAX_PES )

!
! For now, Iam and GroupSize take on the global communicator values
!
      Iam = GID
      GroupSize = Gsize
!
! All PEs send their contribution to the root
!
      L = 0
      IF ( Iam .GT. 0 ) L = SUM( Decomp%NumEntries(1:Iam) )
      DO I=1, Decomp%NumEntries(Iam+1)
        DataBuf( L + I ) = Local( I )
      ENDDO
      CALL mlp_barrier(gid,gsize)
!
      IF ( Iam .EQ. Root ) THEN
!
! On the Root PE receive from every other PE
!
        L = 0
        DO I = 1, GroupSize
!
! This is the simple reverse mapping of that in ParScatter
!
          DO J = 1, SIZE( Decomp%HEAD(I)%StartTags )
            DO K = Decomp%HEAD(I)%StartTags(J),Decomp%HEAD(I)%EndTags(J)
              L = L + 1
              Slice(K) = DataBuf(L)
            ENDDO
          ENDDO
        ENDDO
      ENDIF
!
! The following is needed to ensure that DataBuf can now be reused
!
      CALL mlp_barrier(gid,gsize)
#else
      CALL MPI_COMM_RANK( InComm, Iam, Ierror )
      CALL MPI_COMM_SIZE( InComm, GroupSize, Ierror )
!
! All PEs send their contribution to the root
!
      CALL MPI_ISEND( Local, Decomp%NumEntries(Iam+1),                  &
     &                CPP_MPI_REAL,                                     &
     &                Root, Iam+3001, InComm, Req, Ierror )

      IF ( Iam .EQ. Root ) THEN
        ALLOCATE( RecvBuf( SUM( Decomp%NumEntries ) ) )
!
! On the Root PE receive from every other PE
!
        L = 0
        DO I = 1, GroupSize
```

```
!
! This is a blocking, synchronous recv.  All the
! sends should have been posted so it should not deadlock
!
! WARNING: F90-MPI inconsistency: make sure the indexing below always works
!
          CPP_ASSERT_F90( L .LT. SIZE( RecvBuf ) )
          CALL MPI_RECV( RecvBuf(L+1), Decomp%NumEntries(I),             &
     &                   CPP_MPI_REAL, I-1, I+3000, InComm,              &
     &                   Status, Ierror )
!
! This is the simple reverse mapping of that in ParScatter
!
          DO J = 1, SIZE( Decomp%HEAD(I)%StartTags )
            DO K = Decomp%HEAD(I)%StartTags(J),Decomp%HEAD(I)%EndTags(J)
              L = L + 1
              Slice(K) = RecvBuf(L)
#if defined(DEBUG_PARGATHERREAL)
                PRINT *, " Entry ", L, RecvBuf(L), K, SIZE(Slice)
#endif
            ENDDO
          ENDDO
        ENDDO
        DEALLOCATE( RecvBuf )
      ENDIF
      CALL MPI_WAIT( Req, Stat, Ierror )
!
! The following may be needed on some platforms to avoid an MPI bug.
!
      CALL MPI_BARRIER( InComm, Ierror )
#endif
      CPP_LEAVE_PROCEDURE( "PARGATHERREAL" )
      RETURN
```

71.7.12 ParGatherInt — Gather Slice from all PEs

INTERFACE:

```
SUBROUTINE ParGatherInt ( InComm, Root, Local, Decomp, Slice )
```

USES:

```
USE decompmodule, ONLY:  DecompType, Lists
IMPLICIT NONE
```

INPUT PARAMETERS:

```
INTEGER, INTENT( IN )          :: InComm       ! Communicator
INTEGER, INTENT( IN )          :: Root         ! Root PE
INTEGER, INTENT( IN )          :: Local(*)     ! Local Slice
TYPE(DecompType), INTENT( IN ) :: Decomp       ! Decomp information
```

OUTPUT PARAMETERS:

```
      INTEGER, INTENT( OUT )         :: Slice(*)     ! Global Slice
```

DESCRIPTION:

Given a decomposition of the domain and a local portion of the total slice on each PE, gather together the portions into a global slice on the root PE

SYSTEM ROUTINES:

```
      MPI_ISEND, MPI_RECV, MPI_COMM_RANK
```

REVISION HISTORY:

```
     97.04.14   Sawyer     Creation
     97.04.16   Sawyer     Cleaned up for walk-through
     97.05.01   Sawyer     Use Decomp%Comm for all local info
     97.05.18   Sawyer     DecompType has moved to ParUtilitiesTypes
     97.05.29   Sawyer     Changed 2-D arrays to 1-D
     97.07.03   Sawyer     Reformulated documentation
     97.07.22   Sawyer     DecompType has moved to DecompModule
     97.12.01   Sawyer     Changed MPI_SSEND to MPI_ISEND
     97.12.05   Sawyer     Added InComm and Root as arguments
     97.12.05   Sawyer     Added logic to support intercommunicators
     98.01.24   Sawyer     Removed dependence on MPI derived types TESTED
     98.01.29   Sawyer     Corrected assertions
     98.02.05   Sawyer     Removed the use of intercommunicators
     98.03.31   Sawyer     Stat dimension corrected: MPI_STATUS_SIZE
     98.04.22   Sawyer     Local no longer assumed shape: Local(*)
     99.01.19   Sawyer     Dropped assumed-size arrays
     00.07.07   Sawyer     Removed "1D" references
     00.07.23   Sawyer     Implementation with shared memory arenas

! !LOCAL VARIABLES:
      INTEGER Ierror, I, J, K, L, Iam, GroupSize, Req
#if !defined( USE_ARENAS )
      INTEGER Status( MPI_STATUS_SIZE ), Stat( MPI_STATUS_SIZE )
#endif
      INTEGER, ALLOCATABLE    :: RecvBuf(:)
!
      CPP_ENTER_PROCEDURE( "PARGATHERINT" )
!
#if defined( USE_ARENAS )
!
! Pull the local process information out of the communicator
!
!!!      Iam = MOD( Comm, MAX_PES )
!!!      L   = Comm / MAX_PES
!!!      GroupSize = MOD( L, MAX_PES ) + 1
!!!      L   = L / MAX_PES
!!!      Color = MOD( L, MAX_PES )

!
! For now, Iam and GroupSize take on the global communicator values
!
```

```
      Iam = GID
      GroupSize = Gsize
!
! All PEs send their contribution to the root
!
      L = 0
      IF ( Iam .GT. 0 ) L = SUM( Decomp%NumEntries(1:Iam) )
      DO I=1, Decomp%NumEntries(Iam+1)
        IntBuf( L + I ) = Local( I )
      ENDDO
      CALL mlp_barrier(gid,gsize)
!
      IF ( Iam .EQ. Root ) THEN
!
! On the Root PE receive from every other PE
!
        L = 0
        DO I = 1, GroupSize
!
! This is the simple reverse mapping of that in ParScatter
!
          DO J = 1, SIZE( Decomp%HEAD(I)%StartTags )
            DO K = Decomp%HEAD(I)%StartTags(J),Decomp%HEAD(I)%EndTags(J)
              L = L + 1
              Slice(K) = IntBuf(L)
            ENDDO
          ENDDO
        ENDDO
      ENDIF
!
! The following is needed to ensure that IntBuf can now be reused
!
      CALL mlp_barrier(gid,gsize)
#else
      CALL MPI_COMM_RANK( InComm, Iam, Ierror )
      CALL MPI_COMM_SIZE( InComm, GroupSize, Ierror )
!
! All PEs send their contribution to the root
!
      CALL MPI_ISEND( Local, Decomp%NumEntries(Iam+1), CPP_MPI_INTEGER,       &
     &                Root, Iam+3001, InComm, Req, Ierror )

      IF ( Iam .EQ. Root ) THEN
        ALLOCATE( RecvBuf( SUM( Decomp%NumEntries ) ) )
!
! On the Root PE receive from every other PE
!
        L = 0
        DO I = 1, GroupSize
!
! This is a blocking, synchronous recv.  All the
! sends should have been posted so it should not deadlock
!
! WARNING: F90-MPI inconsistency: make sure the indexing below always works
!
```

```
          CPP_ASSERT_F90( L .LT. SIZE( RecvBuf ) )
          CALL MPI_RECV( RecvBuf(L+1), Decomp%NumEntries(I),                  &
     &                   CPP_MPI_INTEGER, I-1, I+3000, InComm,                &
     &                   Status, Ierror )
!
! This is the simple reverse mapping of that in ParScatter
!
          DO J = 1, SIZE( Decomp%HEAD(I)%StartTags )
            DO K = Decomp%HEAD(I)%StartTags(J),Decomp%HEAD(I)%EndTags(J)
              L = L + 1
              Slice(K) = RecvBuf(L)
#if defined(DEBUG_PARGATHERINT)
                PRINT *, " Entry ", L, RecvBuf(L), K, SIZE(Slice)
#endif
            ENDDO
          ENDDO
        ENDDO
        DEALLOCATE( RecvBuf )
      ENDIF
      CALL MPI_WAIT( Req, Stat, Ierror )
!
! The following may be needed on some platforms to avoid an MPI bug.
!
      CALL MPI_BARRIER( InComm, Ierror )
#endif
      CPP_LEAVE_PROCEDURE( "PARGATHERINT" )
      RETURN
```

71.7.13 ParBeginTransferReal — Start an ASYNC Real Transfer

INTERFACE:

```
       SUBROUTINE ParBeginTransferReal(InComm, NrInPackets, NrOutPackets, &
      &                                Dest, Src, InBuf, InIA,            &
      &                                OutBuf, OutIA )
```

USES:

```
       IMPLICIT NONE
```

INPUT PARAMETERS:

```
       INTEGER, INTENT( IN )       :: InComm       ! Communicator
       INTEGER, INTENT( IN )       :: NrInPackets  ! Number of in packets
       INTEGER, INTENT( IN )       :: NrOutPackets ! Number of out packets
       INTEGER, INTENT( IN )       :: Dest(:)      ! PE destinations
       INTEGER, INTENT( IN )       :: Src(:)       ! PE sources
       REAL(CPP_REAL), INTENT( IN ):: InBuf(:)     ! Input buffer
       INTEGER, INTENT( IN )       :: InIA(:)      ! In packet counter
       INTEGER, INTENT( IN )       :: OutIA(:)     ! Out packet counter
```

OUTPUT PARAMETERS:

```
REAL(CPP_REAL), INTENT( OUT ) :: OutBuf(:)  ! Output buffer
```

DESCRIPTION:

This routine initiates an async. transfer of an array InBuf partitioned into chunks defined by the arrays InIA and Dest to an output array OutBuf on another PE. InIA(1) contains the number of reals to be sent to Dest(1), InIA(2) the number of reals to be sent to Dest(2), etc. Similarly, the array OutBuf on the calling PE is partitioned into chunks by OutIA and Src, with OutIA(1) the number of reals anticipated from Src(1), etc.
The default implementation reads through the contiguous array InBuf and sends the chunks to the PEs designated with an asyncronous MPI_ISEND. Correspondingly it posts the receives with an asynchronous MPI_IRECV. The USE_ARENAS implementation in fully functional, but only for a global communicator. Wait handles InHandle(:) and OutHandle(:) are in common block.

BUGS:

```
It is assumed that the buffers are passed to this routine by
reference!!!!!!!!!!
The buffers may not be accessed until after the call to
ParEndTransferReal.
```

SYSTEM ROUTINES:

```
MPI_COMM_RANK, MPI_ISEND, MPI_IRECV
```

REVISION HISTORY:

```
97.09.26   Sawyer     Creation
97.12.05   Sawyer     Renamed Comm to InComm to avoid collisions
98.02.26   Sawyer     Added Dest, Src and Remote to clean up code
98.04.16   Sawyer     Number of packets become input arguments
98.09.04   Sawyer     Cleaned interface: handles in common, no Remote
99.03.04   Sawyer     Inlined ParCalculateRemote
99.06.01   Sawyer     Changed pointer arrays to INTEGER*8 for SGI
00.08.07   Sawyer     Implementation with shared memory arenas
```

```
! !LOCAL VARIABLES:
      INTEGER Iam, GroupSize, Nr, Icnt, Packet, I, Ierr
#if defined( USE_ARENAS )
      INTEGER Off( 0:Gsize ), inc, base
#endif

      CPP_ENTER_PROCEDURE( "PARBEGINTRANSFERREAL" )
      CPP_ASSERT_F90( NrInPackets .LE. SIZE( Dest ) )
      CPP_ASSERT_F90( NrInPackets .LE. SIZE( InIA ) )
      CPP_ASSERT_F90( NrOutPackets .LE. SIZE( Src ) )
      CPP_ASSERT_F90( NrOutPackets .LE. SIZE( OutIA ) )

#if defined( USE_ARENAS )
!
! The following code has the effect of sorting the packets
! by their destination and filling the DataBuf
!
```

```
      Base = GID*BlockSize   ! Each PE is allotted Blocksize
      Off = 0                ! Vector of offsets
!
! First count how much is going to each PE
!
      DO Packet=1, NrInPackets
        Off(Dest(Packet)+1) = Off(Dest(Packet)+1)+InIA(Packet)
      ENDDO
!
! Calculate the offsets, and save the counts in Volume
!
      DO I=1, Gsize
        Off(I) = Off(I)+Off(I-1)   ! Beware: "Off(0)" is 0
        Volume(I,GID+1) = Off(I)   ! 00.09.09 changed index order
      ENDDO
      CPP_ASSERT_F90( Off(Gsize) .LE. BlockSize )
      inc = 0
      DO Packet=1, NrInPackets
        Icnt = Base + Off(Dest(Packet))
        DO I=1,InIA(Packet)
          inc = inc+1
          DataBuf(Icnt+I) = InBuf( inc )
        ENDDO
        Off(Dest(Packet)) = Off(Dest(Packet)) + InIA(Packet)
      END DO
#else
      CALL MPI_COMM_RANK( InComm, Iam, Ierr )
      CALL MPI_COMM_SIZE( InComm, GroupSize, Ierr )

!
! Increment the ongoing transfer number
      BegTrf = MOD(BegTrf,MAX_TRF) + 1
!
!     MPI: Irecv over all processes
!
      Icnt = 1
      DO Packet = 1, NrOutPackets
        Nr = OutIA( Packet )
        IF ( Nr .GT. 0 ) THEN
#if defined( DEBUG_PARBEGINTRANSFERREAL )
          PRINT *, "Iam ",Iam," posts recv ",Nr," from ", Src( Packet )
#endif
!
! Receive the buffers with MPI_Irecv. Non-blocking
!
          CPP_ASSERT_F90( Icnt+Nr-1 .LE. SIZE( OutBuf ) )
          CALL MPI_IRECV( OutBuf( Icnt ), Nr,                          &
     &          CPP_MPI_REAL, Src( Packet ), Src( Packet ),             &
     &          InComm, OutHandle(Packet,BegTrf), Ierr )
        ELSE
          OutHandle( Packet,BegTrf ) = MPI_REQUEST_NULL
        END IF
        Icnt = Icnt + Nr
      END DO
!
```

```
!     MPI: Isend over all processes
!
      Icnt = 1
      CPP_ASSERT_F90( NrInPackets .LE. SIZE( Dest ) )
      CPP_ASSERT_F90( NrInPackets .LE. SIZE( InIA ) )
      DO Packet = 1, NrInPackets
        Nr = InIA( Packet )
        IF ( Nr .GT. 0 ) THEN
#if defined( DEBUG_PARBEGINTRANSFERREAL )
          PRINT *,"Iam ",Iam," posts send ",Nr," to ",Dest( Packet )
#endif
!
!     Send the individual buffers with non-blocking sends
!
          CPP_ASSERT_F90( Icnt+Nr-1 .LE. SIZE( InBuf ) )
          CALL MPI_ISEND ( InBuf( Icnt ), Nr,                          &
     &          CPP_MPI_REAL, Dest( Packet ), Iam,                     &
     &          InComm, InHandle( Packet,BegTrf ), Ierr )
        ELSE
          InHandle( Packet,BegTrf ) = MPI_REQUEST_NULL
        END IF
        Icnt = Icnt + Nr
      END DO
#endif
!
!
      CPP_LEAVE_PROCEDURE( "PARBEGINTRANSFERREAL" )
      RETURN
```

71.7.14 ParBeginTransferPattern — Start an ASYNC Pattern Transfer

INTERFACE:

```
SUBROUTINE ParBeginTransferPattern( Pattern, InBuf, OutBuf )
```

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
TYPE (ParPatternType), INTENT( IN )  :: Pattern   ! Comm Pattern
REAL(CPP_REAL), INTENT( IN )         :: InBuf(*)  ! Input buffer
```

OUTPUT PARAMETERS:

```
REAL(CPP_REAL), INTENT( OUT )        :: OutBuf(*) ! Output buffer
```

DESCRIPTION:

This routine initiates an async. transfer of an array InBuf. The communication pattern indicates the indices outgoing values of InBuf and incoming values for OutBuf. This routine is fundamentally equivalent to ParBeginTransferReal; the use of a communication pattern is largely a performance enhancement, since it eliminates the need for intermediate buffering.
Wait handles InHandle(:) and OutHandle(:) are in common block. The buffers may not be accessed until after the call to ParEndTransferReal.

BUGS:

```
It is assumed that the buffers are passed to this routine by
reference.  The USE\_ARENAS version is not yet functional.
```

SYSTEM ROUTINES:

```
MPI_COMM_RANK, MPI_ISEND, MPI_IRECV
```

REVISION HISTORY:

```
01.02.14   Sawyer     Creation from ParBeginTransferReal
```

```
! !LOCAL VARIABLES:
      INTEGER TypeDesc, Nr, Icnt, Packet, I, ipe, Ierr
#if defined( USE_ARENAS )
      INTEGER Off( 0:Gsize ), inc, base
#endif

      CPP_ENTER_PROCEDURE( "PARBEGINTRANSFERPATTERN" )

#if defined( USE_ARENAS )
      The USE_ARENAS code is not running yet.  You should not
      be compiling with this option.

!
! The following code has the effect of sorting the packets
! by their destination and filling the DataBuf
!
      Base = GID*BlockSize   ! Each PE is allotted Blocksize
      Off = 0                ! Vector of offsets
!
! First count how much is going to each PE
!
      DO Packet=1, NrInPackets
        Off(Dest(Packet)+1) = Off(Dest(Packet)+1)+InIA(Packet)
      ENDDO
!
! Calculate the offsets, and save the counts in Volume
!
      DO I=1, Gsize
        Off(I) = Off(I)+Off(I-1)   ! Beware: "Off(0)" is 0
        Volume(I,GID+1) = Off(I)   ! 00.09.09 changed index order
      ENDDO
      CPP_ASSERT_F90( Off(Gsize) .LE. BlockSize )
      inc = 0
      DO Packet=1, NrInPackets
        Icnt = Base + Off(Dest(Packet))
```

```
        DO I=1,InIA(Packet)
          inc = inc+1
          DataBuf(Icnt+I) = InBuf( inc )
        ENDDO
        Off(Dest(Packet)) = Off(Dest(Packet)) + InIA(Packet)
      END DO
#else

      BegTrf = MOD(BegTrf,MAX_TRF) + 1
!
! MPI: Irecv over all processes
!

      DO ipe = 1, Pattern%Size
!
! Receive the buffers with MPI_Irecv. Non-blocking
!
        TypeDesc = Pattern%RecvDesc(ipe)
        CALL MPI_IRECV( OutBuf, 1, TypeDesc, ipe-1, ipe-1,              &
     &                  Pattern%Comm, OutHandle(ipe,BegTrf), Ierr )
      END DO
!
! MPI: Isend over all processes
!
      DO ipe = 1, Pattern%Size
!
! Send the individual buffers with non-blocking sends
!
        TypeDesc = Pattern%SendDesc(ipe)
        CALL MPI_ISEND ( InBuf, 1, TypeDesc, ipe-1, Pattern%Iam,        &
     &                   Pattern%Comm, InHandle(ipe,BegTrf), Ierr )
      END DO
#endif
!
!
      CPP_LEAVE_PROCEDURE( "PARBEGINTRANSFERPATTERN" )
      RETURN
```

71.7.15 ParEndTransferReal — Complete an ASYNC Real Transfer

INTERFACE:

```
       SUBROUTINE ParEndTransferReal( InComm, NrInPackets, NrOutPackets,  &
      &                               Dest, Src, InBuf, InIA,             &
      &                               OutBuf, OutIA )
```

USES:

```
       IMPLICIT NONE
```

INPUT PARAMETERS:

```
INTEGER, INTENT( IN )       :: InComm       ! Communicator
INTEGER, INTENT( IN )       :: NrInPackets  ! Number of in packets
INTEGER, INTENT( IN )       :: NrOutPackets ! Number of out packets
INTEGER, INTENT( IN )       :: Dest(:)      ! PE destinations
INTEGER, INTENT( IN )       :: Src(:)       ! PE sources
REAL(CPP_REAL), INTENT( IN ):: InBuf(:)     ! Input buffer
INTEGER, INTENT( IN )       :: InIA(:)      ! Pointer array
INTEGER, INTENT( IN )       :: OutIA(:)     ! Pointer array
```

INPUT/OUTPUT PARAMETERS:

```
REAL(CPP_REAL), INTENT( INOUT ) :: OutBuf(:)! Output buffer
```

DESCRIPTION:

This routine completes an async. transfer of an array partitioned into chunks defined by the array InIA. In the MPI version, neither InBuf nor OutBuf is not used since that information was utilized in ParBeginTransferReal. In the USE_ARENAS version they are utilized.
The link between StartTransfer and EndTransfer is made possible by the InHandle and OutHandle: they reflect the status of the ongoing transfer. When this routine completes, a valid and accessible copy of the OutBuf is ready for use. The USE_ARENAS version is functional, but assumes a global communicator.

BUGS:

```
It is assumed that the buffers are passed to this routine by
reference! The buffers may not be accessed until after the
completion of ParEndTransferReal.
```

SYSTEM ROUTINES:

```
MPI_COMM_RANK, MPI_ISEND, MPI_IRECV
```

REVISION HISTORY:

```
97.09.26   Sawyer     Creation
97.12.05   Sawyer     Renamed Comm to InComm to avoid collisions
98.02.26   Sawyer     Count through packets, not PEs
98.04.16   Sawyer     Number of packets become input arguments
98.09.04   Sawyer     Cleaned interface: handles in common
99.03.05   Sawyer     Support for contiguous communicators in SHMEM
99.04.22   Sawyer     Bug fix: replaced MPI_WAIT with MPI_WAITALL
99.06.03   Sawyer     Bug fix: GroupSize in SHMEM_BARRIER
00.07.28   Sawyer     Implemented with shared memory arenas
```

```
! !LOCAL VARIABLES:
      INTEGER Iam, GroupSize, J, Offset, Packet, Ierr
#if defined( USE_ARENAS)
      INTEGER Off( 0:Gsize ), Pe, PacketSize, inc
#else
      INTEGER InStats(NrInPackets*MPI_STATUS_SIZE)
      INTEGER OutStats(NrOutPackets*MPI_STATUS_SIZE)
#endif

      CPP_ENTER_PROCEDURE( "PARENDTRANSFERREAL" )
#if defined( USE_ARENAS )
!
```

```
! Needed to ensure that DataBuf and Volume have "arrived"
!
      CALL mlp_barrier(gid,gsize)
!
! The packet is plucked out of the appropriate data segment Blocksize*(Pe-1)
! with the offset specified in Volume(Pe,GID+1).  Since there can be
! more than one packet from Pe bound for me (GID), Off(Pe) is advanced
! by the packetsize.  This should yield the Offset to the needed packet.
!
      Off = 0
      inc = 0
      DO Packet=1, NrOutPackets
        Pe = Src(Packet)
        Offset = Off(Pe) + Blocksize*Pe
! 00.09.09 changed index order of Volume
        IF ( GID .GT. 0 ) Offset = Offset + Volume(GID,Pe+1)
        PacketSize = OutIA(Packet)
        DO J=1, PacketSize
          inc = inc + 1
          OutBuf(inc) = DataBuf(Offset+J)
        ENDDO
        Off(Pe) = Off(Pe) + PacketSize
      ENDDO
!
! Needed to ensure that a future call cannot upset current operation
!
      CALL mlp_barrier(gid,gsize)
#else
      EndTrf = MOD(EndTrf,MAX_TRF)+1
      CPP_ASSERT_F90( NrInPackets .LE. MAX_PAX )
      CALL MPI_WAITALL( NrInPackets, InHandle(:,EndTrf), InStats, Ierr )

      CPP_ASSERT_F90( NrOutPackets .LE. MAX_PAX )
      CALL MPI_WAITALL( NrOutPackets, OutHandle(:,EndTrf), OutStats, Ierr )
!
! WS 98.09.22 : This barrier needed to synchronize.
!
      CALL MPI_BARRIER( InComm, Ierr )
#endif
      CPP_LEAVE_PROCEDURE( "PARENDTRANSFERREAL" )
      RETURN
```

71.7.16 ParEndTransferPattern — Complete an ASYNC Pattern Transfer

INTERFACE:

```
      SUBROUTINE ParEndTransferPattern( Pattern, InBuf, OutBuf )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
TYPE (ParPatternType), INTENT( IN )  :: Pattern   ! Comm Pattern
REAL(CPP_REAL), INTENT( IN )         :: InBuf(*)  ! Input buffer
```

INPUT/OUTPUT PARAMETERS:

```
REAL(CPP_REAL), INTENT( INOUT )      :: OutBuf(*) ! Output buffer
```

DESCRIPTION:

This routine completes an async. transfer of an array communicated with a communication pattern. The link between StartTransfer and EndTransfer is made possible by the InHandle and OutHandle: they reflect the status of the ongoing transfer. When this routine completes, a valid and accessible copy of the OutBuf is ready for use. The buffers may not be accessed until after the completion of ParEndTransfer.

BUGS:

```
It is assumed that the buffers are passed to this routine by
reference! USE_ARENAS version not yet functional.
```

SYSTEM ROUTINES:

```
MPI_COMM_RANK, MPI_ISEND, MPI_IRECV
```

REVISION HISTORY:

```
01.02.14   Sawyer     Creation from ParEndTransferReal
```

```
! !LOCAL VARIABLES:
      INTEGER J, Offset, Packet, Ierr
#if defined( USE_ARENAS)
      INTEGER Off( 0:Gsize ), Pe, PacketSize, inc
#else
      INTEGER InStats(Pattern%Size*MPI_STATUS_SIZE)
      INTEGER OutStats(Pattern%Size*MPI_STATUS_SIZE)
#endif

      CPP_ENTER_PROCEDURE( "PARENDTRANSFERPATTERN" )
#if defined( USE_ARENAS )
!      The USE_ARENAS code does not work yet.  You should not
!      be compiling with this CPP token on.
#else
      EndTrf = MOD(EndTrf,MAX_TRF) + 1
      CALL MPI_WAITALL( Pattern%Size, InHandle(:,EndTrf), InStats, Ierr )
      CALL MPI_WAITALL( Pattern%Size, OutHandle(:,EndTrf), OutStats, Ierr )
!
! WS 98.09.22 : This barrier needed to synchronize.
!
      CALL MPI_BARRIER( Pattern%Comm, Ierr )
#endif
      CPP_LEAVE_PROCEDURE( "PARENDTRANSFERPATTERN" )
      RETURN
```

71.7.17 ParExchangeVectorReal — Exchange a sparse packed vector

INTERFACE:

```
      SUBROUTINE ParExchangeVectorReal ( InComm, LenInVector, InVector,  &
     &                                   LenOutVector, OutVector )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      INTEGER, INTENT( IN )   :: InComm            ! Communicator
      INTEGER, INTENT( IN )   :: LenInVector( * )  ! Length on each PE
      REAL(CPP_REAL), INTENT( IN ):: InVector( * ) ! The input buffer
```

OUTPUT PARAMETERS:

```
      INTEGER, INTENT( OUT )  :: LenOutVector( * ) ! Length on each PE
      REAL(CPP_REAL), INTENT( OUT ) :: OutVector( * ) ! The output buffer
```

DESCRIPTION:

This routine exchanges vectors stored in compressed format, i.e., in so-called compressed sparse row (CSR) format, with other PEs. In essence it first exchanges the lengths with MPI_Alltoall, then the exchange of the actual vectors (can be different in size) using MPI_AlltoallV. Since the latter is inefficient, it is simulated using MPI_Isend and MPI_Recv.
The USE_ARENAS version is fully functional, but assumes a global communicator.

SYSTEM ROUTINES:

```
      MPI_ISEND, MPI_RECV, MPI_WAITALL, MPI_ALLTOALL
```

REVISION HISTORY:

```
     98.03.17   Sawyer     Creation from F77 version
     98.03.30   Sawyer     Removed assumed shape arrays due to problems
     99.01.18   Sawyer     Added barrier for safety
     99.03.08   Sawyer     USE_SHMEM version for CRAY only; untested
     99.06.01   Sawyer     USE_SHMEM version revised per comments from Tom
     00.07.28   Sawyer     Implemented with shared memory arenas

!
! !LOCAL VARIABLES:
      INTEGER :: i, iscnt, ircnt, nr, pe, icnt, Nsize, Iam, Ierr
      INTEGER :: Status(MPI_STATUS_SIZE)
      INTEGER :: Reqs( Gsize ), Stats( Gsize*MPI_STATUS_SIZE )
#if defined( USE_ARENAS )
      INTEGER :: offset( MAX_PE ), goffset( MAX_PE )
#endif

      CPP_ENTER_PROCEDURE( "PAREXCHANGEVECTORREAL" )
#if defined( USE_ARENAS )
      DO pe = 1, Gsize
```

```
        volume( pe, gid+1 ) = LenInVector( pe )
      ENDDO
      CALL mlp_barrier(gid,gsize)
      icnt = 0
      DO pe = 1, Gsize
        LenOutVector( pe ) = volume( gid+1, pe )
        offset( pe )  = icnt
        goffset( pe ) = icnt
        DO i = 1, gid
          goffset( pe ) = goffset( pe ) + volume( pe, i )
        ENDDO
        icnt = icnt + sum( volume( pe, 1:Gsize ) )
      ENDDO
      icnt = 0
      DO pe = 1, Gsize
        DO i = goffset(pe)+1, goffset(pe)+LenInVector(pe)
          icnt = icnt + 1
          databuf( i ) = Invector( icnt )
        ENDDO
      ENDDO
      CALL mlp_barrier(gid,gsize)
      DO i = 1, SUM(LenOutVector(1:Gsize))
        OutVector( i ) = databuf( offset(gid+1) + i )
      ENDDO
#else
      CALL MPI_COMM_SIZE( InComm, Nsize, Ierr )
      CALL MPI_COMM_RANK( InComm, Iam, Ierr )

#if defined( MY_ALLTOALL )
      DO pe = 0, Nsize-1
!
! Send the individual buffers with non-blocking sends
!
        nr = LenInVector( pe + 1 )
        CALL MPI_ISEND( nr, 1,                                          &
     &                  MPI_INTEGER, pe, Iam+3000,                      &
     &                  InComm, Reqs( pe+1 ), Ierr )
      ENDDO
      DO pe = 0, Nsize - 1
!
! Receive the buffers with MPI_Recv. Now we are blocking.
!
        CALL MPI_RECV( nr, 1, CPP_MPI_INTEGER, pe, pe+3000,                    &
     &                 InComm, Status, Ierr )
        LenOutVector(pe + 1) = nr
      ENDDO
      CALL MPI_WAITALL( Nsize, Reqs, Stats, Ierr )
#else
      CALL MPI_ALLTOALL( LenInVector, 1, CPP_MPI_INTEGER,                      &
     &                   LenOutVector, 1, CPP_MPI_INTEGER,                     &
     &                   InComm, Ierr )
#endif
!
! Over all processes
!
```

```
      icnt = 1
      DO pe = 0, Nsize-1
!
! Send the individual buffers with non-blocking sends
!
        nr = LenInVector( pe + 1 )
        IF ( nr .gt. 0 ) THEN
          CALL MPI_ISEND( InVector( icnt ), nr,                          &
     &                    CPP_MPI_REAL, pe, Iam+2000,                    &
     &                    InComm, Reqs( pe+1 ), Ierr )
        ELSE
          Reqs( pe+1 ) = MPI_REQUEST_NULL
        ENDIF
        icnt = icnt + nr
      ENDDO

!
! Over all processes
!
      icnt = 1
      DO pe = 0, Nsize - 1
!
! Receive the buffers with MPI_Recv. Now we are blocking.
!
        nr = LenOutVector(pe + 1)
        IF ( nr .gt. 0 ) THEN
          CALL MPI_RECV( OutVector( icnt ), nr,                          &
     &                   CPP_MPI_REAL, pe, pe+2000,                      &
     &                   InComm, Status, Ierr )
        ENDIF
        icnt = icnt + nr
      ENDDO
      CALL MPI_WAITALL( Nsize, Reqs, Stats, Ierr )
!
! WS 98.09.22 : This barrier needed to synchronize.  Why?
!
      CALL MPI_BARRIER( InComm, Ierr )
#endif
      CPP_LEAVE_PROCEDURE( "PAREXCHANGEVECTORREAL" )

      RETURN
```

71.7.18 ParExchangeVectorInt — Exchange a sparse packed vector

INTERFACE:

```
       SUBROUTINE ParExchangeVectorInt ( InComm, LenInVector, InVector,   &
      &                                  LenOutVector, OutVector )
```

USES:

```
       IMPLICIT NONE
```

INPUT PARAMETERS:

```
INTEGER, INTENT( IN )   :: InComm            ! Communicator
INTEGER, INTENT( IN )   :: LenInVector( * )  ! Length on each PE
INTEGER, INTENT( IN )   :: InVector( * )     ! The input buffer
```

OUTPUT PARAMETERS:

```
INTEGER, INTENT( OUT )  :: LenOutVector( * ) ! Length on each PE
INTEGER, INTENT( OUT )  :: OutVector( * )    ! The output buffer
```

DESCRIPTION:

This routine exchanges vectors stored in compressed format, i.e., in so-called compressed sparse row (CSR) format, with other PEs. In essence it first exchanges the lengths with MPI_Alltoall, then the exchange of the actual vectors (can be different in size) using MPI_AlltoallV. Since the latter is inefficient, it is simulated using MPI_Isend and MPI_Recv.
The USE_ARENAS version is fully functional, but assumes a global communicator.

SYSTEM ROUTINES:

```
MPI_ISEND, MPI_RECV, MPI_WAITALL, MPI_ALLTOALL
```

REVISION HISTORY:

```
98.03.17   Sawyer     Creation from F77 version
98.03.30   Sawyer     Removed assumed shape arrays due to problems
99.01.18   Sawyer     Added barrier for safety
99.03.08   Sawyer     USE_SHMEM version for CRAY only; untested
99.06.01   Sawyer     USE_SHMEM version revised per comments from Tom
00.07.28   Sawyer     Implemented with shared memory arenas
```

```
!
! !LOCAL VARIABLES:
      INTEGER :: i, iscnt, ircnt, nr, pe, icnt, Nsize, Iam, Ierr
      INTEGER :: Status(MPI_STATUS_SIZE)
      INTEGER :: Reqs( Gsize ), Stats( Gsize*MPI_STATUS_SIZE )
#if defined( USE_ARENAS )
      INTEGER :: offset( MAX_PE ), goffset( MAX_PE )
#endif

      CPP_ENTER_PROCEDURE( "PAREXCHANGEVECTORINT" )
#if defined( USE_ARENAS )
      DO pe = 1, Gsize
        volume( pe, gid+1 ) = LenInVector( pe )
      ENDDO
      CALL mlp_barrier(gid,gsize)
      icnt = 0
      DO pe = 1, Gsize
        LenOutVector( pe ) = volume( gid+1, pe )
        offset( pe )  = icnt
        goffset( pe ) = icnt
        DO i = 1, gid
          goffset( pe ) = goffset( pe ) + volume( pe, i )
        ENDDO
        icnt = icnt + sum( volume( pe, 1:Gsize ) )
```

```
      ENDDO
      icnt = 0
      DO pe = 1, Gsize
        DO i = goffset(pe)+1, goffset(pe)+LenInVector(pe)
          icnt = icnt + 1
          IntBuf( i ) = Invector( icnt )
        ENDDO
      ENDDO
      CALL mlp_barrier(gid,gsize)
      DO i = 1, SUM(LenOutVector(1:Gsize))
        OutVector( i ) = IntBuf( offset(gid+1) + i )
      ENDDO
#else
      CALL MPI_COMM_SIZE( InComm, Nsize, Ierr )
      CALL MPI_COMM_RANK( InComm, Iam, Ierr )

#if defined( MY_ALLTOALL )
      DO pe = 0, Nsize-1
!
! Send the individual buffers with non-blocking sends
!
        nr = LenInVector( pe + 1 )
        CALL MPI_ISEND( nr, 1,                                          &
     &                  MPI_INTEGER, pe, Iam+3000,                      &
     &                  InComm, Reqs( pe+1 ), Ierr )
      ENDDO
      DO pe = 0, Nsize - 1
!
! Receive the buffers with MPI_Recv. Now we are blocking.
!
        CALL MPI_RECV( nr, 1,                                                  &
     &                 MPI_INTEGER, pe, pe+3000,                               &
     &                 InComm, Status, Ierr )
        LenOutVector(pe + 1) = nr
      ENDDO
      CALL MPI_WAITALL( Nsize, Reqs, Stats, Ierr )
#else
      CALL MPI_ALLTOALL( LenInVector, 1, CPP_MPI_INTEGER,                      &
     &                   LenOutVector, 1, CPP_MPI_INTEGER,                     &
     &                   InComm, Ierr )
#endif
!
! Over all processes
!
      icnt = 1
      DO pe = 0, Nsize-1
!
! Send the individual buffers with non-blocking sends
!
        nr = LenInVector( pe + 1 )
        IF ( nr .gt. 0 ) THEN
          CALL MPI_ISEND( InVector( icnt ), nr,                                &
     &                    CPP_MPI_INTEGER, pe, Iam+2000,                     &
     &                    InComm, Reqs( pe+1 ), Ierr )
        ELSE
```

```
          Reqs( pe+1 ) = MPI_REQUEST_NULL
        ENDIF
        icnt = icnt + nr
      ENDDO

!
! Over all processes
!
      icnt = 1
      DO pe = 0, Nsize - 1
!
! Receive the buffers with MPI_Recv. Now we are blocking.
!
        nr = LenOutVector(pe + 1)
        IF ( nr .gt. 0 ) THEN
          CALL MPI_RECV( OutVector( icnt ), nr,                                &
     &                   CPP_MPI_INTEGER, pe, pe+2000,                       &
     &                   InComm, Status, Ierr )
        ENDIF
        icnt = icnt + nr
      ENDDO
      CALL MPI_WAITALL( Nsize, Reqs, Stats, Ierr )
!
! WS 98.09.22 : This barrier needed to synchronize.  Why?
!
      CALL MPI_BARRIER( InComm, Ierr )
#endif
      CPP_LEAVE_PROCEDURE( "PAREXCHANGEVECTORINT" )

      RETURN
```

71.7.19 ParCollectiveBarrier — Barrier: Simplest collective op. (Source File: parutilities-module.F90)

INTERFACE:

```
SUBROUTINE ParCollectiveBarrier( InComm )
```

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
INTEGER, INTENT( IN ) :: InComm   ! Communicator
```

DESCRIPTION:

This routine performs a barrier only within the communicator InComm
The USE_ARENAS version is fully functional, but assumes a global communicator.

REVISION HISTORY:

```
     00.09.10   Sawyer     Creation

#if !defined( USE_ARENAS )
      INTEGER Ierror
#endif

#if defined( USE_ARENAS )
      CALL mlp_barrier(gid,gsize)
#else
      CALL MPI_Barrier(InComm, Ierror )
#endif
      RETURN
```

71.7.20 ParCollective0D — Perform global Collective of a scalar (Source File: parutilities-module.F90)

INTERFACE:

```
       SUBROUTINE ParCollective0D( InComm, Op, Var )
```

USES:

```
       IMPLICIT NONE
```

INPUT PARAMETERS:

```
       INTEGER, INTENT( IN ) :: InComm   ! Communicator
       INTEGER, INTENT( IN ) :: Op       ! Operation (see header)
```

INPUT/OUTPUT PARAMETERS:

```
       REAL(CPP_REAL), INTENT( INOUT ) :: Var ! partial Var in, Var out
```

DESCRIPTION:

This utility makes a collective operation over all processes in communicator InComm.
The USE_ARENAS version is fully functional, but assumes a global communicator.

REVISION HISTORY:

```
     00.08.07   Sawyer     Creation

#if !defined( USE_ARENAS )
      INTEGER Ierror
      REAL(CPP_REAL)    Tmp
#else
      INTEGER Ipe
#endif

#if defined( USE_ARENAS )
      DataBuf( GID+1 ) = Var
      CALL mlp_barrier(gid,gsize)
```

```
      IF ( Op .EQ. CPP_SUM_OP ) THEN
        Var = 0.0            !  Must be done the same on all PEs
        DO Ipe = 1, Gsize
          Var = Var + DataBuf( Ipe )
        ENDDO
      ELSEIF ( Op .EQ. CPP_MAX_OP ) THEN
        DO Ipe = 1, Gsize
          IF ( Var .LT. DataBuf( Ipe ) ) Var = DataBuf( Ipe+1 )
        ENDDO
      ELSEIF ( Op .EQ. CPP_MIN_OP ) THEN
        DO Ipe = 1, Gsize
          IF ( Var .GT. DataBuf( Ipe ) ) Var = DataBuf( Ipe )
        ENDDO
      ELSEIF ( Op .EQ. CPP_BCST_OP ) THEN
        DO Ipe = 2, Gsize
          DataBuf( Ipe ) = DataBuf( 1 )
        ENDDO
      ELSEIF ( Op .EQ. CPP_BCAST_OP ) THEN
        Var = DataBuf( 1 )
      ENDIF
      CALL mlp_barrier(gid,gsize)
#else
      IF ( Op .EQ. BCSTOP ) THEN
        CALL MPI_BCAST( Var, 1, CPP_MPI_REAL, 0, InComm, Ierror )
      ELSE
        CALL MPI_ALLREDUCE( Var, Tmp, 1, CPP_MPI_REAL,                  &
     &                     Op, InComm, Ierror )
        Var = Tmp
      ENDIF
#endif
      RETURN
```

71.7.21 ParCollective1D — Perform component-wise global Collective of a vector (Source File: parutilitiesmodule.F90)

INTERFACE:

```
      SUBROUTINE ParCollective1D( InComm, Op, Im, Var )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      INTEGER, INTENT( IN ) :: InComm   ! Communicator
      INTEGER, INTENT( IN ) :: Op       ! Operation (see header)
      INTEGER, INTENT( IN ) :: Im       ! Size of 1-D array
```

INPUT/OUTPUT PARAMETERS:

```
      REAL(CPP_REAL), INTENT( INOUT ) :: Var(Im) ! partial Var in, Var out
```

DESCRIPTION:

This utility makes a collective operation over all processes in communicator InComm.
The USE_ARENAS version is fully functional, but assumes a global communicator.

REVISION HISTORY:

```
     00.08.07   Sawyer     Creation

#if !defined( USE_ARENAS )
      INTEGER Ierror
      REAL(CPP_REAL)    Tmp(Im)
#else
      INTEGER Icnt, Ipe, I
#endif

#if defined( USE_ARENAS )
      Icnt   = GID*Im
      DO i=1,Im
        Icnt = Icnt + 1
        DataBuf( Icnt ) = Var( i )
      ENDDO
      CALL mlp_barrier(gid,gsize)

      IF ( Op .EQ. CPP_SUM_OP ) THEN
        Var = 0.0            !  Must be done the same on all PEs
        Icnt = 0
        DO Ipe = 1, Gsize
          DO i=1,Im
            Icnt = Icnt + 1
            Var( i ) = Var( i ) + DataBuf( Icnt )
          ENDDO
        ENDDO
      ELSEIF ( Op .EQ. CPP_MAX_OP ) THEN
        Icnt = 0
        DO Ipe = 0, Gsize-1
          IF ( Ipe .NE. GID ) THEN
            DO i=1,Im
              Icnt = Icnt + 1
              IF ( Var( i ) .LT. DataBuf( Icnt ) )                        &
     &                  Var( i ) = DataBuf( Icnt )
            ENDDO
          ELSE
            Icnt = Icnt + Im
          ENDIF
        ENDDO
      ELSEIF ( Op .EQ. CPP_MIN_OP ) THEN
        Icnt = 0
        DO Ipe = 0, Gsize-1
          IF ( Ipe .NE. GID ) THEN
            DO i=1,Im
              Icnt = Icnt + 1
              IF ( Var( i ) .GT. DataBuf( Icnt ) )                        &
     &                  Var( i ) = DataBuf( Icnt )
```

```
          ENDDO
        ELSE
          Icnt = Icnt + Im
        ENDIF
      ENDDO
    ELSEIF ( Op .EQ. CPP_BCAST_OP ) THEN
      Icnt = 0
      DO i=1,Im
        Icnt = Icnt + 1
        Var( i ) = DataBuf( Icnt )
      ENDDO
    ENDIF
    CALL mlp_barrier(gid,gsize)
#else
    IF ( Op .EQ. BCSTOP ) THEN
      CALL MPI_BCAST( Var, Im, CPP_MPI_REAL, 0, InComm, Ierror )
    ELSE
      CALL MPI_ALLREDUCE( Var, Tmp, Im, CPP_MPI_REAL,                  &
   &                      Op, InComm, Ierror )
      Var = Tmp
    ENDIF
#endif
    RETURN
```

71.7.22 ParCollective2D — Perform component-wise collective operation (Source File: parutilitiesmodule.F90)

INTERFACE:

```
SUBROUTINE ParCollective2D( InComm, Op, Im, Jm, Var )
```

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
INTEGER, INTENT( IN ) :: InComm     ! Communicator
INTEGER, INTENT( IN ) :: Op         ! Operation (see header)
INTEGER, INTENT( IN ) :: Im         ! First dimension of 2-D array
INTEGER, INTENT( IN ) :: Jm         ! Second dimension of 2-D array
```

INPUT/OUTPUT PARAMETERS:

```
REAL(CPP_REAL), INTENT( INOUT ) :: Var(Im,Jm) ! partial Var in, Var out
```

DESCRIPTION:

This utility makes a collective operation over all processes in communicator InComm.
The USE_ARENAS version is fully functional, but assumes a global communicator.

