Land Information System

LIS Reference Manual

June 8, 2010

Revision 1.0

History:

Revision	Summary of Changes	Date
1.0	LIS 6.0 Public Release	June 8, 2010

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

Contents

I LIS Overview **31**

1 Overview of LIS **32**

II LIS offline program **33**

2 LIS offline program **34**
2.0.1 lisdrv (Source File: lisdrv.F90) . . . 34

III LIS core structures and methods **35**

3 LIS core structures and methods **36**
3.1 Fortran: Module Interface LIS_DAobservationsMod (Source File: LIS_DAobservationsMod.F90) 36
3.1.1 LIS_initDAobservations (Source File: LIS_DAobservationsMod.F90) . . . 36
3.1.2 LIS_readDAobservations (Source File: LIS_DAobservationsMod.F90) . . . 37
3.1.3 LIS_perturb_DAobservations (Source File: LIS_DAobservationsMod.F90) . . . 37
3.2 Fortran: Module Interface LIS_LMLCMod (Source File: LIS_LMLCMod.F90) . . . 39
3.2.1 Overview . . . 39
3.2.2 LIS_LMLC_init (Source File: LIS_LMLCMod.F90) . . . 39
3.2.3 LIS_diagnoselandmask (Source File: LIS_LMLCMod.F90) . . . 40
3.2.4 LIS_diagnoseLandcover (Source File: LIS_LMLCMod.F90) . . . 40
3.3 Fortran: Module Interface LIS_PRIV_gridMod (Source File: LIS_PRIV_gridMod.F90) . . . 41
3.3.1 Overview . . . 41
3.4 Fortran: Module Interface LIS_PRIV_rcMod (Source File: LIS_PRIV_rcMod.F90) . . . 41
3.4.1 LIS_PRIV_tileMod (Source File: LIS_PRIV_tileMod.F90) . . . 51
3.4.2 Overview . . . 51
3.5 Fortran: Module Interface LIS_RTMMod (Source File: LIS_RTMMod.F90) . . . 51
3.5.1 LIS_RTM_init (Source File: LIS_RTMMod.F90) . . . 51
3.5.2 LIS_RTM_run (Source File: LIS_RTMMod.F90) . . . 52
3.5.3 LIS_RTM_output (Source File: LIS_RTMMod.F90) . . . 52
3.5.4 LIS_RTM_final (Source File: LIS_RTMMod.F90) . . . 52
3.6 Fortran: Module Interface LIS_albedoMod (Source File: LIS_albedoMod.F90) . . . 52
3.6.1 Overview . . . 53
3.6.2 LIS_albedo_setup (Source File: LIS_albedoMod.F90) . . . 53
3.6.3 LIS_read_mxsnalb (Source File: LIS_albedoMod.F90) . . . 54
3.6.4 LIS_read_albedo (Source File: LIS_albedoMod.F90) . . . 54
3.6.5 LIS_albedo_finalize (Source File: LIS_albedoMod.F90) . . . 55
3.6.6 LIS_diagnosealbedo (Source File: LIS_albedoMod.F90) . . . 55
3.6.7 LIS_albedo_reset (Source File: LIS_albedoMod.F90) . . . 56
3.7 Fortran: Module Interface LIS_baseforcingMod (Source File: LIS_baseforcingMod.F90) . . . 56
3.7.1 Overview . . . 56
3.7.2 LIS_baseforcing_init (Source File: LIS_baseforcingMod.F90) . . . 58
3.7.3 baseforcing_init_offline (Source File: LIS_baseforcingMod.F90) . . . 58
3.7.4 baseforcing_init_coupled (Source File: LIS_baseforcingMod.F90) . . . 58
3.7.5 forcingPerturbSetup (Source File: LIS_baseforcingMod.F90) . . . 59
3.7.6 LIS_perturb_forcing (Source File: LIS_baseforcingMod.F90) . . . 59
3.7.7 LIS_get_base_forcing (Source File