REVISION HISTORY:

```
     00.08.07   Sawyer     Creation

#if !defined( USE_ARENAS )
      INTEGER Ierror
      REAL(CPP_REAL)    Tmp(Im,Jm)
#else
      INTEGER Icnt, Length, Ipe, I, J
#endif

#if defined( USE_ARENAS )
      Length = Im*Jm
      Icnt   = GID*Length
      DO j=1,Jm
        DO i=1,Im
          Icnt = Icnt + 1
          DataBuf( Icnt ) = Var( i,j )
        ENDDO
      ENDDO
      CALL mlp_barrier(gid,gsize)

      IF ( Op .EQ. CPP_SUM_OP ) THEN
        Var = 0.0            !  Must be done the same on all PEs
        Icnt = 0
        DO Ipe = 1, Gsize
          DO j=1,Jm
            DO i=1,Im
              Icnt = Icnt + 1
              Var( i,j ) = Var( i,j ) + DataBuf( Icnt )
            ENDDO
          ENDDO
        ENDDO
      ELSEIF ( Op .EQ. CPP_MAX_OP ) THEN
        Icnt = 0
        DO Ipe = 0, Gsize-1
          IF ( Ipe .NE. GID ) THEN
            DO j=1,Jm
              DO i=1,Im
                Icnt = Icnt + 1
                IF ( Var( i,j ) .LT. DataBuf( Icnt ) )                    &
     &                    Var( i,j ) = DataBuf( Icnt )
              ENDDO
            ENDDO
          ELSE
            Icnt = Icnt + Length
          ENDIF
        ENDDO
      ELSEIF ( Op .EQ. CPP_MIN_OP ) THEN
        Icnt = 0
        DO Ipe = 0, Gsize-1
          IF ( Ipe .NE. GID ) THEN
            DO j=1,Jm
              DO i=1,Im
                Icnt = Icnt + 1
                IF ( Var( i,j ) .GT. DataBuf( Icnt ) )                    &
     &                    Var( i,j ) = DataBuf( Icnt )
              ENDDO
```

```
          ENDDO
        ELSE
          Icnt = Icnt + Length
        ENDIF
      ENDDO
    ELSEIF ( Op .EQ. CPP_BCAST_OP ) THEN
      Icnt = 0
      DO j=1,Jm
        DO i=1,Im
          Icnt = Icnt + 1
          Var( i,j ) = DataBuf( Icnt )
        ENDDO
      ENDDO
    ENDIF
    CALL mlp_barrier(gid,gsize)
#else
    IF ( Op .EQ. BCSTOP ) THEN
      CALL MPI_BCAST( Var, Im*Jm, CPP_MPI_REAL, 0, InComm, Ierror )
    ELSE
      CALL MPI_ALLREDUCE( Var, Tmp, Im*Jm, CPP_MPI_REAL,              &
   &                    Op, InComm, Ierror )
      Var = Tmp
    ENDIF
#endif
    RETURN
```

71.7.23 ParCollective3D — Perform component-wise global Collective of a vector (Source File: parutilitiesmodule.F90)

INTERFACE:

```
SUBROUTINE ParCollective3D( InComm, Op, Im, Jm, Lm, Var )
```

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
INTEGER, INTENT( IN ) :: InComm     ! Communicator
INTEGER, INTENT( IN ) :: Op         ! Operation (see header)
INTEGER, INTENT( IN ) :: Im         ! First dimension of 3-D array
INTEGER, INTENT( IN ) :: Jm         ! Second dimension of 3-D array
INTEGER, INTENT( IN ) :: Lm         ! Third dimension of 3-D array
```

INPUT/OUTPUT PARAMETERS:

```
REAL(CPP_REAL), INTENT( INOUT ):: Var(Im,Jm,LM) ! partial Var in, Var out
```

DESCRIPTION:

This utility makes a collective operation over all processes in communicator InComm. The USE_ARENAS version is fully functional, but assumes a global communicator.

REVISION HISTORY:

```
     00.08.07   Sawyer     Creation

#if !defined( USE_ARENAS )
      INTEGER Ierror
      REAL(CPP_REAL) Tmp(Im,Jm,Lm)
#else
      INTEGER Icnt, Length, Ipe, I, J, L
#endif

#if defined( USE_ARENAS )
      Length = Im*Jm*Lm
      Icnt   = GID*Length
      DO l=1,Lm
        DO j=1,Jm
          DO i=1,Im
            Icnt = Icnt + 1
            DataBuf( Icnt ) = Var( i,j,l )
          ENDDO
        ENDDO
      ENDDO
      CALL mlp_barrier(gid,gsize)

      IF ( Op .EQ. CPP_SUM_OP ) THEN
        Var = 0.0            !  Must be done the same on all PEs
        Icnt = 0
        DO Ipe = 1, Gsize
          DO l=1,Lm
            DO j=1,Jm
              DO i=1,Im
                Icnt = Icnt + 1
                Var( i,j,l ) = Var( i,j,l ) + DataBuf( Icnt )
              ENDDO
            ENDDO
          ENDDO
        ENDDO
      ELSEIF ( Op .EQ. CPP_MAX_OP ) THEN
        Icnt = 0
        DO Ipe = 0, Gsize-1
          IF ( Ipe .NE. GID ) THEN
            DO l=1,Lm
              DO j=1,Jm
                DO i=1,Im
                  Icnt = Icnt + 1
                  IF ( Var( i,j,l ) .LT. DataBuf( Icnt ) )               &
     &                      Var( i,j,l ) = DataBuf( Icnt )
                ENDDO
              ENDDO
            ENDDO
          ELSE
```

```
          Icnt = Icnt + Length
        ENDIF
      ENDDO
    ELSEIF ( Op .EQ. CPP_MIN_OP ) THEN
      Icnt = 0
      DO Ipe = 0, Gsize-1
        IF ( Ipe .NE. GID ) THEN
          DO l=1,Lm
            DO j=1,Jm
              DO i=1,Im
                Icnt = Icnt + 1
                IF ( Var( i,j,l ) .GT. DataBuf( Icnt ) )               &
     &                     Var( i,j,l ) = DataBuf( Icnt )
              ENDDO
            ENDDO
          ENDDO
        ELSE
          Icnt = Icnt + Length
        ENDIF
      ENDDO
    ELSEIF ( Op .EQ. CPP_BCAST_OP ) THEN
      Icnt = 0
      DO l=1,Lm
        DO j=1,Jm
          DO i=1,Im
            Icnt = Icnt + 1
            Var( i,j,l ) = DataBuf( Icnt )
          ENDDO
        ENDDO
      ENDDO
    ENDIF
    CALL mlp_barrier(gid,gsize)
#else
    IF ( Op .EQ. BCSTOP ) THEN
      CALL MPI_BCAST( Var, Im*Jm*Lm, CPP_MPI_REAL, 0, InComm, Ierror )
    ELSE
      CALL MPI_ALLREDUCE( Var, Tmp, Im*Jm*Lm, CPP_MPI_REAL,            &
     &                    Op, InComm, Ierror )
      Var = Tmp
    ENDIF
#endif
    RETURN
```

71.7.24 ParCollective0DInt — Perform global Collective of a scalar (Source File: parutilitiesmodule.F90)

INTERFACE:

```
      SUBROUTINE ParCollective0DInt( InComm, Op, Var )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      INTEGER, INTENT( IN ) :: InComm   ! Communicator
      INTEGER, INTENT( IN ) :: Op       ! Operation (see header)
```

INPUT/OUTPUT PARAMETERS:

```
      INTEGER, INTENT( INOUT ) :: Var ! partial Var in, Var out
```

DESCRIPTION:

This utility makes a collective operation over all processes in communicator InComm. The USE_ARENAS version is fully functional, but assumes a global communicator.

REVISION HISTORY:

```
     00.08.07   Sawyer     Creation
```

```
#if !defined( USE_ARENAS )
      INTEGER Ierror
      INTEGER    Tmp
#else
      INTEGER Ipe
#endif


#if defined( USE_ARENAS )
      IntBuf( GID+1 ) = Var
      CALL mlp_barrier(gid,gsize)

      IF ( Op .EQ. CPP_SUM_OP ) THEN
        Var = 0.0            !  Must be done the same on all PEs
        DO Ipe = 1, Gsize
          Var = Var + IntBuf( Ipe )
        ENDDO
      ELSEIF ( Op .EQ. CPP_MAX_OP ) THEN
        DO Ipe = 1, Gsize
          IF ( Var .LT. IntBuf( Ipe ) ) Var = IntBuf( Ipe+1 )
        ENDDO
      ELSEIF ( Op .EQ. CPP_MIN_OP ) THEN
        DO Ipe = 1, Gsize
          IF ( Var .GT. IntBuf( Ipe ) ) Var = IntBuf( Ipe )
        ENDDO
      ELSEIF ( Op .EQ. CPP_BCAST_OP ) THEN
        Var = IntBuf(1)
      ENDIF
      CALL mlp_barrier(gid,gsize)
#else
      IF ( Op .EQ. BCSTOP ) THEN
        CALL MPI_BCAST( Var, 1, CPP_MPI_INTEGER, 0, InComm, Ierror )
      ELSE
        CALL MPI_ALLREDUCE( Var,Tmp,1,CPP_MPI_INTEGER,Op,InComm,Ierror )
        Var = Tmp
```

```
      ENDIF
#endif
      RETURN
```

71.7.25 ParCollective1DInt — Perform component-wise global (Source File: parutilitiesmodule.F90)

collective operations of int vector

INTERFACE:

```
      SUBROUTINE ParCollective1DInt( InComm, Op, Im, Var )
```

USES:

```
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      INTEGER, INTENT( IN ) :: InComm   ! Communicator
      INTEGER, INTENT( IN ) :: Op       ! Operation (see header)
      INTEGER, INTENT( IN ) :: Im       ! Size of 1-D array
```

INPUT/OUTPUT PARAMETERS:

```
      INTEGER, INTENT( INOUT ) :: Var(Im) ! partial Var in, Var out
```

DESCRIPTION:

This utility makes a collective operation over all processes in communicator InComm.
The USE_ARENAS version is fully functional, but assumes a global communicator.

REVISION HISTORY:

```
     00.08.07   Sawyer     Creation

#if !defined( USE_ARENAS )
      INTEGER Ierror
      INTEGER Tmp(Im)
#else
      INTEGER Icnt, Ipe, I
#endif

#if defined( USE_ARENAS )
      Icnt   = GID*Im
      DO i=1,Im
        Icnt = Icnt + 1
        IntBuf( Icnt ) = Var( i )
      ENDDO
      CALL mlp_barrier(gid,gsize)

      IF ( Op .EQ. CPP_SUM_OP ) THEN
```

```
        Var = 0.0            !  Must be done the same on all PEs
        Icnt = 0
        DO Ipe = 1, Gsize
          DO i=1,Im
            Icnt = Icnt + 1
            Var( i ) = Var( i ) + IntBuf( Icnt )
          ENDDO
        ENDDO
      ELSEIF ( Op .EQ. CPP_MAX_OP ) THEN
        Icnt = 0
        DO Ipe = 0, Gsize-1
          IF ( Ipe .NE. GID ) THEN
            DO i=1,Im
              Icnt = Icnt + 1
              IF ( Var(i) .LT. IntBuf(Icnt) ) Var(i) = IntBuf(Icnt)
            ENDDO
          ELSE
            Icnt = Icnt + Im
          ENDIF
        ENDDO
      ELSEIF ( Op .EQ. CPP_MIN_OP ) THEN
        Icnt = 0
        DO Ipe = 0, Gsize-1
          IF ( Ipe .NE. GID ) THEN
            DO i=1,Im
              Icnt = Icnt + 1
              IF ( Var(i) .GT. IntBuf(Icnt) ) Var(i) = IntBuf(Icnt)
            ENDDO
          ELSE
            Icnt = Icnt + Im
          ENDIF
        ENDDO
      ELSEIF ( Op .EQ. CPP_BCAST_OP ) THEN
        DO i=1,Im
          Var(i) = IntBuf(i)
        ENDDO
      ENDIF
      CALL mlp_barrier(gid,gsize)
#else
      IF ( Op .EQ. BCSTOP ) THEN
        CALL MPI_BCAST( Var, Im, CPP_MPI_INTEGER, 0, InComm, Ierror )
      ELSE
        CALL MPI_ALLREDUCE( Var,Tmp,Im,CPP_MPI_INTEGER,Op,InComm,Ierror )
        Var = Tmp
      ENDIF
#endif
      RETURN
```

71.7.26 scatter — wrapper for PILGRIM utility (Source File: puminterfaces.F90)

INTERFACE:

```
  ****6***0*********0*********0*********0*********0*********0**********72
       subroutine scatter( input, decomp, output, incomm )
  ****6***0*********0*********0*********0*********0*********0**********72
```

USES:

```
       use LIS_precisionMod
       use decompmodule, only : decomptype
 #if defined( SPMD )
       use parutilitiesmodule, only : parscatter
 #endif
       implicit none
```

INPUT PARAMETERS:

```
       real(CPP_REAL) input(*)              ! Input array (global)
       type (decomptype) :: decomp          ! Decomposition
       integer incomm                       ! Communicator
```

OUTPUT PARAMETERS:

```
       real(CPP_REAL) output(*)             ! Output array (local)
```

DESCRIPTION:

Scatter the global input array (on PE 0) according to decomposition to the local output array. This intermediate routine is a way to trick the compiler into passing 1-D arrays to the parutilitiesmodule method parscatter.

REVISION HISTORY:

```
     WS  00.11.28:  Creation
     AAM 01.05.08:  Added communicator as input argument
```

```
#if defined( SPMD )
      call parscatter( incomm, 0, input, decomp, output )
#endif
      return
```

71.7.27 gather — wrapper for PILGRIM utility (Source File: puminterfaces.F90)

INTERFACE:

```
  ****6***0*********0*********0*********0*********0*********0**********72
       subroutine gather( input, decomp, output )
  ****6***0*********0*********0*********0*********0*********0**********72
```

USES:

```
       use LIS_precisionMod
       use decompmodule, only : decomptype
 #if defined( SPMD )
       use parutilitiesmodule, only : commglobal, pargather
 #endif
       implicit none
```

INPUT PARAMETERS:

```
      real(CPP_REAL) input(*)              ! Input array (global)
      type (decomptype) :: decomp          ! Decomposition
```

OUTPUT PARAMETERS:

```
      real(CPP_REAL) output(*)             ! Output array (local)
```

DESCRIPTION:

Gather the local input array according to decomposition to the global output array (on PE 0). This intermediate routine is a way to trick the compiler into passing 1-D arrays to the parutilitiesmodule method pargather.

REVISION HISTORY:

```
     WS  00.11.28:  Creation
```

```
#if defined( SPMD )
      call pargather( commglobal, 0, input, decomp, output )
#endif
      return
```

71.7.28 begintransfer — wrapper for PILGRIM utility (Source File: puminterfaces.F90)

INTERFACE:

```
  ****6***0*********0*********0*********0*********0*********0**********72
      subroutine begintransfer( pattern, input, output )
  ****6***0*********0*********0*********0*********0*********0**********72
```

USES:

```
      use LIS_precisionMod
 #if defined( SPMD )
      use parutilitiesmodule, only : parpatterntype, parbegintransfer
 #endif
      implicit none
```

INPUT PARAMETERS:

```
      type (parpatterntype) :: pattern     ! Decomposition
      real(CPP_REAL) input(*)              ! Input array
```

OUTPUT PARAMETERS:

```
      real(CPP_REAL) output(*)             ! Output array
```

DESCRIPTION:

Initiate an asynchronous collective transfer of the input array to the output array as defined by the communication pattern.

REVISION HISTORY:

```
     WS  01.03.11:  Creation


#if defined( SPMD )
      call parbegintransfer( pattern, input, output )
#endif
      return
```

71.7.29 endtransfer — wrapper for PILGRIM utility (Source File: puminterfaces.F90)

INTERFACE:

```
  ****6***0*********0*********0*********0*********0*********0**********72
       subroutine endtransfer( pattern, input, output )
  ****6***0*********0*********0*********0*********0*********0**********72
```

USES:

```
       use LIS_precisionMod
 #if defined( SPMD )
       use parutilitiesmodule, only : parpatterntype, parendtransfer
 #endif
       implicit none
```

INPUT PARAMETERS:

```
       type (parpatterntype) :: pattern     ! Decomposition
       real(CPP_REAL) input(*)              ! Input array
```

OUTPUT PARAMETERS:

```
       real(CPP_REAL) output(*)             ! Output array
```

DESCRIPTION:

Complete an asynchronous collective transfer of the input array to the output array as defined by the communication pattern.

REVISION HISTORY:

```
     WS  01.03.11:  Creation


#if defined( SPMD )
      call parendtransfer( pattern, input, output )
#endif
      return
```

71.7.30 exchangevector — wrapper for PILGRIM utility (Source File: puminterfaces.F90)

INTERFACE:

```
  ****6***0*********0*********0*********0*********0*********0**********72
       subroutine exchangevector( incomm, inlen, input, outlen, output )
  ****6***0*********0*********0*********0*********0*********0**********72
```

USES:

```
       use LIS_precisionMod
 #if defined( SPMD )
       use parutilitiesmodule, only : parexchangevector
 #endif
       implicit none
```

INPUT PARAMETERS:

```
       integer incomm                      ! Communicator
       integer  inlen(*)                   ! Input lengths per PE
       real(CPP_REAL) input(*)             ! Input array (global)
```

OUTPUT PARAMETERS:

```
       integer  outlen(*)                  ! Output lengths per PE
       real(CPP_REAL) output(*)            ! Output array (local)
```

DESCRIPTION:

Perform a synchronous collective transfer of the input vector, blocked in segments to be sent to each PE in ascending order, and the lengths of the blocks given by inlen. The routine returns the output block lengths (those received on the local PE) and the vector output which is blocked by ascending PE order.

REVISION HISTORY:

```
     WS  01.03.11:  Creation
```

```
#if defined( SPMD )
      call parexchangevector( incomm, inlen, input, outlen, output )
#endif
      return
```

71.8 Fortran: Module Interface redistributemodule (Source File: redistributemodule.F90)

USES:

```
 #include "debug.h"
       use LIS_precisionMod, only: r8
       IMPLICIT NONE
```

DESCRIPTION:

REVISION HISTORY:

```
99.01.18   Sawyer     Creation
99.11.17   Sawyer     Added RedistributeStart, RedistributeFinish
00.07.20   Sawyer     Minor cosmetic changes
00.08.28   Sawyer     Accommodated change to ParEndTranfer interface
01.02.12   Sawyer     Converted to free format
```

PUBLIC TYPES:

```
      PUBLIC RedistributeType
      PUBLIC RedistributeCreate, RedistributeFree, RedistributePerform
      PUBLIC RedistributeStart, RedistributeFinish

  Redistribution info

      TYPE RedistributeType
        INTEGER, POINTER     :: CountA(:) ! Per PE counts in Decomp A
        INTEGER, POINTER     :: CountB(:) ! Per PE counts in Decomp B
        INTEGER, POINTER     :: PermA(:)  ! Permutation in Decomp A
        INTEGER, POINTER     :: PermB(:)  ! Permutation in Decomp B
      END TYPE RedistributeType
```

71.8.1 RedistributeCreate — Create an inter-decomp. structure

INTERFACE:

```
      SUBROUTINE RedistributeCreate( DecompA, DecompB, Inter )
```

USES:

```
      USE decompmodule, ONLY:  DecompType, DecompGlobalToLocal
      USE parutilitiesmodule, ONLY:  GID, Gsize
      IMPLICIT NONE
#include "mpif.h"
```

INPUT PARAMETERS:

```
      TYPE(DecompType), INTENT( IN ) :: DecompA       ! Decomposition A
      TYPE(DecompType), INTENT( IN ) :: DecompB       ! Decomposition B
```

OUTPUT PARAMETERS:

```
      TYPE(RedistributeType), INTENT( OUT ) :: Inter  ! Inter info.
```

DESCRIPTION:

This routine constructs a RedistributeType structure which can be efficiently used in the RedistributePerform routine.

SYSTEM ROUTINES:

```
       ALLOCATE
```

REVISION HISTORY:

```
     99.01.15   Sawyer     Creation
```

BUGS:

```
      Currently untested.


!
! !LOCAL VARIABLES:
      INTEGER IndexA, IndexB, I, J, K, Tag, Local, Pe, Offsets( Gsize )
      LOGICAL Found, First, Search

      CPP_ENTER_PROCEDURE( "REDISTRIBUTECREATE" )
!
! Allocate the number of entries and list head arrays
!
      CPP_ASSERT_F90( SIZE( DecompA%NumEntries ).EQ. Gsize )

      CPP_ASSERT_F90( SIZE( DecompB%NumEntries ).EQ. Gsize )

      ALLOCATE( Inter%CountA( Gsize ) )
      ALLOCATE( Inter%CountB( Gsize ) )

      ALLOCATE( Inter%PermA( DecompA%NumEntries( GID+1 ) ) )
      ALLOCATE( Inter%PermB( DecompB%NumEntries( GID+1 ) ) )

      Inter%CountA = 0
      Inter%CountB = 0
      IndexA       = 0
      IndexB       = 0

      DO I = 1, Gsize
        DO J = 1, SIZE( DecompB%Head(I)%StartTags )
          First = .TRUE.
          DO Tag=DecompB%Head(I)%StartTags(J),DecompB%Head(I)%EndTags(J)

!
! CODE INLINED FOR PERFORMANCE
!
!!!            CALL DecompGlobalToLocal( DecompA, Tag, Local, Pe )

            Search = .TRUE.
            IF ( .NOT. First )                                          &
              Search = First .OR. Tag .GT. DecompA%Head(Pe+1)%EndTags(K)

            IF ( Search ) THEN
              First = .FALSE.

!
! Search over all the PEs
!
              Pe = -1
              Found = .FALSE.
```

```
              DO WHILE ( .NOT. Found )
!
! Copy the number of entries on each PE
!
                Pe = Pe + 1
!
! Search through the local data segment
!
                Local = 1
                K = 1
                DO WHILE ( .NOT. Found .AND.                              &
                    K .LE. SIZE( DecompA%Head(Pe+1)%StartTags ) )
                  IF ( Tag .GE. DecompA%Head(Pe+1)%StartTags(K) .AND.    &
                       Tag .LE. DecompA%Head(Pe+1)%EndTags(K) ) THEN
                    Local = Local+Tag - DecompA%Head(Pe+1)%StartTags(K)
                    Found = .TRUE.
                  ELSE
                    Local = Local + DecompA%Head(Pe+1)%EndTags(K) -      &
                            DecompA%Head(Pe+1)%StartTags(K) + 1
                    K = K+1
                  ENDIF
                ENDDO
!
! Emergency brake
!
                IF ( Pe.EQ.(SIZE(DecompA%Head)-1).AND. .NOT.Found ) THEN
                  Found = .TRUE.
                  Local = 0
                  Pe    = -1
                ENDIF
              ENDDO

!
! END OF INLINING
!
            ELSE
              Local = Local + 1
            ENDIF
!
! Calculate the sorting permutation for A
!
            IF ( Pe .EQ. GID ) THEN
              Inter%CountA( I ) = Inter%CountA( I ) + 1
              IndexA = IndexA + 1
              Inter%PermA( IndexA ) = local
            ENDIF
!
! Calculate the sorting permutation for B
!
            IF ( I-1 .EQ. GID ) THEN
              Inter%CountB( Pe+1 ) = Inter%CountB( Pe+1 ) + 1
              IndexB = IndexB + 1
              Inter%PermB( IndexB ) = Inter%CountB( Pe+1 )*Gsize + Pe
            ENDIF
          ENDDO
```

```
        ENDDO
      ENDDO

!
! Finally decode PermB and add in the proper offsets
!
      Offsets = 0
      DO I=1, Gsize-1
        Offsets( I+1 ) = Offsets( I ) + Inter%CountB( I )
      ENDDO
      DO I=1, IndexB
        Pe = MOD( Inter%PermB( I ), Gsize )
        Inter%PermB( I ) = Inter%PermB(I)/Gsize + Offsets( Pe+1 )
      ENDDO

      CPP_LEAVE_PROCEDURE( "REDISTRIBUTECREATE" )
      RETURN
```

71.8.2 RedistributePerform — Perform the Redistribution

INTERFACE:

```
SUBROUTINE RedistributePerform( Inter, Forward, Input, Output )
```

USES:

```
USE parutilitiesmodule, ONLY : CommGlobal, Gsize,                  &
                               ParExchangeVector,GID
IMPLICIT NONE
```

INPUT PARAMETERS:

```
TYPE(RedistributeType), INTENT( INOUT ) :: Inter ! Inter info.
LOGICAL             :: Forward     ! True: A -> B  False: B -> A
REAL(CPP_REAL), INTENT( IN )  :: Input( * )  ! Input Array
```

OUTPUT PARAMETERS:

```
REAL(CPP_REAL), INTENT( OUT ) :: Output( * ) ! Output Array
```

DESCRIPTION:

This routine performs the redistribution of Input to Output according to the RedistributionType data structure Inter. The redistribution can be from A -¿ B (”forward”) or B -¿ A (”backward”). This feature has been added to avoid the need of a separate Inter (which requires considerable memory) to perform the backward redistribution.

SYSTEM ROUTINES:

```
ALLOCATE, DEALLOCATE
```

BUGS:

```
Currently limited to the global communicator.
```

REVISION HISTORY:

```
     99.01.15   Sawyer     Creation

! !LOCAL VARIABLES:
      INTEGER I, Ierr, LenOutBuf( Gsize )
      REAL(CPP_REAL), ALLOCATABLE :: InBuf(:), OutBuf(:)

      CPP_ENTER_PROCEDURE( "REDISTRIBUTEPERFORM" )

      IF ( Forward ) THEN
!
! Forward redistribution
!
        ALLOCATE( InBuf( SUM( Inter%CountA ) ) )
        ALLOCATE( OutBuf( SUM( Inter%CountB ) ) )
        DO I = 1, SUM( Inter%CountA )
          InBuf( I ) = Input( Inter%PermA( I ) )
        ENDDO

        CALL ParExchangeVector( CommGlobal, Inter%CountA, InBuf,         &
                                LenOutBuf, OutBuf )
        DO I = 1, SUM( Inter%CountB )
          Output( I ) = OutBuf( Inter%PermB( I ) )
        ENDDO
        DEALLOCATE( OutBuf )
        DEALLOCATE( InBuf )

      ELSE
!
! Backward redistribution
!
        ALLOCATE( InBuf( SUM( Inter%CountB ) ) )
        ALLOCATE( OutBuf( SUM( Inter%CountA ) ) )
        DO I = 1, SUM( Inter%CountB )
          InBuf( Inter%PermB( I ) ) = Input( I )
        ENDDO

        CALL ParExchangeVector( CommGlobal, Inter%CountB, InBuf,         &
                                LenOutBuf, OutBuf )
        DO I = 1, SUM( Inter%CountA )
          Output( Inter%PermA( I ) ) = OutBuf( I )
        ENDDO
        DEALLOCATE( OutBuf )
        DEALLOCATE( InBuf )

      ENDIF
      CPP_LEAVE_PROCEDURE( "REDISTRIBUTEPERFORM" )
      RETURN
```

71.8.3 RedistributeFree — Free an inter-decomp. structure

INTERFACE:

```
      SUBROUTINE RedistributeFree( Inter )
```

USES:

```
      IMPLICIT NONE
```

INPUT/OUTPUT PARAMETERS:

```
      TYPE(RedistributeType), INTENT( INOUT ) :: Inter ! Inter info.
```

DESCRIPTION:

This routine frees a RedistributeType structure.

SYSTEM ROUTINES:

```
      DEALLOCATE
```

REVISION HISTORY:

```
    99.01.15   Sawyer     Creation
```

BUGS:

```
     Currently untested.


! !LOCAL VARIABLES:
      INTEGER Ierr

      CPP_ENTER_PROCEDURE( "REDISTRIBUTEFREE" )

      DEALLOCATE( Inter%PermB  )
      DEALLOCATE( Inter%PermA  )
      DEALLOCATE( Inter%CountB )
      DEALLOCATE( Inter%CountA )

      CPP_LEAVE_PROCEDURE( "REDISTRIBUTEFREE" )
      RETURN
```

71.8.4 RedistributeStart — Perform Asynchronous Redistribution

INTERFACE:

```
      SUBROUTINE RedistributeStart( Inter, Forward, Input )
```

USES:

```
      USE parutilitiesmodule, ONLY : CommGlobal, Gsize,                 &
                                     ParBeginTransfer,GID
      IMPLICIT NONE
```

INPUT PARAMETERS:

```
      TYPE(RedistributeType), INTENT( INOUT ) :: Inter ! Inter info.
      LOGICAL             :: Forward     ! True: A -> B  False: B -> A
      REAL(CPP_REAL), INTENT( IN )  :: Input( * )  ! Input Array
```

DESCRIPTION:

This routine starts an asynchronous redistribution of Input to Output according to the RedistributionType data structure Inter. The redistribution can be from A -¿ B (”forward”) or B -¿ A (”backward”). This feature has been added to avoid the need of a separate Inter (which requires considerable memory) to perform the backward redistribution.
Beware: both RedistributeStart and RedistributeFinish *must* be called with the same values of Inter and Forward. Nesting of asynchronous distributions is forbidden. In addition, any other communication in the between RedistributeStart and RedistributeFinish cannot used the communicator ”CommGlobal” provided by parutilitiesmodule.

SYSTEM ROUTINES:

```
ALLOCATE
```

REVISION HISTORY:

```
99.11.17   Sawyer     Creation from RedistributePerform
```

BUGS:

```
Currently limited to the global communicator.
```

```
! !LOCAL VARIABLES:
      INTEGER I, Ierr, Dest( Gsize ), Src( Gsize )

      CPP_ENTER_PROCEDURE( "REDISTRIBUTESTART" )

      DO I = 1, Gsize
        Dest( I ) = I-1
        Src( I )  = I-1
      ENDDO

      IF ( Forward ) THEN
!
! Forward redistribution
!
        ALLOCATE( InStatic( SUM( Inter%CountA ) ) )
        ALLOCATE( OutStatic( SUM( Inter%CountB ) ) )
        DO I = 1, SUM( Inter%CountA )
          InStatic( I ) = Input( Inter%PermA( I ) )
        ENDDO
        CALL ParBeginTransfer( CommGlobal, Gsize, Gsize, Dest, Src,      &
                               InStatic, Inter%CountA,                   &
                               OutStatic, Inter%CountB )

      ELSE
!
! Backward redistribution
!
        ALLOCATE( InStatic( SUM( Inter%CountB ) ) )
        ALLOCATE( OutStatic( SUM( Inter%CountA ) ) )
        DO I = 1, SUM( Inter%CountB )
          InStatic( Inter%PermB( I ) ) = Input( I )
        ENDDO
```

```
      CALL ParBeginTransfer( CommGlobal, Gsize, Gsize, Dest, Src,      &
                             InStatic, Inter%CountB,                   &
                             OutStatic, Inter%CountA )

    ENDIF
    CPP_LEAVE_PROCEDURE( "REDISTRIBUTESTART" )
    RETURN
```

71.8.5 RedistributeFinish — Complete Asynchronous Redistribution

INTERFACE:

```
SUBROUTINE RedistributeFinish( Inter, Forward, Output )
```

USES:

```
USE parutilitiesmodule, ONLY: CommGlobal,Gsize,ParEndTransfer,GID
IMPLICIT NONE
```

INPUT PARAMETERS:

```
TYPE(RedistributeType), INTENT( INOUT ) :: Inter ! Inter info.
LOGICAL             :: Forward     ! True: A -> B  False: B -> A
```

OUTPUT PARAMETERS:

```
REAL(CPP_REAL), INTENT( OUT ) :: Output( * ) ! Output Array
```

DESCRIPTION:

This routine completes an asynchronous redistribution of Input to Output according to the RedistributionType data structure Inter. The redistribution can be from A -¿ B ("forward") or B -¿ A ("backward"). This feature has been added to avoid the need of a separate Inter (which requires considerable memory) to perform the backward redistribution.
See additional documentation in RedistributeStart.

SYSTEM ROUTINES:

```
DEALLOCATE
```

REVISION HISTORY:

```
99.11.17   Sawyer     Creation from RedistributePerform
```

BUGS:

```
Currently limited to the global communicator.
```

```
! !LOCAL VARIABLES:
      INTEGER I, Dest( Gsize ), Src( Gsize )

      CPP_ENTER_PROCEDURE( "REDISTRIBUTEFINISH" )

      DO I = 1, Gsize
```

```
      Dest( I ) = I-1
      Src( I )  = I-1
    ENDDO

    IF ( Forward ) THEN
      CALL ParEndTransfer( CommGlobal, Gsize, Gsize, Dest, Src,        &
                           InStatic, Inter%CountA,                     &
                           OutStatic, Inter%CountB )
      DO I = 1, SUM( Inter%CountB )
        Output( I ) = OutStatic( Inter%PermB( I ) )
      ENDDO
    ELSE
      CALL ParEndTransfer( CommGlobal, Gsize, Gsize, Dest, Src,        &
                           InStatic, Inter%CountB,                     &
                           OutStatic, Inter%CountA )
      DO I = 1, SUM( Inter%CountA )
        Output( Inter%PermA( I ) ) = OutStatic( I )
      ENDDO
    ENDIF
    DEALLOCATE( OutStatic )
    DEALLOCATE( InStatic )

    CPP_LEAVE_PROCEDURE( "REDISTRIBUTEFINISH" )
    RETURN
```

---

72 Catchment Land Surface Model

This section describes the interface implementations for the Fortuna-2_5 version of the Catchment land surface model.

72.1 Fortran: Module Interface (Source File: clsmf25_DragCoefficientsMod.F90)

DragCoefficientsMod – Container for the GEOS drag coefficient utility.

USES:

```
 use clsmf25_MAPL_Constants

   implicit none
   private
```

PUBLIC MEMBER FUNCTIONS:

```
   public GETCDS
   public GETCDM
   public GETCDH
```

INTERFACE:

```
   interface GETCDS ! (TVA,UA,DZ,TVS,Z0,ZT,ZQ,  CT,CM,CQ,CN,RI,DCT,MSK)
```

```
  module procedure GETCDS_1D
  module procedure GETCDS_2D
end interface

interface GETCDM ! (TVA,UA,DZ,TVS,Z0, CM,CN,RI)
  module procedure GETCDM_1D
  module procedure GETCDM_2D
end interface

interface GETCDH ! (TVA,UA,DZ,TVS,ZH,ZQ,CN,RI,  CT,CQ,DCT,DCQ)
  module procedure GETCDH_1D
  module procedure GETCDH_2D
end interface
```

72.1.1 GETCDS – Computes drag coefficients at the surface

A utility component used to compute drag coefficients using the Louis scheme. It computes two coefficients, one for momentum , C_D^u, and one for heat and moisture, C_D^h. It also, optionally, computes the derivative of the drag coefficients with respect to surface temperature and surface humidity.
Surface-air quantities are subscripted a and surface quantities are sucscripted s. The difference of the two is, for example, $\delta T = T_a - T_s$.
The drag coefficients have the form:

$$C_D^m = C_D^n f_m(\mathrm{Ri})$$

and

$$C_D^h = C_D^n f_h(\mathrm{Ri}),$$

where

$$C_D^n = (\frac{\kappa}{\log(\zeta)})^2$$

is the neutral drag coefficient and $\zeta = \frac{\delta z}{z_0} + 1$. The Von Karman constant, κ, is taken as 0.40 and z_0 is the surface roughness. The height corresponding to surface-air quantities is taken as one-half the thickness of the lowest model layer.
The surface bulk Richardson number, Ri, is defined as

$$\mathrm{Ri} = \delta T_v \frac{\frac{g}{\delta z}}{T_o * \frac{u_a}{\delta z}^2}$$

$$= \delta T_v \alpha,$$

where $T_v = T(1 + \epsilon q)$ is the virtual temperature, $\epsilon = \frac{M_a}{M_w} - 1$, M_a and M_w are the molecular weights of dry air and water, u_a is the surface-air wind speed, and α is defined to simplify the calculation. In the code we use T_v in place of T_o, but assume that the factor α is constatnt when differentiating the Ricahrdson number with respect to δT_v.
The two universal functions of the Richardson number, f_m and f_h, are taken from Louis et al (1979). For unstable conditions (Ri≤ 0), they are:

$$f_m = (1 - 2b\psi)$$

and

$$f_h = (1 - 3b\psi),$$

where

$$\psi = \frac{\mathrm{Ri}}{1 + 3bcC_D^n\sqrt{-\mathrm{Ri}\zeta}}.$$

For stable condition (Ri$\geq$ 0), they are

$$f_m = \frac{1}{1.0 + 2b\frac{\mathrm{Ri}}{\psi}}$$

and

$$f_h = \frac{1}{1.0 + 3b\mathrm{Ri}\psi},$$

where

$$\psi = \sqrt{1 + d\mathrm{Ri}}.$$

As in Louis et al (1979) the parameters appearing in these are taken as $b = c = d = 5$.
We also require the derivative of C_D^h with respect to δT_v.

$$\frac{\partial C_D^h}{\partial \delta T_v} = C_D^n \frac{\mathrm{d}f_h}{\mathrm{dRi}}\alpha$$

The derivatives of f_h are as follows.
For unstable conditions:

$$\frac{\mathrm{d}f_h}{\mathrm{dRi}} = -3b\frac{rmd\psi}{\mathrm{dRi}},$$

where

$$\frac{\mathrm{d}\psi}{\mathrm{dRi}} = \frac{\psi}{\mathrm{Ri}}(1 + \frac{3bcC_D^n\psi\zeta}{2\sqrt{-\mathrm{Ri}\zeta}})$$

For stable conditions:

$$\frac{\mathrm{d}f_h}{\mathrm{dRi}} = -f_h^2 3b(\psi + \mathrm{Ri}\frac{d}{2\psi}),$$

INTERFACE:

```
subroutine GETCDS (TVA,UA,DZ,TVS,Z0,LOUIS,  CT,CM,CN,RI,DCT)
```

```
!INPUT ARGUMENTS:
```

TVA	(K)	Virtual surface air temperature
UA	(m s^{-1})	Surface wind speed
UU	(m)	Altitude of UA
TVS	(K)	Virtual surface air temperature
Z0	(m)	Surface roughness
LOUIS		Louis scheme parameter (usually 5)

```
!OUTPUT ARGUMENTS:
```

CT	Drag coefficient for heat and scalars
CM	Drag coefficient for momentum

CN		Neutral drag coefficient
RI		Surface Bulk Richardson number
DCT	(K^{-1})	$\frac{\partial C_D^h}{\partial \delta T_v}$

72.2 Fortran: Module Interface – A container module for global constants (Source File: clsmf25_MAPL_constants.F90)

!PUBLIC VARIABLES:

```
real(kind=8), parameter, public :: MAPL_PI_R8     = 3.14159265358979323846
real, parameter, public :: MAPL_PI     = MAPL_PI_R8
real, parameter, public :: MAPL_GRAV   = 9.80                   ! m^2/s
real, parameter, public :: MAPL_RADIUS = 6376.0E3               ! m
real, parameter, public :: MAPL_OMEGA  = 2.0*MAPL_PI/86164.0    ! 1/s
real, parameter, public :: MAPL_ALHL   = 2.4665E6               ! J/kg @15C
real, parameter, public :: MAPL_ALHF   = 3.3370E5               ! J/kg
real, parameter, public :: MAPL_ALHS   = MAPL_ALHL+MAPL_ALHF    ! J/kg
real, parameter, public :: MAPL_STFBOL = 5.6734E-8              ! W/(m^2 K^4)
real, parameter, public :: MAPL_AIRMW  = 28.97                  ! kg/Kmole
real, parameter, public :: MAPL_H2OMW  = 18.01                  ! kg/Kmole
real, parameter, public :: MAPL_O3MW   = 47.9982                ! kg/Kmole
real, parameter, public :: MAPL_RUNIV  = 8314.3                 ! J/(Kmole K)
real, parameter, public :: MAPL_KAPPA  = 2.0/7.0                ! --
real, parameter, public :: MAPL_RVAP   = MAPL_RUNIV/MAPL_H2OMW  ! J/(kg K)
real, parameter, public :: MAPL_RGAS   = MAPL_RUNIV/MAPL_AIRMW  ! J/(kg K)
real, parameter, public :: MAPL_CP     = MAPL_RGAS/MAPL_KAPPA   ! J/(kg K)
real, parameter, public :: MAPL_P00    = 100000.0               ! Pa
real, parameter, public :: MAPL_CAPICE = 2000.                  ! J/(K kg)
real, parameter, public :: MAPL_CAPWTR = 4218.                  ! J/(K kg)
real, parameter, public :: MAPL_RHOWTR = 1000.                  ! kg/m^3
real, parameter, public :: MAPL_NUAIR  = 1.533E-5               ! m^2/S (@ 18C)
real, parameter, public :: MAPL_TICE   = 273.16                 ! K
real, parameter, public :: MAPL_SRFPRS = 98470                  ! Pa
real, parameter, public :: MAPL_KARMAN = 0.40                   ! --
real, parameter, public :: MAPL_USMIN  = 1.00                   ! m/s
real, parameter, public :: MAPL_VIREPS = MAPL_AIRMW/MAPL_H2OMW-1.0   ! --
real, parameter, public :: MAPL_AVOGAD = 6.023E26               ! 1/kmol

integer,parameter, public :: MAPL_R8 = selected_real_kind(12) ! 8 byte real
integer,parameter, public :: MAPL_R4 = selected_real_kind( 6) ! 4 byte real
integer,parameter, public :: MAPL_RN = kind(1.0)              ! native real
integer,parameter, public :: MAPL_I8 = selected_int_kind (13) ! 8 byte integer
integer,parameter, public :: MAPL_I4 = selected_int_kind ( 6) ! 4 byte integer
integer,parameter, public :: MAPL_IN = kind(1)                ! native integer
```

72.2.1 clsmf25_read_land_parameters3 (Source File: clsmf25_compute_land_parameters.F90)

REVISION HISTORY:

```
12 May 2003 Rolf Reichle, Initial Specification
06 Jun 2005 Rolf Reichle, adapted to read "SiB2_V2" parameters
10 Jul 2006 James Geiger, Implementation in LIS
23 Nov 2012: David Mocko, Added Catchment Fortuna-2.5
```

INTERFACE:

```
subroutine clsmf25_compute_land_parameters(nest)
```

USES:

```
!use tile_coord_types
use clsmf25_constants, only : N_gt
use clsmf25_lsmMod
use LIS_coreMod, only : LIS_rc, LIS_surface
use LIS_logMod,    only : LIS_logunit, LIS_endrun

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Reads in the vegetation, soil properties and topographic parameters from global files and outputs parameters for the domain.
Additional parameters are derived from the ones that have been read from files.
The arguments are:

nest index of the nest

72.3 Fortran: Module Interface clsmf25_diagn_routines (Source File: clsmf25_diagn_routines

This module contains subroutinte for computing various diagnostics in the Catchment model

REVISION HISTORY:

```
Feb 5,  2004, Rolf Reichle:  Initial Specification
Mar 19, 2004, Rolf Reichle:  Revised subroutine, calc_soil_moist
Aug 31, 2004, Rolf Reichle:  Added calc_tsurf
Jun 21, 2005, Rolf Reichle:  Added calc_arX
```

72.3.1 calc_tsurf (Source File: clsmf25_diagn_routines.F90)

INTERFACE:

```
subroutine calc_tsurf( N_cat, cat_param, cat_progn, tsurf )
```

DESCRIPTION:

Calculate diagnostic surface temperature "tsurf" from prognostics

---

72.3.2 calc_arX (Source File: clsmf25_diagn_routines.F90)

INTERFACE:

```
subroutine calc_arX( N_cat, cat_param, cat_progn, ar1, ar2, ar4 )
```

DESCRIPTION:

???

72.3.3 calc_asnow (Source File: clsmf25_diagn_routines.F90)

INTERFACE:

```
subroutine calc_asnow( N_cat, cat_progn, asnow )
```

DESCRIPTION:

Calculate diagnostic snow area from prognostic SWE

72.4 Fortran: Module Interface clsmf25_drv_types (Source File: clsmf25_drv_types.F90)

definition of types and associated operators for Catchment Model driver
IMPORTANT: When adding a field to any of the derived types, must also update the associated assignment and operator definitions. THERE IS NO WARNING/ERROR IF OPERATOR IS NOT DEFINED FOR ALL FIELDS

REVISION HISTORY:

```
reichle, 10 May 2005
reichle, 10 Jun 2005 - converted met_force_type to ALMA
23 Nov 2012: David Mocko, Added RefH to met_force_type
```

INTERFACE:

72.4.1 clsmf25_dynsetup.F90 (Source File: clsmf25_dynsetup.F90)

Sets the time varying parameters for catchment

REVISION HISTORY:

```
28 Jan 2002: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine clsmf25_dynsetup(n)
```

USES:

72.5 Fortran: Module Interface clsmf25_esat_qsat (Source File: clsmf25_esat_qsat.F90)

cleaned up qsat.f and function esat() from process_new.f

REVISION HISTORY:

```
reichle, 13 Jun 2005
```

INTERFACE:

```
module clsmf25_esat_qsat
```

72.5.1 clsmf25_f2t (Source File: clsmf25_f2t.F90)

REVISION HISTORY:

```
16 Dec 2005: Sujay Kumar; Initial Specification
11 Oct 2006: Sujay Kumar; modified to work with ESMF_State design
23 Nov 2012: David Mocko, Added forcing height option
```

INTERFACE:

```
subroutine clsmf25_f2t(n)
```

USES:

```
use ESMF
use LIS_coreMod , only : LIS_rc,LIS_surface
use LIS_FORC_AttributesMod
use LIS_metforcingMod, only : LIS_FORC_State
use LIS_constantsMod,       only : LIS_CONST_TKFRZ
use LIS_logMod,             only : LIS_logunit,LIS_verify,LIS_endrun
use clsmf25_lsmMod, only : clsmf25_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the Catchment model tiles.
The arguments are:

n index of the nest

72.5.2 clsmf25_finalize (Source File: clsmf25_finalize.F90)

Complete the finalize routines for catchment

REVISION HISTORY:

```
11 Feb 2006: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine clsmf25_finalize()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use clsmf25_lsmMod
```

72.5.3 clsmf25_lsmMod.F90 (Source File: clsmf25_lsmMod.F90)

Module for 1-D catchment land model driver variable initialization

REVISION HISTORY:

```
15 Dec 2005; Sujay Kumar, Initial Code
23 Nov 2012: David Mocko, Additional configs for Catchment Fortuna-2.5
```

INTERFACE:

```
module clsmf25_lsmMod
```

USES:

```
use clsmf25_types
use clsmf25_drv_types
use clsmf25_modis_alb_types

PRIVATE

-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: clsmf25_varalloc
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: clsmf25_struc

type, public ::  cat_type_dec
   integer :: catopen             ! Keeps track of opening files
   integer :: numout              ! Counts number of output times for CLSM
   real    :: rstInterval         ! CLSM restart interval (seconds)
   integer :: forc_count
   real    :: ts
   integer :: varid(39)           ! For netcdf output
   real    :: dzsfcrd             ! top soil layer depth from lis.config
   real    :: initSM              ! initial soil moisture
   real    :: initST              ! initial soil temperature
   real    :: RefHfixed           ! reference height [m]
   integer :: turbscheme          ! Catchment turbulence scheme option
```

```
    integer :: uselaiflag
    integer :: usemodisalbflag
    integer :: usegreennessflag
    logical :: modelstart

    character*100 :: rfile         ! restart file
    logical, allocatable               :: good_forcing_mask(:)
    type(cat_param_type), allocatable  :: cat_param(:)
    type(met_force_type), allocatable  :: met_force(:)
    type(cat_progn_type), allocatable  :: cat_progn(:)
    type(cat_diagn_type), allocatable  :: cat_diagn(:)
    type(cat_output_type), allocatable :: cat_output(:)
    type(modis_alb_type), allocatable  :: modis_alb_param(:,:)


end type cat_type_dec

type(cat_type_dec), allocatable :: clsmf25_struc(:)
```

72.5.4 clsmf25_varalloc (Source File: clsmf25_lsmMod.F90)

Reads in runtime catchment parameters, allocates memory for variables

INTERFACE:

```
subroutine clsmf25_varalloc()
```

USES:

```
  use LIS_coreMod, only : LIS_rc
  use LIS_surfaceModelDataMod, only : LIS_sfmodel_struc
  use LIS_timeMgrMod
  use LIS_logMod
```

72.5.5 clsmf25_main (Source File: clsmf25_main.F90)

REVISION HISTORY:

```
13 Jun 2005 Rolf Reichle, Initial Specification
17 Dec 2005 Sujay Kumar, Implementation in LIS
23 Nov 2012: David Mocko, Added Catchment Fortuna-2.5
27 Mar 2013: David Mocko, Updates for ALMA outputs
```

INTERFACE:

```
subroutine clsmf25_main(nid)

  ! reichle, 13 Jun 2005
  ! qliu+reichle, 14 Aug 2008 - major overhaul, use subroutine louissurface()
  ! reichle, 29 Nov 2010 - added Helfand Monin-Obukhov surface layer turbulence scheme
  ! reichle, 20 Dec 2011 - reinstated "PARdrct" and "PARdffs" for MERRA-Land file specs
  ! reichle, 28 Dec 2011 - removed field "totalb" from "cat_diagn" structure
```

USES:

```
use LIS_coreMod
use LIS_FORC_AttributesMod
use LIS_timeMgrMod, only : LIS_isAlarmRinging
use LIS_albedoMod, only : LIS_alb
use LIS_vegDataMod, only : LIS_lai,LIS_gfrac
use LIS_logMod,    only : LIS_logunit, LIS_endrun
use LIS_histDataMod
use LIS_pluginIndices

use clsmf25_lsmMod, only : clsmf25_struc
use clsmf25_constants
use clsmf25_MAPL_constants, ONLY: &
     stefan_boltzmann => MAPL_STFBOL, MAPL_USMIN,                    &
     lhe => MAPL_ALHL, lhf => MAPL_ALHF
use clsmf25_types
use clsmf25_esat_qsat
use clsmf25_model
use clsmf25_diagn_routines

implicit none

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

DESCRIPTION:

This is the entry point for calling the Catchment LSM physics. This routine calls `catchment` routine that performs the land surface computations, to solve water and energy equations.
The arguments are:

nid index of the nest

72.5.6 clsmf25_pmonth (Source File: clsmf25_pmonth_stage4.F90)

REVISION HISTORY:

```
23 Nov 2012: David Mocko, Added Catchment Fortuna-2.5
```

INTERFACE:

```
SUBROUTINE clsmf25_PMONTH (&
       NCH,    ITYP,   JDAY,   ALAT,&
       GREEN, ZLT, Z0, D, BUG,&
       SQSCAT, Z2, DZM,&
       RSOIL1, RSOIL2, SATCAP, RDC,  U2FAC &
       &                  )
```

72.5.7 clsmf25_readcrd (Source File: clsmf25_readcrd.F90)

Routine to read Catchment specific parameters from the card file.

REVISION HISTORY:

```
12 Feb 2006; Sujay Kumar, Initial Code
23 Nov 2012: David Mocko, Additional configs for Catchment Fortuna-2.5
```

INTERFACE:

```
subroutine clsmf25_readcrd()
```

USES:

```
use clsmf25_lsmMod, only : clsmf25_struc
use LIS_logMod, only :LIS_logunit,LIS_verify
use LIS_timeMgrMod
use LIS_coreMod, only : LIS_config,LIS_rc
use ESMF
```

72.5.8 clsmf25_readrst (Source File: clsmf25_readrst.F90)

REVISION HISTORY:

```
16 Dec 2005: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine clsmf25_readrst
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_logMod
use LIS_fileIOMod
use LIS_timeMgrMod
use LIS_historyMod, only : LIS_readvar_restart
use clsmf25_constants
use clsmf25_lsmMod
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
```

DESCRIPTION:

This program reads restart files for Catchment. This includes all relevant water/energy storages, tile information.
The routines invoked are:

drv_readvar_restart (3.17.6)
reads a variable from the restart file

72.5.9 clsmf25_setup (Source File: clsmf25_setup.F90)

REVISION HISTORY:

```
16 Dec 2005: Sujay Kumar; Initial Code
23 Nov 2012: David Mocko, Updates for Catchment Fortuna-2.5
```

INTERFACE:

```
subroutine clsmf25_setup()
```

USES:

```
use LIS_coreMod,  only : LIS_rc,LIS_domain, LIS_surface
use LIS_logMod,   only : LIS_logunit
use LIS_soilsMod, only : LIS_soils
use clsmf25_lsmMod
use LIS_pluginIndices
```

DESCRIPTION:

This routine is the entry point to set up the parameters required for the Catchment Fortuna-2.5 LSM. These include the soils, greenness, albedo, etc.
The routines invoked are:

clsmf25_read_land_parameter (72.5.10)
reads specified 2d land parameter

clsmf25_read_land3d_parameter (72.5.11)
reads specified 3d land parameter

72.5.10 clsmf25_read_land_parameter (Source File: clsmf25_setup.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine clsmf25_read_land_parameter(n,vname,pvalue)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
    use netcdf
#endif
    use LIS_coreMod,        only : LIS_rc, LIS_localPet,&
         LIS_ews_ind, LIS_ewe_ind,&
         LIS_nss_ind, LIS_nse_ind, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
         LIS_nss_halo_ind, LIS_nse_halo_ind
    use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
         LIS_releaseUnitNumber, LIS_endrun, LIS_verify
    use LIS_fileIOMod

    implicit none
```

ARGUMENTS:

```
    integer, intent(in) :: n
    character(len=*)    :: vname
    real                :: pvalue(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine reads the CLSM model parameter
The arguments are:

n index of n

locallc landlc for the region of interest

72.5.11 clsmf25_read_land3d_parameter (Source File: clsmf25_setup.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine clsmf25_read_land3d_parameter(n,vname,pvalue)
```

USES:

```
#if(defined USE_NETCDF3 || defined USE_NETCDF4)
    use netcdf
#endif
    use LIS_coreMod,        only : LIS_rc, LIS_localPet,&
         LIS_ews_ind, LIS_ewe_ind,&
         LIS_nss_ind, LIS_nse_ind, LIS_ews_halo_ind,LIS_ewe_halo_ind, &
         LIS_nss_halo_ind, LIS_nse_halo_ind
    use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
         LIS_releaseUnitNumber, LIS_endrun, LIS_verify
    use LIS_fileIOMod

    implicit none
```

ARGUMENTS:

```
    integer, intent(in) :: n
    character(len=*)    :: vname
    real                :: pvalue(LIS_rc%lnc(n),LIS_rc%lnr(n),12)
```

DESCRIPTION:

This subroutine reads the CLSM model parameter
The arguments are:

n index of n

locallc landlc for the region of interest

72.5.12 clsmf25_turb (Source File: clsmf25_turb.F90)

REVISION HISTORY:

```
23 Nov 2012: David Mocko, Added Catchment Fortuna-2.5
```

INTERFACE:

```
subroutine clsmf25_turb(N_cat, met_force, cat_progn, &
     cat_diagn, zol, dze, lai,  &
     RA1, ETURB1, DEDQA1, DEDTC1, HSTURB1, DHSDQA1, DHSDTC1,                &
     RA2, ETURB2, DEDQA2, DEDTC2, HSTURB2, DHSDQA2, DHSDTC2,                &
     RA4, ETURB4, DEDQA4, DEDTC4, HSTURB4, DHSDQA4, DHSDTC4,                &
     RAS, ETURBS, DEDQAS, DEDTCS, HSTURBS, DHSDQAS, DHSDTCS,                &
     sfc_turb_scheme )


    ! qliu+reichle: 14 Aug 2008 - land interface to subroutine louissurface()
    ! qliu+reichle: 30 Oct 2008 - force derivatives to be non-negative in case
    !                             extra derivatives are included
    ! reichle     : 29 Nov 2010 - added Helfand Monin-Obukhov scheme
    ! reichle     :  2 Feb 2011 - added T2m, Q2m diagnostics
```

72.6 Fortran: Module Interface clsmf25_types (Source File: clsmf25_types.F90)

definition of types and associated operators for Catchment Model
IMPORTANT: When adding a field to any of the derived types, must also update the associated assignment and operator definitions. THERE IS NO WARNING/ERROR IF OPERATOR IS NOT DEFINED FOR ALL FIELDS!

REVISION HISTORY:

```
21 May 2003, Rolf Reichle: Initial Specification
25 Jan 2005, Rolf Reichle: Added cat_force_type
23 Nov 2012: David Mocko, Added greenness/lai from parameter file
```

72.6.1 clsmf25_writerst (Source File: clsmf25_writerst.F90)

REVISION HISTORY:

```
18 Dec 2005: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine clsmf25_writerst(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_logMod
use LIS_timeMgrMod
use LIS_fileIOMod, only : LIS_create_output_directory, &
                          LIS_create_restart_filename
```

```
  use clsmf25_lsmMod
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
```

ARGUMENTS:

```
  implicit none
  integer, intent(in) :: n
```

DESCRIPTION:

This program writes restart files for catchment. This includes all relevant water/energy storage and tile information
The routines invoked are:

LIS_create_output_directory (3.14.6)
creates a timestamped directory for the restart files

LIS_create_restart_filename (3.14.2)
generates a timestamped restart filename

cat_dump_restart (72.6.2)
writes the catchment variables into the restart file

72.6.2 clsmf25_dump_restart (Source File: clsmf25_writerst.F90)

REVISION HISTORY:

```
24 Aug 2004: James Geiger, Initial Specification
```

INTERFACE:

```
subroutine clsmf25_dump_restart(n, ftn, wformat)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_historyMod
use clsmf25_constants
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: ftn
integer, intent(in) :: n
character(len=*), intent(in) :: wformat
```

DESCRIPTION:

This routine gathers the necessary restart variables and performs the actual write statements to create the restart files.
The arguments are:

n index of the nest

ftn unit number for the restart file

wformat restart file format (binary/netcdf)

The routines invoked are:

LIS_writeGlobalHeader_restart (3.17.22)
writes the global header information

LIS_writeHeader_restart (3.17.23)
writes the header information for a variable

LIS_closeHeader_restart (3.17.24)
close the header

LIS_writevar_restart (3.17.3)
writes a variable to the restart file

72.7 Fortran: Module Interface clsmf25_dasnow_Mod (Source File: clsmf25_dasnow_Mod.F9

REVISION HISTORY:

USES:

```
implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: clsmf25_dasnow_init
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
```

72.7.1 clsmf25_dasnow_init (Source File: clsmf25_dasnow_Mod.F90)

INTERFACE:

```
subroutine clsmf25_dasnow_init()
```

USES:

DESCRIPTION:

72.7.2 clsmf25_descale_snow (Source File: clsmf25_descale_snow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine clsmf25_descale_snow(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

72.7.3 clsmf25_getsnowvars (Source File: clsmf25_getsnowvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine clsmf25_getsnowvars(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

72.7.4 clsmf25_getsnwdpred (Source File: clsmf25_getsnwdpred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine clsmf25_getsnwdpred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_surface
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real                   :: obs_pred(LIS_rc%ngrid(n),LIS_rc%nensem(n))
```

72.7.5 clsmf25_qc_snowobs (Source File: clsmf25_qc_snowobs.F90)

REVISION HISTORY:

```
25Feb2008: Sujay Kumar: Initial Specification
```

INTERFACE:

```
subroutine clsmf25_qc_snowobs(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_logMod,  only : LIS_verify
use LIS_constantsMod, only : LIS_CONST_TKFRZ
use LIS_vegDataMod, only : LIS_lai,LIS_gfrac
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

This subroutine performs any model-based QC of the observation prior to data assimilation. Here the snow observations are flagged when LSM indicates that (1) rain is falling (2) ground is fully or partially covered with snow.
The arguments are:

n index of the nest

OBS_State ESMF state container for observations

72.7.6 clsmf25_qcsnow (Source File: clsmf25_qcsnow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine clsmf25_qcsnow(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use clsmf25_lsmMod
use LIS_logMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

72.7.7 clsmf25_scale_snow (Source File: clsmf25_scale_snow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine clsmf25_scale_snow(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

72.7.8 clsmf25_setsnowvars (Source File: clsmf25_setsnowvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine clsmf25_setsnowvars(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod
use clsmf25_lsmMod
use clsmf25_constants
use LIS_logMod, only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

This routine assigns the snow progognostic variables to noah's model space.

72.7.9 clsmf25_updatesnowvars (Source File: clsmf25_updatesnowvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine clsmf25_updatesnowvars(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use clsmf25_lsmMod
use LIS_logMod,   only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

LSM_Incr_State ESMF State container for LSM state increments

72.7.10 clsmf25_wrtswe (Source File: clsmf25_wrtswe.F90)

REVISION HISTORY:

```
13Nov2007: Sujay Kumar; Initial Specification
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine clsmf25_wrtswe(ftn, n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: ftn
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the tskin related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

72.8 Fortran: Module Interface clsmf25_dasoilm_Mod (Source File: clsmf25_dasoilm_Mod.F9

REVISION HISTORY:

USES:

```
implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: clsmf25_dasoilm_init
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
```

72.8.1 clsmf25_dasoilm_init (Source File: clsmf25_dasoilm_Mod.F90)

INTERFACE:

```
subroutine clsmf25_dasoilm_init()
```

USES:

DESCRIPTION:

72.8.2 clsmf25_descale_soilm (Source File: clsmf25_descale_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clsmf25_descale_soilm(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Descales soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

72.8.3 clsmf25_getsmpred (Source File: clsmf25_getsmpred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clsmf25_getsmpred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_surface
use clsmf25_lsmMod
use clsmf25_model
```

72.8.4 clsmf25_getsoilm (Source File: clsmf25_getsoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clsmf25_getsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

72.8.5 clsmf25_qc_soilmobs (Source File: clsmf25_qc_soilmobs.F90)

REVISION HISTORY:

```
25Feb2008: Sujay Kumar: Initial Specification
```

INTERFACE:

```
subroutine clsmf25_qc_soilmobs(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_logMod
use LIS_constantsMod, only : LIS_CONST_TKFRZ
use LIS_vegDataMod, only : LIS_lai,LIS_gfrac
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

This subroutine performs any model-based QC of the observation prior to data assimilation. Here the soil moisture observations are flagged when LSM indicates that (1) rain is falling (2) soil is frozen or (3) ground is fully or partially covered with snow.
The arguments are:

n index of the nest

OBS_State ESMF state container for observations

72.8.6 clsmf25_qcsoilm (Source File: clsmf25_qcsoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clsmf25_qcsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

72.8.7 clsmf25_scale_soilm (Source File: clsmf25_scale_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clsmf25_scale_soilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Scales soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

72.8.8 clsmf25_setsoilm (Source File: clsmf25_setsoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clsmf25_setsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_logMod,  only  : LIS_verify, LIS_logunit
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

This routine assigns the soil moisture prognostic variables to noah's model space.

72.8.9 clsmf25_updatesoilm (Source File: clsmf25_updatesoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clsmf25_updatesoilm(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

This routine assigns the soil moisture prognostic variables to noah's model space.

72.8.10 clsmf25_write_soilm (Source File: clsmf25_write_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine clsmf25_write_soilm(ftn,n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use clsmf25_lsmMod
use LIS_historyMod, only : LIS_writevar_restart
implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: ftn
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

72.9 Fortran: Module Interface clsmf25_datws_Mod (Source File: clsmf25_datws_Mod.F90)

REVISION HISTORY:

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: clsmf25_datws_init
```

PUBLIC TYPES:

72.9.1 clsmf25_datws_init (Source File: clsmf25_datws_Mod.F90)

INTERFACE:

```
subroutine clsmf25_datws_init()
```

USES:

DESCRIPTION:

72.9.2 clsmf25_descale_tws (Source File: clsmf25_descale_tws.F90)

REVISION HISTORY:

```
12 Feb 2006; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine clsmf25_descale_tws(n,LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only  : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(IN)       :: n
type(ESMF_State)          :: LSM_State
type(ESMF_State)          :: LSM_Incr_State
```

DESCRIPTION:

Routine to retrieve soil moisture prognostic variables in catchment.
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

72.9.3 clsmf25_gettws (Source File: clsmf25_gettws.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
28Sep2011: Ben Zaitchik: Applied to GRACE
```

INTERFACE:

```
subroutine clsmf25_gettws(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_logMod,  only  : LIS_verify
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the tws related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

72.9.4 clsmf25_gettwspred (Source File: clsmf25_gettwspred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
28Sep2011: Ben Zaitchik: Applied to GRACE
```

INTERFACE:

```
subroutine clsmf25_gettwspred(n,obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_surface
use clsmf25_lsmMod
use LIS_logMod, only : LIS_logunit, LIS_verify
use clsmf25_tws_DAlogMod, only : CLSMpred_struc
```

72.9.5 clsmf25_qc_twsobs (Source File: clsmf25_qc_twsobs.F90)

REVISION HISTORY:

```
25Feb2008: Sujay Kumar: Initial Specification
29Sep2011: Ben Zaitchik: Applied to GRACE
```

INTERFACE:

```
subroutine clsmf25_qc_twsobs(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_logMod,  only : LIS_verify
use LIS_constantsMod, only : LIS_CONST_TKFRZ
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

This subroutine performs any model-based QC of the observation prior to data assimilation. The arguments are:

n index of the nest

OBS_State ESMF state container for observations

72.9.6 clsmf25_qctws (Source File: clsmf25_qctws.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
29Sep2011: Ben Zaitchik: Applied to GRACE
```

INTERFACE:

```
subroutine clsmf25_qctws(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the TWS related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

72.9.7 clsmf25_scale_tws (Source File: clsmf25_scale_tws.F90)

REVISION HISTORY:

```
12 Feb 2006; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine clsmf25_scale_tws(n,LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only  : LIS_rc
use LIS_logMod,  only  : LIS_verify
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(IN)       :: n
type(ESMF_State)          :: LSM_State
```

DESCRIPTION:

Routine to retrieve soil moisture prognostic variables in catchment.
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

72.9.8 clsmf25_settws (Source File: clsmf25_settws.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
29Sep2011: Ben Zaitchik: Applied to GRACE
```

INTERFACE:

```
subroutine clsmf25_settws(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_logMod,  only  : LIS_logunit,LIS_verify
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

This routine assigns the TWS prognostic variables to CLSM's model space.

72.9.9 clsmf25_update_tws (Source File: clsmf25_update_tws.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
29Sep2011: Ben Zaitchik: Applied to GRACE
```

INTERFACE:

```
subroutine clsmf25_update_tws(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_logMod,  only  : LIS_verify, LIS_logunit
use clsmf25_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

This routine assigns the soil moisture prognostic variables to noah's model space.

72.9.10 clsmf25_write_tws (Source File: clsmf25_write_tws.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
29Sep2011: Ben Zaitchik: Applied to GRACE
```

INTERFACE:

```
subroutine clsmf25_write_tws(ftn,n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use clsmf25_lsmMod
use LIS_historyMod, only : LIS_writevar_restart
implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: ftn
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the TWS related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

72.9.11 clsmf25_getirrigationstates (Source File: clsmf25_getirrigationstates.F90)

INTERFACE:

```
subroutine clsmf25_getirrigationstates(n,irrigState)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_logMod
use clsmf25_lsmMod
use LIS_vegDataMod
```

DESCRIPTION:

Calculate water requirement and apply the amount to precipitation.

Irrigate when root zone soil moisture falls below 50 the field capacity (reference soil moiture) at 6 am LST. The root zone is actual maximum root depth rather than NOAH root zone. Method of irrigation is by precipitation between 6-10 am LST.
Irrigation amount is scaled to grid total crop fraction when intensity is less than the fraction. Irrigation is expanded to non-crop, non-forest, non-baresoil/urban tiles if intensity exceeds grid total crop fraction. In latter case, scaled irrigation is applied to grassland first, then further applied over the rest of tiles equally if the intensity exceeds grassland fraction as well.
Optionally efficiency correction is applied to account for field loss.
Optionally outputs amount of water put into the system to a text file.
This version includes modifications to irr4 as follows: 1) Use location specific growing season threshold (402) Allow irrigation in non-crop/non-forest tiles when irrigation intensity exceeds total crop fraction
REVISION HISTORY:
Aug 2008: Hiroko Kato; Initial code for Noah LSM Feb 2014: Sujay Kumar; Implemented in LIS based on the work of John Bolten and student.

72.9.12 clsmf25_getrunoffs (Source File: clsmf25_getrunoffs.F90)

REVISION HISTORY:

```
6 May 2011: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine clsmf25_getrunoffs(n)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_routingMod, only : LIS_runoff_state
use LIS_logMod,     only : LIS_verify
use LIS_constantsMod, only : LIS_CONST_RHOFW
use LIS_historyMod
use clsmf25_lsmMod, only : clsmf25_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

72.9.13 clsmf25_getrunoffs_mm (Source File: clsmf25_getrunoffs_mm.F90)

REVISION HISTORY:

```
6 May 2011: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine clsmf25_getrunoffs_mm(n)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_routingMod, only : LIS_runoff_state
use LIS_logMod,     only : LIS_verify
use LIS_historyMod
use clsmf25_lsmMod, only : clsmf25_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

---

73 WRSI Model

This section describes the interface implementations for the version 2.0 of the Water Requirement Satisfaction Index (WRSI) model.

73.0.14 geowrsi2_dynsetup (Source File: geowrsi2_dynsetup.F90)

INTERFACE:

```
subroutine geowrsi2_dynsetup(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_historyMod, only : LIS_readvar_restart
use LIS_logMod,     only : LIS_logunit, LIS_endrun, LIS_verify
use geowrsi2_lsmMod
use geowrsi2_module
use geowrsi2_physics_module, only : offsetTStepByNumTSteps, stepnFromTStep, &
         gTStepsBeforeSeasonStarts, gTStepsAfterSeasonEnds, gTimeStepsPerYear, &
         DifferenceOf2TSteps

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine sets up and ingests time-dependent (dynamic) variables for GeoWRSI model, including yearly SOS/SOSa files.
The arguments are:

n index of the nest

73.0.15 geowrsi2_f2t (Source File: geowrsi2_f2t.F90)

REVISION HISTORY:

```
31Jul2011: Brad Wind;   Initial Definition
28Feb2013: KR Arsenault/JV Geiger; Removed original FEWSNET forcing calls
25Oct2013: KR Arsenault;  Added GeoWRSI2.0 model to LIS-7
```

INTERFACE:

```
subroutine geowrsi2_f2t(n)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc, LIS_surface
use LIS_logMod,         only : LIS_verify
use LIS_metforcingMod,  only : LIS_FORC_State
use LIS_FORC_AttributesMod
use geowrsi2_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the WRSI model tiles
The arguments are:

n index of the nest

73.0.16 geowrsi2_finalize (Source File: geowrsi2_finalize.F90)

INTERFACE:

```
subroutine geowrsi2_finalize()
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use geowrsi2_lsmMod, only : geowrsi2_struc
```

ARGUMENTS:

DESCRIPTION:

This routine cleans up the allocated memory structures in the geowrsi2 (forcing-only option)
The arguments are:

n index of the nest

73.1 Fortran: Module Interface geowrsi2_lsmMod (Source File: geowrsi2_lsmMod.F90)

Module for 1-D land model driver variable initialization

count variable to keep track of the number of timesteps before an output

outInterval output writing interval

geowrsi2 WRSI LSM specific variables

REVISION HISTORY:

```
31 Jul 2011: Brad Wind; Initial Definition
10 Jul 2013: KR Arsenault, JV Geiger;  Additional updates to code
25 Oct 2013: KR Arsenault; Added GeoWRSI2.0 model to LIS-7
```

USES:

```
use ESMF
use LIS_timeMgrMod
use geowrsi2_module
use geowrsi2_physics_module, only : gCalcSOSmode
use fbil_module, only : charN

implicit none

PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
public :: geowrsi2_lsm_ini
public :: outVar
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
public :: geowrsi2_struc
public :: num_growing_seasons
public :: geowrsi2_lsmRunMode
public :: geowrsi2_CalcSOSlsmRunMode
public :: geowrsi2_udef
```

73.1.1 geowrsi2_lsm_ini (Source File: geowrsi2_lsmMod.F90)

INTERFACE:

```
subroutine geowrsi2_lsm_ini()
```

USES:

```
use ESMF
use LIS_surfaceModelDataMod, only : LIS_sfmodel_struc
use LIS_timeMgrMod
use LIS_coreMod,  only : LIS_rc
use LIS_logMod,   only : LIS_logunit, LIS_verify, LIS_endrun
```

DESCRIPTION:

73.1.2 geowrsi2_main (Source File: geowrsi2_main.F90)

REVISION HISTORY:

```
31 Jul 2011: Brad Wind; Initial Definition
19 Dec 2012: KR Arsenault; Removed test code
19 Feb 2013: KR Arsenault; Removed param array for multi season component
25 Oct 2013: KR Arsenault; Added GeoWRSI2.0 model to LIS-7
```

INTERFACE:

```
subroutine geowrsi2_main(n)
```

USES:

```
use LIS_coreMod,   only : LIS_rc, LIS_domain
use LIS_logMod,    only : LIS_logunit, LIS_endrun
use LIS_timeMgrMod
use LIS_histDataMod
use geowrsi2_module
use geowrsi2_lsmMod, only : geowrsi2_struc, geowrsi2_CalcSOSlsmRunMode, &
                            num_growing_seasons, geowrsi2_udef, outVar

use geowrsi2_physics_module, only :  &
    offsetTStepByNumTSteps, CalcWRSI, CalcWRSIAnomalies,      &
    CalcSOS, resetWBdataFor1point, gTimeStepsPerYear,         &
    gCurrentTStep, gCurrentYear, gInitialTStep, gInitialYear, &
    gPW_EnablePermanentWilting, gPermanentWiltingIsFromSWI,   &
    gPermanentWiltingIsFromWRSI, gPermanentWiltingThresholdValue, &
    gCrop_c1, gCrop_c2, gCrop_F1, gCrop_F2, gCrop_F3, gCrop_K1,   &
    gCrop_K2, gCrop_K3, gCrop_kp, gCrop_rini, gCrop_r, gPreSeason_Kc, &
    gLastCurrentTStep, gLastCurrentYear, &
    gSOScalcUseRainThreshold, gSOScalcTStep1Threshold,     &
    gSOScalcTStep2Plus3Threshold, gSOScalcUseSWIThreshold, &
    gSOScalcSoilMoistureThreshold, gSOScalcDesiredPlantingTStep,  &
    gSOScalcUseNearestToDesiredPracticalPlantingTStep,      &
    gSOScalcUseWRSIThresholdInsteadofSWIThresholdToRestart, &
    gSOScalcRestartThresholdWithWRSI, gSOScalcRestartThresholdWithSWI,&
    gSOScalcUseWRSIThresholdInsteadofSWIThresholdToRestart, &
    gSOScalcRestartCropWhenFailed, &
    gSOScalcCropCanStillRestartWithinPctLGPThreshold,   &
    gFinalYear, gFinalTStep, gSOScalcIgnoreClimatology, &
    gSOScalcMaxTStepsLate, gSOScalcMaxTStepsEarly,      &
    gSOScalcExcludeIncompleteAreasFromSOS,              &
```

```
    gSOScalcAcceptablePercentOfTotalSeasonalGrowth,     &
    gMASK_INC, gWRSI_NA, gKF2_SEASONTOSTARTDANDORACCUM, &
    DifferenceOf2TSteps

use fbil_module, only : CInt2, CInt4

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Calls the run routines for the forcing-only option (geowrsi2)
The arguments are:

n index of the nest

73.2 Fortran: Module Interface geowrsi2_module.F90 (Source File: geowrsi2_module.F90)

REVISION HISTORY:

```
31 Jul 2011: Brad Wind; Initial Definition
11 Jan 2013: KR Arsenault; Cleaned up comments/documentation
13 Feb 2013: KR Arsenault; Added new single-input parameter declarations
                           (to replace the old "e1v" routines)
25 Oct 2013: KR Arsenault;  Added GeoWRSI2.0 model to LIS-7
```

DESCRIPTION:

WRSI LSM option (WRSI) variables

USES:

```
use geowrsi2_physics_module, only : gSOScalcNtStepsOfForcingUsed
```

73.3 Fortran: Module Interface geowrsi2_physics_module (Source File: geowrsi2_physics_mo

REVISION HISTORY:

```
31 Jul 2011: Brad Wind; Initial Definition
20 Dec 2012: KR Arsenault; Cleaned-up code
25 Oct 2013: KR Arsenault; Added GeoWRSI2.0 model to LIS-7
```

USES:

```
use geowrsi2_arraymgmt_module
use fbil_module
```

73.3.1 geowrsi2_readInputSettings (Source File: geowrsi2_readInputSettings.F90)

REVISION HISTORY:

```
23 Jan 2013: KR Arsenault;  Added new user-input parameter file
25 Oct 2013: KR Arsenault;  Added GeoWRSI2.0 model to LIS-7
```

INTERFACE:

```
subroutine geowrsi2_readInputSettings()
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc, LIS_domain, LIS_localPet
use LIS_logMod,    only : LIS_logunit, LIS_verify, LIS_endrun
use LIS_fileIOMod, only : LIS_read_param
use geowrsi2_module
use geowrsi2_lsmMod
use geowrsi2_physics_module, only : offsetTStepByNumTSteps, stepnFromTStep, &
         gTStepsBeforeSeasonStarts, gTStepsAfterSeasonEnds, gTimeStepsPerYear, &
         DifferenceOf2TSteps
use geowrsi2_arraymgmt_module, only: nullify_ptr, alloc_arr, dealloc_arr
use fbil_module
```

DESCRIPTION:

This subroutine is designed to read in GeoWRSI-based user settings input file.
The GeoWRSI-based user settings file can be generated by initializing GeoWRSI, or set up independently by the user based on the input file example provided in the WRSI LSM test case.

73.3.2 geowrsi2_readSOS_Bilfile (Source File: geowrsi2_readSOS_Bilfile.F90)

REVISION HISTORY:

```
25 Apr 2014: KR Arsenault;  Implemented separate SOS-file reader
```

INTERFACE:

```
subroutine geowrsi2_readSOS_Bilfile( n, numpts, year, SOS, SOSa, SOS_offset )
```

USES:

```
use ESMF
use LIS_coreMod,  only : LIS_rc, LIS_domain, &
                         LIS_masterproc, LIS_npes
use LIS_logMod,   only : LIS_logunit, LIS_endrun
use geowrsi2_lsmMod, only: geowrsi2_struc
use geowrsi2_physics_module,   only : offsetTStepByNumTSteps
use geowrsi2_arraymgmt_module, only : nullify_ptr, alloc_arr, dealloc_arr
use fbil_module

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n     ! nest
integer, intent(in)      :: numpts
integer, intent(in)      :: year
integer*4, intent(inout) :: SOS(numpts)
integer*4, intent(inout) :: SOSa(numpts)
integer*4, intent(in)    :: SOS_offset
```

DESCRIPTION:

This routine reads the SOS and SOS anomaly fields from BIL formatted files.
The arguments are:

n index of the nest

73.3.3 geowrsi2_readcrd (Source File: geowrsi2_readcrd.F90)

REVISION HISTORY:

```
31 Jul 2011: Brad Wind; Initial Definition
14 Jan 2013: KR Arsenault; Updated config file input options
25 Oct 2013: KR Arsenault;  Added GeoWRSI2.0 model to LIS-7
```

INTERFACE:

```
subroutine geowrsi2_readcrd()
```

USES:

```
use ESMF
use LIS_coreMod,     only : LIS_rc, LIS_config
use LIS_logMod,      only : LIS_logunit, LIS_verify
use LIS_timeMgrMod,  only : LIS_parseTimeString
use geowrsi2_lsmMod, only : geowrsi2_struc, num_growing_seasons, &
                            geowrsi2_lsmRunMode
```

DESCRIPTION:

This routine reads the options specific to WRSI LSM options from the LIS configuration file.

73.3.4 geowrsi2_readrst (Source File: geowrsi2_readrst.F90)

REVISION HISTORY:

```
30 Jan 2014: KR Arsenault; Initial code set-up
```

INTERFACE:

```
subroutine geowrsi2_readrst()
```

USES:

```
  use LIS_coreMod,    only : LIS_rc, LIS_masterproc
  use LIS_historyMod, only : LIS_readvar_restart
  use LIS_logMod,     only : LIS_logunit, LIS_endrun, LIS_verify,&
          LIS_getNextUnitNumber, LIS_releaseUnitNumber
  use geowrsi2_lsmMod

#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
```

ARGUMENTS:

DESCRIPTION:

This program reads restart and start-of-season (SOS) files for GeoWRSI 2.0, including important tile information.
The following is the list of variables specified in the GeoWRSI2.0 restart file:

```
nc,nr,ntiles - grid and tile space dimensions
sos          - GeoWRSI2 SOS
sosanom      - GeoWRSI2 SOSanom
```

The routines invoked are:

LIS_readvar_restart (3.17.6)
Reads a variable from the restart file

73.3.5 geowrsi2_set_gcoords (Source File: geowrsi2_set_gcoords.F90)

REVISION HISTORY:

```
28 Jun 2013: James Geiger; Initial implementation
25 Oct 2013: KR Arsenault;  Added GeoWRSI2.0 model to LIS-7
```

INTERFACE:

```
subroutine geowrsi2_set_gcoords(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use fbil_module

implicit none
```

ARGUMENTS:

```
integer :: n
```

DESCRIPTION:

This routine sets the gCoords data structure. This data structure contains the running domain specifications that are used by the BIL routines.
Note that the BIL library expects grid-cells to be defined by their upper left corner. LIS defines grid-cells on their centers. When setting gCoords, nudge the values to conform to the BIL library.
The arguments are:

n index of the nest

73.3.6 geowrsi2_setup (Source File: geowrsi2_setup.F90)

REVISION HISTORY:

```
31 Jul 2011: Brad Wind; Initial Definition
21 Dec 2012: KR Arsenault; Cleaned up code and comments
25 Oct 2013: KR Arsenault; Added GeoWRSI2.0 model to LIS-7
```

INTERFACE:

```
subroutine geowrsi2_setup()
```

USES:

```
use LIS_coreMod,   only : LIS_rc
use LIS_logMod,    only : LIS_logunit, LIS_endrun

use geowrsi2_module
use geowrsi2_lsmMod
use geowrsi2_physics_module, only : offsetTStepByNumTSteps, &
         gSOScalcNtStepsOfForcingUsed, &
         gPPTDATALIMITEDORMISSINGTHRESHOLD, gTimeStepsPerYear, &
         gWILTING1_DEFAULT, gWILTING2_DEFAULT, gMASK_EXC, &
         gSOS_INITIAL, gSOS_NOSTART, gCurrentYear, gCurrentTStep, &
         DifferenceOf2TSteps

use fbil_module
```

DESCRIPTION:

This routine is the entry point to complete the set up parameters required for the FEWSNET WRSI LSM. These include everything from crop phenology water requirement parameterizations, to soil water holding capacities, to start and end of crop growing season, to analyst options and state variables initializations in WRSI.

73.3.7 geowrsi2_writeSOS_Bilfile (Source File: geowrsi2_writeSOS_Bilfile.F90)

REVISION HISTORY:

```
31 Jul 2011: Brad Wind; Initial setup
15 Jan 2013: KR Arsenault;  Implemented LIS-SOS BIL write routine into LIS
25 Oct 2013: KR Arsenault;  Added GeoWRSI2.0 model to LIS-7
```

INTERFACE:

```
subroutine geowrsi2_writeSOS_Bilfile(n)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_domain, &
                           LIS_masterproc, LIS_npes
use LIS_logMod,     only : LIS_logunit, LIS_endrun
use LIS_fileIOMod,  only : LIS_create_output_directory
use LIS_historyMod, only : LIS_gather_gridded_output
use geowrsi2_module
use geowrsi2_lsmMod
use geowrsi2_physics_module, only : gMASK_EXC, &
          gWILTING1_DEFAULT, gWILTING2_DEFAULT,&
          gSOS_INITIAL, gSOS_NOSTART, gSOSCLIM_NA
use fbil_module

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine writes the SOS and SOS anomaly fields into BIL formatted files.
The arguments are:

n index of the nest

73.3.8 geowrsi2_writerst (Source File: geowrsi2_writerst.F90)

INTERFACE:

```
subroutine geowrsi2_writerst(n)
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use LIS_logMod,      only : LIS_logunit
use geowrsi2_lsmMod, only : geowrsi2_CalcSOSlsmRunMode, geowrsi2_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Forcing-only (geowrsi2) option for calling the write restart routines.
This routine also writes the SOS/SOSa files from the CalcSOS run mode.
The arguments are:

n index of the nest

---

74 HySSIB Land Surface Model

This section describes the interface implementations for the HySSIB land surface model.

74.0.9 hyssib_coldstart (Source File: hyssib_coldstart.F90)

REVISION HISTORY:

```
15 Dec 2003: Luis-Gustavo Goncalves, Initial version
29 Apr 2004: David Mocko, Conversion from NOAH to HY-SSiB
25 Aug 2007: Chuck Alonge, Updates for LIS 5.0 compliance
```

INTERFACE:

```
subroutine hyssib_coldstart()
```

USES:

```
use LIS_coreMod, only: LIS_rc
use LIS_timeMgrMod, only : LIS_date2time
use LIS_logMod, only : LIS_logunit
use hyssib_lsmMod    ! HY-SSiB tile variables
```

DESCRIPTION:

This routine initializes the hyssib state variables with some predefined values uniformly for the entire domain. These initial values will be overwritten by the values read from the supplied HYSSIB model restart file.

74.0.10 hyssib_dynsetup (Source File: hyssib_dynsetup.F90)

REVISION HISTORY:

```
15 Apr 2002: Sujay Kumar, Initial Specification
23 Apr 2004: David Mocko, Conversion from NOAH to HY-SSiB
25 Aug 2007: Chuck Alonge, Updates for LIS 5.0 compliance
```

INTERFACE:

```
subroutine hyssib_dynsetup(n)
```

DESCRIPTION:

This routine sets up the time-dependent variables in Hyssib. Currently this routine is a placeholder

---

74.0.11 hyssib_f2t (Source File: hyssib_f2t.F90)

REVISION HISTORY:

```
21 Apr 2004: David Mocko, Conversion from NOAH to HY-SSiB
25 Aug 2007: Chuck Alonge, Updates for LIS 5.0 Compliance
27 Oct 2010: David Mocko, changes for HY-SSiB in LIS6.1
```

INTERFACE:

```
subroutine hyssib_f2t(n)
```

USES:

```
use ESMF
use LIS_coreMod       , only : LIS_rc, LIS_surface
use LIS_FORC_AttributesMod
use LIS_metforcingMod, only : LIS_FORC_State
use LIS_logMod,         only : LIS_verify
use hyssib_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the Hyssib model tiles.
The arguments are:

n index of the nest

74.0.12 hyssib_finalize (Source File: hyssib_finalize.F90)

REVISION HISTORY:

```
28 Apr 2002: Sujay Kumar; Initial code
05 Sep 2007: Sujay Kumar; Addition of Hyssib to LIS 5.0
```

INTERFACE:

```
subroutine hyssib_finalize()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use hyssib_lsmMod
use hyssibveg_module
use hyssibalb_module
```

DESCRIPTION:

This routine cleans up the allocated memory structures in Hyssib

---

74.0.13 hyssib_gfrac.F90 (Source File: hyssib_gfrac.F90)

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault, Added SSiB LSM to LDAS, initial code
15 Feb 2004: David Mocko, Conversion from SSiB to HY-SSiB
28 Oct 2007: Chuck Alonge, Updates for LIS 5.0
```

INTERFACE:

```
subroutine hyssib_gfrac()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use hyssib_lsmMod
use hyssibveg_module
```

DESCRIPTION:

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

---

74.1 Fortran: Module Interface hyssib_lsmMod (Source File: hyssib_lsmMod.F90)

This module provides the definition of derived data type used to control the operation of Hyssib LSM. It also provides the entry method for the initialization of Hyssib-specific variables. The derived data type `hyssib_struc` includes the variables that specify the runtime options and other control variables as described below:

rfile name of the hyssib restart file

vfile name of the static vegetation parameter table

topostdfile name of the file for std. dev. of topography

count variable to keep track of the number of timesteps before an output

hyssibopen variable to keep track of opened files

numout number of output times

nvegp number of static vegetation parameters in the table

nvegip number of vegetation based albedo parameters in the table

npr number of prognostic state variables used in data assimilation

zh reference height of T and q forcing

zm reference height of u and v forcing

initsm initial soil moisture for a cold start run

initTemp initial soil temperature for a cold start run

rstInterval restart writing interval

hyssib hyssib LSM tile specific variables

REVISION HISTORY:

```
   Apr 2003: Sujay Kumar, Initial Code
21 Feb 2004: David Mocko, Conversion from NOAH to HY-SSiB
25 Aug 2007: Chuck Alonge, Updates for LIS 5.0 Compliance
27 Oct 2010: David Mocko, changes for HY-SSiB in LIS6.1
```

USES:

```
use hyssib_module

 implicit none

 PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
 public :: hyssib_lsm_ini
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
 public :: hyssib_struc
```

74.1.1 hyssib_lsm_ini (Source File: hyssib_lsmMod.F90)

INTERFACE:

```
subroutine hyssib_lsm_ini()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod,   only : LIS_clock, LIS_calendar, &
     LIS_update_timestep, LIS_registerAlarm
use LIS_logMod,       only : LIS_verify
```

DESCRIPTION:

Reads in runtime HY-SSiB parameters, allocates memory for variables
The routines invoked are:

hyssib_readcrd (74.2.1)
reads the runtime options for Hyssib LSM

74.1.2 hyssib_main (Source File: hyssib_main.F90)

REVISION HISTORY:

```
21 Apr 2004: David Mocko, Initial Code
30 Apr 2004: David Mocko, Revisions to fix MPI version
25 Nov 2008: Chuck Alonge, Updates for LIS 5.0
27 Oct 2010: David Mocko, changes for HY-SSiB in LIS6.1
```

INTERFACE:

```
subroutine hyssib_main(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_surface,LIS_domain
use LIS_timeMgrMod, only : LIS_isAlarmRinging
use LIS_albedoMod, only : LIS_alb
use LIS_logMod,    only : LIS_logunit, LIS_endrun
use LIS_histDataMod
use hyssib_lsmMod  ! HY-SSiB tile variables
use hyssibalb_module

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

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.
The arguments are:

n index of the nest

74.1.3 hyssib_mapvegc.F90 (Source File: hyssib_mapvegc.F90)

REVISION HISTORY:

```
28 Apr 2002: K Arsenault, Added SSIB LSM to LDAS
   Feb 2004: David Mocko, Conversion from SSiB to HY-SSiB
25 Nov 2007: Chuck Alonge, Updates for LIS 5.0
```

INTERFACE:

```
SUBROUTINE HYSSIB_MAPVEGC(VEGT)

    implicit none
```

ARGUMENTS:

```
    INTEGER, INTENT(INOUT) :: VEGT
```

DESCRIPTION:

This subroutine converts the UMD classes to the SIB classes used by SSIB LSM (v 2.5). (Originally from Dag Lohmann at NCEP)
The arguments are:

vegt UMD Vegetation class number to be converted to SIB class

---

74.2 Fortran: Module Interface hyssib_module (Source File: hyssib_module.F90)

The code in this file provides a description of the data structure containing the hyssib 1-d variables. The variables specified in the data structure include:

ts time step in seconds (integer)

maxt maximum number of tiles per grid

vegt HYSSIB vegetation class index value

count number of timesteps between outputs

snowtcount number of timesteps with snow (for averaging)

albedocount number of timesteps for albedo averaging

sliqfraccount number of timesteps for snow cover averaging

vegp static monthly vegetation parameters

vegip monthly albedo vegetation lookup table values

tempbot bottom temperature value

tempbot1 month 1 bottom temp value

tempbot2 month 2 bottom temp value

tempstd standard deviation of topography (m)

sgfg flag which indicates if snow model is active

sdens snow density

tc canopy temperature

tg ground temperature

tsn temperature of the snow pack

www soil moisture (volumetric) for each soil layer (3)

capac canopy/ground water content (1/2)

snow canopy/ground snow water equivalent

swnet net shortwave radiation flux

lwnet net longwave radiation flux

qle latent heat flux

qh sensible heat flux

qg ground heat flux

qf energy of fusion

qv energy of sublimation

qtau momentum flux from/to surface

qa advective energy

delsurfheat change in surface heat storage

delcoldcont change in snow cold content

snowf snowfall

rainf rainfall

evap total evaporation

qs surface runoff

qrec recharge

qsb subsurface runoff

qsm snowmelt

qfz re-freezing of water in the snow

qst snow throughfall

delsoilmoist change in soil moisture

delswe change in snow water equivalent

delintercept change in interception storage

snowt snow surface temperature (K)

vegtc vegetation canopy temperature

radteff surface radiative temperature

albedo surface albedo

swe snow water equivalent

sweveg snow water equivalent intercepted by veg

soilmoist soil moisture layers 1-3 (kg/m2)

soiltemp soil temperature (K)

soilwet total column soil wetness

potevap potential evaporation

ecanop interception evaporation

tveg transpiration

esoil bare soil evaporation

rootmoist root zone soil moisture

canopint canopy interception

subsnow snow sublimation

subsurf sublimation of snow free area

acond areodynamic conductance

ccond canopy conductance

snowfrac percent of tile covered by snow

sliqfrac snow liquid fraction

ect transpiration from canopy

eci evaporation of canopy interepted water

egs evaporation of ground intercepted water

hc sensible heat flux canopy

hg sensible heat flux ground

radnvisdir net radiation

radnvisdif net radiation

radnnirdir net radiation

radnnifdif net radiation

salbvisdir visible component of surface albedo - direct

salbvisdif visible component of surface albedo - diffuse

salbnirdir near ir component of surface albedo - direct

salbnirdif near ir component of surface albedo - diffuse

radtc net radiation canopy

radtg net radiation ground

watcan canopy interception

watgrd ground interception

snocan swe in canopy

snogrd swe on ground

wet1 soil wetness layer 1

wet2 soil wetness layer 2

wet3 soil wetness layer 3

tair air temperature forcing

qair specific humidity forcing

swdown downward shortwave forcing

lwdown downward longwave forcing

uwind u-wind component forcing

vwind v-wind component forcing

psurf surface pressure forcing

rainf_in total rainfall forcing

rainf_cp convective rainfall forcing

snowf_in total snowfall forcing

REVISION HISTORY:

```
28 Apr 2002: K. Arsenault, added NOAH LSM 2.5 code to LDAS
14 Nov 2002: Sujay Kumar, Optimized version for LIS
21 Apr 2004: David Mocko, Conversion from NOAH to HY-SSiB
25 Aug 2007: Chuck Alonge, Updates for LIS 5.0 compliance
27 Oct 2010: David Mocko, changes for HY-SSiB in LIS6.1
```

74.2.1 hyssib_readcrd (Source File: hyssib_readcrd.F90)

REVISION HISTORY:

```
14 Oct 2003: Sujay Kumar, Initial Code
21 Apr 2004: David Mocko, Conversion from NOAH to HY-SSiB
 5 Sep 2007: Chuck Alonge, Updates for LIS 5.0
27 Oct 2010: David Mocko, changes for HY-SSiB in LIS6.1
```

INTERFACE:

```
subroutine hyssib_readcrd()
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc, LIS_config
use LIS_timeMgrMod, only : LIS_parseTimeString
use LIS_logMod,    only : LIS_logunit, LIS_verify
use hyssib_lsmMod, only : hyssib_struc
```

DESCRIPTION:

Routine to read HY-SSiB specific parameters from the LIS configuration file

---

74.2.2 hyssib_readrst (Source File: hyssib_readrst.F90)

REVISION HISTORY:

```
21 Apr 2004: David Mocko, Conversion from NOAH to HY-SSiB
25 Aug 2007: Chuck Alonge, Updates for LIS 5.0 Compliance
```

INTERFACE:

```
subroutine hyssib_readrst
```

USES:

```
  use LIS_coreMod, only : LIS_rc, LIS_masterproc
  use LIS_historyMod, only : LIS_readvar_restart
  use LIS_logMod
  use hyssib_lsmMod
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
```

DESCRIPTION:

This program reads restart files for HY-SSiB. This includes all relevant water/energy storages, tile information, and time information. The following is the list of variables specified in the Hyssib restart file:

```
nc,nr,ntiles    - grid and tile space dimensions
tc           - hyssib canopy temperature
tg           - hyssib ground temperature
tsn          - hyssib snow on ground temperature
td           - hyssib deep soil temperature
www(3)       - hyssib soil wetness (for each layer)
capac(2)     - hyssib liq equiv. water&snow on canopy/ground
snow(2)      - hyssib snow storage (swe) on snow/ground
sgfg         - hyssib density of bulk snow layer
sdens        - hyssib momentum exchange coefficient
```

The routines invoked are:

drv_readvar_restart (3.17.6)
reads a variable from the restart file

74.2.3 hyssib_settbot (Source File: hyssib_settbot.F90)

REVISION HISTORY:

```
03 Jun 2005: David Mocko, Conversion from NOAH to HY-SSiB
05 Sep 2007: Chuck Alonge, Updates for LIS 5.0 compliance
```

INTERFACE:

```
subroutine hyssib_settbot
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_surface
use LIS_fileIOMod, only : LIS_read_param
use hyssib_lsmMod
```

DESCRIPTION:

This subroutine retrieves HY-SSiB bottom soil temperature Currently a static data is being used. If a climatology is available, it can be used by setting the appropriate frequency of the climatology
The routines invoked are:

LIS_read_param (3.14.4)
retrieves the bottom temperature data

74.2.4 hyssib_settopostd (Source File: hyssib_settopostd.F90)

REVISION HISTORY:

```
03 Jun 2005: David Mocko, Conversion from NOAH to HY-SSiB
27 Sep 2007: Chuck ALonge, Updates for LIS 5.0
```

INTERFACE:

```
subroutine hyssib_settopostd
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_surface
use LIS_logMod,    only : LIS_logunit
use hyssib_lsmMod
```

DESCRIPTION:

This subroutine retrieves standard deviation of topography data for the Hyssib LSM. Currently static data is being used.

74.2.5 hyssib_setup (Source File: hyssib_setup.F90)

REVISION HISTORY:

```
21 Apr 2004: David Mocko, Conversion from NOAH to HY-SSiB
27 Sep 2007: Chuck Alonge, Updates for LIS 5.0
27 Oct 2010: David Mocko, changes for HY-SSiB in LIS6.1
```

INTERFACE:

```
subroutine hyssib_setup()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use hyssib_lsmMod
use hyssibveg_module
use hyssibalb_module
```

DESCRIPTION:

This routine is the entry point to set up the parameters required for Hyssib LSM. These include the vegetation, albedo, bottom temperature and the initialization of state variables in Hyssib.
The routines invoked are:

hyssibveg_ini (74.4.1)
initializes the Hyssib vegetation parameter module

hyssibalb_vegini (74.3.1)
initializes the Hyssib albedo parameter module

hyssib_setvegparms (74.2.6)
initializes the vegetation-related parameters in Hyssib

hyssib_settbot (74.2.3)
initializes the bottom temperature fields

hyssib_settopostd (74.2.4)
initializes the std. dev. of topography in hyssib

hyssib_gfrac (74.0.13)
initializes monthly vegetation parameters in hyssib

hyssib_coldstart (74.0.9)
initializes the hyssib state variables

74.2.6 hyssib_setvegparms (Source File: hyssib_setvegparms.F90)

This subroutine retrieves HY-SSiB vegetation parameters

REVISION HISTORY:

```
21 Apr 2004: David Mocko, Conversion from NOAH to HY-SSiB
05 Sep 2007: Chuck Alonge, Updates for LIS 5.0
```

INTERFACE:

```
subroutine hyssib_setvegparms
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_surface
use LIS_logMod, only : LIS_logunit
use hyssib_lsmMod
use hyssibveg_module
```

DESCRIPTION:

This subroutine retrieves Hyssib veg./soil parameters. The current implementation uses a table-based lookup based on SIB classes to initialize the following parameters:

```
chil    - leaf angle distribution factor
topt    - optimum temperature for rst calculation
tll     - low temp for rst calculation
tu      - top temp for rst calculation
defac   - dew factor for rst calculation
```

```
ph1     - stome slope factor
ph2     - point at which stomates close
rootd   - rooting depth - canopy
rootd   - rooting depth - ground
rstpar1 - PAR influence on stomatal resist. coefficients
rstpar2 - PAR influence on stomatal resist. coefficients
rstpar3 - PAR influence on stomatal resist. coefficients
phsat   - soil moisture potential at saturation
poros   - soil porosity
bee     - Clapp-Hornberger emperical const.
satco   - sat. hydraulic conductivity
slope   - average topographic slope %
zdepth1 - depth of soil layer 1
zdepth2 - depth of soil layer 2
zdepth3 - depth of soil layer 3
```

74.2.7 hyssib_totinit (Source File: hyssib_totinit.F90)

REVISION HISTORY:

```
14 Jun 2002: Sujay Kumar, Initial Specification
21 Apr 2004: David Mocko, Conversion from NOAH to HY-SSiB
28 Aug 2007: Chuck Alonge, Updates for LIS 5.0
27 Oct 2010: David Mocko, changes for HY-SSiB in LIS6.1
```

INTERFACE:

```
subroutine hyssib_totinit(n)
```

USES:

```
use hyssib_lsmMod
use LIS_coreMod, only : LIS_rc

implicit none

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

DESCRIPTION:

This routine resets the time-averaged Hyssib variables after a model output. The routine is called after a model output call. The arguments are:

n index of the nest

74.2.8 hyssib_writerst (Source File: hyssib_writerst.F90)

REVISION HISTORY:

```
21 Apr 2004: David Mocko, Conversion from NOAH to HY-SSiB
25 Aug 2007: Chuck Alonge, Updates for LIS 5.0 Compliance
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(n)
```

USES:

```
  use ESMF
  use LIS_coreMod, only : LIS_rc, LIS_masterproc
  use LIS_timeMgrMod, only : LIS_isAlarmRinging
  use LIS_logMod
  use lis_fileIOMod, only : LIS_create_output_directory, &
                            LIS_create_restart_filename
  use hyssib_lsmMod
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This program writes restart files for Hyssib. This includes all relevant water/energy storage and tile information
The routines invoked are:

LIS_create_output_directory (3.14.6)
creates a timestamped directory for the restart files

LIS_create_restart_filename (3.14.2)
generates a timestamped restart filename

hyssib_dump_restart (74.2.9)
writes the hyssib variables into the restart file

74.2.9 hyssib_dump_restart (Source File: hyssib_writerst.F90)

REVISION HISTORY:

```
24 Aug 2004: James Geiger, Initial Specification
27 Sep 2005: Chuck Alonge, Updates for LIS 5.0
```

INTERFACE:

```
subroutine hyssib_dump_restart(n, ftn,wformat)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_historyMod
use hyssib_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: ftn
character(len=*)    :: wformat
```

DESCRIPTION:

This routine gathers the necessary restart variables and performs the actual write statements to create the restart files.
The arguments are:

n index of the nest

ftn unit number for the restart file

wformat restart file format (binary/netcdf)

The following is the list of variables written in the Hyssib restart file:

```
nc,nr,ntiles    - grid and tile space dimensions
tc           - hyssib canopy temperature
tg           - hyssib ground temperature
tsn          - hyssib snow on ground temperature
td           - hyssib deep soil temperature
www(3)       - hyssib soil wetness (for each layer)
capac(2)     - hyssib liq equiv. water&snow on canopy/ground
snow(2)      - hyssib snow storage (swe) on snow/ground
sgfg         - hyssib density of bulk snow layer
sdens        - hyssib momentum exchange coefficient
```

The routines invoked are:

LIS_writeGlobalHeader_restart (3.17.22)
writes the global header information

LIS_writeHeader_restart (3.17.23)
writes the header information for a variable

LIS_closeHeader_restart (3.17.24)
close the header

LIS_writevar_restart (3.17.3)
writes a variable to the restart file

74.3 Fortran: Module Interface hyssibalb_module (Source File: hyssibalb_module.F90)

In order to use the HySSiB albedo calculations, UMD vegetation types need to be mapped to SSiB vegetation types. The code in this file provides a description of the data structure used to contain the vegetation specific constants used in HySSiB's albedo calculations. The varaiables specified in the data structure include:

cedfu

cedir

cedfu1

cedir1

cedfu2

xmiu

cedir2

clefu

cledir

cether

xmiw

REVISION HISTORY:

```
 01 Dec 2007: Chuck Alonge; Initial Specification
implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
 public :: hyssibalb_ini
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
 public :: hyssibalb
```

74.3.1 hyssibalb_ini (Source File: hyssibalb_module.F90)

INTERFACE:

```
subroutine hyssibalb_ini()
```

USES:

```
  use LIS_coreMod, only : LIS_rc
  use hyssib_lsmMod ! HY-SSiB tile variables
```

DESCRIPTION:

Reads in runtime HY-SSiB albedo parameters to populate constants used in Hyssib albedo calculations

---

74.4 Fortran: Module Interface hyssibveg_module (Source File: hyssibveg_module.F90)

In order to use the HySSiB vegetation table data efficiently, SiB vegetation parameter data (static and monthly) are read into this module to reduce model i/o when updating monthly values. The varaiables specified in this data structure include:

rstpar par influence on stomatal resist. coefficients (rst)

chil leaf angle distribution factor

topt optimum temperature for rst calculation

tll bottom temperature for rst calculation

tu top temperature for rst calculation

defac dew factor for rst calculation

ph1 stome slope factor

ph2 point at which stomates close

rootd rooting depth for canopy/ground cover

bee Clapp-Hornberger empirical constant

phsat soil moisture potential at saturation

satco saturation hydraulic conductivity

poros soil porosity

zdepth exact independent depth of 3 soil layers

slope avg. topographic slope in

green green leaf fraction

vcover fraction of vegetation cover [fpar/greenness]

zlt leaf area index

z0 roughness height

z2 canopy top height

z1 canopy base height

rdc ground to canopy air space resistance coeff.

rbc bulk canopy boundary layer resistance coeff.

REVISION HISTORY:

```
    01 Dec 2007: Chuck Alonge; Initial Specification
   implicit none
   PRIVATE
  ------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
---------------------------------------------------------------------------
 public :: hyssibveg_ini
---------------------------------------------------------------------------
```

PUBLIC TYPES:

```
---------------------------------------------------------------------------
 public :: hyssibveg
```

74.4.1 hyssibveg_ini (Source File: hyssibveg_module.F90)

INTERFACE:

```
subroutine hyssibveg_ini()
```

USES:

```
  use LIS_coreMod, only : LIS_rc
  use hyssib_lsmMod ! HY-SSiB model control variables
```

DESCRIPTION:

Reads in runtime HY-SSiB vegetation parameters to populate constant and monthly varying values Hyssib calculations

---

75 Mosaic Land Surface Model

This section describes the interface implementations for the Mosaic land surface model.

75.0.2 mos_descale_soilm (Source File: mos_descale_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
24Mar2005: Xiwu Zhan; modified for Mos model
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine mos_descale_soilm(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use mos_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

75.0.3 mos_getsoilm (Source File: mos_getsoilm.F90)

REVISION HISTORY:

```
06 Oct 2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine mos_getsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use mos_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

75.0.4 mos_getsynsmpred (Source File: mos_getsynsmpred.F90)

REVISION HISTORY:

```
06 Oct 2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine mos_getsynsmpred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_surface
use mos_lsmMod
```

75.0.5 mos_qcsoilm (Source File: mos_qcsoilm.F90)

REVISION HISTORY:

```
06 Oct 2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine mos_qcsoilm(n, LSM_State)
```

USES:

```
use ESMF
use mos_lsmMod
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Qc's the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

75.0.6 mos_scale_soilm (Source File: mos_scale_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine mos_scale_soilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use mos_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

75.0.7 mos_setsoilm (Source File: mos_setsoilm.F90)

REVISION HISTORY:

```
06 Oct 2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine mos_setsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use mos_lsmMod
use LIS_logMod,    only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

This routine assigns the soil moisture prognostic variables to mosaic's model space.

75.0.8 mos_updatesoilm (Source File: mos_updatesoilm.F90)

REVISION HISTORY:

```
06 Oct 2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine mos_updatesoilm(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use mos_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Qc's the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

75.0.9 mos_write_soilm (Source File: mos_write_soilm.F90)

REVISION HISTORY:

```
06 Oct 2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine mos_write_soilm(ftn, n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use mos_lsmMod
use LIS_logMod,    only : LIS_logunit
use LIS_historyMod, only : LIS_writevar_restart

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: ftn
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

This routine assigns the soil moisture prognostic variables to mosaic's model space.

75.0.10 mapsib2umd (Source File: mapsib2umd.F90)

REVISION HISTORY:

```
13 Apr 2001: Urszula Jambor; Initial code, based on calc_albedo.f scheme
03 Feb 2002: Jon Gottschalck; Added sections to use Koster tilespace
             (may be better to put the tile space distinction in
             umd_sibalb.F90 at a later time)
```

INTERFACE:

```
subroutine mapsib2umd ()
```

USES:

```
use sibalb_module    ! SiB-based coefficients for albedo calculation.

implicit none
DESCRIPTION:
 This routine assigns then maps 7 SiB vegetation types to 13 UMD
 vegetation types, defining 4 coefficient arrays.
 **NOTE: current form of this routine only has mapping based on North
 America, and not also for the globe.**
```

75.0.11 mos_coldstart (Source File: mos_coldstart.F90)

REVISION HISTORY:

```
3 Jun  2003: Jon Gottschalck; Initial code
25 Sep 2007: Sujay Kumar; Upgraded for LIS5.0
```

INTERFACE:

```
subroutine mos_coldstart(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_logMod, only : LIS_logunit
use LIS_timeMgrMod, only : LIS_date2time
use mos_lsmMod
```

DESCRIPTION:

This routine initializes the mosaic state variables with some predefined values uniformly for the entire domain. These initial values will be overwritten by the values read from the supplied Mosaic model restart file.

75.0.12 mos_f2t (Source File: mos_f2t.F90)

REVISION HISTORY:

```
 28 Jan 2002: Jon Gottschalck; Added option for different number of forcing variables
25 Sep 2007: Sujay Kumar; Upgraded for LIS5.0
```

INTERFACE:

```
subroutine mos_f2t(n)
```

USES:

```
use ESMF
use LIS_coreMod ,       only : LIS_rc, LIS_surface
use LIS_FORC_AttributesMod
use LIS_metforcingMod,  only : LIS_FORC_State
use LIS_logMod,         only : LIS_verify
use mos_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the mosaic model tiles.
The arguments are:

n index of the nest

75.0.13 mos_finalize (Source File: mos_finalize.F90)

REVISION HISTORY:

```
04 Oct 2005: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine mos_finalize()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use mos_lsmMod
```

DESCRIPTION:

This routine cleans up the allocated memory structures in Mosaic

---

75.0.14 mos_lsmMod (Source File: mos_lsmMod.F90)

This module provides the definition of derived data type used to control the operation of Mosaic LSM. It also provides the entry method for the initialization of Mosaic-specific variables. The derived data type `mos_struc` includes the variables that specify the runtime options and other control variables as described below:

mos_rfile name of the mosaic restart file

mos_vfile name of the static vegetation parameter table

mos_sfile name of the soil parameter table

mos_mvfile name of monthly vegetation parameter file

mos_pfile name of general parameter file (levs, depths)

count variable to keep track of the number of timesteps before an output

mosopen variable to keep track of opened files

numout number of output times

mos_nvegp number of static vegetation parameters in the table

mos_nmvegp number of monthly vegetation parameters in the table

mos_ism initial soil moisture for a cold start run

mos_it initial soil temperature for a cold start run

outInterval output writing interval

rstInterval restart writing interval

forcing_z flag indicating whether to use the observation height from the forcing

forcing_ch flag indicating whether to use aerodynamic conductance field from forcing

mos Mosaic LSM specific variables

REVISION HISTORY:

```
Jun 2003; Jon Gottschalck, Initial Code
Sept 2007: Sujay Kumar, Upgraded the data structures for LIS 5.0
```

INTERFACE:

USES:

```
use mos_module
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: mos_lsm_ini
```

PUBLIC TYPES:

```
public :: mos_struc
```

75.0.15 mos_lsm_ini (Source File: mos_lsmMod.F90)

INTERFACE:

```
subroutine mos_lsm_ini()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_surfaceModelDataMod, only : LIS_sfmodel_struc
use LIS_timeMgrMod,   only : LIS_clock, LIS_calendar, &
     LIS_update_timestep, LIS_registerAlarm
use LIS_logMod,       only : LIS_verify
```

DESCRIPTION:

This routine creates the datatypes and allocates memory for mosaic-specific variables. It also invokes the routine to read the runtime specific options for mosaic from the configuration file.
The routines invoked are:

readmoscrd (75.1.7)
reads the runtime options for Mosaic LSM

75.0.16 mos_main (Source File: mos_main.F90)

REVISION HISTORY:

```
11  Apr 2000: Brian Cosgrove; Initial Code
12  May 2000: Brian Cosgrove/Jared Entin; Added code to prevent
              numerical instablility due to small values of wind
              and humidity
22 Aug. 2000: Brian Cosgrove; Modified code for output of
              standard LDAS output variables.  Added LDAS and MOS
              modules into call for MOSTILE and also added many
              new variables in the MOS%RETURN section
25 Aug. 2000: Brian Cosgrove; Fixed snowcover fraction output variable
21 Sep. 2000: Brian Cosgrove; Fixed code so that rain output field
```

```
               is set to zero when snow output field is greater than zero
  27 Sep. 2000: Brian Cosgrove; Fixed code so that 1 meter soil moisture
               is correctly calculated (FACTOR variable had been
               incorrectly computed before this fix)
  23 Mar. 2001: Jon Radakovich; Updated for PSAS temperature assimilation
  19 Apr. 2001  Updated scheme for calculating albedo, using sibalb module
  05 Sep. 2001: Brian Cosgrove; Added in volumetric variables soilwm,soilww
               for use in LDAS output.  Zthick changed from 50 to 10 sometime
               by someone else.  Added volumetric output at 4 more levels
               corresponding to OK mesonet levels.  Changed calculation
               of soil wetness output variables
  05 Feb. 2002: Brian Cosgrove; Changed Zthick back to 50 from 10 after talk
               with Jon Radokovich.  10 is correct for observation height, but
               was causing fluxes that were too high, so changed back to
               'incorrect' value of 50
  22 Apr. 2002: Urszula Jambor; Added conditional to suppress calculation
               of dewpt temp. if using Aaron Berg's reanalysis ECMWF data.
  13 Sep. 2002: Urszula Jambor; Reversed suppression of dew point temp.
               calculation if using Aaron Berg's reanalysis ECMWF data.
  27 Nov. 2002: Urszula Jambor; Restricted mos%snow to max. of 100000.0
               to prevent problems in GRIB output snow fields.
  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.
   25 Sep 2007: Sujay Kumar; Upgraded for LIS5.0
```

INTERFACE:

```
subroutine mos_main(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_surface
use LIS_constantsMod,  only : LIS_CONST_RHOFW
use LIS_timeMgrMod, only : LIS_isAlarmRinging
use LIS_logMod, only : LIS_logunit
use LIS_albedoMod, only : LIS_alb
use LIS_histDataMod
use mos_lsmMod
use sibalb_module

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This is the entry point for calling the Mosaic LSM physics. This routine calls the `MOSTILE` routine that performs the land surface computations, to solve for water and energy equations. For documentation of the `MOSTILE` and other Mosaic routines, please see the reference: Koster, R. D., and M. J. Suarez, Energy and Water Balance Calculations in the MOSAIC LSM. NASA Technical Memorandum 104606, 9, 76 pp., 1996. The arguments are:

n index of the nest

75.1 Fortran: Module Interface mos_module (Source File: mos_module.F90)

The code in this file provides a description of the data structure containing the mosaic 1-d variables. The variables specified in the data structure include:

vegt vegetation type of tile

soiltype soil type of tile

vegp static vegetation parameter values

vegip interpolated monthly parameter values

soilp static soil parameter values

lai leaf area index of the tile

green greeness fraction value of the tile

dsai stem area index of the tile

ct canopy/soil temperature

qa canopy humidity

ics interception canopy storage

snow snow depth

sot deep soil temperature

sowet soil wetness

tair 2m air temperature forcing

qair 2m specific humidity forcing

swdown downward shortwave forcing

lwdown downward longwave forcing

uwind u-wind component forcing

vwind v-wind component forcing

psurf surface pressure forcing

rainf total rainfall forcing

rainf_c convective rainfall forcing

snowf total snowfall forcing

swnet net shortwave radiation (W/m2)

lwnet net longwave radiation (W/m2)

qle latent heat flux (W/m2)

qh sensible heat flux (W/m2)

qg ground heat flux (W/m2)

snowfall snowfall (kg/m2s)

pcp rainfall (kg/m2s)

evap evapotranspiration (kg/m2s)

qs surface runoff (kg/m2s)

qsb subsurface runoff (kg/m2s)

qsm snow melt (kg/m2s)

avgsurft average surface temperature (K)

albedo surface albedo (-)

swe snow water equivalent (kg/m2)

soilmoist1 soil moisture for layer1 (kg/m2)

soilmoist2 soil moisture for layer2 (kg/m2)

soilmoist3 soil moisture for layer3 (kg/m2)

soilmoist1m soil moisture for top 1m of soil (kg/m2)

soilwet total column soil wetness (-)

soilwetrz total column soil wetness (-)

ecanop interception evaporation (kg/m2s)

tveg vegetation transpiration (kg/m2s)

canopint total canopy water storage (kg/m2s)

rootmoist root zone soil moisture (kg/m2)

soilm_prev soil moisture from the previous model output

swe_prev snow water equivalent from the previous model output

water1 ??

water2 ??

water3 ??

dtcanal change in temperature based on analysis

REVISION HISTORY:

```
11 Feb 2002: Jon Gottschalck; Added AVHRR derived variables
25 Sep 2007: Sujay Kumar; Upgraded for LIS5.0
09 Nov 2007: Chuck Alonge; Removed nslay - unused
```

INTERFACE:

```
module mos_module
```

75.1.1 mos_readrestart (Source File: mos_readrestart.F90)

REVISION HISTORY:

```
19 Jan 2001: Brian Cosgrove; Initial Code
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.
23 Jan 2003: Urszula Jambor; Switch index order of GEOS forcing
              array.  Snow is 12, soil wetness is 13.
25 Sep 2007: Sujay Kumar, Upgraded for LIS 5.0
10 Jun 2012: Sujay Kumar, added support for netcdf formats
```

INTERFACE:

```
subroutine mos_readrestart
```

USES:

```
  use LIS_coreMod, only : LIS_rc
  use mos_lsmMod
  use LIS_historyMod, only : LIS_readvar_restart
  use LIS_logMod
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
```

DESCRIPTION:

This program reads restart files for Mosaic. This includes all relevant water/energy storages and tile information. The following is the list of variables specified in the Mosaic restart file:

```
nc,nr,ntiles    - grid and tile space dimensions
ct           - mosaic canopy temperature
qa           - mosaic canopy humidity
ics          - mosaic interception canopy storage
snow         - mosaic snow depth
sot          - mosaic deep soil temperature
soWet        - mosaic soil wetness
```

The routines invoked are:

LIS_readvar_restart (3.17.6)
reads a variable from the restart file

75.1.2 mos_setup (Source File: mos_setup.F90)

REVISION HISTORY:

```
3 Jun  2003: Jon Gottschalck; Initial Code
25 Sep 2007: Sujay Kumar; Upgraded for LIS5.0
```

INTERFACE:

```
subroutine mos_setup()
```

USES:

```
  use LIS_coreMod, only : LIS_rc
  use mos_lsmMod

  implicit none
```

DESCRIPTION:

This routine is the entry point to set up the parameters required for Mosaic LSM. These include the soils and topography and the initialization of state variables in Mosaic.
The routines invoked are:

setmosp (75.1.8)
initializes the static soil and topography fields

mapsib2umd (75.0.10)
maps the SIB vegetation types to UMD classification

mos_coldstart (75.0.11)
initializes the noah state variables

75.1.3 mos_writerst (Source File: mos_writerst.F90)

REVISION HISTORY:

```
    19 Jan 2001: Brian Cosgrove; Initial Code
    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.
    23 Jan 2003: Urszula Jambor; Switch index order of GEOS forcing
                  array.  Snow is 12, soil wetness is 13.
    25 Sep 2007: Sujay Kumar, Upgraded for LIS 5.0
    10 Jun 2012: Sujay Kumar, added support for netcdf formats
```

INTERFACE:

```
subroutine mos_writerst(n)
```

USES:

```
  use LIS_coreMod, only : LIS_rc, LIS_masterproc
  use LIS_timeMgrMod, only  : LIS_isAlarmRinging
  use mos_lsmMod, only    : mos_struc
  use LIS_logMod
  use lis_fileIOMod, only : LIS_create_output_directory, &
       LIS_create_restart_filename
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This program writes restart files for Mosaic. This includes all relevant water/energy storage and tile information
The routines invoked are:

LIS_create_output_directory (3.14.6)
creates a timestamped directory for the restart files

LIS_create_restart_filename (3.14.2)
generates a timestamped restart filename

mos_dump_restart (75.1.4)
writes the Mosaic variables into the restart file

75.1.4 mos_dump_restart (Source File: mos_writerst.F90)

REVISION HISTORY:

```
25 Sep 2007: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine mos_dump_restart(ftn,n,wformat)
```

USES:

```
use LIS_coreMod, only  : LIS_rc, LIS_masterproc
use mos_lsmMod
use LIS_historyMod

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: ftn
character(len=*), intent(in) :: wformat
```

DESCRIPTION:

This routine gathers the necessary restart variables and performs the actual write statements to create the restart files.
The arguments are:

n index of the nest

ftn unit number for the restart file

wformat restart file format (binary/netcdf)

The following is the list of variables written in the Mosaic restart file:

```
nc,nr,ntiles    - grid and tile space dimensions
ct           - mosaic canopy temperature
qa           - mosaic canopy humidity
ics          - mosaic interception canopy storage
snow         - mosaic snow depth
sot          - mosaic deep soil temperature
soWet        - mosaic soil wetness
```

The routines invoked are:

LIS_writeGlobalHeader_restart (3.17.22)
writes the global header information

LIS_writeHeader_restart (3.17.23)
writes the header information for a variable

LIS_closeHeader_restart (3.17.24)
close the header

LIS_writevar_restart (3.17.3)
writes a variable to the restart file

75.1.5 mosdynp (Source File: mosdynp.F90)

REVISION HISTORY:

```
6  Apr 2001: Matt Rodell; Initial Specification
11 Feb 2002: Jon Gottschalck; Added use of AVHRR derived LAI/Greenness
01 Oct 2002: Jon Gottschalck; Modified to allow for MODIS LAI
25 Sep 2007: Sujay Kumar; Upgraded for LIS5.0
```

INTERFACE:

```
subroutine mosdynp(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc,LIS_domain,LIS_surface
use LIS_timeMgrMod, only : LIS_date2time
use LIS_vegDataMod, only : LIS_gfrac,LIS_lai, LIS_sai, LIS_roughness
use mos_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine take all the monthly varying parameters and the date and determine the actual value of the parameter for that date this actual value is returned to the main program The assumption is that the data point is valid for the 16th of the given month at 00hr

---

75.1.6 paramb (Source File: paramb.F90)

REVISION HISTORY:

```
11 Mar 2001 : Matt Rodell, Initial Specification
```

INTERFACE:

```
subroutine paramb (ntiles,tex,b)

  implicit none
```

ARGUMENTS:

```
  integer :: ntiles
  integer :: tex(ntiles)
  real    :: b(ntiles)
```

DESCRIPTION:

This subroutine assigns the dimensionless b parameter based on values modified from the soil texture class and the table of cosby et al. [1984].
The arguments are:

ntiles number of tiles

tex texture class

b b parameter

75.1.7 readmoscrd (Source File: readmoscrd.F90)

REVISION HISTORY:

```
15 Oct 2003; Sujay Kumar, Initial Code
25 Sep 2007: Sujay Kumar, Upgraded for LIS 5.0
12 Nov 2007: Chuck Alonge, Added entry # of soil textures
```

INTERFACE:

```
subroutine readmoscrd()
```

USES:

```
  use ESMF
  use LIS_timeMgrMod, only : LIS_parseTimeString
  use LIS_logMod, only : LIS_logunit, LIS_verify
  use LIS_coreMod, only : LIS_rc, LIS_config
  use mos_lsmMod, only : mos_struc
```

DESCRIPTION:

This routine reads the options specific to Mosaic LSM from the LIS configuration file.

75.1.8 setmosp (Source File: setmosp.F90)

REVISION HISTORY:

```
31 Jul 2001: Matt Rodell; Initial Specification
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.
25 Sep 2007: Sujay Kumar; Upgraded for LIS5.0
```

INTERFACE:

```
subroutine setmosp(n)
```

USES:

```
use LIS_coreMod,  only : LIS_rc, LIS_surface
use LIS_soilsMod, only : LIS_soils
use LIS_logMod,   only : LIS_logunit, LIS_endrun
use LIS_fileIOMod, only : LIS_read_param
use mos_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine sets the static mosaic parameters related to vegetation, soils and topography.

75.1.9 mos_sfc2cmem3 (Source File: mos_sfc2cmem3.F90)

REVISION HISTORY:

```
37 Mar 2009: Sujay Kumar; Initial Code
2  May 2011: Yudong Tian; Modified for Mos 3.2
```

INTERFACE:

```
subroutine mos_sfc2cmem3(n, sfcState)
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc, LIS_surface
use LIS_logMod,    only : LIS_verify
use LIS_soilsMod, only : LIS_soils
use LIS_historyMod, only : LIS_patch2tile
use LIS_constantsMod,  only : LIS_CONST_RHOFW

use mos_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: sfcState

FUNCTIONS
```

DESCRIPTION:

This subroutine assigns the mos specific surface variables to CMEM3.

75.2 Fortran: Module Interface sibalb_module (Source File: sibalb_module.F90)

In order to use the MOSAIC-SiB albedo calculation, SiB vegetation types are mapped to UMD vegetation types. For both SiB and UMD, 4 coefficient arrays are defined for calculating the albedos of soil in the subroutine sibalb, representing the following bands:
visible, direct solar radiation,
infra-red, direct solar radiation,
visible, diffuse solar radiation, and
infra-red, diffuse solar radiation.

MOSAIC/SiB ITYP: Vegetation type as follows:
1: BROADLEAF EVERGREEN TREES
2: BROADLEAF DECIDUOUS TREES
3: NEEDLELEAF TREES
4: GROUND COVER
5: BROADLEAF SHRUBS
6: DWARF TREES (TUNDRA)
7: BARE SOIL

UMD ITYP: Vegetation type as follows:
1. Evergreen Needleleaf Forest
2. Evergreen Broadleaf Forest
3. Deciduous Needleleaf Forest
4. Deciduous Broadleaf Forest
5. Mixed Cover
6. Woodland
7. Wooded Grassland
8. Closed Shrubland
9. Open Shrubland
10. Grassland
11. Cropland
12. Bare Ground
13. Urban and Built-Up

REVISION HISTORY:

```
04 Apr 2001: Urszula Jambor; Initial code, using old calc_albedo.f scheme
```

75.2.1 texture (Source File: texture.F90)

REVISION HISTORY:

```
7 Mar 2001 : Matt Rodell, Initial Specification
```

INTERFACE:

```
subroutine texture (sand,silt,clay,tex)
```

USES:

```
implicit none
```

ARGUMENTS:

```
real,    intent(in)    :: sand,silt,clay
integer, intent(inout) :: tex
```

DESCRIPTION:

Finds the USDA texture class based on percentages of sand, silt, and clay.

---

76 Noah land surface model version 2.7.1

This section describes the interface implementations for the Noah land surface model. The community Noah Land Surface Model is a stand-alone, uncoupled, 1-D column model freely available at the National Centers for Environmental Prediction (NCEP; [2]). The name is an acronym representing the various developers of the model (N: NCEP; O: Oregon State University, Dept. of Atmospheric Sciences; A: Air Force (both AFWA and AFRL - formerly AFGL, PL); and H: Hydrologic Research Lab - NWS (now Office of Hydrologic Development – OHD)). Noah can be executed in either coupled or uncoupled mode. It has been coupled with the operational NCEP mesoscale Eta model [7] and its companion Eta Data Assimilation System (EDAS) [12], and the NCEP Global Forecast System (GFS) and its companion Global Data Assimilation System (GDAS). When Noah is executed in uncoupled mode, near-surface atmospheric forcing data (e.g., precipitation, radiation, wind speed, temperature, humidity) is required as input. Noah simulates soil moisture (both liquid and frozen), soil temperature, skin temperature, snowpack depth, snowpack water equivalent, canopy water content, and the energy flux and water flux terms of the surface energy balance and surface water balance. The model applies finite-difference spatial discretization methods and a Crank-Nicholson time-integration scheme to numerically integrate the governing equations of the physical processes of the soil vegetation-snowpack medium, including the surface energy balance equation, Richards' [11] equation for soil hydraulics, the diffusion equation for soil heat transfer, the energy-mass balance equation for the snowpack, and the Jarvis [8] equation for the conductance of canopy transpiration.

76.0.2 noah271_gce_f2t (Source File: noah271_gce_f2t.F90)

REVISION HISTORY:

```
15 Oct 2002: Sujay Kumar; Initial Code
```

INTERFACE:

```
subroutine noah271_gce_f2t(n)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc, LIS_localPet
use LIS_FORC_AttributesMod
```

```
use LIS_metforcingMod, only : LIS_FORC_State
use LIS_logMod,         only : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the Noah model tiles.
The arguments are:

n index of the nest

76.0.3 noah271_setgceexport.F90 (Source File: noah271_setgceexport.F90)

Defines the export states from Noah to GCE in coupled mode

REVISION HISTORY:

```
02 Dec 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
subroutine noah271_setgceexport(n, expState)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc,LIS_domain
use LIS_historyMod, only : LIS_tile2grid
use lisGCEGridCompMod, only : lisgce_export
use noah271_lsmMod

implicit none
integer, intent(in) :: n
type(ESMF_State)    :: expState

integer :: i
real :: temp(LIS_rc%ntiles(n))
real :: t2v, rho
real :: temp1(LIS_rc%lnc(n), LIS_rc%lnr(n))
```

76.0.4 noah271_setwrfesmfexport.F90 (Source File: noah271_setwrfesmfexport.F90)

Defines the export states from Noah to WRF in coupled mode

REVISION HISTORY:

```
02 Dec 2003; Sujay Kumar, Initial Version
17 Nov 2008; Sujay Kumar, Modified for the ESMF coupled version
```

INTERFACE:

```
subroutine noah271_setwrfesmfexport(n, LISWRF_ExpState)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_historyMod, only : LIS_tile2grid
use LIS_logMod,     only : LIS_verify
 use module_lis_component, only : LISWRF_ExpState
use noah271_lsmMod

implicit none
integer, intent(in) :: n
type(ESMF_State)    :: LISWRF_ExpState
```

76.0.5 noah271_wrf_esmff2t (Source File: noah271_wrf_esmff2t.F90)

REVISION HISTORY:

```
15 Oct 2002: Sujay Kumar; Initial Code
```

INTERFACE:

```
subroutine noah271_wrf_esmff2t(n)
```

USES:

```
use ESMF
use LIS_coreMod ,      only : LIS_rc
use LIS_metforcingMod, only : LIS_FORC_State
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use noah271_lsmMod,        only : noah271_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the Noah model tiles in a coupled mode to WRF
The arguments are:

n index of the nest

76.0.6 noah271_setwrfexport.F90 (Source File: noah271_setwrfexport.F90)

Defines the export states from Noah to WRF in coupled mode

REVISION HISTORY:

```
02 Dec 2003; Sujay Kumar, Initial Version
17 Nov 2008; Sujay Kumar, Modified for the ESMF coupled version
```

INTERFACE:

```
subroutine noah271_setwrfexport(n, expState)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_historyMod, only : LIS_tile2grid
use LIS_logMod,     only : LIS_verify
use LISWRFGridCompMod, only : LISWRF_export
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)      :: expState
```

76.0.7 noah271_wrf_f2t (Source File: noah271_wrf_f2t.F90)

REVISION HISTORY:

```
15 Oct 2002: Sujay Kumar; Initial Code
```

INTERFACE:

```
subroutine noah271_wrf_f2t(n)
```

USES:

```
use ESMF
use LIS_coreMod ,      only : LIS_rc
use LIS_metforcingMod, only : LIS_FORC_State
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use noah271_lsmMod,        only : noah271_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the Noah model tiles in a coupled mode to WRF
The arguments are:

n index of the nest

76.0.8 noah271_descale_multism (Source File: noah271_descale_multism.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_descale_multism(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Descales soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.9 noah271_getmultism (Source File: noah271_getmultism.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_getmultism(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.10 noah271_getmultismpred (Source File: noah271_getmultismpred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_getmultismpred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use noah271_lsmMod
```

76.0.11 noah271_qcmultism (Source File: noah271_qcmultism.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_qcmultism(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.12 noah271_scale_multism (Source File: noah271_scale_multism.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_scale_multism(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Scales soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.13 noah271_setmultism (Source File: noah271_setmultism.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_setmultism(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

This routine assigns the soil moisture prognostic variables to noah's model space.

76.0.14 noah271_updatemultism (Source File: noah271_updatemultism.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_updatemultism(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

This routine assigns the soil moisture prognostic variables to noah's model space.

76.0.15 noah271_write_multism (Source File: noah271_write_multism.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_write_multism(ftn,n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod
use LIS_historyMod, only : LIS_writevar_restart
implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: ftn
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.16 noah271_descale_scf (Source File: noah271_descale_scf.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_descale_scf(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.17 noah271_getscfpred (Source File: noah271_getscfpred.F90)

REVISION HISTORY:

```
13Nov2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah271_getscfpred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real                   :: obs_pred(LIS_rc%ngrid(n),LIS_rc%nensem(n))
```

DESCRIPTION:

Returns the snow cover obs pred (model's estimate of observations) for data assimilation
The arguments are:

n index of the nest

obs_pred model's estimate of observations

76.0.18 noah271_getscfvars (Source File: noah271_getscfvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_getscfvars(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use noah271_lsmMod, only : noah271_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.19 noah271_map_scf (Source File: noah271_map_scf.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_map_scf(n,OBS_State,LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_constantsMod, only  : LIS_CONST_TKFRZ
use LIS_logMod,   only  : LIS_logunit, LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
type(ESMF_State)         :: LSM_Incr_State
```

DESCRIPTION:

This subroutine directly maps the observation state to the corresponding variables in the LSM state for SCF data assimilation.
The arguments are:

n index of the nest

OBS_State ESMF State for observations

LSM_State ESMF State for LSM state variables

76.0.20 noah271_qc_scf (Source File: noah271_qc_scf.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_qc_scf(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.21 noah271_scale_scf (Source File: noah271_scale_scf.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_scale_scf(n, LSM_State)
```

USES:

```
use ESMF

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.22 noah271_setscfvars (Source File: noah271_setscfvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_setscfvars(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use noah271_lsmMod, only : noah271_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine assigns the snow progognostic variables to noah's model space.

76.0.23 noah271_transform_scf (Source File: noah271_transform_scf.F90)

REVISION HISTORY:

```
25Jun2006: Sujay Kumar: Initial Specification
```

INTERFACE:

```
subroutine noah271_transform_scf(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_logMod,  only : LIS_verify
use noah271_lsmMod
```

76.0.24 noah271_updatescfvars (Source File: noah271_updatescfvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_updatescfvars(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod
use LIS_logMod,   only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

LSM_Incr_State ESMF State container for LSM state increments

76.0.25 noah271_descale_snow (Source File: noah271_descale_snow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_descale_snow(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.26 noah271_getscapred (Source File: noah271_getscapred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_getscapred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real                   :: obs_pred(LIS_rc%ngrid(n),LIS_rc%nensem(n))
```

76.0.27 noah271_getsnowvars (Source File: noah271_getsnowvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_getsnowvars(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.28 noah271_getsynswepred (Source File: noah271_getsynswepred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_getsynswepred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real                   :: obs_pred(LIS_rc%ngrid(n),LIS_rc%nensem(n))
```

76.0.29 noah271_map_snow (Source File: noah271_map_snow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_map_snow(n,OBS_State,LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_constantsMod, only  : LIS_CONST_TKFRZ
use LIS_logMod,   only  : LIS_logunit, LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
type(ESMF_State)         :: LSM_Incr_State
```

DESCRIPTION:

This subroutine directly maps the observation state to the corresponding variables in the LSM state for SCA data assimilation.
The arguments are:

n index of the nest

OBS_State ESMF State for observations

LSM_State ESMF State for LSM state variables

76.0.30 noah271_map_swe (Source File: noah271_map_swe.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_map_swe(n,OBS_State,LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_constantsMod, only  : LIS_CONST_TKFRZ
use LIS_logMod,   only  : LIS_logunit, LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
type(ESMF_State)         :: LSM_State
```

DESCRIPTION:

This subroutine directly maps the observation state to the corresponding variables in the LSM state for SWE data assimilation.
The arguments are:

n index of the nest

OBS_State ESMF State for observations

LSM_State ESMF State for LSM state variables

76.0.31 noah271_qc_snowobs (Source File: noah271_qc_snowobs.F90)

REVISION HISTORY:

```
25Feb2008: Sujay Kumar: Initial Specification
```

INTERFACE:

```
subroutine noah271_qc_snowobs(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_logMod,  only : LIS_verify
use LIS_constantsMod, only : LIS_CONST_TKFRZ
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

This subroutine performs any model-based QC of the observation prior to data assimilation. Here the snow observations are flagged when LSM indicates that (1) rain is falling (2) ground is fully or partially covered with snow.
The arguments are:

n index of the nest

OBS_State ESMF state container for observations

76.0.32 noah271_qcsnow (Source File: noah271_qcsnow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_qcsnow(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod
use LIS_logMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.33 noah271_scale_snow (Source File: noah271_scale_snow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_scale_snow(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.34 noah271_setsnowvars (Source File: noah271_setsnowvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_setsnowvars(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod
use LIS_logMod, only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine assigns the snow progognostic variables to noah's model space.

76.0.35 noah271_transform_snow (Source File: noah271_transform_snow.F90)

REVISION HISTORY:

```
25Jun2006: Sujay Kumar: Initial Specification
```

INTERFACE:

```
subroutine noah271_transform_snow(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_logMod,  only : LIS_verify
use noah271_lsmMod
```

76.0.36 noah271_updatesnowvars (Source File: noah271_updatesnowvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_updatesnowvars(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod
use LIS_logMod,   only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Updates and returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

LSM_Incr_State ESMF State container for LSM state increments

76.0.37 noah271_wrtswe (Source File: noah271_wrtswe.F90)

REVISION HISTORY:

```
13Nov2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah271_wrtswe(ftn, n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: ftn
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the tskin related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.38 noah271_descale_snwd (Source File: noah271_descale_snwd.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_descale_snwd(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the snwd related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.39 noah271_getsnwdpred (Source File: noah271_getsnwdpred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_getsnwdpred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real                   :: obs_pred(LIS_rc%ngrid(n),LIS_rc%nensem(n))
```

76.0.40 noah271_getsnwdvars (Source File: noah271_getsnwdvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_getsnwdvars(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.41 noah271_qcsnwd (Source File: noah271_qcsnwd.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_qcsnwd(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod
use LIS_logMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.42 noah271_scale_snwd (Source File: noah271_scale_snwd.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_scale_snwd(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the snwd related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.43 noah271_setsnwdvars (Source File: noah271_setsnwdvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_setsnwdvars(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod
use LIS_logMod, only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine assigns the snwd progognostic variables to noah's model space.

76.0.44 noah271_updatesnwdvars (Source File: noah271_updatesnwdvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_updatesnwdvars(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod
use LIS_logMod,   only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

LSM_Incr_State ESMF State container for LSM state increments

76.0.45 noah271_descale_soilm (Source File: noah271_descale_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_descale_soilm(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Descales soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.46 noah271_getsoilm (Source File: noah271_getsoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_getsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.47 noah271_getsynsmpred (Source File: noah271_getsynsmpred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_getsynsmpred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use noah271_lsmMod
```

76.0.48 noah271_map_sm (Source File: noah271_map_sm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_map_sm(n,OBS_State,LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_logMod,  only : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
type(ESMF_State)         :: LSM_State
```

DESCRIPTION:

This subroutine directly maps the observation state to the corresponding variables in the LSM state for soil moisture data assimilation.
The arguments are:

n index of the nest

OBS_State ESMF State for observations

LSM_State ESMF State for LSM state variables

76.0.49 noah271_qc_soilmobs (Source File: noah271_qc_soilmobs.F90)

REVISION HISTORY:

```
25Feb2008: Sujay Kumar: Initial Specification
```

INTERFACE:

```
subroutine noah271_qc_soilmobs(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_logMod,  only : LIS_verify
use LIS_constantsMod, only : LIS_CONST_TKFRZ
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

This subroutine performs any model-based QC of the observation prior to data assimilation. Here the soil moisture observations are flagged when LSM indicates that (1) rain is falling (2) soil is frozen or (3) ground is fully or partially covered with snow.
The arguments are:

n index of the nest

OBS_State ESMF state container for observations

76.0.50 noah271_qcsoilm (Source File: noah271_qcsoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_qcsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.51 noah271_scale_soilm (Source File: noah271_scale_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_scale_soilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Scales soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.52 noah271_setsoilm (Source File: noah271_setsoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_setsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

This routine assigns the soil moisture prognostic variables to noah's model space.

76.0.53 noah271_transform_synsm (Source File: noah271_transform_synsm.F90)

REVISION HISTORY:

```
25Jun2006: Sujay Kumar: Initial Specification
```

INTERFACE:

```
subroutine noah271_transform_synsm(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_logMod,  only : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

This subroutine transforms the Synthetic observation state (mm) to a volumetric soil moisture state. The arguments are:

n index of the nest

OBS_State ESMF state container for observations

76.0.54 noah271_updatesoilm (Source File: noah271_updatesoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_updatesoilm(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

This routine assigns the soil moisture prognostic variables to noah's model space.

76.0.55 noah271_write_soilm (Source File: noah271_write_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_write_soilm(ftn,n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod
use LIS_historyMod, only : LIS_writevar_restart
implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: ftn
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.56 noah271_descale_swe (Source File: noah271_descale_swe.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_descale_swe(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the swe related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.57 noah271_getswepred (Source File: noah271_getswepred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_getswepred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real                   :: obs_pred(LIS_rc%ngrid(n),LIS_rc%nensem(n))
```

76.0.58 noah271_getswevars (Source File: noah271_getswevars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_getswevars(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.59 noah271_qcswe (Source File: noah271_qcswe.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_qcswe(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod
use LIS_logMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.60 noah271_scale_swe (Source File: noah271_scale_swe.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_scale_swe(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the swe related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.61 noah271_setswevars (Source File: noah271_setswevars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_setswevars(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod
use LIS_logMod, only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine assigns the swe progognostic variables to noah's model space.

76.0.62 noah271_updateswevars (Source File: noah271_updateswevars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_updateswevars(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah271_lsmMod
use LIS_logMod,   only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

LSM_Incr_State ESMF State container for LSM state increments

76.0.63 noah271_descale_tskin (Source File: noah271_descale_tskin.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_descale_tskin(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.64 noah271_getTskinPred (Source File: noah271_getTskinPred.F90)

REVISION HISTORY:

```
13Nov2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah271_getTskinPred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use noah271_lsmMod
```

76.0.65 noah271_gettskin (Source File: noah271_gettskin.F90)

REVISION HISTORY:

```
13Nov2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah271_gettskin(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the tskin related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.66 noah271_qc_tskinobs (Source File: noah271_qc_tskinobs.F90)

REVISION HISTORY:

```
25Feb2008: Sujay Kumar: Initial Specification
```

INTERFACE:

```
subroutine noah271_qc_tskinobs(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_logMod,  only : LIS_verify
use LIS_constantsMod, only : LIS_CONST_TKFRZ
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

This subroutine performs any model-based QC of the observation prior to data assimilation. Here the tskin observations are flagged when LSM indicates that (1) rain is falling (2) ground is fully or partially covered with snow.
The arguments are:

n index of the nest

OBS_State ESMF state container for observations

76.0.67 noah271_qctskin (Source File: noah271_qctskin.F90)

REVISION HISTORY:

```
13Nov2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah271_qctskin(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify, LIS_logunit
use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

QC's the tskin related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.68 noah271_scale_tskin (Source File: noah271_scale_tskin.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_scale_tskin(n, LSM_State)
```

USES:

```
use ESMF

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.69 noah271_settskin (Source File: noah271_settskin.F90)

REVISION HISTORY:

```
13Nov2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah271_settskin(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_domain
use noah271_lsmMod
use LIS_logMod,  only  : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

This routine assigns the tskin prognostic variables to noah's model space.

76.0.70 noah271_update_tskin (Source File: noah271_update_tskin.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah271_update_tskin(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_domain
use noah271_lsmMod
use LIS_logMod, only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.71 noah271_wrttskin (Source File: noah271_wrttskin.F90)

REVISION HISTORY:

```
13Nov2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah271_wrttskin(ftn, n, LSM_State)
```

USES:

```
use ESMF

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: ftn
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the tskin related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.0.72 noah_physics.F90 (Source File: module_sf_noah271lsm.F90)

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
 31 May 2004: Kristi Arsenault; Updated Noah LSM to 2.7 version
 24 Nov 2004: Kristi Arsenault; Updated Noah LSM to 2.7.1
 06 Apr 2005: Jesse Meng; Fix FDOWN, RTDIS, ETA, EVP
 16 Oct 2006: Jesse Meng; Complete Noah 2.7.1
-----------------------------------------------------------------------
module module_sf_noah271lsm
```

```
-- 1. MAIN PHYSICS SUBROUTINE ==>  SUBROUTINE SFLX  ------------------

PRIVATE

public :: SFLX
public :: CSNOW
public :: TDFCND
public :: PENMAN
public :: SFCDIF
public :: SNFRAC

contains
```

INTERFACE:

```
    SUBROUTINE SFLX (ntl, &
     SBETA, FXEXP, CZIL, &
     CFACTR,CMCMAX,RSMAX,TOPT,REFDK,REFKDT,CSOIL,FRZK,&
     FFROZP,ICE,DT,ZLVL,NSOIL,SLDPTH, &
     LWDN,SOLDN,SOLNET,SFCPRS,PRCP,SFCTMP,Q2,SFCSPD, &
     TH2,Q2SAT,DQSDT2, &
     SLOPE,SHDFAC,SHDMIN,PTU,ALB,SNOALB, &
     RSMIN,RGL,HS,SNUP,ZO,EMISSI, 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, ETA_KINEMATIC, &
     EC,EDIR,ET,ETT,ESNOW,DRIP,DEW, &
     BETA,ETP,SSOIL, &
     FLX1,FLX2,FLX3, &
     SNOMLT,SNCOVR, &
     RUNOFF1,RUNOFF2,RUNOFF3, &
     RC,PC,RCS,RCT,RCQ,RCSOIL,Q1,&
     SOILW,SOILT,SOILM,ETANRG, &
Save soil surface temperature and emissivity for output - D. Mocko
     TSOIL,EMISSI_OUT)
```

76.0.73 noah271_calc_tskin (Source File: noah271_calc_tskin.F90)

REVISION HISTORY:

```
10 Apr 2008: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine noah271_calc_tskin (n, stc,t1)
```

USES:

```
use LIS_coreMod,      only : LIS_rc
use LIS_constantsMod, only : LIS_CONST_TKFRZ
use LIS_vegDataMod,     only : LIS_gfrac
use noah271_lsmMod,      only : noah271_struc
use module_sf_noah271lsm, only : csnow, tdfcnd, penman
```

ARGUMENTS:

```
implicit none

real, parameter         :: sbeta = -2.0
real, parameter         :: sigma = 5.67E-8
real, parameter         :: cpice = 2.106E+3
real, parameter         :: cph2o = 4.218E+3
real, parameter         :: LSUBC = 2.501000E+6
real, parameter         :: LSUBS = 2.83E+6
integer, intent(in)     :: n
real, intent(in)        :: stc(LIS_rc%npatch(n,LIS_rc%lsm_index))
 real, intent(out)       :: t1(LIS_rc%npatch(n,LIS_rc%lsm_index))
real                    :: t1(LIS_rc%npatch(n,LIS_rc%lsm_index))
```

DESCRIPTION:

Diagnostic routine to update noah variables based on an updated T1 value

---

76.0.74 noah271_coldstart (Source File: noah271_coldstart.F90)

REVISION HISTORY:

```
28 Apr 2002: Sujay Kumar; Initial version
27 Oct 2010: David Mocko, changes for Noah2.7.1 in LIS6.1
```

INTERFACE:

```
subroutine noah271_coldstart()
```

USES:

```
use LIS_coreMod
use LIS_logMod,       only : LIS_logunit
use LIS_timeMgrMod,   only : LIS_date2time
use noah271_lsmMod
use LIS_fileIOMod,      only : LIS_readData
```

DESCRIPTION:

This routine initializes the Noah2.7.1 state variables with some predefined values uniformly for the entire domain. These initial values will be overwritten by the values read from the supplied Noah2.7.1 model restart file.

76.0.75 noah271_dynsetup (Source File: noah271_dynsetup.F90)

REVISION HISTORY:

```
28 Jan 2002: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine noah271_dynsetup(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_snowMod, only : LIS_snow_struc
use noah271_lsmMod, only : noah271_struc
```

DESCRIPTION:

This routine sets up the time-dependent variables in Noah2.7.1 Currently this routine is a placeholder

---

76.0.76 noah271_f2t (Source File: noah271_f2t.F90)

REVISION HISTORY:

```
15 Oct 2002: Sujay Kumar; Initial Code
27 Oct 2010: David Mocko, changes for Noah2.7.1 in LIS6.1
```

INTERFACE:

```
subroutine noah271_f2t(n)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_FORC_AttributesMod
use LIS_metforcingMod, only : LIS_FORC_State
use LIS_logMod,         only : LIS_verify
use noah271_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the Noah2.7.1 model tiles.
The arguments are:

n index of the nest

76.0.77 noah271_finalize (Source File: noah271_finalize.F90)

REVISION HISTORY:

```
28 Apr 2002: Sujay Kumar; Initial code
27 Oct 2010: David Mocko, changes for Noah2.7.1 in LIS6.1
```

INTERFACE:

```
subroutine noah271_finalize()
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use noah271_lsmMod
```

DESCRIPTION:

This routine cleans up the allocated memory structures in Noah2.7.1

76.1 Fortran: Module Interface noah271_lsmMod (Source File: noah271_lsmMod.F90)

This module provides the definition of derived data type used to control the operation of Noah2.7.1 LSM. It also provides the entry method for the initialization of Noah2.7.1-specific variables. The derived data type `noah2.7.1_struc` includes the variables that specify the runtime options and other control variables as described below:

rfile name of the noah2.7.1 restart file

vfile name of the static vegetation parameter table

sfile name of the soil parameter table

useptf whether PTFs should be used to map soil properties

count variable to keep track of the number of timesteps before an output

noah271open variable to keep track of opened files

soilscheme type of soil mapping scheme usd (1-zobler,2-statsgo)

numout number of output times

nslay number of soil layers

nvegp number of static vegetation parameters in the table

nsoilp number of soil parameters in the table

nstxts number of soil texture classes in the classification scheme

param_rst flag to indicate if parameters are to be read from a restart file (e.g. calibrated parameters output by the optUE codes)

prstfile parameter restart filename (e.g. GA output file)

forcing_ch flag indicating if the heat exchange coefficient is present in the forcing inputs

inittemp initial soil temperatures for a cold start run

initsm initial total soil moisture for a cold start run

initsmliq initial liquid soil moisture for a cold start run

initskintemp initial value of skin temperature for a cold start run

initcanopywater initial value of canopy water for a cold start run

initsnowdepth initial value of snow depth for a cold start run

initsnoweqiv initial value of snow water equivalent for a cold start run

outInterval output writing interval

rstInterval restart writing interval

rstAlarm restart alarm object

outAlarm output alarm object

zh reference height of T and q forcing

zm reference height of u and v forcing

forcing_ch flag indicating whether to use aerodynamic conductance field from forcing

lyrthk thickness of soil layers

noah Noah2.7.1 LSM specific variables

REVISION HISTORY:

```
Apr 2003; Sujay Kumar, Initial Code
 27 Oct 2010: David Mocko, changes for Noah2.7.1 in LIS6.1
```

USES:

```
use noah271_module

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: noah271_lsm_ini
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: noah271_struc
```

76.1.1 noah271_lsm_ini (Source File: noah271_lsmMod.F90)

INTERFACE:

```
subroutine noah271_lsm_ini()
```

USES:

```
  use LIS_coreMod,      only : LIS_rc
  use LIS_surfaceModelDataMod, only : LIS_sfmodel_struc
  use LIS_timeMgrMod,   only : LIS_clock, LIS_calendar,  &
       LIS_update_timestep, LIS_registerAlarm
  use LIS_logMod,       only : LIS_verify
```

DESCRIPTION:

This routine creates the datatypes and allocates memory for Noah2.7.1-specific variables. It also invokes the routine to read the runtime specific options for Noah2.7.1 from the configuration file.
The routines invoked are:

noah271_readcrd (76.2.1)
reads the runtime options for Noah2.7.1 LSM

76.1.2 noah271_main (Source File: noah271_main.F90)

REVISION HISTORY:

```
27 Oct 2010: David Mocko, changes for Noah2.7.1 in LIS6.1
17 Jan 2011: David Mocko, fixes to output diagnostics for energy
                          & water balance and fix to conversion of
                          U,V forcing height to same level as T,q
27 Oct 2011: Chris Franks, added output of near sfc RH as well as
                           heat & momemtum exchange coefficients
```

INTERFACE:

```
subroutine noah271_main(n)
```

USES:

```
use LIS_coreMod
use LIS_vegDataMod,    only : LIS_gfrac
use LIS_timeMgrMod,    only : LIS_isAlarmRinging
use LIS_albedoMod,     only : LIS_alb
use LIS_constantsMod,  only : LIS_CONST_RHOFW, LIS_CONST_TKFRZ, &
                              LIS_CONST_LATVAP
use LIS_logMod,        only : LIS_logunit, LIS_endrun
use LIS_histDataMod
use module_sf_noah271lsm, only : SFLX, SFCDIF, SNFRAC
use noah271_lsmMod
use LIS_tbotAdjustMod, only: LIS_tbotTimeUtil,LIS_updateTbot

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This is the entry point for calling the Noah2.7.1 LSM physics. This routine calls the `SFLX` routine that performs the land surface computations, to solve for water and energy equations. For documentation of the `SFLX` and other Noah2.7.1 routines, please see the "NOAH LSM USERS GUIDE" maintained at the public server at NOAA NCEP: http://www.emc.ncep.noaa.gov/mmb/gcp/noahlsm/Noah_LSM_USERGUIDE_2.7.1.htm The arguments are:

n index of the nest

76.1.3 sh2o_init.F90 (Source File: noah271_main.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
```

76.2 Fortran: Module Interface noah271_module (Source File: noah271_module.F90)

The code in this file provides a description of the data structure containing the Noah2.7.1 1-d variables. The variables specified in the data structure include:

soiltype soil type (integer index)

nroot Number of root layers, depends on the vegetation type

rsmin Minimum canopy resistance (s/m)

rgl Solar radiation term of canopy resistance function

hs vapor pressure deficit term of canopy resistance function

snup threshold snow depth (in water equivalent m) that implies 100% snow cover

z0 roughness length

lai leaf area index

vegip interpolated vegetation fraction

smcmax maximum soil moisture content (porosity)

psisat saturated matric potential

dksat saturated hydraulic conductivity

bexp b parameter value

quartz quartz fraction

smcwlt wilting point (volumetric)

smcref reference soil moisture (onset of soil moisture stress in transpiration, volumetric)

dwsat saturated soil diffusivity

mxsnalb maximum albedo expected over deep snow

tempbot bottom boundary temperature

t1 skin temperature (K)

cmc canopy water content

snowh actual snow depth (m)

sneqv snow water equivalent (m)

stc soil temperature for different layers

smc soil moisture for different layers

sh2o liquid-only soil moisture for different layers

ch heat/moisture exchange coefficient

cm momentum exchange coefficient

vegt vegetation type of tile

tair_agl_min minimum 2 meter AGL temperature

rhmin minimum 2 meter relative humidity

relsmc Volumetric relative soil moisture (m^3/m^3) (smc - smcwlt) / (porosity - smcwlt)

REVISION HISTORY:

```
28 Apr 2002: K. Arsenault added NOAH LSM 2.5 code to LDAS.
14 Nov 2002: Sujay Kumar Optimized version for LIS
27 Oct 2010: David Mocko, changes for Noah2.7.1 in LIS6.1
```

76.2.1 noah271_readcrd (Source File: noah271_readcrd.F90)

REVISION HISTORY:

```
14 Oct 2003; Sujay Kumar, Initial Code
27 Oct 2010: David Mocko, changes for Noah2.7.1 in LIS6.1
```

INTERFACE:

```
subroutine noah271_readcrd()
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_timeMgrMod, only : LIS_parseTimeString
use LIS_logMod,  only : LIS_logunit, LIS_verify, LIS_warning
use noah271_lsmMod, only : noah271_struc
```

DESCRIPTION:

This routine reads the options specific to Noah2.7.1 LSM from the LIS configuration file.

76.2.2 noah271_readrst (Source File: noah271_readrst.F90)

REVISION HISTORY:

```
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 Noah2.5 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.
27 Oct 2010: David Mocko, changes for Noah2.7.1 in LIS6.1
```

INTERFACE:

```
subroutine noah271_readrst
```

USES:

```
  use LIS_coreMod
  use LIS_historyMod, only : LIS_readvar_restart
  use LIS_logMod,     only : LIS_logunit, LIS_endrun, &
       LIS_getNextUnitNumber, LIS_releaseUnitNumber, LIS_verify
  use noah271_lsmMod
  use LIS_tbotAdjustMod, only: LIS_readTmnUpdateRestart
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
```

DESCRIPTION:

This program reads restart files for Noah2.7.1. This includes all relevant water/energy storages and tile information. The following is the list of variables specified in the Noah2.7.1 restart file:

```
nc,nr,ntiles    - grid and tile space dimensions
t1            - Noah2.7.1 skin temperature
cmc           - Noah2.7.1 canopy moisture storage
snowh         - Noah2.7.1 snow depth
sneqv         - Noah2.7.1 snow water equivalent
stc           - Noah2.7.1 soil temperature (for each layer)
smc           - Noah2.7.1 soil moisture (for each layer)
sh2o          - Noah2.7.1 liquid only soil moisture (for each layer)
ch            - Noah2.7.1 heat/moisture exchange coefficient
cm            - Noah2.7.1 momentum exchange coefficient
```

The routines invoked are:

LIS_readvar_restart (3.17.6)
reads a variable from the restart file

noah_coldstart (76.0.74)
initializes the Noah2.7.1 state variables

noah271_read_paramrst (76.2.3)
reads the OPTUE output to read the optimized LSM parameters

76.2.3 noah271_read_paramrst (Source File: noah271_readrst.F90)

INTERFACE:

```
subroutine noah271_read_paramrst(n)
```

USES:

```
use LIS_coreMod
use LIS_logMod,       only : LIS_getNextUnitNumber, LIS_releaseUnitNumber
use LIS_historyMod,   only : LIS_readvar_gridded
use noah271_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine reads the output file of the OPT/UE algorithm to read the optimized set of model parameters. The optimized parameters are then used to overwrite the LSM's default parameters.

76.2.4 noah271_setsoils (Source File: noah271_setsoils.F90)

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault;  Added Noah2.5 LSM, Initial Code
13 Oct 2003: Sujay Kumar; Domain independent modifications
15 Sep 2005: Matthew Garcia; update for ARMS project (Windows capability)
04 Oct 2005: Matthew Garcia; additions for Noah Unified
 9 Jul 2008: Chris Franks/AFWA; Save wilting point for calculation of relative soil moisture
27 Oct 2010: David Mocko, changes for Noah2.7.1 in LIS6.1
```

INTERFACE:

```
subroutine noah271_setsoils(mtype)
```

USES:

```
use LIS_coreMod
use LIS_soilsMod,  only : LIS_soils
use LIS_logMod,    only : LIS_logunit, LIS_endrun
use noah271_lsmMod
```

DESCRIPTION:

This subroutine sets the soil parameters in Noah2.7.1. Noah2.7.1 uses a lookup table based on the input soil texture to derive these parameters.
The routines invoked are:

LIS_mapSoilType (3.22.15)
method to derive the soil texture type from the sand, silt, and clay fractions.

76.2.5 noah271_settbot (Source File: noah271_settbot.F90)

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault;  Added Noah2.5 LSM, Initial Code
13 Oct 2003: Sujay Kumar; Domain independent modifications
```

INTERFACE:

```
subroutine noah271_settbot(mtype)
```

USES:

```
use LIS_coreMod
use LIS_tbotAdjustMod, only : LIS_initTmnUpdateTile
use LIS_fileIOMod

use noah271_lsmMod
```

DESCRIPTION:

This subroutine retrieves bottom boundary temperature for Noah2.7.1 LSM. Currently a static data is being used. If a climatology is available, it can be used by setting the appropriate frequency of the climatology
The routines invoked are:

LIS_read_param (3.14.4)
retrieves the bottom temperature data

76.2.6 noah271_setup (Source File: noah271_setup.F90)

REVISION HISTORY:

```
28 Apr 2002: Sujay Kumar; Initial version
```

INTERFACE:

```
subroutine noah271_setup()
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use noah271_lsmMod
```

DESCRIPTION:

This routine is the entry point to set up the parameters required for Noah2.7.1 LSM. These include the soils, greenness, albedo, bottom temperature and the initialization of state variables in Noah2.7.1.
The routines invoked are:

noah271_setvegparms (76.2.7)
initializes the vegetation-related parameters in Noah2.7.1

noah271_settbot (76.2.5)
initializes the bottom temperature fields

noah271_setsoils (76.2.4)
initializes the soil parameters

76.2.7 noah271_setvegparms (Source File: noah271_setvegparms.F90)

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault;  Added Noah2.5 LSM, Initial Code
13 Oct 2003: Sujay Kumar; Domain independent modifications
25 Jul 2005: Sujay Kumar; Removed the dependency to SIB classes
04 Oct 2005: Matthew Garcia; additions for Noah Unified
04 Oct 2005: Matthew Garcia; additions for Distributed Noah Router
```

INTERFACE:

```
subroutine noah271_setvegparms(mtype)
```

USES:

```
use LIS_coreMod
use noah271_lsmMod
```

DESCRIPTION:

This subroutine retrieves Noah2.7.1 vegetation parameters. The current implementation uses a table-based lookup based on vegetation classes to initialize the following parameters

```
nroot  - number of root zones
rsmin  - minimum canopy resistance
rgl    - solar radiation term in canopy resistance function
hs     - vapor pressure deficit term
snup   - threshold snow depth
z0     - roughness length
lai    - leaf area index
```

76.2.8 noah271_writerst (Source File: noah271_writerst.F90)

REVISION HISTORY:

```
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 Noah2.5 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
27 Oct 2010: David Mocko, changes for Noah2.7.1 in LIS6.1
```

INTERFACE:

```
subroutine noah271_writerst(n)
```

USES:

```
  use LIS_coreMod,   only : LIS_rc, LIS_masterproc
  use LIS_logMod,    only : LIS_logunit, LIS_getNextUnitNumber, &
       LIS_releaseUnitNumber, LIS_verify
  use LIS_timeMgrMod, only : LIS_isAlarmRinging
  use LIS_fileIOMod, only : LIS_create_output_directory, &
                            LIS_create_restart_filename
  use noah271_lsmMod
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This program writes restart files for Noah2.7.1. This includes all relevant water/energy storage and tile information.
The routines invoked are:

LIS_create_output_directory (3.14.6)
creates a timestamped directory for the restart files

LIS_create_restart_filename (3.14.2)
generates a timestamped restart filename

noah271_dump_restart (76.2.9)
writes the Noah2.7.1 variables into the restart file

76.2.9 noah271_dump_restart (Source File: noah271_writerst.F90)

REVISION HISTORY:

```
24 Aug 2004: James Geiger, Initial Specification
```

INTERFACE:

```
subroutine noah271_dump_restart(n, ftn, wformat)
```

USES:

```
  use LIS_coreMod,    only : LIS_rc, LIS_masterproc
  use LIS_historyMod
  use LIS_logMod,     only : LIS_logunit
  use noah271_lsmMod
  use LIS_tbotAdjustMod, only : LIS_writeTmnUpdateRestart

  implicit none

  integer, intent(in) :: ftn
  integer, intent(in) :: n
  character(len=*), intent(in) :: wformat
```

DESCRIPTION:

This routine gathers the necessary restart variables and performs the actual write statements to create the restart files.
The arguments are:

n index of the nest

ftn unit number for the restart file

The following is the list of variables written in the Noah2.7.1 restart file:

```
nc,nr,ntiles    - grid and tile space dimensions
t1           - Noah2.7.1 skin temperature
cmc          - Noah2.7.1 canopy moisture storage
snowh        - Noah2.7.1 snow depth
sneqv        - Noah2.7.1 snow water equivalent
stc          - Noah2.7.1 soil temperature (for each layer)
smc          - Noah2.7.1 soil moisture (for each layer)
sh2o         - Noah2.7.1 liquid only soil moisture (for each layer)
ch           - Noah2.7.1 heat/moisture exchange coefficient
cm           - Noah2.7.1 momentum exchange coefficient
```

The routines invoked are:

LIS_writevar_restart (3.17.3)
writes a variable to the restart file

76.2.10 noah271_getpeobspred_wgPBMRsm (Source File: noah271_getpeobspred_wgPBMRsm.F90)

REVISION HISTORY:

```
06 Feb 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah271_getpeobspred_wgPBMRsm(Obj_Func)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_soilsMod,  only : LIS_soils
use noah271_lsmMod, only : noah271_struc
use LIS_logMod,       only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: Obj_Func
```

DESCRIPTION:

This routine defines the obspred for the walnut gulch PBMR soil moisture observations, which are in mass units.

76.3 Fortran: Module Interface noah271_peMod (Source File: noah271_peMod.F90)

This module contains the definitions of the Noah model parameters used in parameter estimation. The data structure is used to expose the LSM parameters to be used in opt/ue.

REVISION HISTORY:

```
12 Jan 2012; Sujay Kumar, Initial Code
```

USES:

```
use ESMF

implicit none

PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
public :: noah271_setup_pedecvars
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
public :: noah271_pe_struc
```

76.3.1 noah271_setup_pedecvars (Source File: noah271_peMod.F90)

REVISION HISTORY:

```
06 Feb 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah271_setup_pedecvars(DEC_State, Feas_State)
```

USES:

```
use ESMF
use LIS_coreMod,       only : LIS_rc, LIS_config,LIS_vecTile
use LIS_logMod,        only : LIS_logunit, LIS_verify
use noah271_lsmMod,    only : noah271_struc

implicit none
```

ARGUMENTS:

```
character*100               :: decSpaceAttribsFile
type(ESMF_State)            :: DEC_State
type(ESMF_State)            :: Feas_State
```

DESCRIPTION:

This routine determines the list of parameters to be used in parameter estimation, initializes them, and updates the LIS decision space.

76.3.2 initializeDecSpace (Source File: noah271_peMod.F90)

INTERFACE:

```
subroutine initializeDecSpace(n,var,varmin,varmax)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_numerRecipesMod, only : LIS_rand_func
```

DESCRIPTION:

This subroutine initialize the decision space variable. If the decision space initialization mode is zero, then the population is initialized with the default LSM parameters. Otherwise the population is initialized with randomly sampled values based on the input range.

76.3.3 noah271_set_pedecvars (Source File: noah271_set_pedecvars.F90)

REVISION HISTORY:

```
06 Feb 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah271_set_pedecvars(DEC_State, Feas_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_soilsMod,  only : LIS_soils
use LIS_logMod,       only : LIS_logunit,LIS_verify
use noah271_lsmMod, only : noah271_struc
use noah271_peMod,  only : noah271_pe_struc

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: DEC_State
type(ESMF_State)       :: Feas_State
```

DESCRIPTION:

This routine assigns the decision space to Noah's model variables.

76.3.4 noah271_setupobspred_wgPBMRsm (Source File: noah271_setupobspred_wgPBMRsm.F90)

REVISION HISTORY:

```
16 Jul 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah271_setupobspred_wgPBMRsm(OBSPred)
```

USES:

```
use ESMF
use LIS_coreMod,      only : LIS_vecTile
use LIS_logMod,       only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: OBSPred
```

DESCRIPTION:

This routine generates a placeholder entry in the OBSPRED object for creating the obspred from the LSM for the walnut gulch PBMR soil moisture observations.

76.3.5 noah271_getrunoffs (Source File: noah271_getrunoffs.F90)

REVISION HISTORY:

```
6 May 2011: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah271_getrunoffs(n)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_routingMod, only : LIS_runoff_state
use LIS_logMod,     only : LIS_verify
use LIS_historyMod, only : LIS_gather_gridded_output
use noah271_lsmMod, only : noah271_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

76.3.6 noah271_sfc2cmem3 (Source File: noah271_sfc2cmem3.F90)

REVISION HISTORY:

```
37 Mar 2009: Sujay Kumar; Initial Code
```

INTERFACE:

```
subroutine noah271_sfc2cmem3(n, sfcState)
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc
use LIS_logMod,    only : LIS_verify
use LIS_constantsMod,  only : LIS_CONST_RHOFW

use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: sfcState

FUNCTIONS
```

DESCRIPTION:

This subroutine assigns the noah271 specific surface variables to CMEM3.

76.3.7 noah271_sfc2crtm (Source File: noah271_sfc2crtm.F90)

REVISION HISTORY:

```
37 Mar 2009: Sujay Kumar; Initial Code
```

INTERFACE:

```
subroutine noah271_sfc2crtm(n, sfcState)
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc
use LIS_logMod,    only : LIS_verify
use LIS_constantsMod,  only : LIS_CONST_RHOFW

use noah271_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: sfcState

FUNCTIONS
```

DESCRIPTION:

This subroutine assigns the noah271 specific surface variables to CRTM.

76.3.8 noah32_setwrfexport.F90 (Source File: noah32_setwrfexport.F90)

Defines the export states from Noah to WRF in coupled mode

REVISION HISTORY:

```
02 Dec 2003; Sujay Kumar, Initial Version
17 Nov 2008; Sujay Kumar, Modified for the ESMF coupled version
```

INTERFACE:

```
subroutine noah32_setwrfexport(n, expState)
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_historyMod, only : LIS_tile2grid
use LIS_logMod,     only : LIS_verify
use LISWRFGridCompMod, only : LISWRF_export
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)      :: expState
```

76.3.9 noah32_wrf_f2t (Source File: noah32_wrf_f2t.F90)

REVISION HISTORY:

```
15 Oct 2002: Sujay Kumar; Initial Code
```

INTERFACE:

```
subroutine noah32_wrf_f2t(n)
```

USES:

```
use ESMF
use LIS_coreMod ,      only : LIS_rc
use LIS_metforcingMod, only : LIS_FORC_State
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use noah32_lsmMod,        only : noah32_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the Noah model tiles in a coupled mode to WRF
The arguments are:

n index of the nest

76.4 Fortran: Module Interface noah32_dascf_Mod (Source File: noah32_dascf_Mod.F90)

di_opt Choice of DI methodology: "standard" – Rodell and Houser 2004, "customized" – Liu et al 2013

di_addswe Amount of SWE (mm) to add to model

di_maxmelt Maximum SWE melt rate (mm/day)

di_minswe Threshold of model SWE to be removed at once

di_ndaymelt Length of snowmelt period in days

di_scaobs1 Threshold of observed SCA for snow presence

di_scaobs2 Threshold of observed SCA for snow non-presence

di_swemod Threshold of model SWE(mm) for snow non-presence

REVISION HISTORY:

USES:

```
none

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: noah32_dascf_init
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: noah32_dascf_struc
```

76.4.1 noah32_dascf_init (Source File: noah32_dascf_Mod.F90)

INTERFACE:

```
subroutine noah32_dascf_init()
```

USES:

```
  use LIS_coreMod,      only : LIS_rc
  use LIS_logMod,       only : LIS_verify
```

DESCRIPTION:

76.4.2 noah32_descale_snow (Source File: noah32_descale_snow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah32_descale_snow(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.4.3 noah32_getsnowvars (Source File: noah32_getsnowvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah32_getsnowvars(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.4.4 noah32_getsnwdpred (Source File: noah32_getsnwdpred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah32_getsnwdpred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real                   :: obs_pred(LIS_rc%ngrid(n),LIS_rc%nensem(n))
```

76.4.5 noah32_getswepred (Source File: noah32_getswepred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah32_getswepred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real                   :: obs_pred(LIS_rc%ngrid(n),LIS_rc%nensem(n))
```

76.4.6 noah32_map_snow (Source File: noah32_map_snow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
20 Aug 2013: Yuqiong Liu; Modified to combine the two DI approaches,
  i.e., 'standard' - Rodell & House (2004, JHM) and 'customized' - Liu et al. (2013, AWR)
```

INTERFACE:

```
subroutine noah32_map_snow(n,OBS_State,LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_constantsMod, only  : LIS_CONST_TKFRZ
use LIS_logMod,   only  : LIS_logunit, LIS_verify
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
type(ESMF_State)         :: LSM_Incr_State
```

DESCRIPTION:

This subroutine directly maps the observation state to the corresponding variables in the LSM state for SCA data assimilation.
The arguments are:

n index of the nest

OBS_State ESMF State for observations

LSM_State ESMF State for LSM state variables

76.4.7 noah32_map_swe (Source File: noah32_map_swe.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah32_map_swe(n,OBS_State,LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_constantsMod, only  : LIS_CONST_TKFRZ
use LIS_logMod,   only  : LIS_logunit, LIS_verify
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
type(ESMF_State)         :: LSM_State
```

DESCRIPTION:

This subroutine directly maps the observation state to the corresponding variables in the LSM state for SWE data assimilation.
The arguments are:

n index of the nest

OBS_State ESMF State for observations

LSM_State ESMF State for LSM state variables

76.4.8 noah32_qc_snowobs (Source File: noah32_qc_snowobs.F90)

REVISION HISTORY:

```
25Feb2008: Sujay Kumar: Initial Specification
```

INTERFACE:

```
subroutine noah32_qc_snowobs(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_logMod,  only : LIS_verify
use LIS_constantsMod, only : LIS_CONST_TKFRZ
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

This subroutine performs any model-based QC of the observation prior to data assimilation. Here the snow observations are flagged when LSM indicates that (1) rain is falling (2) ground is fully or partially covered with snow.
The arguments are:

n index of the nest

OBS_State ESMF state container for observations

76.4.9 noah32_qcsnow (Source File: noah32_qcsnow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah32_qcsnow(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah32_lsmMod
use LIS_logMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.4.10 noah32_scale_snow (Source File: noah32_scale_snow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah32_scale_snow(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.4.11 noah32_setsnowvars (Source File: noah32_setsnowvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah32_setsnowvars(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah32_lsmMod
use LIS_logMod, only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine assigns the snow progognostic variables to noah's model space.

76.4.12 noah32_transform_snow (Source File: noah32_transform_snow.F90)

REVISION HISTORY:

```
25Jun2006: Sujay Kumar: Initial Specification
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah32_transform_snow(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_logMod,  only : LIS_verify
use noah32_lsmMod
```

76.4.13 noah32_updatesnowvars (Source File: noah32_updatesnowvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah32_updatesnowvars(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah32_lsmMod
use LIS_logMod,   only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

LSM_Incr_State ESMF State container for LSM state increments

76.4.14 noah32_wrtswe (Source File: noah32_wrtswe.F90)

REVISION HISTORY:

```
13Nov2007: Sujay Kumar; Initial Specification
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah32_wrtswe(ftn, n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: ftn
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the tskin related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.4.15 noah32_descale_soilm (Source File: noah32_descale_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah32_descale_soilm(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Descales soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.4.16 noah32_getsmpred (Source File: noah32_getsmpred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah32_getsmpred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use noah32_lsmMod
```

76.4.17 noah32_getsoilm (Source File: noah32_getsoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah32_getsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.4.18 noah32_qc_soilmobs (Source File: noah32_qc_soilmobs.F90)

REVISION HISTORY:

```
25Feb2008: Sujay Kumar: Initial Specification
```

INTERFACE:

```
subroutine noah32_qc_soilmobs(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_logMod,  only : LIS_verify
use LIS_constantsMod, only : LIS_CONST_TKFRZ
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

This subroutine performs any model-based QC of the observation prior to data assimilation. Here the soil moisture observations are flagged when LSM indicates that (1) rain is falling (2) soil is frozen or (3) ground is fully or partially covered with snow.
The arguments are:

n index of the nest

OBS_State ESMF state container for observations

76.4.19 noah32_qcsoilm (Source File: noah32_qcsoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah32_qcsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.4.20 noah32_scale_soilm (Source File: noah32_scale_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah32_scale_soilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Scales soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.4.21 noah32_setsoilm (Source File: noah32_setsoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah32_setsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify, LIS_logunit
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

This routine assigns the soil moisture prognostic variables to noah's model space.

76.4.22 noah32_updatesoilm (Source File: noah32_updatesoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah32_updatesoilm(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

This routine assigns the soil moisture prognostic variables to noah's model space.

76.4.23 noah32_write_soilm (Source File: noah32_write_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah32_write_soilm(ftn,n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah32_lsmMod
use LIS_historyMod, only : LIS_writevar_restart
implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: ftn
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.4.24 noah32_coldstart (Source File: noah32_coldstart.F90)

REVISION HISTORY:

```
28 Apr 2002: Sujay Kumar; Initial version
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
```

INTERFACE:

```
subroutine noah32_coldstart(mtype)
```

USES:

```
use LIS_coreMod,      only : LIS_rc
use LIS_logMod,       only : LIS_logunit
use LIS_timeMgrMod,   only : LIS_date2time
use noah32_lsmMod
```

DESCRIPTION:

This routine initializes the Noah3.2 state variables with some predefined values uniformly for the entire domain. These initial values will be overwritten by the values read from the supplied Noah3.2 model restart file.

76.4.25 noah32_dynsetup (Source File: noah32_dynsetup.F90)

REVISION HISTORY:

```
28 Jan 2002: Sujay Kumar, Initial Specification
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
```

INTERFACE:

```
subroutine noah32_dynsetup(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, LIS_surface
use LIS_snowMod, only : LIS_snow_struc
use LIS_vegDataMod,    only : LIS_gfrac, LIS_lai, LIS_roughness
use LIS_emissMod,      only : LIS_emiss
use LIS_albedoMod,     only : LIS_alb
use noah32_lsmMod, only : noah32_struc
use LIS_timeMgrMod, only        : LIS_date2time,LIS_tick
use module_sf_noah32lsm, only : month_d, calc_localtime
```

DESCRIPTION:

This routine sets up the time-dependent variables in Noah3.2

76.4.26 noah32_f2t (Source File: noah32_f2t.F90)

REVISION HISTORY:

```
 15 Oct 2002: Sujay Kumar; Initial Code
  8 May 2009: Sujay Kumar; additions for Noah3.1
 27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
  7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
```

INTERFACE:

```
subroutine noah32_f2t(n)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc, LIS_surface
use LIS_FORC_AttributesMod
use LIS_metforcingMod, only : LIS_FORC_State
use LIS_logMod,         only : LIS_verify
use noah32_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the Noah3.2 model tiles.
The arguments are:

n index of the nest

76.4.27 noah32_finalize (Source File: noah32_finalize.F90)

REVISION HISTORY:

```
28 Apr 2002: Sujay Kumar; Initial code
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
```

INTERFACE:

```
subroutine noah32_finalize()
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use noah32_lsmMod
```

DESCRIPTION:

This routine cleans up the allocated memory structures in Noah3.2

76.5 Fortran: Module Interface noah32_lsmMod (Source File: noah32_lsmMod.F90)

This module provides the definition of derived data type used to control the operation of Noah3.2 LSM. It also provides the entry method for the initialization of Noah3.2-specific variables. The derived data type `noah3.2_struc` includes the variables that specify the runtime options and other control variables as described below:

rfile name of the noah3.2 restart file

vfile name of the static vegetation parameter table

sfile name of the soil parameter table

gfile name of the general parameter table

useptf whether PTFs should be used to map soil properties

count variable to keep track of the number of timesteps before an output

soilscheme type of soil mapping scheme usd (1-zobler,2-statsgo)

nslay number of soil layers

nvegp number of static vegetation parameters in the table

nstxts number of soil texture classes in the classification scheme

varid NETCDF ids for variables (used for netcdf output)

inittemp initial soil temperatures for a cold start run

initsm initial total soil moisture for a cold start run

initsmliq initial liquid soil moisture for a cold start run

rstInterval restart writing interval

zh reference height of T and q forcing

zm reference height of u and v forcing

forcing_ch flag indicating whether to use aerodynamic conductance field from forcing

lyrthk thickness of soil layers

noah Noah3.2 LSM specific variables

REVISION HISTORY:

```
Apr 2003; Sujay Kumar, Initial Code
  8 May 2009: Sujay Kumar; additions for Noah3.1
 27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
  7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
```

USES:

```
use noah32_module

implicit none

PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
public :: noah32_lsm_ini
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
public :: noah32_struc
```

76.5.1 noah32_lsm_ini (Source File: noah32_lsmMod.F90)

INTERFACE:

```
subroutine noah32_lsm_ini()
```

USES:

```
use LIS_coreMod,      only : LIS_rc
use LIS_surfaceModelDataMod, only : LIS_sfmodel_struc
use LIS_timeMgrMod,   only : LIS_clock, LIS_calendar, &
     LIS_update_timestep, LIS_registerAlarm
use LIS_logMod,       only : LIS_verify
```

DESCRIPTION:

This routine creates the datatypes and allocates memory for Noah3.2-specific variables. It also invokes the routine to read the runtime specific options for Noah3.2 from the configuration file.
The routines invoked are:

noah32_readcrd (76.6.1)
reads the runtime options for Noah3.2 LSM

76.5.2 noah32_main (Source File: noah32_main.F90)

REVISION HISTORY:

```
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
20 Jan 2011: David Mocko, fixes to output diagnostics for energy
                          & water balance and fix to conversion of
                          U,V forcing height to same level as T,q
27 Oct 2011: Chris Franks, added calculation and output of minimum RH
                           and relative soil moisture
```

INTERFACE:

```
subroutine noah32_main(n)
```

USES:

```
use LIS_coreMod
use LIS_timeMgrMod,    only : LIS_isAlarmRinging
use LIS_albedoMod,     only : LIS_alb
use LIS_constantsMod,  only : LIS_CONST_RHOFW, LIS_CONST_TKFRZ, &
                              LIS_CONST_LATVAP
use LIS_logMod,        only : LIS_logunit, LIS_endrun
use LIS_histDataMod
use module_sf_noah32lsm, only : SFLX, DQSDT, SFCDIF_OFF, CALHUM, SNFRAC
use module_sfcdif_wrf_32, only: SFCDIF_MYJ
use noah32_lsmMod
use LIS_tbotAdjustMod, only: LIS_tbotTimeUtil,LIS_updateTbot

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This is the entry point for calling the Noah3.2 LSM physics. This routine calls the `SFLX` routine that performs the land surface computations, to solve for water and energy equations. For documentation of the `SFLX` and other Noah3.2 routines, please see the Noah3.2 webpage at NCAR:
http://www.ral.ucar.edu/research/land/technology/lsm.php
The arguments are:

n index of the nest

76.6 Fortran: Module Interface noah32_module (Source File: noah32_module.F90)

The code in this file provides a description of the data structure containing the Noah3.2 1-d variables. The variables specified in the data structure include:

soiltype soil type (integer index)

slopetype slope type (integer index)

nroot Number of root layers, depends on the vegetation type

rsmin Minimum canopy resistance (s/m)

rgl Solar radiation term of canopy resistance function

hs vapor pressure deficit term of canopy resistance function

snup threshold snow depth (in water equivalent m) that implies 100% snow cover

z0 roughness length

lai leaf area index

shdfac greenness vegetation fraction

shdmax maximum greenness vegetation fraction

shdmin minimum greenness vegetation fraction

smcmax maximum soil moisture content (porosity)

slope slope coefficient for baseflow

psisat saturated matric potential

dksat saturated hydraulic conductivity

bexp b parameter value

quartz quartz fraction

smcwlt wilting point (volumetric)

smcref reference soil moisture (onset of soil moisture stress in transpiration, volumetric)

dwsat saturated soil diffusivity

mxsnalb maximum albedo expected over deep snow

tempbot bottom boundary temperature

t1 skin temperature (K)

cmc canopy water content

snowh actual snow depth (m)

sneqv snow water equivalent (m)

stc soil temperature for different layers

smc soil moisture for different layers

sh2o liquid-only soil moisture for different layers

ch heat/moisture exchange coefficient

cm momentum exchange coefficient

vegt vegetation type of tile

tair_agl_min minimum 2 meter AGL temperature

rhmin minimum 2 meter relative humidity

relsmc Volumetric relative soil moisture (m^3/m^3) (smc - smcwlt) / (porosity - smcwlt)

REVISION HISTORY:

```
28 Apr 2002: K. Arsenault added Noah2.5 LSM code to LDAS.
14 Nov 2002: Sujay Kumar Optimized version for LIS
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
```

76.6.1 noah32_readcrd (Source File: noah32_readcrd.F90)

REVISION HISTORY:

```
14 Oct 2003; Sujay Kumar, Initial Code
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
```

INTERFACE:

```
subroutine noah32_readcrd()
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_timeMgrMod, only : LIS_parseTimeString
use LIS_logMod,  only : LIS_logunit, LIS_verify
use noah32_lsmMod, only : noah32_struc
```

DESCRIPTION:

This routine reads the options specific to Noah3.2 LSM from the LIS configuration file.

76.6.2 noah32_readrst (Source File: noah32_readrst.F90)

REVISION HISTORY:

```
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 Noah2.5 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.
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
```

INTERFACE:

```
subroutine noah32_readrst()
```

USES:

```
  use LIS_coreMod,    only : LIS_rc, LIS_masterproc
  use LIS_historyMod, only : LIS_readvar_restart
  use LIS_logMod,     only : LIS_logunit, LIS_endrun, &
       LIS_getNextUnitNumber, LIS_releaseUnitNumber, LIS_verify
  use module_sfcdif_wrf_32, only: MYJSFCINIT
  use noah32_lsmMod
  use LIS_tbotAdjustMod, only: LIS_readTmnUpdateRestart
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
```

DESCRIPTION:

This program reads restart files for Noah3.2. This includes all relevant water/energy storages and tile information. The following is the list of variables specified in the Noah3.2 restart file:

```
nc,nr,ntiles    - grid and tile space dimensions
t1            - Noah3.2 skin temperature
cmc           - Noah3.2 canopy moisture storage
snowh         - Noah3.2 snow depth
sneqv         - Noah3.2 snow water equivalent
stc           - Noah3.2 soil temperature (for each layer)
smc           - Noah3.2 soil moisture (for each layer)
sh2o          - Noah3.2 liquid only soil moisture (for each layer)
ch            - Noah3.2 heat/moisture exchange coefficient
cm            - Noah3.2 momentum exchange coefficient
```

The routines invoked are:

LIS_readvar_restart (3.17.6)
reads a variable from the restart file

noah32_coldstart (76.4.24)
initializes the Noah3.2 state variables

76.6.3 noah32_setsoils (Source File: noah32_setsoils.F90)

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault;  Added Noah2.5 LSM, Initial Code
13 Oct 2003: Sujay Kumar; Domain independent modifications
15 Sep 2005: Matthew Garcia; update for ARMS project (Windows capability)
04 Oct 2005: Matthew Garcia; additions for Noah Unified
 9 Jul 2008: Chris Franks/AFWA; Save wilting point for calculation of relative soil moisture
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
20 Jan 2011: David Mocko, added slope type
```

INTERFACE:

```
subroutine noah32_setsoils(mtype)
```

USES:

```
use LIS_coreMod,   only : LIS_rc, LIS_domain, LIS_surface
use LIS_soilsMod,  only : LIS_soils
use LIS_logMod,    only : LIS_logunit, LIS_endrun, &
                          LIS_getNextUnitNumber, LIS_releaseUnitNumber
use LIS_fileIOMod, only : LIS_read_param
use noah32_lsmMod
```

DESCRIPTION:

This subroutine sets the soil parameters in Noah3.2. Noah3.2 uses a lookup table based on the input soil texture to derive these parameters.
The routines invoked are:

LIS_mapSoilType (3.22.15)
method to derive the soil texture type from the sand, silt, and clay fractions.

76.6.4 noah32_settbot (Source File: noah32_settbot.F90)

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault;  Added Noah2.5 LSM, Initial Code
13 Oct 2003: Sujay Kumar; Domain independent modifications
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
```

INTERFACE:

```
subroutine noah32_settbot(mtype)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_surface
use LIS_logMod,     only : LIS_logunit
use LIS_tbotAdjustMod, only : LIS_initTmnUpdateTile
use LIS_fileIOMod,  only : LIS_read_param
use noah32_lsmMod
```

DESCRIPTION:

This subroutine retrieves bottom boundary temperature for Noah3.2 LSM. Currently a static data is being used. If a climatology is available, it can be used by setting the appropriate frequency of the climatology
The routines invoked are:

LIS_read_param (3.14.4)
retrieves the bottom temperature data

76.6.5 noah32_setup (Source File: noah32_setup.F90)

REVISION HISTORY:

```
28 Apr 2002: Sujay Kumar; Initial version
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
```

INTERFACE:

```
subroutine noah32_setup()
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use noah32_lsmMod
```

DESCRIPTION:

This routine is the entry point to set up the parameters required for Noah3.2 LSM. These include the soils, greenness, albedo, bottom temperature and the initialization of state variables in Noah3.2.
The routines invoked are:

noah32_setvegparms (76.6.6)
initializes the vegetation-related parameters in Noah3.2

noah32_settbot (76.6.4)
initializes the bottom temperature fields

noah32_setsoils (76.6.3)
initializes the soil parameters

76.6.6 noah32_setvegparms (Source File: noah32_setvegparms.F90)

REVISION HISTORY:

```
07 May 2009: Sujay Kumar; Initial Implementation
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
20 Jan 2011: David Mocko, added max/min greenness
```

INTERFACE:

```
subroutine noah32_setvegparms(mtype)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_surface
use LIS_vegDataMod,    only : LIS_gfrac, LIS_read_shdmax, &
     LIS_read_shdmin, LIS_lai, LIS_roughness
use LIS_emissMod,      only : LIS_emiss
use LIS_albedoMod,     only : LIS_alb
use LIS_logMod,  only : LIS_logunit, &
                        LIS_getNextUnitNumber, LIS_releaseUnitNumber
use LIS_timeMgrMod, only        : LIS_date2time,LIS_tick
use module_sf_noah32lsm, only : month_d, calc_localtime
use noah32_lsmMod
```

DESCRIPTION:

This subroutine retrieves Noah3.2 vegetation parameters. The current implementation uses a table-based lookup based on vegetation classes to initialize the following parameters

```
 nroot  - number of root zones
 rsmin  - minimum canopy resistance
 rgl    - solar radiation term in canopy resistance function
 hs     - vapor pressure deficit term
 snup   - threshold snow depth
 shdfac - greenness vegetation fraction
 shdmax - maximum greenness vegetation fraction during year
 shdmin - minimum greenness vegetation fraction during year
The following variables are also a function of shdfac:
 albedo - background (snow-free) surface albedo
 embrd  - background (snow-free) emissivity
 z0     - roughness length
 lai    - leaf area index
```

76.6.7 noah32_writerst (Source File: noah32_writerst.F90)

REVISION HISTORY:

```
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 Noah2.5 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
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
```

INTERFACE:

```
subroutine noah32_writerst(n)
```

USES:

```
  use LIS_coreMod,    only : LIS_rc, LIS_masterproc
  use LIS_timeMgrMod, only : LIS_isAlarmRinging
  use LIS_logMod,     only : LIS_logunit, LIS_getNextUnitNumber, &
       LIS_releaseUnitNumber, LIS_verify
  use LIS_fileIOMod,  only : LIS_create_output_directory, &
                             LIS_create_restart_filename
  use noah32_lsmMod
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This program writes restart files for Noah3.2. This includes all relevant water/energy storage and tile information.
The routines invoked are:

LIS_create_output_directory (3.14.6)
creates a timestamped directory for the restart files

LIS_create_restart_filename (3.14.2)
generates a timestamped restart filename

noah32_dump_restart (76.6.8)
writes the Noah3.2 variables into the restart file

76.6.8 noah32_dump_restart (Source File: noah32_writerst.F90)

REVISION HISTORY:

```
24 Aug 2004: James Geiger, Initial Specification
```

INTERFACE:

```
subroutine noah32_dump_restart(n, ftn, wformat)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_historyMod
use LIS_logMod,     only : LIS_logunit
use noah32_lsmMod
use LIS_tbotAdjustMod, only : LIS_writeTmnUpdateRestart

implicit none

integer,          intent(in) :: ftn
integer,          intent(in) :: n
character(len=*), intent(in) :: wformat
```

DESCRIPTION:

This routine gathers the necessary restart variables and performs the actual write statements to create the restart files.
The arguments are:

n index of the nest

ftn unit number for the restart file

wformat restart file format (binary/netcdf)

The following is the list of variables written in the Noah3.2 restart file:

```
nc,nr,ntiles    - grid and tile space dimensions
t1           - Noah3.2 skin temperature
cmc          - Noah3.2 canopy moisture storage
snowh        - Noah3.2 snow depth
sneqv        - Noah3.2 snow water equivalent
stc          - Noah3.2 soil temperature (for each layer)
smc          - Noah3.2 soil moisture (for each layer)
sh2o         - Noah3.2 liquid only soil moisture (for each layer)
ch           - Noah3.2 heat/moisture exchange coefficient
cm           - Noah3.2 momentum exchange coefficient
z0           - Noah3.2 roughness length
emiss        - Noah3.2 emissivity
albedo       - Noah3.2 albedo
snotime1     - Noah3.2 snow timing variable
q1           - Noah3.2 mixing ratio
```

The routines invoked are:

LIS_writeGlobalHeader_restart (3.17.22)
writes the global header information

LIS_writeHeader_restart (3.17.23)
writes the header information for a variable

LIS_closeHeader_restart (3.17.24)
close the header

LIS_writevar_restart (3.17.3)
writes a variable to the restart file

76.6.9 noah32_getpeobspred_ARMobs (Source File: noah32_getpeobspred_ARMobs.F90)

REVISION HISTORY:

```
06 Feb 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah32_getpeobspred_ARMobs(Obj_Func)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_soilsMod,  only : LIS_soils
use noah32_lsmMod, only : noah32_struc
use LIS_logMod,       only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: Obj_Func
```

DESCRIPTION:

This routine assigns the decision space to Noah's model variables.

76.6.10 noah32_getpeobspred_template (Source File: noah32_getpeobspred_template.F90)

REVISION HISTORY:

```
06 Feb 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah32_getpeobspred_template(Obj_Func)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_soilsMod,  only : LIS_soils
use noah32_lsmMod, only : noah32_struc
use LIS_logMod,       only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: Obj_Func
```

DESCRIPTION:

This routine assigns the decision space to Noah's model variables.

76.7 Fortran: Module Interface noah32_peMod (Source File: noah32_peMod.F90)

This module contains the definitions of the Noah model parameters used in parameter estimation. The data structure is used to expose the LSM parameters to be used in opt/ue.

REVISION HISTORY:

```
12 Jan 2012; Sujay Kumar, Initial Code
```

USES:

```
use ESMF

implicit none

PRIVATE
--------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
--------------------------------------------------------------------------------
public :: noah32_setup_pedecvars
--------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
--------------------------------------------------------------------------------
public :: noah32_pe_struc
```

76.7.1 noah32_setup_pedecvars (Source File: noah32_peMod.F90)

REVISION HISTORY:

```
06 Feb 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah32_setup_pedecvars(DEC_State, Feas_State)
```

USES:

```
use ESMF
use LIS_coreMod,       only : LIS_rc, LIS_config,LIS_vecTile
use LIS_logMod,        only : LIS_logunit, LIS_verify
use noah32_lsmMod,     only : noah32_struc

implicit none
```

ARGUMENTS:

```
character*100               :: decSpaceAttribsFile
type(ESMF_State)            :: DEC_State
type(ESMF_State)            :: Feas_State
```

DESCRIPTION:

This routine determines the list of parameters to be used in parameter estimation, initializes them, and updates the LIS decision space.

76.7.2 initializeDecSpace (Source File: noah32_peMod.F90)

INTERFACE:

```
subroutine initializeDecSpace(n,var,varmin,varmax)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_numerRecipesMod, only : LIS_rand_func
```

DESCRIPTION:

This subroutine initialize the decision space variable. If the decision space initialization mode is zero, then the population is initialized with the default LSM parameters. Otherwise the population is initialized with randomly sampled values based on the input range.

76.7.3 noah32_set_pedecvars (Source File: noah32_set_pedecvars.F90)

REVISION HISTORY:

```
06 Feb 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah32_set_pedecvars(DEC_State, Feas_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_soilsMod,  only : LIS_soils
use LIS_logMod,       only : LIS_logunit,LIS_verify
use noah32_lsmMod, only : noah32_struc
use noah32_peMod,  only : noah32_pe_struc

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: DEC_State
type(ESMF_State)       :: Feas_State
```

DESCRIPTION:

This routine assigns the decision space to Noah's model variables.

76.7.4 noah32_setupobspred_ARMobs (Source File: noah32_setupobspred_ARMobs.F90)

REVISION HISTORY:

```
16 Jul 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah32_setupobspred_ARMobs(OBSPred)
```

USES:

```
use ESMF
use LIS_coreMod,      only : LIS_vecTile
use LIS_logMod,       only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: OBSPred
```

DESCRIPTION:

This routine creates an entry in the Obs pred object used for parameter estimation

---

76.7.5 noah32_setupobspred_template (Source File: noah32_setupobspred_template.F90)

REVISION HISTORY:

```
16 Jul 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah32_setupobspred_template(OBSPred)
```

USES:

```
use ESMF
use LIS_coreMod,      only : LIS_vecTile
use LIS_logMod,       only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: OBSPred
```

DESCRIPTION:

This routine creates an entry in the Obs pred object used for parameter estimation

---

76.7.6 noah32_getrunoffs (Source File: noah32_getrunoffs.F90)

REVISION HISTORY:

```
6 May 2011: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah32_getrunoffs(n)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_masterproc
use LIS_routingMod, only : LIS_runoff_state
use LIS_logMod,     only : LIS_verify
use LIS_historyMod, only : LIS_gather_gridded_output
use noah32_lsmMod, only : noah32_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

76.7.7 noah32_sfc2cmem3 (Source File: noah32_sfc2cmem3.F90)

REVISION HISTORY:

```
37 Mar 2009: Sujay Kumar; Initial Code
2  May 2011: Yudong Tian; Modified for Noah 3.2
```

INTERFACE:

```
subroutine noah32_sfc2cmem3(n, sfcState)
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc
use LIS_logMod,    only : LIS_verify
use LIS_constantsMod,  only : LIS_CONST_RHOFW

use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: sfcState

FUNCTIONS
```

DESCRIPTION:

This subroutine assigns the noah32 specific surface variables to CMEM3.

76.7.8 noah32_sfc2crtm (Source File: noah32_sfc2crtm.F90)

REVISION HISTORY:

```
 37 Mar 2009: Sujay Kumar; Initial Code
 2  May 2011: Yudong Tian; Modified for Noah 3.2
```

INTERFACE:

```
subroutine noah32_sfc2crtm(n, sfcState)
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc
use LIS_logMod,    only : LIS_verify
use LIS_constantsMod,  only : LIS_CONST_RHOFW

use noah32_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: sfcState

FUNCTIONS
```

DESCRIPTION:

This subroutine assigns the noah32 specific surface variables to CRTM.

76.7.9 noah33_setwrfexport.F90 (Source File: noah33_setwrfexport.F90)

Defines the export states from Noah to WRF in coupled mode

REVISION HISTORY:

```
02 Dec 2003; Sujay Kumar, Initial Version
17 Nov 2008; Sujay Kumar, Modified for the ESMF coupled version
```

INTERFACE:

```
subroutine noah33_setwrfexport(n)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_historyMod, only : LIS_patch2tile
use LIS_logMod
use LISWRFGridCompMod, only : LISWRF_export
use noah33_lsmMod

implicit none
```

ARGUMENTS:

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

76.7.10 noah33_wrf_f2t (Source File: noah33_wrf_f2t.F90)

REVISION HISTORY:

```
15 Oct 2002: Sujay Kumar; Initial Code
```

INTERFACE:

```
subroutine noah33_wrf_f2t(n)
```

USES:

```
use ESMF
use LIS_coreMod ,      only : LIS_rc,LIS_surface
use LIS_metforcingMod, only : LIS_FORC_State
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use noah33_lsmMod,        only : noah33_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the Noah model tiles in a coupled mode to WRF
The arguments are:

n index of the nest

76.8 Fortran: Module Interface noah33_dasnow_Mod (Source File: noah33_dasnow_Mod.F90

REVISION HISTORY:

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: noah33_dasnow_init
```

PUBLIC TYPES:

76.8.1 noah33_dasnow_init (Source File: noah33_dasnow_Mod.F90)

INTERFACE:

```
subroutine noah33_dasnow_init()
```

USES:

DESCRIPTION:

76.8.2 noah33_descale_snow (Source File: noah33_descale_snow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah33_descale_snow(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.8.3 noah33_getswfpred (Source File: noah33_getscfpred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
19 Mar 2012: Yuqiong Liu, modified for SCF assimilation
```

INTERFACE:

```
subroutine noah33_getscfpred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_surface
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real                   :: obs_pred(LIS_rc%ngrid(n),LIS_rc%nensem(n))
```

76.8.4 noah33_getsnowpred_PMWsnow (Source File: noah33_getsnowpred_PMWsnow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
Mar 14 2014, Yuqiong Liu: Modified to asismilate SWE or snow depth
```

INTERFACE:

```
subroutine noah33_getsnowpred_PMWsnow(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_surface
use noah33_lsmMod
use PMW_snow_Mod, only : PMW_snow_struc
use LIS_logMod,   only: LIS_logunit

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real                   :: obs_pred(LIS_rc%ngrid(n),LIS_rc%nensem(n))
```

76.8.5 noah33_getsnowvars (Source File: noah33_getsnowvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah33_getsnowvars(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.8.6 noah33_getsnwdpred (Source File: noah33_getsnwdpred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah33_getsnwdpred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_surface
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real                   :: obs_pred(LIS_rc%ngrid(n),LIS_rc%nensem(n))
```

76.8.7 noah33_getswepred (Source File: noah33_getswepred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah33_getswepred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_surface
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real                   :: obs_pred(LIS_rc%ngrid(n),LIS_rc%nensem(n))
```

76.8.8 noah33_map_snow (Source File: noah33_map_snow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah33_map_snow(n,OBS_State,LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_surface
use LIS_constantsMod, only  : LIS_CONST_TKFRZ
use LIS_logMod,   only  : LIS_logunit, LIS_verify
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
type(ESMF_State)         :: LSM_Incr_State
```

DESCRIPTION:

This subroutine directly maps the observation state to the corresponding variables in the LSM state for SCA data assimilation.
The arguments are:

n index of the nest

OBS_State ESMF State for observations

LSM_State ESMF State for LSM state variables

76.8.9 noah33_map_snow_DI (Source File: noah33_map_snow_DI.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
12 Jun 2013, May 31 2014: Yuqiong Liu; Modified to combine different DI approaches,
  Please refer to Rodell & House (2004, JHM), De Lannoy (2012, WRR) and Liu et al. (2013, AWR)
```

INTERFACE:

```
subroutine noah33_map_snow_DI(n,OBS_State,LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_constantsMod, only  : LIS_CONST_TKFRZ
use LIS_logMod,   only  : LIS_logunit, LIS_verify, &
       LIS_getNextUnitNumber, LIS_releaseUnitNumber
use noah33_lsmMod
use ANSASCFsnow_Mod, only : ANSASCFsnow_struc
 use LIS_topoMod, only : LIS_topo

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
type(ESMF_State)         :: LSM_Incr_State
```

DESCRIPTION:

This subroutine directly maps the observation state to the corresponding variables in the LSM state for SCA data assimilation.
The arguments are:

n index of the nest

OBS_State ESMF State for observations

LSM_State ESMF State for LSM state variables

76.8.10 noah33_map_swe (Source File: noah33_map_swe.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah33_map_swe(n,OBS_State,LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_surface
use LIS_constantsMod, only  : LIS_CONST_TKFRZ
use LIS_logMod,   only  : LIS_logunit, LIS_verify
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
type(ESMF_State)         :: LSM_State
```

DESCRIPTION:

This subroutine directly maps the observation state to the corresponding variables in the LSM state for SWE data assimilation.
The arguments are:

n index of the nest

OBS_State ESMF State for observations

LSM_State ESMF State for LSM state variables

76.8.11 noah33_qc_PMWsnowobs (Source File: noah33_qc_PMWsnowobs.F90)

REVISION HISTORY:

```
25Feb2008: Sujay Kumar: Initial Specification
13Mar2014: Yuqiong Liu, modified for qc PMW SWE and snow depth obs
```

INTERFACE:

```
subroutine noah33_qc_PMWsnowobs(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_logMod,  only : LIS_verify
use LIS_constantsMod, only : LIS_CONST_TKFRZ
use noah33_lsmMod
use PMW_snow_Mod, only : PMW_snow_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

This subroutine performs any model-based QC of the observation prior to data assimilation. Here the snow observations are flagged when LSM indicates that (1) rain is falling (2) ground is fully or partially covered with snow.
The arguments are:

n index of the nest

OBS_State ESMF state container for observations

76.8.12 noah33_qc_scfobs (Source File: noah33_qc_scfobs.F90)

REVISION HISTORY:

```
25Feb2008: Sujay Kumar: Initial Specification
```

INTERFACE:

```
subroutine noah33_qc_scfobs(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_logMod,  only : LIS_verify
use LIS_constantsMod, only : LIS_CONST_TKFRZ
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

This subroutine performs any model-based QC of the observation prior to data assimilation. Here the snow observations are flagged when LSM indicates that (1) rain is falling (2) ground is fully or partially covered with snow.
The arguments are:

n index of the nest

OBS_State ESMF state container for observations

76.8.13 noah33_qc_snowobs (Source File: noah33_qc_snowobs.F90)

REVISION HISTORY:

```
25Feb2008: Sujay Kumar: Initial Specification
```

INTERFACE:

```
subroutine noah33_qc_snowobs(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_logMod,  only : LIS_verify
use LIS_constantsMod, only : LIS_CONST_TKFRZ
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

This subroutine performs any model-based QC of the observation prior to data assimilation. Here the snow observations are flagged when LSM indicates that (1) rain is falling (2) ground is fully or partially covered with snow.
The arguments are:

n index of the nest

OBS_State ESMF state container for observations

76.8.14 noah33_qcsnow (Source File: noah33_qcsnow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah33_qcsnow(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah33_lsmMod
use LIS_logMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.8.15 noah33_scale_snow (Source File: noah33_scale_snow.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah33_scale_snow(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.8.16 noah33_setsnowvars (Source File: noah33_setsnowvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah33_setsnowvars(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah33_lsmMod
use LIS_logMod, only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine assigns the snow progognostic variables to noah's model space.

76.8.17 noah33_transform_snow (Source File: noah33_transform_snow.F90)

REVISION HISTORY:

```
25Jun2006: Sujay Kumar: Initial Specification
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah33_transform_snow(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use noah33_lsmMod
```

76.8.18 noah33_updatesnowvars (Source File: noah33_updatesnowvars.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah33_updatesnowvars(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah33_lsmMod
use LIS_logMod,   only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

LSM_Incr_State ESMF State container for LSM state increments

76.8.19 noah33_updatesnowvars_scfda (Source File: noah33_updatesnowvars_scfda.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
may 30 2014: Yuqiong Liu; adpated for SCF EnKF assimilation
```

INTERFACE:

```
subroutine noah33_updatesnowvars_scfda(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_surface
use noah33_lsmMod
use LIS_logMod,   only : LIS_logunit, LIS_verify
!use enkf_Mod,  only: enkf_struc
use LIS_DAobservationsMod, only : LIS_OBS_State
use ANSASCFsnow_Mod, only: ANSASCFsnow_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Returns the snow related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

LSM_Incr_State ESMF State container for LSM state increments

76.8.20 noah33_wrtswe (Source File: noah33_wrtswe.F90)

REVISION HISTORY:

```
13Nov2007: Sujay Kumar; Initial Specification
 02 Mar 2010: Sujay Kumar; Modified for Noah 3.1
```

INTERFACE:

```
subroutine noah33_wrtswe(ftn, n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: ftn
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the tskin related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.9 Fortran: Module Interface noah33_dasoilm_Mod (Source File: noah33_dasoilm_Mod.F9

REVISION HISTORY:

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: noah33_dasoilm_init
```

PUBLIC TYPES:

76.9.1 noah33_dasoilm_init (Source File: noah33_dasoilm_Mod.F90)

INTERFACE:

```
subroutine noah33_dasoilm_init()
```

USES:

DESCRIPTION:

76.9.2 noah33_descale_soilm (Source File: noah33_descale_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah33_descale_soilm(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

Descales soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.9.3 noah33_getLbandTbPred (Source File: noah33_getLbandTbPred.F90)

REVISION HISTORY:

```
13 Sept 2012: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah33_getLbandTbPred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_surface
use LIS_RTMMod,  only : LIS_forwardState, LIS_RTM_run
use LIS_logMod,  only : LIS_verify

use noah33_lsmMod
```

76.9.4 noah33_getsmpred (Source File: noah33_getsmpred.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah33_getsmpred(n, obs_pred)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc,LIS_surface
use noah33_lsmMod
```

76.9.5 noah33_getsoilm (Source File: noah33_getsoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah33_getsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.9.6 noah33_qc_soilmobs (Source File: noah33_qc_soilmobs.F90)

REVISION HISTORY:

```
25Feb2008: Sujay Kumar: Initial Specification
```

INTERFACE:

```
subroutine noah33_qc_soilmobs(n,OBS_State)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_logMod,  only : LIS_verify
use LIS_constantsMod, only : LIS_CONST_TKFRZ
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: n
type(ESMF_State)         :: OBS_State
```

DESCRIPTION:

This subroutine performs any model-based QC of the observation prior to data assimilation. Here the soil moisture observations are flagged when LSM indicates that (1) rain is falling (2) soil is frozen or (3) ground is fully or partially covered with snow.
The arguments are:

n index of the nest

OBS_State ESMF state container for observations

76.9.7 noah33_qcsoilm (Source File: noah33_qcsoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah33_qcsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.9.8 noah33_scale_soilm (Source File: noah33_scale_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah33_scale_soilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Scales soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.9.9 noah33_setsoilm (Source File: noah33_setsoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah33_setsoilm(n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify, LIS_logunit
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

This routine assigns the soil moisture prognostic variables to noah's model space.

76.9.10 noah33_updatesoilm (Source File: noah33_updatesoilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah33_updatesoilm(n, LSM_State, LSM_Incr_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_logMod,  only  : LIS_verify
use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
type(ESMF_State)       :: LSM_Incr_State
```

DESCRIPTION:

This routine assigns the soil moisture prognostic variables to noah's model space.

76.9.11 noah33_write_soilm (Source File: noah33_write_soilm.F90)

REVISION HISTORY:

```
27Feb2005: Sujay Kumar; Initial Specification
25Jun2006: Sujay Kumar: Updated for the ESMF design
```

INTERFACE:

```
subroutine noah33_write_soilm(ftn,n, LSM_State)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use noah33_lsmMod
use LIS_historyMod, only : LIS_writevar_restart
implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: ftn
integer, intent(in)    :: n
type(ESMF_State)       :: LSM_State
```

DESCRIPTION:

Returns the soilmoisture related state prognostic variables for data assimilation
The arguments are:

n index of the nest

LSM_State ESMF State container for LSM state variables

76.9.12 noah33_getirrigationstates (Source File: noah33_getirrigationstates.F90)

INTERFACE:

```
subroutine noah33_getirrigationstates(n,irrigState)
```

USES:

```
use ESMF
use LIS_coreMod
use LIS_logMod
use noah33_lsmMod
```

DESCRIPTION:

Calculate water requirement and apply the amount to precipitation.
Irrigate when root zone soil moisture falls below 50 the field capacity (reference soil moiture) at 6 am LST. The root zone is actual maximum root depth rather than NOAH root zone. Method of irrigation is by precipitation between 6-10 am LST.
Irrigation amount is scaled to grid total crop fraction when intensity is less than the fraction. Irrigation is expanded to non-crop, non-forest, non-baresoil/urban tiles if intensity exceeds grid total crop fraction. In latter case, scaled irrigation is applied to grassland first, then further applied over the rest of tiles equally if the intensity exceeds grassland fraction as well.
Optionally efficiency correction is applied to account for field loss.
Optionally outputs amount of water put into the system to a text file.
This version includes modifications to irr4 as follows: 1) Use location specific growing season threshold (402) Allow irrigation in non-crop/non-forest tiles when irrigation intensity exceeds total crop fraction
REVISION HISTORY:
Aug 2008: Hiroko Kato; Initial code Nov 2012: Sujay Kumar, Incorporated into LIS

76.9.13 noah33_coldstart (Source File: noah33_coldstart.F90)

REVISION HISTORY:

```
28 Apr 2002: Sujay Kumar; Initial version
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
 9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
```

INTERFACE:

```
subroutine noah33_coldstart(mtype)
```

USES:

```
use LIS_coreMod,      only : LIS_rc
use LIS_logMod,       only : LIS_logunit
use LIS_timeMgrMod,   only : LIS_date2time
use noah33_lsmMod
```

DESCRIPTION:

This routine initializes the Noah3.3 state variables with some predefined values uniformly for the entire domain. These initial values will be overwritten by the values read from the supplied Noah3.3 model restart file.

76.9.14 noah33_dynsetup (Source File: noah33_dynsetup.F90)

REVISION HISTORY:

```
 28 Jan 2002: Sujay Kumar, Initial Specification
  8 May 2009: Sujay Kumar; additions for Noah3.1
 27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
  7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
  9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
 14 Jan 2014: David Mocko, reconfirmed Noah3.3 in LIS7.0
```

INTERFACE:

```
subroutine noah33_dynsetup(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, LIS_surface
use LIS_snowMod, only : LIS_snow_struc
use LIS_vegDataMod,    only : LIS_gfrac, LIS_lai, LIS_roughness
use LIS_emissMod,      only : LIS_emiss
use LIS_albedoMod,     only : LIS_alb
use noah33_lsmMod, only : noah33_struc
use LIS_timeMgrMod, only        : LIS_date2time,LIS_tick
use module_sf_noah33lsm, only : month_d, calc_localtime
```

DESCRIPTION:

This routine sets up the time-dependent variables in Noah3.3

76.9.15 noah33_f2t (Source File: noah33_f2t.F90)

REVISION HISTORY:

```
 15 Oct 2002: Sujay Kumar; Initial Code
  8 May 2009: Sujay Kumar; additions for Noah3.1
 27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
  7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
  9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
```

INTERFACE:

```
subroutine noah33_f2t(n)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc, LIS_surface
use LIS_FORC_AttributesMod
use LIS_metforcingMod, only : LIS_FORC_State
use LIS_logMod,         only : LIS_verify
use noah33_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the Noah3.3 model tiles.
The arguments are:

n index of the nest

76.9.16 noah33_finalize (Source File: noah33_finalize.F90)

REVISION HISTORY:

```
28 Apr 2002: Sujay Kumar; Initial code
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
 9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
```

INTERFACE:

```
subroutine noah33_finalize()
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use noah33_lsmMod
```

DESCRIPTION:

This routine cleans up the allocated memory structures in Noah3.3

76.10 Fortran: Module Interface noah33_lsmMod (Source File: noah33_lsmMod.F90)

This module provides the definition of derived data type used to control the operation of Noah3.3 LSM. It also provides the entry method for the initialization of Noah3.3-specific variables. The derived data type `noah3.3_struc` includes the variables that specify the runtime options and other control variables as described below:

rfile name of the noah3.3 restart file

vfile name of the static vegetation parameter table

sfile name of the soil parameter table

gfile name of the general parameter table

useptf whether PTFs should be used to map soil properties

count variable to keep track of the number of timesteps before an output

soilscheme type of soil mapping scheme usd (1-zobler,2-statsgo)

nslay number of soil layers

nvegp number of static vegetation parameters in the table

nstxts number of soil texture classes in the classification scheme

varid NETCDF ids for variables (used for netcdf output)

inittemp initial soil temperatures for a cold start run

initsm initial total soil moisture for a cold start run

initsmliq initial liquid soil moisture for a cold start run

rstInterval restart writing interval

zh reference height of T and q forcing

zm reference height of u and v forcing

forcing_ch flag indicating whether to use aerodynamic conductance field from forcing

lyrthk thickness of soil layers

noah Noah3.3 LSM specific variables

REVISION HISTORY:

```
Apr 2003; Sujay Kumar, Initial Code
  8 May 2009: Sujay Kumar; additions for Noah3.1
 27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
  7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
  9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
```

USES:

```
use noah33_module

implicit none

PRIVATE
-----------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: noah33_lsm_ini
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: noah33_struc
```

76.10.1 noah33_lsm_ini (Source File: noah33_lsmMod.F90)

INTERFACE:

```
subroutine noah33_lsm_ini()
```

USES:

```
use LIS_coreMod,      only : LIS_rc
use LIS_surfaceModelDataMod, only : LIS_sfmodel_struc
use LIS_timeMgrMod,   only : LIS_clock, LIS_calendar, &
     LIS_update_timestep, LIS_registerAlarm
use LIS_logMod,       only : LIS_verify
```

DESCRIPTION:

This routine creates the datatypes and allocates memory for Noah3.3-specific variables. It also invokes the routine to read the runtime specific options for Noah3.3 from the configuration file.
The routines invoked are:

noah33_readcrd (76.11.1)
reads the runtime options for Noah3.3 LSM

76.10.2 noah33_main (Source File: noah33_main.F90)

REVISION HISTORY:

```
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
20 Jan 2011: David Mocko, fixes to output diagnostics for energy
                          & water balance and fix to conversion of
                          U,V forcing height to same level as T,q
 9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
14 Jan 2014: David Mocko, reconfirmed Noah3.3 in LIS7.0
```

INTERFACE:

```
subroutine noah33_main(n)
```

USES:

```
$ use omp_lib
 use LIS_coreMod
 use LIS_timeMgrMod,    only : LIS_isAlarmRinging
 use LIS_albedoMod,     only : LIS_alb
 use LIS_constantsMod,  only : LIS_CONST_RHOFW, LIS_CONST_TKFRZ, &
                               LIS_CONST_LATVAP
 use LIS_logMod,        only : LIS_logunit, LIS_endrun
 use LIS_histDataMod
 use module_sf_noah33lsm, only : SFLX, DQSDT, SFCDIF_OFF, CALHUM, SNFRAC
 use module_sfcdif_wrf_33, only: SFCDIF_MYJ
 use module_sf_noah33lsm_glacial, only : SFLX_glacial
 use noah33_lsmMod
 use LIS_tbotAdjustMod, only: LIS_tbotTimeUtil,LIS_updateTbot

 implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This is the entry point for calling the Noah3.3 LSM physics. This routine calls the `SFLX` routine that performs the land surface computations, to solve for water and energy equations. For documentation of the `SFLX` and other Noah3.3 routines, please see the Noah3.3 webpage at NCAR: http://www.ral.ucar.edu/research/land/technology/lsm.ph
The arguments are:

n index of the nest

76.11 Fortran: Module Interface noah33_module (Source File: noah33_module.F90)

The code in this file provides a description of the data structure containing the Noah3.3 1-d variables. The variables specified in the data structure include:

soiltype soil type (integer index)

slopetype slope type (integer index)

nroot Number of root layers, depends on the vegetation type

rsmin Minimum canopy resistance (s/m)

rgl Solar radiation term of canopy resistance function

hs vapor pressure deficit term of canopy resistance function

snup threshold snow depth (in water equivalent m) that implies 100% snow cover

z0 roughness length

lai leaf area index

shdfac greenness vegetation fraction

shdmax maximum greenness vegetation fraction

shdmin minimum greenness vegetation fraction

smcmax maximum soil moisture content (porosity)

slope slope coefficient for baseflow

psisat saturated matric potential

dksat saturated hydraulic conductivity

bexp b parameter value

quartz quartz fraction

smcwlt wilting point (volumetric)

smcref reference soil moisture (onset of soil moisture stress in transpiration, volumetric)

dwsat saturated soil diffusivity

mxsnalb maximum albedo expected over deep snow

tempbot bottom boundary temperature

t1 skin temperature (K)

cmc canopy water content

snowh actual snow depth (m)

sneqv snow water equivalent (m)

stc soil temperature for different layers

smc soil moisture for different layers

sh2o liquid-only soil moisture for different layers

ch heat/moisture exchange coefficient

cm momentum exchange coefficient

vegt vegetation type of tile

tair_agl_min minimum 2 meter AGL temperature

rhmin minimum 2 meter relative humidity

relsmc Volumetric relative soil moisture (m^3/m^3) (smc - smcwlt) / (porosity - smcwlt)

REVISION HISTORY:

```
28 Apr 2002: K. Arsenault added Noah2.5 LSM code to LDAS.
14 Nov 2002: Sujay Kumar Optimized version for LIS
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
 9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
```

76.11.1 noah33_readcrd (Source File: noah33_readcrd.F90)

REVISION HISTORY:

```
14 Oct 2003; Sujay Kumar, Initial Code
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
 9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
```

INTERFACE:

```
subroutine noah33_readcrd()
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_timeMgrMod, only : LIS_parseTimeString
use LIS_logMod,  only : LIS_logunit, LIS_verify
use noah33_lsmMod, only : noah33_struc
```

DESCRIPTION:

This routine reads the options specific to Noah3.3 LSM from the LIS configuration file.

76.11.2 noah33_readrst (Source File: noah33_readrst.F90)

REVISION HISTORY:

```
 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 Noah2.5 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.
  8 May 2009: Sujay Kumar; additions for Noah3.1
 27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
  7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
  9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
 10 Jun 2012: Sujay Kumar, added support for netcdf formats
```

INTERFACE:

```
subroutine noah33_readrst()
```

USES:

```
  use LIS_coreMod,    only : LIS_rc, LIS_masterproc
  use LIS_historyMod, only : LIS_readvar_restart
  use LIS_logMod,     only : LIS_logunit, LIS_endrun, &
       LIS_getNextUnitNumber, LIS_releaseUnitNumber,&
       LIS_verify
  use module_sfcdif_wrf_33, only: MYJSFCINIT
  use noah33_lsmMod
  use LIS_tbotAdjustMod, only: LIS_readTmnUpdateRestart
#if (defined USE_NETCDF3 || defined USE_NETCDF4)
  use netcdf
#endif
```

DESCRIPTION:

This program reads restart files for Noah3.3. This includes all relevant water/energy storages and tile information. The following is the list of variables specified in the Noah3.3 restart file:

```
nc,nr,ntiles    - grid and tile space dimensions
t1            - Noah3.3 skin temperature
cmc           - Noah3.3 canopy moisture storage
snowh         - Noah3.3 snow depth
sneqv         - Noah3.3 snow water equivalent
stc           - Noah3.3 soil temperature (for each layer)
smc           - Noah3.3 soil moisture (for each layer)
sh2o          - Noah3.3 liquid only soil moisture (for each layer)
ch            - Noah3.3 heat/moisture exchange coefficient
cm            - Noah3.3 momentum exchange coefficient
```

The routines invoked are:

LIS_readvar_restart (3.17.6)
reads a variable from the restart file

noah33_coldstart (76.9.13)
initializes the Noah3.3 state variables

76.11.3 noah33_reset (Source File: noah33_reset.F90)

REVISION HISTORY:

```
28 Apr 2002: Sujay Kumar; Initial version
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
 9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
```

INTERFACE:

```
subroutine noah33_reset()
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use noah33_lsmMod
use LIS_surfaceModelDataMod, only : LIS_sfmodel_struc
use LIS_timeMgrMod,   only : LIS_clock, LIS_calendar, &
     LIS_update_timestep
use LIS_logMod,       only : LIS_verify, LIS_logunit
```

DESCRIPTION:

This routine is the entry point to set up the parameters required for Noah3.3 LSM. These include the soils, greenness, albedo, bottom temperature and the initialization of state variables in Noah3.3.
The routines invoked are:

noah33_resetvegparms (76.11.9)
initializes the vegetation-related parameters in Noah3.3

noah33_resettbot (76.11.6)
initializes the bottom temperature fields

noah33_resetsoils (**??**)
initializes the soil parameters

76.11.4 noah33_setsoils (Source File: noah33_setsoils.F90)

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault;  Added Noah2.5 LSM, Initial Code
13 Oct 2003: Sujay Kumar; Domain independent modifications
15 Sep 2005: Matthew Garcia; update for ARMS project (Windows capability)
04 Oct 2005: Matthew Garcia; additions for Noah Unified
 9 Jul 2008: Chris Franks/AFWA; Save wilting point for calculation of relative soil moisture
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
 9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
20 Jan 2011: David Mocko, added slope type
```

INTERFACE:

```
subroutine noah33_setsoils(mtype)
```

USES:

```
use LIS_coreMod,   only : LIS_rc, LIS_domain, LIS_surface
use LIS_soilsMod,  only : LIS_soils
use LIS_logMod,    only : LIS_logunit, LIS_endrun, &
                          LIS_getNextUnitNumber, LIS_releaseUnitNumber
use LIS_fileIOMod, only : LIS_read_param
use noah33_lsmMod
```

DESCRIPTION:

This subroutine sets the soil parameters in Noah3.3. Noah3.3 uses a lookup table based on the input soil texture to derive these parameters.
The routines invoked are:

LIS_mapSoilType (3.22.15)
method to derive the soil texture type from the sand, silt, and clay fractions.

76.11.5 noah33_settbot (Source File: noah33_settbot.F90)

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault;  Added Noah2.5 LSM, Initial Code
13 Oct 2003: Sujay Kumar; Domain independent modifications
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
 9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
23 May 2014: David Mocko, changes for Noah3.3 in LIS7.0
```

INTERFACE:

```
subroutine noah33_settbot(mtype)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_surface
use LIS_logMod,     only : LIS_logunit
use LIS_tbotAdjustMod, only : LIS_initTmnUpdateTile
use LIS_fileIOMod,  only : LIS_read_param
use noah33_lsmMod
```

DESCRIPTION:

This subroutine retrieves bottom boundary temperature for Noah3.3 LSM. Currently a static data is being used. If a climatology is available, it can be used by setting the appropriate frequency of the climatology
The routines invoked are:

LIS_read_param (3.14.4)
retrieves the bottom temperature data

76.11.6 noah33_resettbot (Source File: noah33_settbot.F90)

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault;  Added Noah2.5 LSM, Initial Code
13 Oct 2003: Sujay Kumar; Domain independent modifications
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
 9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
```

INTERFACE:

```
subroutine noah33_resettbot(mtype)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_surface
use LIS_logMod,     only : LIS_logunit
use LIS_fileIOMod,  only : LIS_read_param
use noah33_lsmMod
```

DESCRIPTION:

This subroutine retrieves bottom boundary temperature for Noah3.3 LSM. Currently a static data is being used. If a climatology is available, it can be used by resetting the appropriate frequency of the climatology
The routines invoked are:

LIS_read_param (3.14.4)
retrieves the bottom temperature data

76.11.7 noah33_setup (Source File: noah33_setup.F90)

REVISION HISTORY:

```
28 Apr 2002: Sujay Kumar; Initial version
 8 May 2009: Sujay Kumar; additions for Noah3.1
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
 9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
```

INTERFACE:

```
subroutine noah33_setup()
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use noah33_lsmMod
```

DESCRIPTION:

This routine is the entry point to set up the parameters required for Noah3.3 LSM. These include the soils, greenness, albedo, bottom temperature and the initialization of state variables in Noah3.3.
The routines invoked are:

noah33_setvegparms (76.11.8)
initializes the vegetation-related parameters in Noah3.3

noah33_settbot (76.11.5)
initializes the bottom temperature fields

noah33_setsoils (76.11.4)
initializes the soil parameters

76.11.8 noah33_setvegparms (Source File: noah33_setvegparms.F90)

REVISION HISTORY:

```
07 May 2009: Sujay Kumar; Initial Implementation
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
20 Jan 2011: David Mocko, added max/min greenness
 9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
14 Jan 2014: David Mocko, reconfirmed Noah3.3 in LIS7.0
```

INTERFACE:

```
subroutine noah33_setvegparms(mtype)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_surface
use LIS_vegDataMod,    only : LIS_gfrac, LIS_read_shdmax, &
     LIS_read_shdmin, LIS_lai, LIS_roughness
use LIS_emissMod,      only : LIS_emiss
use LIS_albedoMod,     only : LIS_alb
use LIS_logMod,  only : LIS_logunit, &
                        LIS_getNextUnitNumber, LIS_releaseUnitNumber
use LIS_timeMgrMod, only        : LIS_date2time,LIS_tick
use module_sf_noah33lsm, only : month_d, calc_localtime
use noah33_lsmMod
```

DESCRIPTION:

This subroutine retrieves Noah3.3 vegetation parameters. The current implementation uses a table-based lookup based on vegetation classes to initialize the following parameters

```
 nroot  - number of root zones
 rsmin  - minimum canopy resistance
 rgl    - solar radiation term in canopy resistance function
 hs     - vapor pressure deficit term
 snup   - threshold snow depth
 shdfac - greenness vegetation fraction
 shdmax - maximum greenness vegetation fraction during year
 shdmin - minimum greenness vegetation fraction during year
The following variables are also a function of shdfac:
 albedo - background (snow-free) surface albedo
 embrd  - background (snow-free) emissivity
 z0     - roughness length
 lai    - leaf area index
```

76.11.9 noah33_resetvegparms (Source File: noah33_setvegparms.F90)

REVISION HISTORY:

```
07 May 2009: Sujay Kumar; Initial Implementation
27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
 7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
20 Jan 2011: David Mocko, added max/min greenness
 9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
```

INTERFACE:

```
subroutine noah33_resetvegparms(mtype)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_surface
use LIS_vegDataMod,    only : LIS_gfrac, LIS_read_shdmax, &
     LIS_read_shdmin, LIS_lai, LIS_roughness
use LIS_emissMod,      only : LIS_emiss
use LIS_albedoMod,     only : LIS_alb
use LIS_logMod,  only : LIS_logunit
use LIS_timeMgrMod, only        : LIS_date2time,LIS_tick
use module_sf_noah33lsm, only : month_d, calc_localtime
use noah33_lsmMod
```

DESCRIPTION:

This subroutine resets Noah3.3 vegetation parameters. The current implementation uses a table-based lookup based on vegetation classes to initialize the following parameters

```
 nroot  - number of root zones
 rsmin  - minimum canopy resistance
 rgl    - solar radiation term in canopy resistance function
 hs     - vapor pressure deficit term
 snup   - threshold snow depth
 shdfac - greenness vegetation fraction
 shdmax - maximum greenness vegetation fraction during year
 shdmin - minimum greenness vegetation fraction during year
The following variables are also a function of shdfac:
 albedo - background (snow-free) surface albedo
 embrd  - background (snow-free) emissivity
 z0     - roughness length
 lai    - leaf area index
```

76.11.10 noah33_writerst (Source File: noah33_writerst.F90)

REVISION HISTORY:

```
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 Noah2.5 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
     8 May 2009: Sujay Kumar; additions for Noah3.1
    27 Oct 2010: David Mocko, changes for Noah3.1 in LIS6.1
     7 Nov 2010: David Mocko, changes for Noah3.2 in LIS6.1
     9 Sep 2011: David Mocko, changes for Noah3.3 in LIS6.1
    10 Jun 2012: Sujay Kumar, added support for netcdf formats
```

INTERFACE:

```
 subroutine noah33_writerst(n)
```

USES:

```
   use LIS_coreMod,   only : LIS_rc, LIS_masterproc
   use LIS_timeMgrMod, only : LIS_isAlarmRinging
   use LIS_logMod,    only : LIS_logunit, LIS_getNextUnitNumber, &
        LIS_releaseUnitNumber, LIS_verify
   use LIS_fileIOMod, only : LIS_create_output_directory, &
                             LIS_create_restart_filename
   use noah33_lsmMod
 #if (defined USE_NETCDF3 || defined USE_NETCDF4)
   use netcdf
 #endif

   implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This program writes restart files for Noah3.3. This includes all relevant water/energy storage and tile information.
The routines invoked are:

LIS_create_output_directory (3.14.6)
creates a timestamped directory for the restart files

LIS_create_restart_filename (3.14.2)
generates a timestamped restart filename

noah33_dump_restart (76.11.11)
writes the Noah3.3 variables into the restart file

76.11.11 noah33_dump_restart (Source File: noah33_writerst.F90)

REVISION HISTORY:

```
    24 Aug 2004: James Geiger, Initial Specification
```

INTERFACE:

```
subroutine noah33_dump_restart(n, ftn, wformat)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_historyMod
use LIS_logMod,     only : LIS_logunit
use noah33_lsmMod
use LIS_tbotAdjustMod, only : LIS_writeTmnUpdateRestart

implicit none

integer, intent(in) :: ftn
integer, intent(in) :: n
character(len=*), intent(in) :: wformat
```

DESCRIPTION:

This routine gathers the necessary restart variables and performs the actual write statements to create the restart files.
The arguments are:

n index of the nest

ftn unit number for the restart file

wformat restart file format (binary/netcdf)

The following is the list of variables written in the Noah3.3 restart file:

```
nc,nr,ntiles    - grid and tile space dimensions
t1            - Noah3.3 skin temperature
cmc           - Noah3.3 canopy moisture storage
snowh         - Noah3.3 snow depth
sneqv         - Noah3.3 snow water equivalent
stc           - Noah3.3 soil temperature (for each layer)
smc           - Noah3.3 soil moisture (for each layer)
sh2o          - Noah3.3 liquid only soil moisture (for each layer)
ch            - Noah3.3 heat/moisture exchange coefficient
cm            - Noah3.3 momentum exchange coefficient
z0            - Noah3.3 roughness length
emiss         - Noah3.3 emissivity
albedo        - Noah3.3 albedo
snotime1      - Noah3.3 snow timing variable
q1            - Noah3.3 mixing ratio
```

The routines invoked are:

LIS_writeGlobalHeader_restart (3.17.22)
writes the global header information

LIS_writeHeader_restart (3.17.23)
writes the header information for a variable

LIS_closeHeader_restart (3.17.24)
close the header

LIS_writevar_restart (3.17.3)
writes a variable to the restart file

76.12 Fortran: Module Interface noah33_peMod (Source File: noah33_peMod.F90)

This module contains the definitions of the Noah model parameters used in parameter estimation. The data structure is used to expose the LSM parameters to be used in opt/ue.

REVISION HISTORY:

```
12 Jan 2012; Sujay Kumar, Initial Code
```

USES:

```
use ESMF
use LIS_numerRecipesMod, only : LIS_rand_func

implicit none

PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
public :: noah33_setup_pedecvars
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
public :: noah33_pe_struc
```

76.12.1 noah33_setup_pedecvars (Source File: noah33_peMod.F90)

REVISION HISTORY:

```
06 Feb 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah33_setup_pedecvars(DEC_State, Feas_State)
```

USES:

```
use ESMF
use LIS_coreMod,       only : LIS_rc, LIS_config,LIS_vecPatch, LIS_surface, LIS_localPET
use LIS_logMod,        only : LIS_logunit, LIS_verify
use noah33_lsmMod,     only : noah33_struc

implicit none
```

ARGUMENTS:

```
character*100               :: decSpaceAttribsFile
type(ESMF_State)            :: DEC_State
type(ESMF_State)            :: Feas_State
```

DESCRIPTION:

This routine determines the list of parameters to be used in parameter estimation, initializes them, and updates the LIS decision space.

76.12.2 noah33_set_pedecvars (Source File: noah33_set_pedecvars.F90)

REVISION HISTORY:

```
06 Feb 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah33_set_pedecvars(DEC_State, Feas_State)
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc, LIS_surface
use LIS_logMod,    only : LIS_logunit,LIS_verify
use noah33_lsmMod, only : noah33_struc
use noah33_peMod,  only : noah33_pe_struc

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: DEC_State
type(ESMF_State)       :: Feas_State
```

DESCRIPTION:

This routine assigns the decision space to Noah's model variables.

76.12.3 randArray (Source File: noah33_set_pedecvars.F90)

INTERFACE:

```
subroutine getvardata(n,DEC_State,vname, vdata, statusStateGet)
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc
use LIS_logMod,    only : LIS_logunit,LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: DEC_State
character*100          :: vname
real          :: vdata(LIS_rc%npatch(n,LIS_rc%lsm_index))
```

DESCRIPTION:

This routine assigns the decision space to Noah's model variables.

76.12.4 noah33_getpeobspred_ARMobs (Source File: noah33_getpeobspred_ARMobs.F90)

REVISION HISTORY:

```
06 Feb 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah33_getpeobspred_ARMobs(Obj_Func)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_soilsMod,  only : LIS_soils
use noah33_lsmMod, only : noah33_struc
use LIS_logMod,       only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: Obj_Func
```

DESCRIPTION:

This routine assigns the decision space to Noah's model variables.

76.12.5 noah33_setupobspred_ARMobs (Source File: noah33_setupobspred_ARMobs.F90)

REVISION HISTORY:

```
16 Jul 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah33_setupobspred_ARMobs(OBSPred)
```

USES:

```
use ESMF
use LIS_coreMod,      only : LIS_rc, LIS_vecPatch
use LIS_logMod,       only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: OBSPred
```

DESCRIPTION:

This routine creates an entry in the Obs pred object used for parameter estimation

---

76.12.6 noah33_getpeobspred_FLUXNETobs (Source File: noah33_getpeobspred_FLUXNETobs.F90)

REVISION HISTORY:

```
06 Feb 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah33_getpeobspred_FLUXNETobs(Obj_Func)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_soilsMod,  only : LIS_soils
use noah33_lsmMod, only : noah33_struc
use LIS_logMod,       only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: Obj_Func
```

DESCRIPTION:

This routine assigns the decision space to Noah's model variables.

76.12.7 noah33_setupobspred_FLUXNETobs (Source File: noah33_setupobspred_FLUXNETobs.F90)

REVISION HISTORY:

```
16 Jul 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah33_setupobspred_FLUXNETobs(OBSPred)
```

USES:

```
use ESMF
use LIS_coreMod,      only : LIS_rc, LIS_vecPatch
use LIS_logMod,       only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: OBSPred
```

DESCRIPTION:

This routine creates an entry in the Obs pred object used for parameter estimation

---

76.12.8 noah33_getpeobspred_LPRM_AMSREsmObs (Source File: noah33_getpeobspred_LPRMsmobs.F90)

REVISION HISTORY:

```
06 Feb 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah33_getpeobspred_LPRM_AMSREsmObs(Obj_Func)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_soilsMod,  only : LIS_soils
use noah33_lsmMod, only : noah33_struc
use LIS_logMod,       only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: Obj_Func
```

DESCRIPTION:

This routine assigns the decision space to Noah's model variables.

76.12.9 noah33_setupobspred_LPRM_AMSREsmObs (Source File: noah33_setupobspred_LPRMsmobs.F90)

REVISION HISTORY:

```
16 Jul 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah33_setupobspred_LPRM_AMSREsmObs(OBSPred)
```

USES:

```
use ESMF
use LIS_coreMod,      only : LIS_rc, LIS_vecPatch
use LIS_logMod,       only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: OBSPred
```

DESCRIPTION:

This routine creates an entry in the Obs pred object used for parameter estimation

---

76.12.10 noah33_getpeobspred_USDA_ARSsmObs (Source File: noah33_getpeobspred_USDA_ARSsmobs.F9

REVISION HISTORY:

```
06 Feb 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah33_getpeobspred_USDA_ARSsmObs(Obj_Func)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_soilsMod,  only : LIS_soils
use noah33_lsmMod, only : noah33_struc
use LIS_logMod,       only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: Obj_Func
```

DESCRIPTION:

This routine assigns the decision space to Noah's model variables.

76.12.11 noah33_setupobspred_USDA_ARSsmObs (Source File: noah33_setupobspred_USDA_ARSsmobs.F9

REVISION HISTORY:

```
16 Jul 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah33_setupobspred_USDA_ARSsmObs(OBSPred)
```

USES:

```
use ESMF
use LIS_coreMod,      only : LIS_rc, LIS_vecPatch
use LIS_logMod,       only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: OBSPred
```

DESCRIPTION:

This routine creates an entry in the Obs pred object used for parameter estimation

---

76.12.12 noah33_getpeobspred_template (Source File: noah33_getpeobspred_template.F90)

REVISION HISTORY:

```
06 Feb 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah33_getpeobspred_template(Obj_Func)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc
use LIS_soilsMod,  only : LIS_soils
use noah33_lsmMod, only : noah33_struc
use LIS_logMod,       only : LIS_verify

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: Obj_Func
```

DESCRIPTION:

This routine assigns the decision space to Noah's model variables.

76.12.13 noah33_setupobspred_template (Source File: noah33_setupobspred_template.F90)

REVISION HISTORY:

```
16 Jul 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah33_setupobspred_template(OBSPred)
```

USES:

```
use ESMF

implicit none
```

ARGUMENTS:

```
type(ESMF_State)       :: OBSPred
```

DESCRIPTION:

This routine creates an entry in the Obs pred object used for parameter estimation

---

76.12.14 noah33_getrunoffs (Source File: noah33_getrunoffs.F90)

REVISION HISTORY:

```
6 May 2011: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah33_getrunoffs(n)
```

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_routingMod, only : LIS_runoff_state
use LIS_logMod,     only : LIS_verify
use LIS_historyMod
use noah33_lsmMod, only : noah33_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

76.12.15 noah33_getrunoffs_mm (Source File: noah33_getrunoffs_mm.F90)

REVISION HISTORY:

```
6 May 2011: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine noah33_getrunoffs_mm(n)
```

USES:

USES:

```
use ESMF
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_routingMod, only : LIS_runoff_state
use LIS_logMod,     only : LIS_verify
use LIS_constantsMod
use LIS_historyMod
use noah33_lsmMod, only : noah33_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

76.12.16 noah33_sfc2cmem3 (Source File: noah33_sfc2cmem3.F90)

REVISION HISTORY:

```
37 Mar 2009: Sujay Kumar; Initial Code
2  May 2011: Yudong Tian; Modified for Noah 3.2
```

INTERFACE:

```
subroutine noah33_sfc2cmem3(n, sfcState)
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc
use LIS_logMod,    only : LIS_verify
use LIS_historyMod, only : LIS_patch2tile
use LIS_constantsMod,  only : LIS_CONST_RHOFW

use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: sfcState

FUNCTIONS
```

DESCRIPTION:

This subroutine assigns the noah33 specific surface variables to CMEM3.

76.12.17 noah33_sfc2crtm (Source File: noah33_sfc2crtm.F90)

REVISION HISTORY:

```
37 Mar 2009: Sujay Kumar; Initial Code
2  May 2011: Yudong Tian; Modified for Noah 3.2
```

INTERFACE:

```
subroutine noah33_sfc2crtm(n, sfcState)
```

USES:

```
use ESMF
use LIS_coreMod,   only : LIS_rc
use LIS_logMod,    only : LIS_verify
use LIS_constantsMod,  only : LIS_CONST_RHOFW

use noah33_lsmMod

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: sfcState

FUNCTIONS
```

DESCRIPTION:

This subroutine assigns the noah33 specific surface variables to CRTM.

76.12.18 RDHM356_coldstart (Source File: RDHM356_coldstart.F90)

REVISION HISTORY:

```
This subroutine is generated with the Model Implementation Toolkit developed
by Shugong Wang for the NASA Land Information System Version 7. The initial
specification of the subroutine is defined by Sujay Kumar.
 11/5/13: Shugong Wang; initial implementation for LIS 7 and RDHM356
```

INTERFACE:

```
subroutine RDHM356_coldstart(mtype)
```

USES:

```
use LIS_coreMod, only: LIS_rc
use LIS_logMod, only: LIS_logunit
use LIS_timeMgrMod, only: LIS_date2time
use RDHM356_lsmMod
```

DESCRIPTION:

This routine initializes the RDHM356 state variables with some predefined values constantly for the entire domain.

76.12.19 RDHM356_dynsetup (Source File: RDHM356_dynsetup.F90)

REVISION HISTORY:

```
This subroutine is generated with the Model Implementation Toolkit developed
by Shugong Wang for the NASA Land Information System Version 7. The initial
specification of the subroutine is defined by Sujay Kumar.
 11/5/13: Shugong Wang; initial implementation for LIS 7 and RDHM356
```

INTERFACE:

```
subroutine RDHM356_dynsetup(n)
```

USES:

```
 use LIS_logMod, only     : LIS_logunit
 use LIS_coreMod, only    : LIS_rc, LIS_domain, LIS_surface
 use LIS_timeMgrMod, only : LIS_date2time, LIS_tick
 use LIS_fileIOMod, only  : LIS_read_param
 use RDHM356_lsmMod, only : RDHM356_struc
!use any other modules
```

DESCRIPTION:

This routine sets up the time-dependent variables in RDHM356
The routines invoked are:

LIS_read_param (3.14.4)
retrieves LIS parameter data from NetCDF file

RDHM356_gen_date_str (**??**)
generate date-time string for reading dynamic spatial parameters
format: YYYY, YYYY_MM, YYYY_MM_DD, YYYY_MM_DD_HH

76.12.20 RDHM356_f2t (Source File: RDHM356_f2t.F90)

REVISION HISTORY:

```
This subroutine is generated with the Model Implementation Toolkit developed
by Shugong Wang for the NASA Land Information System Version 7. The initial
specification of the subroutine is defined by Sujay Kumar.

11/6/13: Shugong Wang; Initial implementation for LIS 7 and RDHM356
```

INTERFACE:

```
subroutine RDHM356_f2t(n)
```

USES:

```
 use ESMF
 use LIS_coreMod, only       : LIS_rc , LIS_surface
 use LIS_metforcingMod, only : LIS_FORC_State
 use LIS_logMod, only        : LIS_verify
 use LIS_FORC_AttributesMod
 use RDHM356_lsmMod

 implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing into the RDHM356 model tiles.
The arguments are:

n index of the nest

76.12.21 RDHM356_finalize (Source File: RDHM356_finalize.F90)

REVISION HISTORY:

```
This subroutine is generated with the Model Implementation Toolkit developed
by Shugong Wang for the NASA Land Information System Version 7. The initial
specification of the subroutine is defined by Sujay Kumar.
 11/5/13: Shugong Wang; initial implementation for RDHM356 with LIS-7
```

INTERFACE:

```
subroutine RDHM356_finalize(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use RDHM356_lsmMod
```

DESCRIPTION:

This routine cleans up the allocated memory structures in RDHM356

76.13 Fortran: Module Interface RDHM356_lsmMod (Source File: RDHM356_lsmMod.F90

This module provides the definition of derived data type used to control the operation of RDHM356 model. It also provides the entry method for the initialization of RDHM356-specific variables. The derived data type `RDHM356_struc` includes the variables that specify the runtime options and other control variables as described below:

rfile name of the RDHM356 restart file

rformat format of restart file (binary or netcdf) for RDHM356

LDT_ncvar_SoilAlb LDT NetCDF variable name for snow free ALBEDO (default value 0.15)

LDT_ncvar_SnowAlb LDT NetCDF variable name for snow ALBEDO (default value 0.7)

LDT_ncvar_SOILTYP LDT NetCDF variable name for Soil type

LDT_ncvar_VEGETYP LDT NetCDF variable name for Vegetation type

LDT_ncvar_UZTWM LDT NetCDF variable name for upper zone tension water maximum storage

LDT_ncvar_UZFWM LDT NetCDF variable name for upper zone free water maximum storage

LDT_ncvar_UZK LDT NetCDF variable name for upper zone free water latent depletion rate

LDT_ncvar_PCTIM LDT NetCDF variable name for impervious fraction of the watershad area

LDT_ncvar_ADIMP LDT NetCDF variable name for additional impervious area

LDT_ncvar_RIVA LDT NetCDF variable name for riparian vegetation area

LDT_ncvar_ZPERC LDT NetCDF variable name for maximum percolation rate

LDT_ncvar_REXP LDT NetCDF variable name for exponent of the percolation equation (percolation parameter)

LDT_ncvar_LZTWM LDT NetCDF variable name for lower zone tension water maximum storage

LDT_ncvar_LZFSM LDT NetCDF variable name for lower zone supplemental free water (fast) maximum storage

LDT_ncvar_LZFPM LDT NetCDF variable name for lower zone primary free water (slow) maximum storage

LDT_ncvar_LZSK LDT NetCDF variable name for lower zone supplemental free water depletion rate

LDT_ncvar_LZPK LDT NetCDF variable name for lower zone primary free water depletion rate

LDT_ncvar_PFREE LDT NetCDF variable name for fraction percolation from upper to lower free water storage

LDT_ncvar_SIDE LDT NetCDF variable name for ratio of deep recharge to channel base flow

LDT_ncvar_RSERV LDT NetCDF variable name for fraction of lower zone free water not transferable to tension water

LDT_ncvar_EFC LDT NetCDF variable name for fraction of forest cover

LDT_ncvar_TBOT LDT NetCDF variable name for bottom boundary soil temperature

LDT_ncvar_RSMAX LDT NetCDF variable name for maximum residual porosity (usually = 0.58)

LDT_ncvar_CKSL LDT NetCDF variable name for ratio of frozen to non-frozen surface (increase in frozen ground contact, usually = 8 s/m)

LDT_ncvar_ZBOT LDT NetCDF variable name for lower boundary depth (negative value, usually = -2.5 m)

LDT_ncvar_ALON LDT NetCDF variable name for logitude

LDT_ncvar_ALAT LDT NetCDF variable name for latitude

LDT_ncvar_SCF LDT NetCDF variable name for snow fall correction factor

LDT_ncvar_MFMAX LDT NetCDF variable name for maximum melt factor

LDT_ncvar_MFMIN LDT NetCDF variable name for minimum melt factor

LDT_ncvar_NMF LDT NetCDF variable name for maximum negative melt factor

LDT_ncvar_UADJ LDT NetCDF variable name for the average wind function during rain-on-snow periods

LDT_ncvar_SI LDT NetCDF variable name for areal water-equivalent above which 100 percent areal snow cover

LDT_ncvar_MBASE LDT NetCDF variable name for base temperature for non-rain melt factor

LDT_ncvar_PXTEMP LDT NetCDF variable name for temperature which spereates rain from snow

LDT_ncvar_PLWHC LDT NetCDF variable name for maximum amount of liquid-water held against gravity drainage

LDT_ncvar_TIPM LDT NetCDF variable name for antecedent snow temperature index parameter

LDT_ncvar_GM LDT NetCDF variable name for daily ground melt

LDT_ncvar_ELEV LDT NetCDF variable name for elevation

LDT_ncvar_LAEC LDT NetCDF variable name for snow-rain split temperature

ts RDHM356 model time step in second

count variable to keep track of the number of timesteps before an output

rstInterval restart writing interval

outInterval output writing interval

rdhm356 RDHM356 model specific variables

forc_count counter of forcing data

TempHeight observation height of temperature of humidity

WindHeight observation height of wind

DT_SAC_SNOW17 simulation time interval of SAC model and Snow-17

DT_FRZ simulation time interval of frozen soil model

FRZ_VER_OPT version number of frozen soil model. 1: old version, 2: new version

SNOW17_OPT option for snow-17. If SNOW17_OPT=1, use SNOW-17, otherwise, don't use

NSTYP number of soil types

NVTYP number of vegetation types

NDINTW number of desired soil layers for total and liquid soil moisture

NDSINT number of desired soil layers for soil temperature

NORMALIZE normalization flag for total and liquid soil moisture output (1-normalized, 0-not)

DSINTW thickness of desired soil layers for liquid and total soil moisture

DSINT thickness of desired soil layers for soil temperature

PETADJ_MON adjustment of PET for 12 months

vegRCMIN minimal stomatal resistance table for SACHTET, 14 values

climRCMIN climate dependent miminal stomatal resistance for SACHTET, 14 values

RGL solar radiation threshold table for SACHTET, 14 values

HS vapor pressure resistance factor table for SACHTET, 14 values

LAI leaf area index table for SACHTET, 14 values

D50 the depth (cm) table at which 50% roots are allocated for SACHTET, 14 values

CROOT root distribution parameter table for SACHTET, 14 values

Z0 roughness coefficient of surface

CLAY clay content for SACHTET, 12 values

SAND sand content for sACHTET, 12 values

SATDK saturated hydraulic conductivityfor SACHTET, 12 values

CZIL default=0.12 Zilitinkevich

FXEXP FXEXP(fxexp),(default=2.0) bare soil

vegRCMAX RCMAX,(default=5000s/m) maximum stomatal resistance

TOPT TOPT,(default=298K)optimum air

PC plant coef. default pc = -1, 0.6 - 0.8

PET_OPT if PET_OPT = 0, use non Penmann-based ETP;if penpt ¿ 0 empirical Penmann equation; if penpt ¡ 0, use energy based Pennman

RDST default=1 means noah option,this constant allows selection of tension water redistribution option, if rdst = 0 (ohd), use OHD version of SRT subroutine this SRT uses reference gradient instead an actual. if rdst = 1 (noah), use Noah version of SRT subroutine

thresholdRCMIN this constant allows change of RCMIN (0.5)

SFCREF reference wind speed for PET adjustment (4 m/s)

BAREADJ Ek-Chen evaporation threshold switch. Bare soil evaporation option changes according to greenness.

SNOW17_SWITCH switch variable change liquid water freezing version, 0: Victor's version, 1: Eric's version

REVISION HISTORY:

```
This module is generated with the Model Implementation Toolkit developed
by Shugong Wang for the NASA Land Information System Version 7. The initial
specification of the module is defined by Sujay Kumar.
11/5/13: Shugong Wang Initial implementation for LIS 7 and RDHM356
```

USES:

```
use RDHM356_module

implicit none

PRIVATE
!-----------------------------------------------------------------------
! PUBLIC MEMBER FUNCTIONS
!-----------------------------------------------------------------------
public :: RDHM356_ini
!-----------------------------------------------------------------------
! PUBLIC TYPES
!-----------------------------------------------------------------------
public :: RDHM356_struc
```

76.13.1 RDHM356_ini (Source File: RDHM356_lsmMod.F90)

INTERFACE:

```
subroutine RDHM356_ini()
```

USES:

```
    use LIS_coreMod, only             : LIS_rc
    use LIS_surfaceModelDataMod, only : LIS_sfmodel_struc
    use LIS_timeMgrMod, only          : LIS_clock, LIS_calendar, &
                                        LIS_update_timestep,     &
                                        LIS_registerAlarm
    use LIS_logMod, only              : LIS_verify
```

DESCRIPTION:

This routine creates the datatypes and allocates memory for RDHM356-specific variables. It also invokes the routine to read the runtime specific options for RDHM356 from the configuration file.
The routines invoked are:

RDHM356_readcrd (76.14.1)
reads the runtime options for RDHM356 model

76.13.2 RDHM356_main (Source File: RDHM356_main.F90)

REVISION HISTORY:

```
This subroutine is generated with the Model Implementation Toolkit developed
by Shugong Wang for the NASA Land Information System Version 7. The initial
specification of the subroutine is defined by Sujay Kumar.
 3/7/14: Shugong Wang; initial implementation for RDHM356 with LIS-7
```

INTERFACE:

```
subroutine RDHM356_main(n)
```

USES:

```
 use LIS_coreMod
 use LIS_histDataMod
 use LIS_timeMgrMod, only : LIS_isAlarmRinging
 use LIS_logMod, only     : LIS_logunit, LIS_endrun
 use LIS_FORC_AttributesMod
 use RDHM356_lsmMod
!use other modules

 implicit none
```

ARGUMENTS:

```
 integer, intent(in)  :: n
 integer              :: t
 integer              :: i
 real                 :: dt
 real                 :: lat, lon
 integer              :: row, col
 logical              :: alarmCheck
```

DESCRIPTION:

This is the entry point for calling the RDHM356 physics. This routine calls the `RDHM_356` routine that performs the land surface computations, to solve for water and energy equations.
The arguments are:

n index of the nest

76.14 Fortran: Module Interface RDHM356_module (Source File: RDHM356_module.F90)

The code in this file provides a description of the data structure containing the RDHM356 1-d variables.
The variables specified in the data structure include:

Tair air temperature. unit: K

Psurf surface air pressure. unit: Pa

Wind_E eastward wind. unit: m/s

Wind_N northward wind. unit: m/s

Qair near surface specific humidity. unit: kg/kg

Rainf rainfall rate. unit: kg/m2s

Snowf snowfall rate. unit: kg/m2s

Swdown incident shortwave radiation. unit: W/m2

Lwdown incident longwave radiation. unit: W/m2

Tair_min daily minimum air temperature. unit: K

Tair_max daily maximum air temperature. unit: K

TempHeight observation height of temperature of humidity. unit: m

WindHeight observation height of wind. unit: m

DT_SAC_SNOW17 simulation time interval of SAC model and Snow-17. unit: s

DT_FRZ simulation time interval of frozen soil model. unit: s

FRZ_VER_OPT version number of frozen soil model. 1: old version, 2: new version. unit: -

SNOW17_OPT option for snow-17. If SNOW17_OPT=1, use SNOW-17, otherwise, don't use. unit: -

NSTYP number of soil types. unit: -

NVTYP number of vegetation types. unit: -

NDINTW number of desired soil layers for total and liquid soil moisture. unit: -

NDSINT number of desired soil layers for soil temperature. unit: -

NORMALIZE normalization flag for total and liquid soil moisture output (1-normalized, 0-not). unit: -

DSINTW thickness of desired soil layers for liquid and total soil moisture. unit: cm

DSINT thickness of desired soil layers for soil temperature. unit: cm

PET_MON multiband monthly PET climatology, time series of spatial parameter. unit: mm

GRN_MON multiband monthly greenness climatology, time series of spatial parameter [-]. unit: -

PETADJ_MON adjustment of PET for 12 months. unit: -

SoilAlb snow free ALBEDO (default value 0.15). unit: -

SnowAlb snow ALBEDO (default value 0.7). unit: -

SOILTYP Soil type. unit: -

VEGETYP Vegetation type. unit: -

UZTWM upper zone tension water maximum storage. unit: mm

UZFWM upper zone free water maximum storage. unit: mm

UZK upper zone free water latent depletion rate. unit: day^{-1}

PCTIM impervious fraction of the watershad area. unit: -

ADIMP additional impervious area. unit: -

RIVA riparian vegetation area. unit: -

ZPERC maximum percolation rate. unit: -

REXP exponent of the percolation equation (percolation parameter). unit: -

LZTWM lower zone tension water maximum storage. unit: mm

LZFSM lower zone supplemental free water (fast) maximum storage. unit: mm

LZFPM lower zone primary free water (slow) maximum storage. unit: mm

LZSK lower zone supplemental free water depletion rate. unit: day^{-1}

LZPK lower zone primary free water depletion rate. unit: day^{-1}

PFREE fraction percolation from upper to lower free water storage. unit: day^{-1}

SIDE ratio of deep recharge to channel base flow. unit: -

RSERV fraction of lower zone free water not transferable to tension water. unit: -

EFC fraction of forest cover. unit: -

TBOT bottom boundary soil temperature. unit: °C

RSMAX maximum residual porosity (usually = 0.58). unit: -

CKSL ratio of frozen to non-frozen surface (increase in frozen ground contact, usually = 8 s/m). unit: s/m

ZBOT lower boundary depth (negative value, usually = -2.5 m). unit: m

vegRCMIN minimal stomatal resistance table for SACHTET, 14 values. unit: s/m

climRCMIN climate dependent miminal stomatal resistance for SACHTET, 14 values. unit: s/m

RGL solar radiation threshold table for SACHTET, 14 values. unit: W/m2

HS vapor pressure resistance factor table for SACHTET, 14 values. unit: -

LAI leaf area index table for SACHTET, 14 values. unit: -

D50 the depth (cm) table at which 50% roots are allocated for SACHTET, 14 values. unit: cm

CROOT root distribution parameter table for SACHTET, 14 values. unit: -

Z0 roughness coefficient of surface. unit: m

CLAY clay content for SACHTET, 12 values. unit: -

SAND sand content for sACHTET, 12 values. unit: -

SATDK saturated hydraulic conductivityfor SACHTET, 12 values. unit: m/s

CZIL default=0.12 Zilitinkevich. unit: -

FXEXP FXEXP(fxexp),(default=2.0) bare soil. unit: -

vegRCMAX RCMAX,(default=5000s/m) maximum stomatal resistance. unit: s/m

TOPT TOPT,(default=298K)optimum air. unit: K

PC plant coef. default pc = -1, 0.6 - 0.8. unit: -

PET_OPT if PET_OPT = 0, use non Penmann-based ETP;if penpt > 0 empirical Penmann equation; if penpt < 0, use energy based Pennman. unit: -

RDST default=1 means noah option,this constant allows selection of tension water redistribution option, if rdst = 0 (ohd), use OHD version of SRT subroutine this SRT uses reference gradient instead an actual. if rdst = 1 (noah), use Noah version of SRT subroutine. unit: -

thresholdRCMIN this constant allows change of RCMIN (0.5). unit: s/m

SFCREF reference wind speed for PET adjustment (4 m/s). unit: m/s

BAREADJ Ek-Chen evaporation threshold switch. Bare soil evaporation option changes according to greenness.. unit: -

ALON logitude. unit: -

ALAT latitude. unit: -

SCF snow fall correction factor. unit: -

MFMAX maximum melt factor. unit: mm/(6hr°C)

MFMIN minimum melt factor. unit: mm/(6hr°C)

NMF maximum negative melt factor. unit: mm/(6hr°C)

UADJ the average wind function during rain-on-snow periods. unit: mm/mb

SI areal water-equivalent above which 100 percent areal snow cover. unit: mm

MBASE base temperature for non-rain melt factor. unit: °C

PXTEMP temperature which spereates rain from snow. unit: °C

PLWHC maximum amount of liquid-water held against gravity drainage. unit: -

TIPM antecedent snow temperature index parameter. unit: -

GM daily ground melt. unit: mm/day

ELEV elevation. unit: m

LAEC snow-rain split temperature. unit: °C

ADC multiband Snow-17 curve coordinates. unit: -

SNOW17_SWITCH switch variable change liquid water freezing version, 0: Victor's version, 1: Eric's version. unit: -

UZTWC upper zone tension water storage content. unit: mm

UZFWC upper zone free water storage content. unit: mm

LZTWC lower zone tension water storage content. unit: mm

LZFPC lower zone primary free water storage content. unit: mm

LZFSC lower zone supplemental free water storage content. unit: mm

ADIMC additional impervious area content. unit: mm

TS0 first soil layer temperature. unit: °C

TS1 second soil layer temperature. unit: °C

TS2 third soil layer temperature. unit: °C

TS3 fourth soil layer temperature. unit: °C

TS4 fifth soil layer temperature. unit: °C

UZTWH unfrozen upper zone tension water. unit: mm

UZFWH unfrozen uppeer zone free water. unit: mm

LZTWH unfrozen lower zone tension water. unit: mm

LZFSH unfrozen lower zone supplemental free water. unit: mm

LZFPH unfrozen lower zone primary free water. unit: mm

SMC volumetric content of total soil moisture at each layer. unit: m3/m3

SH2O volumetric content of liquid soil moisture at each layer. unit: m3/m3

WE snow water equivalent without liquid water. unit: mm

LIQW liquid water in snow. unit: mm

NEGHS negative snow heat. unit: mm

TINDEX antecedent temperature index. unit: °C

ACCMAX cumulated snow water including liquid. unit: mm

SNDPT snow depth. unit: cm

SNTMP average snow temperature. unit: °C

SB the last highest snow water equivalent before any snow fall. unit: °C

SBAESC internal snow state during melt & new snow fall (checked with Victor). unit: -

SBWS internal snow state during melt & new snow fall (checked with Victor). unit: -

STORAGE snow liquid water attenuation storage. unit: mm

AEADJ adjusted areal snow cover fraction. unit: -

EXLAG array of lagged liquid water values. unit: -

NEXLAG number of ordinates in lagged liquid water array (EXLAG). unit: -

TA_PREV air temperature of previous time step. unit: -

SWINT total volumetric soil moisture contents at desired soil layers (can be different from soil layers). unit: -

SWHINT liquid volumetric soil moisture contents at desired soil layers (can be different from soil layers). unit: -

TSINT soil temperature at desired soil layers (can be different from soil layers). unit: -

FRZDUP depth of the upper border of frozen ground from surface. unit: m

FRZDBT depth of the bottom border of frozen ground from surface. unit: m

FROST frost index. unit: -

ALBEDO land surface albedo. unit: -

SURF Qs <=> SURF simulated fast runoff (surface runoff). unit: mm/s

GRND Qsb <=> GRND simulated slow runoff (baseflow). unit: mm/s

TET Evap <=> TET simulated actual evapotranspiration. unit: mm/s

EDMND PotEvap <=> EDMND potential evapotranspiration. unit: mm/s

CH surface layer exchage coefficient for heat and moisture. unit: s/m

CM surface layer exchange coefficient for momentum (drag coefficient). unit: s/m

SnowFrac snow cover fraction, Snow-17. unit: -

SWE snow water equivalent, Snow-17. unit: kg/m2

SnowDepth snow depth, Snow-17. unit: m

RM rain + melt. unit: mm

REVISION HISTORY:

```
This module is generated with the Model Implementation Toolkit developed
by Shugong Wang for the NASA Land Information System Version 7. The initial
specification of the module is defined by Sujay Kumar.
11/5/13: Shugong Wang Initial implementation for LIS 7 and RDHM356
```

76.14.1 RDHM356_readcrd (Source File: RDHM356_readcrd.F90)

REVISION HISTORY:

```
This subroutine is generated with the Model Implementation Toolkit developed
by Shugong Wang for the NASA Land Information System Version 7. The initial
specification of the subroutine is defined by Sujay Kumar.
 11/5/13 : Shugong Wang, initial implementation for LIS 7 and RDHM356
```

INTERFACE:

```
subroutine RDHM356_readcrd()
```

USES:

```
use ESMF
use LIS_coreMod, only    : LIS_rc , LIS_config
use LIS_timeMgrMod, only : LIS_parseTimeString
use LIS_logMod, only     : LIS_logunit , LIS_verify
use RDHM356_lsmMod, only : RDHM356_struc
```

DESCRIPTION:

This routine reads the options specific to RDHM356 model from the LIS configuration file.

76.14.2 RDHM356_readrst (Source File: RDHM356_readrst.F90)

REVISION HISTORY:

```
This subroutine is generated with the Model Implementation Toolkit developed
by Shugong Wang for the NASA Land Information System Version 7. The initial
specification of the subroutine is defined by Sujay Kumar.
 3/6/14: Shugong Wang; initial implementation for LIS 7 and RDHM356
```

INTERFACE:

```
subroutine RDHM356_readrst()
```

USES:

```
    use LIS_coreMod, only    : LIS_rc, LIS_masterproc
    use LIS_historyMod, only : LIS_readvar_restart
    use LIS_logMod, only     : LIS_logunit, LIS_endrun, &
                               LIS_getNextUnitNumber,   &
                               LIS_releaseUnitNumber,   &
                               LIS_verify
    use RDHM356_lsmMod

#if (defined USE_NETCDF3 || defined USE_NETCDF4)
    use netcdf
#endif
```

DESCRIPTION:

This program reads restart files for RDHM356. This includes all relevant water/energy storages and tile information. The following is the list of variables specified in the RDHM356 restart file:

```
nc, nr, ntiles             - grid and tile space dimensions
UZTWC                      - RDHM356 upper zone tension water storage content [mm]
UZFWC                      - RDHM356 upper zone free water storage content [mm]
LZTWC                      - RDHM356 lower zone tension water storage content [mm]
LZFPC                      - RDHM356 lower zone primary free water storage content [mm]
LZFSC                      - RDHM356 lower zone supplemental free water storage content [mm]
ADIMC                      - RDHM356 additional impervious area content [mm]
TS0                        - RDHM356 first soil layer temperature [C]
TS1                        - RDHM356 second soil layer temperature [C]
TS2                        - RDHM356 third soil layer temperature [C]
TS3                        - RDHM356 fourth soil layer temperature [C]
TS4                        - RDHM356 fifth soil layer temperature [C]
UZTWH                      - RDHM356 unfrozen upper zone tension water [mm]
UZFWH                      - RDHM356 unfrozen uppeer zone free water [mm]
LZTWH                      - RDHM356 unfrozen lower zone tension water [mm]
LZFSH                      - RDHM356 unfrozen lower zone supplemental free water [mm]
LZFPH                      - RDHM356 unfrozen lower zone primary free water [mm]
SMC                        - RDHM356 volumetric content of total soil moisture at each layer [m3/
SH2O                       - RDHM356 volumetric content of liquid soil moisture at each layer [m3/
WE                         - RDHM356 snow water equivalent without liquid water [mm]
LIQW                       - RDHM356 liquid water in snow [mm]
NEGHS                      - RDHM356 negative snow heat [mm]
TINDEX                     - RDHM356 antecedent temperature index [C]
ACCMAX                     - RDHM356 cumulated snow water including liquid [mm]
```

```
SNDPT                       - RDHM356 snow depth [cm]
SNTMP                       - RDHM356 average snow temperature [C]
SB                          - RDHM356 the last highest snow water equivalent before any snow fall
SBAESC                      - RDHM356 internal snow state during melt & new snow fall (checked wit
SBWS                        - RDHM356 internal snow state during melt & new snow fall (checked wit
STORAGE                     - RDHM356 snow liquid water attenuation storage [mm]
AEADJ                       - RDHM356 adjusted areal snow cover fraction [-]
EXLAG                       - RDHM356 array of lagged liquid water values [-]
NEXLAG                      - RDHM356 number of ordinates in lagged liquid water array (EXLAG) [-]
TA_PREV                     - RDHM356 air temperature of previous time step [-]
```

The routines invoked are:

LIS_readvar_restart (3.17.6)
reads a variable from the restart file

RDHM356_coldstart (76.12.18)
initializes the RDHM356 state variables

76.14.3 RDHM356_setup (Source File: RDHM356_setup.F90)

REVISION HISTORY:

```
This subroutine is generated with the Model Implementation Toolkit developed
by Shugong Wang for the NASA Land Information System Version 7. The initial
specification of the subroutine is defined by Sujay Kumar.
 3/4/14: Shugong Wang; initial implementation for LIS 7 and RDHM356
```

INTERFACE:

```
subroutine RDHM356_setup()
```

USES:

```
use LIS_logMod,    only: LIS_logunit, LIS_verify, LIS_endrun
use LIS_fileIOMod, only: LIS_read_param, LIS_convertParamDataToLocalDomain
use LIS_coreMod,   only: LIS_rc, LIS_surface
use RDHM356_lsmMod
```

DESCRIPTION:

This routine is the entry point to set up the parameters required for RDHM356. These include:
SoilAlb - snow free ALBEDO (default value 0.15) [-]
SnowAlb - snow ALBEDO (default value 0.7) [-]
SOILTYP - Soil type [-]
VEGETYP - Vegetation type [-]
UZTWM - upper zone tension water maximum storage [mm]
UZFWM - upper zone free water maximum storage [mm]
UZK - upper zone free water latent depletion rate [day^{-1}]
PCTIM - impervious fraction of the watershad area [-]
ADIMP - additional impervious area [-]
RIVA - riparian vegetation area [-]
ZPERC - maximum percolation rate [-]
REXP - exponent of the percolation equation (percolation parameter) [-]

LZTWM - lower zone tension water maximum storage [mm]
LZFSM - lower zone supplemental free water (fast) maximum storage [mm]
LZFPM - lower zone primary free water (slow) maximum storage [mm]
LZSK - lower zone supplemental free water depletion rate [day^{-1}]
LZPK - lower zone primary free water depletion rate [day^{-1}]
PFREE - fraction percolation from upper to lower free water storage [day^{-1}]
SIDE - ratio of deep recharge to channel base flow [-]
RSERV - fraction of lower zone free water not transferable to tension water [-]
EFC - fraction of forest cover [-]
TBOT - bottom boundary soil temperature [C]
RSMAX - maximum residual porosity (usually = 0.58) [-]
CKSL - ratio of frozen to non-frozen surface (increase in frozen ground contact, usually = 8 s/m) [s/m]
ZBOT - lower boundary depth (negative value, usually = -2.5 m) [m]
vegRCMIN - minimal stomatal resistance table for SACHTET, 14 values [s/m]
climRCMIN - climate dependent miminal stomatal resistance for SACHTET, 14 values [s/m]
RGL - solar radiation threshold table for SACHTET, 14 values [W/m2]
HS - vapor pressure resistance factor table for SACHTET, 14 values [-]
LAI - leaf area index table for SACHTET, 14 values [-]
D50 - the depth (cm) table at which 50CROOT - root distribution parameter table for SACHTET, 14 values [-]
Z0 - roughness coefficient of surface [m]
CLAY - clay content for SACHTET, 12 values [-]
SAND - sand content for sACHTET, 12 values [-]
SATDK - saturated hydraulic conductivityfor SACHTET, 12 values [m/s]
ALON - logitude [-]
ALAT - latitude [-]
SCF - snow fall correction factor [-]
MFMAX - maximum melt factor [mm/(6hrC)]
MFMIN - minimum melt factor [mm/(6hrC)]
NMF - maximum negative melt factor [mm/(6hrC)]
UADJ - the average wind function during rain-on-snow periods [mm/mb]
SI - areal water-equivalent above which 100 percent areal snow cover [mm]
MBASE - base temperature for non-rain melt factor [C]
PXTEMP - temperature which spereates rain from snow [C]
PLWHC - maximum amount of liquid-water held against gravity drainage [-]
TIPM - antecedent snow temperature index parameter [-]
GM - daily ground melt [mm/day]
ELEV - elevation [m]
LAEC - snow-rain split temperature [C]

The routines invoked are:

LIS_read_param (3.14.4)
retrieves LIS parameter data from NetCDF file

RDHM356_read_multilevel_param (**??**)
retrieves multilevel spatial parameter from NetCDF file

76.14.4 RDHM356_read_multilevel_param (Source File: RDHM356_setup.F90)

REVISION HISTORY:

```
03 Sept 2004: Sujay Kumar; Initial Specification for read_laiclimo
30 Oct  2013: Shugong Wang; Generalization for reading multilevel spatial parameter
```

INTERFACE:

```
subroutine RDHM356_read_multilevel_param(n, ncvar_name, level, placeholder)
```

USES:

```
    use esmf
    use netcdf
    use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_localPet,   &
                            LIS_ews_halo_ind, LIS_ewe_halo_ind, &
                            LIS_nss_halo_ind, LIS_nse_halo_ind
    use LIS_logMod,  only : LIS_logunit, LIS_verify, LIS_endrun
    use LIS_fileIOMod, only: LIS_read_param, LIS_convertParamDataToLocalDomain
    implicit none
```

ARGUMENTS:

```
    integer, intent(in)          :: n
    integer, intent(in)          :: level
    character(len=*), intent(in) :: ncvar_name
    real, intent(out)            :: placeholder(LIS_rc%lnc(n), LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine reads multilevel parameters from the LIS NetCDF parameter data file
The arguments are:

n index of n

level level index (month, quarter, soil layer, snow layer) of the data to be read

array array containing returned values

76.14.5 RDHM356_writerst (Source File: RDHM356_writerst.F90)

REVISION HISTORY:

```
This subroutine is generated with the Model Implementation Toolkit developed
by Shugong Wang for the NASA Land Information System Version 7. The initial
specification of the subroutine is defined by Sujay Kumar.
 11/5/13: Shugong Wang; initial implementation for LIS 7 and RDHM356
```

INTERFACE:

```
subroutine RDHM356_writerst(n)
```

USES:

```
    use LIS_coreMod, only    : LIS_rc, LIS_masterproc
    use LIS_timeMgrMod, only : LIS_isAlarmRinging
    use LIS_logMod, only     : LIS_logunit, LIS_getNextUnitNumber, &
                               LIS_releaseUnitNumber , LIS_verify
    use LIS_fileIOMod, only  : LIS_create_output_directory, &
                               LIS_create_restart_filename
    use RDHM356_lsmMod

#if (defined USE_NETCDF3 || defined USE_NETCDF4)
    use netcdf
#endif

    implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This program writes restart files for RDHM356. This includes all relevant water/energy storage and tile information.
The routines invoked are:

LIS_create_output_directory (3.14.6)
creates a timestamped directory for the restart files

LIS_create_restart_filename (3.14.2)
generates a timestamped restart filename

RDHM356_dump_restart (76.14.6)
writes the RDHM356 variables into the restart file

76.14.6 RDHM356_dump_restart (Source File: RDHM356_writerst.F90)

REVISION HISTORY:

```
This subroutine is generated with the Model Implementation Toolkit developed
by Shugong Wang for the NASA Land Information System Version 7. The initial
specification of the subroutine is defined by Sujay Kumar.
11/5/13: Shugong Wang, initial implementation for LIS 7 and RDHM356
```

INTERFACE:

```
subroutine RDHM356_dump_restart(n, ftn, wformat)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_logMod, only  : LIS_logunit
use LIS_historyMod
use RDHM356_lsmMod

implicit none

integer, intent(in) :: ftn
integer, intent(in) :: n
character(len=*), intent(in) :: wformat
```

DESCRIPTION:

This routine gathers the necessary restart variables and performs the actual write statements to create the restart files.
The arguments are:

n index of the nest

ftn unit number for the restart file

wformat restart file format (binary/netcdf)

The following is the list of variables written in the RDHM356 restart file:

```
nc, nr, ntiles  - grid and tile space dimensions
UZTWC           - RDHM356 upper zone tension water storage content [mm]
UZFWC           - RDHM356 upper zone free water storage content [mm]
LZTWC           - RDHM356 lower zone tension water storage content [mm]
LZFPC           - RDHM356 lower zone primary free water storage content [mm]
LZFSC           - RDHM356 lower zone supplemental free water storage content [mm]
ADIMC           - RDHM356 additional impervious area content [mm]
TS0             - RDHM356 first soil layer temperature [C]
TS1             - RDHM356 second soil layer temperature [C]
TS2             - RDHM356 third soil layer temperature [C]
TS3             - RDHM356 fourth soil layer temperature [C]
TS4             - RDHM356 fifth soil layer temperature [C]
UZTWH           - RDHM356 unfrozen upper zone tension water [mm]
UZFWH           - RDHM356 unfrozen uppeer zone free water [mm]
LZTWH           - RDHM356 unfrozen lower zone tension water [mm]
LZFSH           - RDHM356 unfrozen lower zone supplemental free water [mm]
LZFPH           - RDHM356 unfrozen lower zone primary free water [mm]
SMC             - RDHM356 volumetric content of total soil moisture at each layer [m3/m3]
SH2O            - RDHM356 volumetric content of liquid soil moisture at each layer [m3/m3]
WE              - RDHM356 snow water equivalent without liquid water [mm]
LIQW            - RDHM356 liquid water in snow [mm]
NEGHS           - RDHM356 negative snow heat [mm]
TINDEX          - RDHM356 antecedent temperature index [C]
ACCMAX          - RDHM356 cumulated snow water including liquid [mm]
SNDPT           - RDHM356 snow depth [cm]
SNTMP           - RDHM356 average snow temperature [C]
SB              - RDHM356 the last highest snow water equivalent before any snow fall [C]
SBAESC          - RDHM356 internal snow state during melt & new snow fall (checked with Victor) [-]
SBWS            - RDHM356 internal snow state during melt & new snow fall (checked with Victor) [-]
STORAGE         - RDHM356 snow liquid water attenuation storage [mm]
AEADJ           - RDHM356 adjusted areal snow cover fraction [-]
EXLAG           - RDHM356 array of lagged liquid water values [-]
NEXLAG          - RDHM356 number of ordinates in lagged liquid water array (EXLAG) [-]
TA_PREV         - RDHM356 air temperature of previous time step [-]
```

The routines invoked are:

LIS_writeGlobalHeader_restart (3.17.22)
writes the global header information

LIS_writeHeader_restart (3.17.23)
writes the header information for a variable

LIS_closeHeader_restart (3.17.24)
close the header

LIS_writevar_restart (3.17.3)
writes a variable to the restart file

---

77 Template

This is a dummy placeholder implementation of a LSM, to be used as a template for new implementations.

77.0.7 template_dynsetup (Source File: template_dynsetup.F90)

INTERFACE:

```
subroutine template_dynsetup(n)
```

USES:

```
implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Currently this routine is a placeholder
The arguments are:

n index of the nest

77.0.8 template_f2t (Source File: template_f2t.F90)

REVISION HISTORY:

```
21 Jul 2004: Sujay Kumar   Initial Specification
23 Oct 2007: Kristi Arsenault, Implemented code for LISv5.0
```

INTERFACE:

```
subroutine template_f2t(n)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc, LIS_surface
use LIS_metforcingMod, only : LIS_FORC_State
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use template_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Forcing-only option (template) for calling the forcing transfer routines.
The arguments are:

n index of the nest

77.0.9 template_finalize (Source File: template_finalize.F90)

INTERFACE:

```
subroutine template_finalize()
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use template_lsmMod, only : template_struc
```

ARGUMENTS:

DESCRIPTION:

This routine cleans up the allocated memory structures in the template (forcing-only option)
The arguments are:

n index of the nest

77.0.10 template_getrunoffs (Source File: template_getrunoffs.F90)

REVISION HISTORY:

```
6 May 2011: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine template_getrunoffs(n)
```

USES:

```
implicit none
```

ARGUMENTS:

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

DESCRIPTION:

77.1 Fortran: Module Interface template_lsmMod (Source File: template_lsmMod.F90)

Module for 1-D land model driver variable initialization

count variable to keep track of the number of timesteps before an output

numout number of output times

outInterval output writing interval

templateopen variable to keep track of opened files

template Template LSM specific variables

REVISION HISTORY:

```
   Apr 2003; Sujay Kumar, Initial Code
23 Oct 2007; Kristi Arsenault, Updated for V5.0
```

USES:

```
use template_module

implicit none

PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
public :: template_lsm_ini
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
public :: template_struc
```

77.1.1 template_lsm_ini (Source File: template_lsmMod.F90)

INTERFACE:

```
subroutine template_lsm_ini()
```

USES:

```
 use ESMF
 use LIS_surfaceModelDataMod, only : LIS_sfmodel_struc
 use LIS_coreMod, only : LIS_rc
 use LIS_timeMgrMod,   only : LIS_clock, LIS_calendar, &
      LIS_update_timestep, LIS_registerAlarm
 use LIS_logMod,       only : LIS_verify
```

DESCRIPTION:

77.1.2 template_main (Source File: template_main.F90)

77.1.3 template_main.F90 (Source File: template_main.F90)

Apr 2003; Sujay Kumar, Initial Code 23 Oct 2007; Kristi Arsenault, Updated code for LISv50

INTERFACE:

```
subroutine template_main(n)

#if ( defined TBOT_TESTING )
  use LIS_coreMod,     only : LIS_rc, LIS_domain
  use LIS_histDataMod, only : LIS_diagnoseSurfaceOutputVar, &
                              LIS_MOC_TEMPBOT
#endif
  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Calls the run routines for the forcing-only option (template)
The arguments are:

n index of the nest

77.2 Fortran: Module Interface template_module.F90 (Source File: template_module.F90)

Declare forcing-only option (template) variables

forcing Array of meteorological forcing

tair 2m air temperature forcing

qair 2m specific humidity forcing

swdown downward shortwave forcing

lwdown downward longwave forcing

uwind u-wind component forcing

vwind v-wind component forcing

psurf surface pressure forcing

rainf total rainfall forcing

rainf_c convective rainfall forcing

snowf total snowfall forcing

REVISION HISTORY:

```
21 Jul 2004: Sujay Kumar; Initial Specification
23 Oct 2007: Kristi Arsenault;  Added code for LISv5.0
```

77.2.1 template_readcrd (Source File: template_readcrd.F90)

REVISION HISTORY:

```
21 Jul 2004; Sujay Kumar, Initial Code
23 Oct 2007; Kristi Arsenault, Updated for use in LISv5.0
23 May 2014: David Mocko, Updated for LIS-7.0
```

INTERFACE:

```
subroutine template_readcrd()
```

USES:

```
use ESMF
use LIS_coreMod,     only : LIS_rc, LIS_config
use LIS_timeMgrMod, only : LIS_parseTimeString
use LIS_logMod,      only : LIS_logunit, LIS_verify
use template_lsmMod, only : template_struc
```

DESCRIPTION:

This routine reads the options specific to template LSM option from the LIS configuration file.

77.2.2 template_readrst (Source File: template_readrst.F90)

INTERFACE:

```
subroutine template_readrst()
```

USES:

```
implicit none
```

ARGUMENTS:

DESCRIPTION:

Calls the restart reading routines for the forcing-only option (template)
The arguments are:

n index of the nest

77.2.3 template_setup (Source File: template_setup.F90)

REVISION HISTORY:

```
21 Jul 2004; Sujay Kumar: Initial Specification
23 Feb 2007: Kristi Arsenault; Updated for LISv5.0
```

INTERFACE:

```
subroutine template_setup()
```

USES:

ARGUMENTS:

DESCRIPTION:

Complete the setup routines for Template option (forcing-only)

77.2.4 template_writerst (Source File: template_writerst.F90)

INTERFACE:

```
subroutine template_writerst(n)
```

USES:

```
implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Forcing-only (template) option for calling the write restart routines.
The arguments are:

n index of the nest

78 VIC land surface model version 4.1.1

This section describes the interface implementations for the Variable Infiltration Capacity (VIC) Macroscale Hydrologic Model, version 4.1.1, available from
http://www.hydro.washington.edu/Lettenmaier/Models/VIC/index.shtml

78.1 Notes regarding incorporating VIC 4.1.1 into LIS

VIC's control loop behaves oppositely from that of LIS (and the other land surface models that it contains). LIS' control loop has time as the outer loop and space as the inner loop; whereas, VIC's control loop has space as the outer loop and time as the inner loop.
Viewing this in terms of data, LIS processes the desired running domain as a 2d gridded field that is one time-level deep; whereas, VIC processes the same running domain one grid cell at a time as a one grid cell

structure that is many time-levels deep. In other words, for LIS, at time t_1, LIS processes forcing for the whole domain, then it advances the model state for the whole domain. Then LIS advances time to t_2 and repeats. For VIC, VIC selects grid cell x_1, then it processes forcing for that grid cell from the beginning to the end of the simulation, then it advances the model state from the beginning to the end of the simulation. Then VIC selects grid cell x_2 and repeats.
This difference in approach complicated incorporating VIC into LIS. To overcome this without modifying the VIC source code, two steps were taken. Step 1 was to create an intermediate layer of data structures and supporting routines. Step 2 was to tell VIC that the simulation was only one time step long. These two steps allow LIS to pass data into VIC, have VIC advance its model state forward one time step, and to store the new state for use in the next call to VIC at the next LIS time step.

78.2 Limitations of VIC 4.1.1 within LIS

VIC has many compile time and run time options that modify the behavior of its underlying physics. VIC within LIS (LIS-VIC) has been tested only for VIC's basic water balance configuration and for VIC's basic energy balance configuration. Other configuration options are not guaranteed to work.

78.2.1 vic411_diagnoseoutputvar (Source File: vic411_diag.F90)

REVISION HISTORY:

```
Nov 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
```

INTERFACE:

```
subroutine vic411_diagnoseoutputvar(nest, tile, moc_index, vlevel, value, &
                                    units, lenu, direction, lend)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_histDataMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)             :: nest, tile, moc_index, vlevel
integer, intent(in)             :: lenu, lend
real*8, intent(in)              :: value
character(len=lenu), intent(in) :: units
character(len=lend), intent(in) :: direction
```

DESCRIPTION:

This routine calls LIS_diagnoseOutputVar to process VIC's output variables.
The arguments are:

nest index of the nest

tile index of the tile

moc_index index identifying which output variable to process

vlevel vertical level to process

value output value to process

units units of the output variable

lenu length of the units string

direction direction of the output variable

lend length of the direction string

78.2.2 vic411_dynsetup (Source File: vic411_dynsetup.F90)

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
```

INTERFACE:

```
subroutine vic411_dynsetup(n)
```

USES:

```
implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine sets up the time-dependent variables in VIC. Currently this routine is a placeholder. The arguments are:

n index of the nest

78.2.3 vic411_f2t (Source File: vic411_f2t.F90)

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
14 Aug 2013; Shugong Wang, Modified for LIS-7
```

INTERFACE:

```
subroutine vic411_f2t(n)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc, LIS_surface
use LIS_timeMgrMod,     only : LIS_isAlarmRinging
use LIS_metforcingMod,  only : LIS_FORC_State
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use vic411_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the VIC 4.1.1 model tiles.
The arguments are:

n index of the nest

78.2.4 vic411_finalize (Source File: vic411_finalize.F90)

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
```

INTERFACE:

```
subroutine vic411_finalize()
```

USES:

```
use LIS_coreMod,   only : LIS_rc
use vic411_lsmMod, only : vic411_struc
```

ARGUMENTS:

DESCRIPTION:

This routine cleans up the allocated memory structures in the VIC 4.1.1 lsm.
The arguments are:

n index of the nest

78.3 Fortran: Module Interface vic411_lsmMod (Source File: vic411_lsmMod.F90)

This module provides the definition of derived data type used to control the operation of the VIC LSM. It also provides the entry method for the initialization of VIC-specific variables. The derived data type `vic411_struc` includes the variables that specify the runtime options and other control variables as described below:

vicopen variable to keep track of opened files

numout number of output times

rstInterval restart writing interval

ts model time-step

snowstep snow-step for water-balance mode

tscount time-step count, used to know which snow-step is being processed when running in water-balance mode.

veg_tiling_scheme specifies whether sub-grid tiling is determined by LIS or by VIC.
0 = VIC
1 = LIS

NT total number of vegetation types for the vegetation parameter file

debugging_convert_units debugging flag – remove specifies whether to convert the units of temperature from Celcius to Kelvin.
0 = do not convert
1 = convert
Final version of forcing reader will always perform this conversion.

MAX_SNOW_TEMP maximum snow temperature

global_param VIC's global parameter file/configuration file When running in energy-balance mode, VIC should run every time-step. However, when running in water-balance mode, VIC runs daily, but it expects to be given an array of forcing values (sized by the number of snow-steps). This alarm is necessary to allow LIS to run at a sub-daily time-step to gather all the forcing needed to run VIC daily.

vic VIC LSM specific variables

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
14 Aug 2013; Shugong Wang, Implementation of VIC 4.1.1 into LIS-7.
```

USES:

```
use vic411_module

implicit none

PRIVATE
-------------------------------------------------------------------------------
```

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
public :: vic411_lsm_ini
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
public :: vic411_struc
```

78.3.1 vic411_lsm_ini (Source File: vic411_lsmMod.F90)

INTERFACE:

```
subroutine vic411_lsm_ini()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod,   only : LIS_clock, LIS_calendar, &
                             LIS_update_timestep, LIS_registerAlarm
use LIS_surfaceModelDataMod, only : LIS_sfmodel_struc
use LIS_logMod,       only : LIS_verify
```

DESCRIPTION:

78.3.2 vic411_main (Source File: vic411_main.F90)

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
03 Apr 2012; Shugong Wang, add support to restart file with MPI-safe tile id
03 Sep 2012; Shugong Wang, add support to output VIC PET variables
```

INTERFACE:

```
subroutine vic411_main(n)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_timeMgrMod,     only : LIS_isAlarmRinging
use LIS_histDataMod
use vic411_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This is the entry point for calling the VIC 4.1.1 LSM physics.
The arguments are:

n index of the nest

78.4 Fortran: Module Interface vic411_module.F90 (Source File: vic411_module.F90)

The code in this file provides a description of the data structure containing the VIC 4.1.1 1-d variables. The variables specified in the data structure include:

min_Tfactor Minimum change in temperature due to elevation (C) in each snow elevation band. This value is determined by finding the minimum value of Tfactor from VIC's soil_con data structure.

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
```

78.4.1 vic411_readcard (Source File: vic411_readcard.F90)

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
```

INTERFACE:

```
subroutine vic411_readcard()
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_config
use LIS_timeMgrMod, only : LIS_parseTimeString
use LIS_logMod,     only : LIS_logunit, LIS_verify, LIS_endrun
use vic411_lsmMod,  only : vic411_struc
```

DESCRIPTION:

This routine reads the options specific to VIC 4.1.1 LSM option from the LIS configuration file.

78.4.2 vic411_readrst (Source File: vic411_readrst.F90)

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
```

INTERFACE:

```
subroutine vic411_readrst()
```

USES:

```
implicit none
```

ARGUMENTS:

DESCRIPTION:

This program reads restart files for VIC. This includes all relevant water/energy storages and tile information. Actually, VIC contains support to read its initial state file/restart file in the initialize_model_state routine. This support cannot be easily pulled out into its own routine because the support to process the initial state file/restart file requires air temperature to be present. (stand-alone VIC completely processes all forcing data before it initializes the model states.)
Thus reading a restart file is handled in the vic411_run routine at the first time-step of the run.

78.4.3 vic411_setup (Source File: vic411_setup.F90)

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
01 Jun 2012; Shugong Wang, Retrieve vegetation classes of grid and
                           send to C routine to support COMPUTE_TREELINE
14 Aug 2013; Shugong Wang, modifed for LIS-7
```

INTERFACE:

```
subroutine vic411_setup()
```

USES:

```
   use LIS_coreMod,   only : LIS_rc, LIS_domain
   use vic411_lsmMod, only : vic411_struc
   use LIS_LMLCMod,   only : LIS_LMLC         ! added by Shugong Wang
```

ARGUMENTS:

DESCRIPTION:

This routine initialized VIC's internal data structures.

78.4.4 vic411_writerst (Source File: vic411_writerst.F90)

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
01 May 2012; Shugong Wang. Add vegetation tiling scheme as an option, to support
             writing restart file using VIC based vegetation tiling and LIS based
             vegetation tiling. When using VIC based vegetation tiling, grid id will
             be the key of restart file. Otherwise, tile id will be the key of restart
             file.
```

INTERFACE:

```
subroutine vic411_writerst(n)
```

USES:

```
   use LIS_coreMod,    only : LIS_rc, LIS_masterproc, LIS_localPet, LIS_npes, LIS_domain
   use LIS_timeMgrMod, only : LIS_isAlarmRinging
   use LIS_logMod,     only : LIS_logunit
   use LIS_fileIOMod,  only : LIS_create_output_directory, &
                              LIS_create_restart_filename
   use vic411_lsmMod,  only : vic411_struc
#if (defined SPMD)
   use LIS_mpiMod
#endif

   implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine writes restart files for VIC. This includes all relevant water/energy storage and tile information. This routine calls VIC's write_model_state routine (via the call to vic411_write_state). When running in parallel, this call must be made by all processes, resulting in one restart file per process. This routine then concatenates these files into one (via system calls to Unix commands).
The arguments are:

n index of the nest

78.4.5 finalize_vic411 (Source File: finalize_vic411.c)

REVISION HISTORY:

```
12 Mar 2012  James Geiger; Initial implementation
```

INTERFACE:

```
void FTN(finalize_vic411)(int * LIS_NEST, int * LIS_NTILES)
```

DESCRIPTION:

This routine frees memory and closes opened files.
The arguments are:

LIS_NEST nest index, considered intent(in)

LIS_NTILES number of sub-grid tiles, considered intent(in)

78.4.6 setup_vic411 (Source File: setup_vic411.c)

REVISION HISTORY:

```
04 Aug 2011  James Geiger; Initial adaptation from VIC 4.1.1 (vicNl.c).
06 Jun 2012  Shugong Wang; Add support to VIC COMPUTE_TREELINE
```

INTERFACE:

```
void FTN(setup_vic411)(int *NGRIDS, int *NTILES,
                       float *lat_array, float *lon_array,
                       int *t2gindex, int *vegclasses,
                       int *vtscheme, int *NT,
                       float *veg_fracs_all,
                       char *global_param_filename, int *dummy)
```

DESCRIPTION:

This routine performs variable initialization. It was adapted from VIC 4.1.1.
For details about variables, input files and subroutines check:
http://ce.washington.edu/ hydro/Lettenmaier/Models/VIC/VIC_home.html
UNITS: unless otherwise marked:
all water balance components are in mm
all energy balance components are in mks
depths and lengths are in m

The arguments are:

NGRIDS number of grid cells, considered intent(in)

NTILES number of sub-grid tiles, considered intent(in)

lat_array array of latitude values for each grid cell, considered intent(in)

lon_array array of longitude values for each grid cell, considered intent(in)

t2gindex array mapping tile indices to parent grid index, considered intent(in)

vegclasses array of vegetation classification values for each tile, considered intent(in)

vtscheme vegetation tiling scheme, considered intent(in)

NT total number of vegetation types for VIC lookup table, considered intent(in)

global_param_filename the name of VIC configuration file

dummy dummy variable to capture length of global_param_filename

---

78.4.7 vic411_add_atmosdata (Source File: vic411_add_atmosdata.c)

REVISION HISTORY:

```
Dec 2011  James Geiger; Initial implementation
```

INTERFACE:

```
void FTN(vic411_add_atmosdata)(int *LIS_tile, int *step, int *VAR,
                               float *forcing)
```

DESCRIPTION:

This routine packs the forcing data extracted from LIS_FORC_State into VIC's atmos data structure. The arguments are:

LIS_TILE tile index, considered intent(in)

step step index, considered intent(in) VIC's forcing data are arrays of sized either by the number of snow-steps when running in water-balance mode or by 1 when running in energy-balance mode. This index specifies which snow-step (or 1 in the case of energy-balance mode) is being added to the atmos data structure.

VAR variable index, considered intent(in)

forcing forcing value, considered intent(in)

78.4.8 vic411_nr_atmosdata (Source File: vic411_add_atmosdata.c)

REVISION HISTORY:

```
Dec 2011  James Geiger; Initial implementation
```

INTERFACE:

```
void FTN(vic411_nr_atmosdata)(int *NTILES)
```

DESCRIPTION:

For water-balance mode (vic411_NR > 0), this routine iterates over every tile and computes the model step average or sum value for the forcing fields.
This routine computes the average or sum based off the most common computations found in VIC's initialize_atmos routine. VIC's forcing data processing is quite complex, so the computations performed here may not exactly match the ones performed by VIC's initialize_atmos routine. Thus the forcing for some VIC test-cases may not reproduced exactly by this routine.
The arguments are:

NTILES number of tiles, considered intent(in)

78.4.9 find_min_tfactor (Source File: vic411_find_min_tfactor.c)

REVISION HISTORY:

```
21 Dec 2011  James Geiger; Initial implementation
```

INTERFACE:

```
double FTN(vic411_find_min_tfactor)(int *LIS_NEST, int *LIS_TILE)
```

DESCRIPTION:

This routine returns the minimum Tfactor value from the soil_con.Tfactor array.
The arguments are:

LIS_NEST nest index, considered intent(in)

LIS_TILE tile index, considered intent(in)

78.4.10 fsvp (Source File: vic411_fsvp.c)

REVISION HISTORY:

```
12 Dec 2011  James Geiger; Initial implementation
```

INTERFACE:

```
double FTN(vic411_fsvp)(double * temp)
```

DESCRIPTION:

This routine returns the saturated vapor pressure (vic411_svp) corresponding to the given temperature by calling VIC's vic411_svp routine.
The arguments are:

temp temperature, in Celcius

78.4.11 get_max_snow_temp (Source File: vic411_get_max_snow_temp.c)

REVISION HISTORY:

```
21 Dec 2011  James Geiger; Initial implementation
```

INTERFACE:

```
double FTN(vic411_get_max_snow_temp)(void)
```

DESCRIPTION:

This routine returns the MAX_SNOW_TEMP value from the global_param data structure.

78.4.12 vic411_lis_read_soilparam_arc (Source File: vic411_lis_read_soilparam_arc.c)

REVISION HISTORY:

```
Sept 2011  James Geiger; Initial implementation
```

INTERFACE:

```
vic411_soil_con_struct vic411_lis_read_soilparam_arc(FILE  *soilparam,
                            char  *soilparamdir,
                                       float *lat,
                                       float *lng)
```

DESCRIPTION:

This routine reads soil parameters for each grid cell from an ASCII ARC/INFO output grid.
This routine was modified from the VIC 4.1.1 read_soilparam_arc.c file.
The arguments are:

soilparam filename of the soil parameter file

soilparamdir directory containing the soil parameter file

lat latitude of the desired grid-cell

lng longitude of the desired grid-cell

78.4.13 lis_scan_soilparam_flat_ascii (Source File: vic411_lis_scan_soilparam_flat_ascii.c)

REVISION HISTORY:

```
Feb 2012  James Geiger; Initial implementation
```

INTERFACE:

```
void vic411_lis_scan_soilparam_flat_ascii(FILE  *soilparam,
                                   float *lat,
                                   float *lng)
```

DESCRIPTION:

This routine scans the flat ASCII soil parameters for the given grid cell, and then resets the position within the FILE stream for subsequent reading.
The arguments are:

soilparam filename of the soil parameter file

lat latitude of the desired grid-cell

lng longitude of the desired grid-cell

78.4.14 vic411_run (Source File: vic411_run.c)

REVISION HISTORY:

```
    02 Aug 2011  James Geiger; Initial implementation
    01 May 2012  Shugong Wang; add LIS_VT_SCHEME to pass vegetation tiling
                               scheme to vic411_run.
    07 May 2012  Shugong Wang; add argument vegclass to calculate
                               MPI-safe tile id
    19 Jul 2013  Shugong Wang; add WRSI and SWI output variables for FLDAS
```

INTERFACE:

```
 void FTN(vic411_run)(int *LIS_TILE, int *LIS_TS, int *LIS_VT_SCHEME,
                      int *LIS_VEGCLASS,
                      int *LIS_NEST,
                      int *LIS_MOC_EVAP,
                      int *LIS_MOC_QS,
                      int *LIS_MOC_QSB,
                      int *LIS_MOC_CANOPINT,
                      int *LIS_MOC_SMLIQFRAC,
                      int *LIS_MOC_SWNET,
                      int *LIS_MOC_ECANOP,
                      int *LIS_MOC_TVEG,
                      int *LIS_MOC_ESOIL,
                      int *LIS_MOC_ARESIST,
                      int *LIS_MOC_AVGSURFT,
                      int *LIS_MOC_ALBEDO,
                      int *LIS_MOC_LWNET,
                      int *LIS_MOC_SUBSNOW,
                      int *LIS_MOC_RADT,
                      int *LIS_MOC_QLE,
                      int *LIS_MOC_QH,
                      int *LIS_MOC_QG,
                      int *LIS_MOC_QF,
                      int *LIS_MOC_ACOND,
                      int *LIS_MOC_BARESOILT,
                      int *LIS_MOC_DELCOLDCONT,
                      int *LIS_MOC_DELINTERCEPT,
                      int *LIS_MOC_DELSOILMOIST,
                      int *LIS_MOC_DELSURFSTOR,
                      int *LIS_MOC_QFZ,
                      int *LIS_MOC_QSM,
```

```
int *LIS_MOC_QV,
int *LIS_MOC_RAINF,
int *LIS_MOC_ROOTMOIST,
int *LIS_MOC_SMFROZFRAC,
int *LIS_MOC_SNFRALBEDO,
int *LIS_MOC_SNOWCOVER,
int *LIS_MOC_SNOWDEPTH,
int *LIS_MOC_SNOWF,
int *LIS_MOC_SNOWT,
int *LIS_MOC_SOILMOIST,
int *LIS_MOC_SWDOWNFORC,
int *LIS_MOC_SWE,
int *LIS_MOC_TAIRFORC,
int *LIS_MOC_TOTALPRECIP,
int *LIS_MOC_VIC_PET_SATSOIL, // PET
int *LIS_MOC_VIC_PET_H2OSURF,
int *LIS_MOC_VIC_PET_SHORT,
int *LIS_MOC_VIC_PET_TALL,
int *LIS_MOC_VIC_PET_NATVEG,
int *LIS_MOC_VIC_PET_VEGNOCR,
int *LIS_MOC_SOILTEMP,
int *LIS_MOC_WRSI,
int *LIS_MOC_SWI)
```

DESCRIPTION:

This routine calls VIC's physics routines and diagnoses output. It is based off VIC's vicNl.c file.
The arguments are:

LIS_TILE sub-grid tile index, considered intent(in)

LIS_TS time-step, considered intent(in)

LIS_VT_SCHEME vegetation tiling scheme:
0 = VIC based tiling;
1 = LIS based tiling

LIS_VEGCLASS land use type from LIS

LIS_NEST nest index, considered intent(in)

LIS_MOC_EVAP output variable index for evaporation, considered intent(in)

LIS_MOC_QS output variable index for surface vic411_runoff, considered intent(in)

LIS_MOC_QSB output variable index for sub-surface vic411_runoff, considered intent(in)

LIS_MOC_CANOPINT output variable index for canopy interception, considered intent(in)

LIS_MOC_SMLIQFRAC output variable index for soil moisture liquid fraction, considered intent(in)

LIS_MOC_SWNET output variable index for shortwave net radiation, considered intent(in)

LIS_MOC_ECANOP output variable index for interception evaporation, considered intent(in)

LIS_MOC_TVEG output variable index for vegetation vic411_transpiration, considered intent(in)

LIS_MOC_ESOIL output variable index for bare soil evaporation, considered intent(in)

LIS_MOC_ARESIST output variable index for aerodynamic resistance, considered intent(in)

LIS_MOC_AVGSURFT output variable index for average surface temperature, considered intent(in)

LIS_MOC_ALBEDO output variable index for albedo, considered intent(in)

LIS_MOC_LWNET output variable index for longwave net radiation, considered intent(in)

LIS_MOC_SUBSNOW output variable index for snow sublimation, considered intent(in)

LIS_MOC_RADT output variable index for surface radiative temperature, considered intent(in)

LIS_MOC_QLE output variable index for latent heat flux, considered intent(in)

LIS_MOC_QH output variable index for sensible heat flux, considered intent(in)

LIS_MOC_QG output variable index for ground heat flux, considered intent(in)

LIS_MOC_QF output variable index for energy of fusion, considered intent(in)

LIS_MOC_ACOND aerodynamic conductance considered intent(in)

LIS_MOC_BARESOILT temperature of bare soil considered intent(in)

LIS_MOC_DELCOLDCONT change in snow water content considered intent(in)

LIS_MOC_DELINTERCEPT change in interception storage considered intent(in)

LIS_MOC_DELSOILMOIST change in soil moisture content considered intent(in)

LIS_MOC_DELSURFSTOR change in surface water storage considered intent(in)

LIS_MOC_QFZ refreezing of water in the snowpack considered intent(in)

LIS_MOC_QSM snowmelt (kg/m2s) considered intent(in)

LIS_MOC_QV energy of sublimation considered intent(in)

LIS_MOC_RAINF rainfall rate considered intent(in)

LIS_MOC_ROOTMOIST root zone soil moisture considered intent(in)

LIS_MOC_SMFROZFRAC average layer fraction of frozen moisture considered intent(in)

LIS_MOC_SNFRALBEDO albedo of snow-covered fraction of grid considered intent(in)

LIS_MOC_SNOWCOVER fraction of snow-covered area in grid considered intent(in)

LIS_MOC_SNOWDEPTH snow depth considered intent(in)

LIS_MOC_SNOWF snowfall rate considered intent(in)

LIS_MOC_SNOWT snow surface temperature considered intent(in)

LIS_MOC_SOILMOIST aoil moisture contents in all soil layers considered intent(in)

LIS_MOC_SWDOWNFORC downward shortwave radiation considered intent(in)

LIS_MOC_SWE snow water equivalent considered intent(in)

LIS_MOC_TAIRFORC air temperature considered intent(in)

LIS_MOC_TOTALPRECIP total precipitation considered intent(in)

78.4.15 vic411_set_vegparam (Source File: vic411_set_vegparam.c)

REVISION HISTORY:

```
22 Feb 2012  James Geiger; Initial adaptation from VIC 4.1.1 (vic411_read_vegparam.c).
01 Jun 2012  Shugong Wang; Add support to COMPUTE_TREELINE option
```

INTERFACE:

```
vic411_veg_con_struct *vic411_set_vegparam(int tile_idx,
                              int vegclass,
                              int Nveg_type,
                              int gridcel,
                              FILE *fp_vegparam,
                              float *veg_fracs,
                              int Nlc_type)
```

DESCRIPTION:

This routine initializes the vegetation parameters veg_con data structure based on the given vegclass value. The arguments are:

tile_idx index of current tile

vegclass vegetation classification, as determined by LIS, of the current tile, considered intent(in)

Nveg_type number of vegetation types in the veg_lib parameter file, considered intent(in)

gridcel grid id in VIC soil file

fp_veg_param file pointer to vegetation parameter file

veg_fracs fractions of all vegetation classes

Nlc_type number of vegetation classes in landcover dataset

78.4.16 vic411_write_atmos (Source File: vic411_write_atmos.c)

REVISION HISTORY:

```
Dec 2011  James Geiger; Initial implementation
```

INTERFACE:

```
void FTN(vic411_write_atmos)(int *LIS_NTILES)
```

DESCRIPTION:

This routine iterates over every grid-cell to write out the values of the forcing data contain within the atmos data structure.
The actual writing is performed by VIC's write_atmosdata routine.
The arguments are:

LIS_NTILES number of sub-grid tiles, considered intent(in)

78.4.17 vic411_bin_atmosdata (Source File: vic411_write_bin_atmosdata.c)

REVISION HISTORY:

```
Nov 2011  James Geiger; Initial implementation
```

INTERFACE:

```
void vic411_write_bin_atmosdata(vic411_atmos_data_struct *atmos, int nrecs,
                        float lat, float lng)
```

DESCRIPTION:

This routine writes atmospheric data to a file in a binary format.
The arguments are:

atmos given atmos data structure

nrecs length of atmos data structure (number of time-step records)

lat latitude of corresponding grid-cell

lng longitude of corresponding grid-cell

78.4.18 write_lakecon (Source File: vic411_write_lakecon.c)

REVISION HISTORY:

```
Sept 2011  James Geiger; Initial implementation
```

INTERFACE:

```
void vic411_write_lakecon(vic411_lake_con_struct *lake_con)
```

DESCRIPTION:

This routine writes contents of the lake_con data structure to a log file, used primarily for debugging.
The arguments are:

lake_con given lake_con data structure

78.4.19 write_prcp (Source File: vic411_write_prcp.c)

REVISION HISTORY:

```
Aug 2011  James Geiger; Initial implementation
```

INTERFACE:

```
void vic411_write_prcp(vic411_dist_prcp_struct *prcp, int Nveg)
```

DESCRIPTION:

This routine writes contents of the prcp data structure to stdout, used primarily for debugging.
The arguments are:

prcp given prcp data structure

Nveg number of vegetation types

78.4.20 vic411_write_state (Source File: vic411_write_state.c)

REVISION HISTORY:

```
09 Feb 2012  James Geiger; Initial implementation
01 May 2012  Shugong Wang; Add vegetation tiling scheme as an argument of
             vic411_write_state.
07 May 2012  Shugong Wang; Add vegclasses as an argument of
             vic411_write_state to support MPI-safe tile id.
```

INTERFACE:

```
void FTN(vic411_write_state)(int * LIS_NEST,
                             int * LIS_NTILES,
                             int *LIS_VT_SCHEME,
                             int *LIS_VEGCLASSES,
                             int * year, int * month, int * day,
                             char * filename, int * lenf)
```

DESCRIPTION:

This routine calls VIC's write_model_state routine to write the restart state.
The arguments are:

LIS_NEST nest index, considered intent(in)

LIS_NTILES number of tiles, considered intent(in)

LIS_VT_SCHEME vic411_flag of vegetation tiling (in)
0 = VIC based tiling;
1 = LIS based tiling

LIS_VEGCLASSES land use types

year current year, considered intent(in)

month current month, considered intent(in)

day current day, considered intent(in)

filename name of the restart file, considered intent(in)

lenf length of the filename string (this is a hidden argument), considered intent(in)

79 VIC land surface model version 4.1.2.l

This section describes the interface implementations for the Variable Infiltration Capacity (VIC) Macroscale Hydrologic Model, version 4.1.2.l, available from
http://www.hydro.washington.edu/Lettenmaier/Models/VIC/index.shtml

79.1 Notes regarding incorporating VIC 4.1.2.l into LIS

VIC's control loop behaves oppositely from that of LIS (and the other land surface models that it contains). LIS' control loop has time as the outer loop and space as the inner loop; whereas, VIC's control loop has space as the outer loop and time as the inner loop.
Viewing this in terms of data, LIS processes the desired running domain as a 2d gridded field that is one time-level deep; whereas, VIC processes the same running domain one grid cell at a time as a one grid cell structure that is many time-levels deep. In other words, for LIS, at time t_1, LIS processes forcing for the whole domain, then it advances the model state for the whole domain. Then LIS advances time to t_2 and repeats. For VIC, VIC selects grid cell x_1, then it processes forcing for that grid cell from the beginning to the end of the simulation, then it advances the model state from the beginning to the end of the simulation. Then VIC selects grid cell x_2 and repeats.
This difference in approach complicated incorporating VIC into LIS. To overcome this without modifying the VIC source code, two steps were taken. Step 1 was to create an intermediate layer of data structures and supporting routines. Step 2 was to tell VIC that the simulation was only one time step long. These two steps allow LIS to pass data into VIC, have VIC advance its model state forward one time step, and to store the new state for use in the next call to VIC at the next LIS time step.

79.2 Limitations of VIC 4.1.2.l within LIS

VIC has many compile time and run time options that modify the behavior of its underlying physics. VIC within LIS (LIS-VIC) has been tested only for VIC's basic water balance configuration and for VIC's basic energy balance configuration. Other configuration options are not guaranteed to work.

79.2.1 vic412_diagnoseoutputvar (Source File: vic412_diag.F90)

REVISION HISTORY:

```
Nov 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
```

INTERFACE:

```
subroutine vic412_diagnoseoutputvar(nest, tile, moc_index, vlevel, value, &
                                    units, lenu, direction, lend)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_histDataMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)             :: nest, tile, moc_index, vlevel
integer, intent(in)             :: lenu, lend
real*8, intent(in)              :: value
character(len=lenu), intent(in) :: units
character(len=lend), intent(in) :: direction
```

DESCRIPTION:

This routine calls LIS_diagnoseOutputVar to process VIC's output variables.
The arguments are:

nest index of the nest

tile index of the tile

moc_index index identifying which output variable to process

vlevel vertical level to process

value output value to process

units units of the output variable

lenu length of the units string

direction direction of the output variable

lend length of the direction string

79.2.2 vic412_dynsetup (Source File: vic412_dynsetup.F90)

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
```

INTERFACE:

```
subroutine vic412_dynsetup(n)
```

USES:

```
implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine sets up the time-dependent variables in VIC. Currently this routine is a placeholder.
The arguments are:

n index of the nest

79.2.3 vic412_f2t (Source File: vic412_f2t.F90)

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
14 Aug 2013; Shugong Wang, Modified for LIS-7
```

INTERFACE:

```
subroutine vic412_f2t(n)
```

USES:

```
use ESMF
use LIS_coreMod,        only : LIS_rc, LIS_surface
use LIS_timeMgrMod,     only : LIS_isAlarmRinging
use LIS_metforcingMod,  only : LIS_FORC_State
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use vic412_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the VIC 4.1.1 model tiles.
The arguments are:

n index of the nest

79.2.4 vic412_finalize (Source File: vic412_finalize.F90)

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
```

INTERFACE:

```
subroutine vic412_finalize()
```

USES:

```
use LIS_coreMod,   only : LIS_rc
use vic412_lsmMod, only : vic412_struc
```

ARGUMENTS:

DESCRIPTION:

This routine cleans up the allocated memory structures in the VIC 4.1.1 lsm.
The arguments are:

n index of the nest

79.3 Fortran: Module Interface vic412_lsmMod (Source File: vic412_lsmMod.F90)

This module provides the definition of derived data type used to control the operation of the VIC LSM. It also provides the entry method for the initialization of VIC-specific variables. The derived data type `vic412_struc` includes the variables that specify the runtime options and other control variables as described below:

vicopen variable to keep track of opened files

numout number of output times

rstInterval restart writing interval

ts model time-step

snowstep snow-step for water-balance mode

tscount time-step count, used to know which snow-step is being processed when running in water-balance mode.

veg_tiling_scheme specifies whether sub-grid tiling is determined by LIS or by VIC.
0 = VIC
1 = LIS

NT total number of vegetation types for the vegetation parameter file

debugging_convert_units debugging flag – remove specifies whether to convert the units of temperature from Celcius to Kelvin.
0 = do not convert
1 = convert
Final version of forcing reader will always perform this conversion.

MAX_SNOW_TEMP maximum snow temperature

global_param VIC's global parameter file/configuration file When running in energy-balance mode, VIC should run every time-step. However, when running in water-balance mode, VIC runs daily, but it expects to be given an array of forcing values (sized by the number of snow-steps). This alarm is necessary to allow LIS to run at a sub-daily time-step to gather all the forcing needed to run VIC daily.

vic VIC LSM specific variables

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
14 Aug 2013; Shugong Wang, Implementation of VIC 4.1.1 into LIS-7.
```

USES:

```
use vic412_module

implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: vic412_lsm_ini
```

PUBLIC TYPES:

```
public :: vic412_struc
```

79.3.1 vic412_lsm_ini (Source File: vic412_lsmMod.F90)

INTERFACE:

```
subroutine vic412_lsm_ini()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod,   only : LIS_clock, LIS_calendar, &
                             LIS_update_timestep, LIS_registerAlarm
use LIS_surfaceModelDataMod, only : LIS_sfmodel_struc
use LIS_logMod,       only : LIS_verify
```

DESCRIPTION:

79.3.2 vic412_main (Source File: vic412_main.F90)

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
03 Apr 2012; Shugong Wang, add support to restart file with MPI-safe tile id
03 Sep 2012; Shugong Wang, add support to output VIC PET variables
```

INTERFACE:

```
subroutine vic412_main(n)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_timeMgrMod,     only : LIS_isAlarmRinging
use LIS_histDataMod
use vic412_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This is the entry point for calling the VIC 4.1.1 LSM physics.
The arguments are:

n index of the nest

79.4 Fortran: Module Interface vic412_module.F90 (Source File: vic412_module.F90)

The code in this file provides a description of the data structure containing the VIC 4.1.1 1-d variables. The variables specified in the data structure include:

min_Tfactor Minimum change in temperature due to elevation (C) in each snow elevation band. This value is determined by finding the minimum value of Tfactor from VIC's soil_con data structure.

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
```

79.4.1 vic412_readcard (Source File: vic412_readcard.F90)

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
```

INTERFACE:

```
subroutine vic412_readcard()
```

USES:

```
use ESMF
use LIS_coreMod,    only : LIS_rc, LIS_config
use LIS_timeMgrMod, only : LIS_parseTimeString
use LIS_logMod,     only : LIS_logunit, LIS_verify, LIS_endrun
use vic412_lsmMod,  only : vic412_struc
```

DESCRIPTION:

This routine reads the options specific to VIC 4.1.1 LSM option from the LIS configuration file.

79.4.2 vic412_readrst (Source File: vic412_readrst.F90)

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
15 May 2014; Shugong Wang, Add NetCDF restart support to VIC 4.1.2.1 for LIS 7
```

INTERFACE:

```
subroutine vic412_readrst()
```

USES:

```
    use LIS_coreMod, only    : LIS_rc, LIS_masterproc, LIS_domain
    use LIS_historyMod, only : LIS_readvar_restart
    use LIS_logMod, only     : LIS_logunit, LIS_endrun, &
                               LIS_getNextUnitNumber,   &
                               LIS_releaseUnitNumber,   &
                               LIS_verify
    use vic412_lsmMod
    use netcdf
  implicit none
```

ARGUMENTS:

DESCRIPTION:

This program reads restart files for VIC. This includes all relevant water/energy storages and tile information.

79.4.3 vic412_setup (Source File: vic412_setup.F90)

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
01 Jun 2012; Shugong Wang, Retrieve vegetation classes of grid and
                           send to C routine to support COMPUTE_TREELINE
14 Aug 2013; Shugong Wang, modifed for LIS-7
```

INTERFACE:

```
subroutine vic412_setup()
```

USES:

```
use LIS_coreMod,   only : LIS_rc, LIS_domain
use vic412_lsmMod, only : vic412_struc
use LIS_LMLCMod,   only : LIS_LMLC         ! added by Shugong Wang
use LIS_surfaceModelDataMod
```

ARGUMENTS:

DESCRIPTION:

This routine initialized VIC's internal data structures.

79.4.4 vic412_writerst (Source File: vic412_writerst.F90)

REVISION HISTORY:

```
02 Aug 2011; James Geiger, Initial implementation of VIC 4.1.1 into LIS.
01 May 2012; Shugong Wang. Add vegetation tiling scheme as an option, to support
             writing restart file using VIC based vegetation tiling and LIS based
             vegetation tiling. When using VIC based vegetation tiling, grid id will
             be the key of restart file. Otherwise, tile id will be the key of restart
             file.
```

INTERFACE:

```
subroutine vic412_writerst(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_masterproc, LIS_localPet, LIS_npes, LIS_domain
use LIS_timeMgrMod, only : LIS_isAlarmRinging
use LIS_logMod,     only : LIS_logunit, LIS_verify
```

```
   use LIS_fileIOMod,  only : LIS_create_output_directory, &
                              LIS_create_restart_filename
   use vic412_lsmMod,  only : vic412_struc
   use LIS_historyMod
   use netcdf
#if (defined SPMD)
   use LIS_mpiMod
#endif

   implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine writes restart files for VIC. This includes all relevant water/energy storage and tile information. This routine calls VIC's write_model_state routine (via the call to vic412_write_state). When running in parallel, this call must be made by all processes, resulting in one restart file per process. This routine then concatenates these files into one (via system calls to Unix commands).
The arguments are:

n index of the nest

79.4.5 finalize_vic412 (Source File: finalize_vic412.c)

REVISION HISTORY:

```
   12 Mar 2012  James Geiger; Initial implementation
```

INTERFACE:

```
void FTN(finalize_vic412)(int * LIS_NEST, int * LIS_NTILES)
```

DESCRIPTION:

This routine frees memory and closes opened files.
The arguments are:

LIS_NEST nest index, considered intent(in)

LIS_NTILES number of sub-grid tiles, considered intent(in)

79.4.6 find_min_tfactor (Source File: find_min_tfactor.c)

REVISION HISTORY:

```
   21 Dec 2011  James Geiger; Initial implementation
```

INTERFACE:

```
double FTN(find_min_tfactor)(int *LIS_NEST, int *LIS_TILE)
```

DESCRIPTION:

This routine returns the minimum Tfactor value from the soil_con.Tfactor array.
The arguments are:

LIS_NEST nest index, considered intent(in)

LIS_TILE tile index, considered intent(in)

79.4.7 fsvp (Source File: fsvp.c)

REVISION HISTORY:

```
12 Dec 2011  James Geiger; Initial implementation
```

INTERFACE:

```
double FTN(fsvp)(double * temp)
```

DESCRIPTION:

This routine returns the saturated vapor pressure (svp) corresponding to the given temperature by calling VIC's svp routine.
The arguments are:

temp temperature, in Celcius

79.4.8 get_max_snow_temp (Source File: get_max_snow_temp.c)

REVISION HISTORY:

```
21 Dec 2011  James Geiger; Initial implementation
```

INTERFACE:

```
double FTN(get_max_snow_temp)(void)
```

DESCRIPTION:

This routine returns the MAX_SNOW_TEMP value from the global_param data structure.

79.4.9 lis_scan_soilparam_flat_ascii (Source File: lis_scan_soilparam_flat_ascii.c)

REVISION HISTORY:

```
Feb 2012  James Geiger; Initial implementation
```

INTERFACE:

```
void lis_scan_soilparam_flat_ascii(FILE  *soilparam,
                                   float *lat,
                                   float *lng)
```

DESCRIPTION:

This routine scans the flat ASCII soil parameters for the given grid cell, and then resets the position within the FILE stream for subsequent reading.
The arguments are:

soilparam filename of the soil parameter file

lat latitude of the desired grid-cell

lng longitude of the desired grid-cell

79.4.10 set_vegparam (Source File: set_vegparam.c)

REVISION HISTORY:

```
22 Feb 2012  James Geiger; Initial adaptation from VIC 4.1.1 (read_vegparam.c).
01 Jun 2012  Shugong Wang; Add support to COMPUTE_TREELINE option
```

INTERFACE:

```
veg_con_struct *set_vegparam(int tile_idx,
                             int vegclass,
                             int Nveg_type,
                             int gridcel,
                             FILE *fp_vegparam,
                             float *veg_fracs,
                             int Nlc_type)
```

DESCRIPTION:

This routine initializes the vegetation parameters veg_con data structure based on the given vegclass value.
The arguments are:

tile_idx index of current tile

vegclass vegetation classification, as determined by LIS, of the current tile, considered intent(in)

Nveg_type number of vegetation types in the veg_lib parameter file, considered intent(in)

gridcel grid id in VIC soil file

fp_veg_param file pointer to vegetation parameter file

veg_fracs fractions of all vegetation classes

Nlc_type number of vegetation classes in landcover dataset

79.4.11 setup_vic412 (Source File: setup_vic412.c)

REVISION HISTORY:

```
04 Aug 2011  James Geiger; Initial adaptation from VIC 4.1.1 (vicNl.c).
06 Jun 2012  Shugong Wang; Add support to VIC COMPUTE_TREELINE
```

INTERFACE:

```
void FTN(setup_vic412)(int *NGRIDS, int *NTILES,
                       float *lat_array, float *lon_array,
                       int *t2gindex, int *vegclasses,
                       int *vtscheme, int *NT,
                       float *veg_fracs_all,
                       int *syr, int *smn, int *sda, int *shr,
                       int *eyr, int *emn, int *eda, int *ehr,
                       float *ts, float *outInterval,
                       int *vic412_state_chunk_size,
                       char *global_param_filename,
                       int *dummy)
```

DESCRIPTION:

This routine performs variable initialization. It was adapted from VIC 4.1.1.
For details about variables, input files and subroutines check:
http://ce.washington.edu/ hydro/Lettenmaier/Models/VIC/VIC_home.html
UNITS: unless otherwise marked:
all water balance components are in mm
all energy balance components are in mks
depths and lengths are in m

The arguments are:

NGRIDS number of grid cells, considered intent(in)

NTILES number of sub-grid tiles, considered intent(in)

lat_array array of latitude values for each grid cell, considered intent(in)

lon_array array of longitude values for each grid cell, considered intent(in)

t2gindex array mapping tile indices to parent grid index, considered intent(in)

vegclasses array of vegetation classification values for each tile, considered intent(in)

vtscheme vegetation tiling scheme, considered intent(in)

NT total number of vegetation types for VIC lookup table, considered intent(in)

global_param_filename the name of VIC configuration file

dummy dummy variable to capture length of global_param_filename

---

79.4.12 vic412_add_atmosdata (Source File: vic412_add_atmosdata.c)

REVISION HISTORY:

```
Dec 2011  James Geiger; Initial implementation
```

INTERFACE:

```
void FTN(vic412_add_atmosdata)(int *LIS_tile, int *step, int *VAR,
                               float *forcing)
```

DESCRIPTION:

This routine packs the forcing data extracted from LIS_FORC_State into VIC's atmos data structure.
The arguments are:

LIS_TILE tile index, considered intent(in)

step step index, considered intent(in) VIC's forcing data are arrays of sized either by the number of snow-steps when running in water-balance mode or by 1 when running in energy-balance mode. This index specifies which snow-step (or 1 in the case of energy-balance mode) is being added to the atmos data structure.

VAR variable index, considered intent(in)

forcing forcing value, considered intent(in)

79.4.13 vic412_nr_atmosdata (Source File: vic412_add_atmosdata.c)

REVISION HISTORY:

```
Dec 2011  James Geiger; Initial implementation
```

INTERFACE:

```
void FTN(vic412_nr_atmosdata)(int *NTILES)
```

DESCRIPTION:

For water-balance mode (NR > 0), this routine iterates over every tile and computes the model step average or sum value for the forcing fields.
This routine computes the average or sum based off the most common computations found in VIC's initialize_atmos routine. VIC's forcing data processing is quite complex, so the computations performed here may not exactly match the ones performed by VIC's initialize_atmos routine. Thus the forcing for some VIC test-cases may not reproduced exactly by this routine.
The arguments are:

NTILES number of tiles, considered intent(in)

79.4.14 vic412_initialize (Source File: vic412_intialize.c)

REVISION HISTORY:

```
02 Aug 2011  James Geiger; Initial implementation
01 May 2012  Shugong Wang; add LIS_VT_SCHEME to pass vegetation tiling
                           scheme to vic412_initialize.
07 May 2012  Shugong Wang; add argument vegclass to calculate MPI-safe
                           tile id
19 Jul 2013  Shugong Wang; add WRSI and SWI output variables for FLDAS
20 May 2014  Shugong Wang;
```

INTERFACE:

```
void FTN(vic412_initialize)(int *LIS_TILE,
                            int *LIS_TS,
                            int *LIS_VT_SCHEME,
                            int *LIS_VEGCLASS,
                            int *LIS_NEST,
                            float *LIS_STATES,
                            int * LIS_START_MODE)
```

DESCRIPTION:

This routine calls VIC's physics routines and diagnoses output. It is based off VIC's vicNl.c file.
The arguments are:

LIS_TILE sub-grid tile index, considered intent(in)

LIS_TS time-step, considered intent(in)

LIS_VT_SCHEME vegetation tiling scheme:
0 = VIC based tiling;
1 = LIS based tiling

LIS_VEGCLASS land use type from LIS

LIS_NEST nest index, considered intent(in)

79.4.15 vic412_run (Source File: vic412_run.c)

REVISION HISTORY:

```
02 Aug 2011  James Geiger; Initial implementation
01 May 2012  Shugong Wang; add LIS_VT_SCHEME to pass vegetation tiling scheme to vic412_run.
07 May 2012  Shugong Wang; add argument vegclass to calculate MPI-safe tile id
19 Jul 2013  Shugong Wang; add WRSI and SWI output variables for FLDAS
```

INTERFACE:

```
void FTN(vic412_run)(int *LIS_TILE, int *LIS_TS, int *LIS_VT_SCHEME, int *LIS_VEGCLASS,
                     int *LIS_NEST,
                     int *LIS_MOC_EVAP,
                     int *LIS_MOC_QS,
                     int *LIS_MOC_QSB,
```

```
int *LIS_MOC_CANOPINT,
int *LIS_MOC_SMLIQFRAC,
int *LIS_MOC_SWNET,
int *LIS_MOC_ECANOP,
int *LIS_MOC_TVEG,
int *LIS_MOC_ESOIL,
int *LIS_MOC_ARESIST,
int *LIS_MOC_AVGSURFT,
int *LIS_MOC_ALBEDO,
int *LIS_MOC_LWNET,
int *LIS_MOC_SUBSNOW,
int *LIS_MOC_RADT,
int *LIS_MOC_QLE,
int *LIS_MOC_QH,
int *LIS_MOC_QG,
int *LIS_MOC_QF,
int *LIS_MOC_ACOND,
int *LIS_MOC_BARESOILT,
int *LIS_MOC_DELCOLDCONT,
int *LIS_MOC_DELINTERCEPT,
int *LIS_MOC_DELSOILMOIST,
int *LIS_MOC_DELSURFSTOR,
int *LIS_MOC_QFZ,
int *LIS_MOC_QSM,
int *LIS_MOC_QV,
int *LIS_MOC_RAINF,
int *LIS_MOC_ROOTMOIST,
int *LIS_MOC_SMFROZFRAC,
int *LIS_MOC_SNFRALBEDO,
int *LIS_MOC_SNOWCOVER,
int *LIS_MOC_SNOWDEPTH,
int *LIS_MOC_SNOWF,
int *LIS_MOC_SNOWT,
int *LIS_MOC_SOILMOIST,
int *LIS_MOC_SWDOWNFORC,
int *LIS_MOC_SWE,
int *LIS_MOC_TAIRFORC,
int *LIS_MOC_TOTALPRECIP,
int *LIS_MOC_VIC_PET_SATSOIL, // PET
int *LIS_MOC_VIC_PET_H2OSURF,
int *LIS_MOC_VIC_PET_SHORT,
int *LIS_MOC_VIC_PET_TALL,
int *LIS_MOC_VIC_PET_NATVEG,
int *LIS_MOC_VIC_PET_VEGNOCR,
int *LIS_MOC_SOILTEMP,
int *LIS_MOC_WRSI,
int *LIS_MOC_SWI)
```

DESCRIPTION:

This routine calls VIC's physics routines and diagnoses output. It is based off VIC's vicNl.c file. The arguments are:

LIS_TILE sub-grid tile index, considered intent(in)

LIS_TS time-step, considered intent(in)

LIS_VT_SCHEME vegetation tiling scheme:
0 = VIC based tiling;
1 = LIS based tiling

LIS_VEGCLASS land use type from LIS

LIS_NEST nest index, considered intent(in)

LIS_MOC_EVAP output variable index for evaporation, considered intent(in)

LIS_MOC_QS output variable index for surface runoff, considered intent(in)

LIS_MOC_QSB output variable index for sub-surface runoff, considered intent(in)

LIS_MOC_CANOPINT output variable index for canopy interception, considered intent(in)

LIS_MOC_SMLIQFRAC output variable index for soil moisture liquid fraction, considered intent(in)

LIS_MOC_SWNET output variable index for shortwave net radiation, considered intent(in)

LIS_MOC_ECANOP output variable index for interception evaporation, considered intent(in)

LIS_MOC_TVEG output variable index for vegetation transpiration, considered intent(in)

LIS_MOC_ESOIL output variable index for bare soil evaporation, considered intent(in)

LIS_MOC_ARESIST output variable index for aerodynamic resistance, considered intent(in)

LIS_MOC_AVGSURFT output variable index for average surface temperature, considered intent(in)

LIS_MOC_ALBEDO output variable index for albedo, considered intent(in)

LIS_MOC_LWNET output variable index for longwave net radiation, considered intent(in)

LIS_MOC_SUBSNOW output variable index for snow sublimation, considered intent(in)

LIS_MOC_RADT output variable index for surface radiative temperature, considered intent(in)

LIS_MOC_QLE output variable index for latent heat flux, considered intent(in)

LIS_MOC_QH output variable index for sensible heat flux, considered intent(in)

LIS_MOC_QG output variable index for ground heat flux, considered intent(in)

LIS_MOC_QF output variable index for energy of fusion, considered intent(in)

LIS_MOC_ACOND aerodynamic conductance considered intent(in)

LIS_MOC_BARESOILT temperature of bare soil considered intent(in)

LIS_MOC_DELCOLDCONT change in snow water content considered intent(in)

LIS_MOC_DELINTERCEPT change in interception storage considered intent(in)

LIS_MOC_DELSOILMOIST change in soil moisture content considered intent(in)

LIS_MOC_DELSURFSTOR change in surface water storage considered intent(in)

LIS_MOC_QFZ refreezing of water in the snowpack considered intent(in)

LIS_MOC_QSM snowmelt (kg/m2s) considered intent(in)

LIS_MOC_QV energy of sublimation considered intent(in)

LIS_MOC_RAINF rainfall rate considered intent(in)

LIS_MOC_ROOTMOIST root zone soil moisture considered intent(in)

LIS_MOC_SMFROZFRAC average layer fraction of frozen moisture considered intent(in)

LIS_MOC_SNFRALBEDO albedo of snow-covered fraction of grid considered intent(in)

LIS_MOC_SNOWCOVER fraction of snow-covered area in grid considered intent(in)

LIS_MOC_SNOWDEPTH snow depth considered intent(in)

LIS_MOC_SNOWF snowfall rate considered intent(in)

LIS_MOC_SNOWT snow surface temperature considered intent(in)

LIS_MOC_SOILMOIST aoil moisture contents in all soil layers considered intent(in)

LIS_MOC_SWDOWNFORC downward shortwave radiation considered intent(in)

LIS_MOC_SWE snow water equivalent considered intent(in)

LIS_MOC_TAIRFORC air temperature considered intent(in)

LIS_MOC_TOTALPRECIP total precipitation considered intent(in)

79.4.16 vic412_write_atmos (Source File: vic412_write_atmos.c)

REVISION HISTORY:

```
Dec 2011  James Geiger; Initial implementation
```

INTERFACE:

```
void FTN(vic412_write_atmos)(int *LIS_NTILES)
```

DESCRIPTION:

This routine iterates over every grid-cell to write out the values of the forcing data contain within the atmos data structure.
The actual writing is performed by VIC's write_atmosdata routine.
The arguments are:

LIS_NTILES number of sub-grid tiles, considered intent(in)

79.4.17 vic412_write_state (Source File: vic412_write_state.c)

REVISION HISTORY:

```
09 Feb 2012  James Geiger; Initial implementation
01 May 2012  Shugong Wang; Add vegetation tiling scheme as an argument of
                           vic412_write_state.
07 May 2012  Shugong Wang; Add vegclasses as an argument of
                           vic412_write_state to support MPI-safe tile id.
15 May 2014  Shugong Wang; Use LIS 7 to deal with restart writing
```

INTERFACE:

```
void FTN(vic412_write_state)(int * LIS_NEST,
                             int * LIS_NTILES,
                             int *LIS_VT_SCHEME,
                             int *LIS_VEGCLASSES,
                             int * year, int * month, int * day,
                             //char * filename, int * lenf,
                             float *state_chunk, int *tid)
```

DESCRIPTION:

This routine calls VIC's write_model_state routine to write the restart state.
The arguments are:

LIS_NEST nest index, considered intent(in)

LIS_NTILES number of tiles, considered intent(in)

LIS_VT_SCHEME flag of vegetation tiling (in)
0 = VIC based tiling;
1 = LIS based tiling

LIS_VEGCLASSES land use types

year current year, considered intent(in)

month current month, considered intent(in)

day current day, considered intent(in)

filename name of the restart file, considered intent(in)

lenf length of the filename string (this is a hidden argument), considered intent(in)

79.4.18 vic412_bin_atmosdata (Source File: write_bin_atmosdata.c)

REVISION HISTORY:

```
Nov 2011  James Geiger; Initial implementation
```

INTERFACE:

```
void write_bin_atmosdata(atmos_data_struct *atmos, int nrecs,
                         float lat, float lng)
```

DESCRIPTION:

This routine writes atmospheric data to a file in a binary format.
The arguments are:

atmos given atmos data structure

nrecs length of atmos data structure (number of time-step records)

lat latitude of corresponding grid-cell

lng longitude of corresponding grid-cell

79.4.19 write_lakecon (Source File: write_lakecon.c)

REVISION HISTORY:

```
Sept 2011  James Geiger; Initial implementation
```

INTERFACE:

```
void write_lakecon(lake_con_struct *lake_con)
```

DESCRIPTION:

This routine writes contents of the lake_con data structure to a log file, used primarily for debugging.
The arguments are:

lake_con given lake_con data structure

79.4.20 write_prcp (Source File: write_prcp.c)

REVISION HISTORY:

```
Aug 2011  James Geiger; Initial implementation
```

INTERFACE:

```
void write_prcp(dist_prcp_struct *prcp, int Nveg)
```

DESCRIPTION:

This routine writes contents of the prcp data structure to stdout, used primarily for debugging.
The arguments are:

prcp given prcp data structure

Nveg number of vegetation types

References

[1] CLM. http://www.cgd.ucar.edu/tss/clm.

[2] Noah. ftp://ftp.ncep.noaa.gov/pub/gcp/ldas/noahlsm/.

[3] M.-D. Chous. A solar radiation model for use in climate studies. *Journal of Atmospheric Science*, 49:762–772, 1992.

[4] G. J. Collatz, C Grivet, J. T. Ball, and J. A. Berry. Physiological and environmental regulation of stomatal conducatance: Photosynthesis and transpiration: A model that includes a laminar boundary layer. *Agric. For. Meteorol.*, 5:107–136, 1991.

[5] B. Cosgrove, D. Lohmann, K. E. Mitchell, P. R. Houser, E. F. Wood, J. C. Schaake, A. Robock, C. Marshall, J. Sheffield, Q. Duan, L. Luo, R. W. Higgins, R. T. Pinker, J. D. Tarpley, and J. Meng. Real-time and retrospective forcing in the north american land data assimilation system (NLDAS) project. *Journal of Geophysical Research*, 108, 2003.

[6] J. C. Derber, D.F. Parrish, and S.J. Lord. The new global operational analysis system at the national meteorological center. *Weather and Forecasting*, 6:538–547, 1991.

[7] Chen. F., Mitchell. K., Schaake. J, Xue. J, Pan. H, Koren. V., Ek. M Duan, and A. Betts. Modeling of land-surface evaporation by four schemes and comparison with fife observations. *J. Geophys. Res.*, 101(D3):7251–7268, 1996.

[8] P. G. Jarvis. The interpretation of leaf water potential and stomatal conductance found in canopies of the field. *Phil. Trans. R. Soc.*, 273:593–610, 1976.

[9] J.-J. Morcrette, L. Smith, and Y. Fouquart. Pressure and temperature dependence of the absorption in longwave radiation parameterizations. *Beiträge zur Physik der Atmosphäre*, 59:455–469, 1986.

[10] J.J. Morcrette. Radiation and cloud radiative properties in the ECMWF opertational weather forecast model. *Journal of Geophysical Research*, 96D:9121–9132, 1991.

[11] L. A. Richards. Capillary conduction of liquids in porous media. *Physics*, 1:318–333, 1931.

[12] E. Rogers, T. L. Black, D. G. Deaven, G. J. DiMego, Q. Zhao, M. Baldwin, N. W. Junker, and Y. Lin. Changes to the operational "early" eta analysis/forecast system at the national centers of environmental prediction. *Wea. Forecasting*, 11:391–413, 1996.

[13] L.S. Rothman, R.R. Gamache, A. Barbe, A. Goldman, J.R. Gillis, L.R. Brown, R.A. Roth, J.-M. Flaud, and C. Camy-Peyret. Afgl atmospheric absorption line parameters compilation. *Applied Optics*, 22:2247–2256, 1982.

[14] M. D. Schwarzkopf and S. B.Fels. The simplified exchange method revisited: An accurate rapid method for computation of infrared cooling rates and fluxes. *Journal of Geophysical Research*, 96:9075–9096, 1991.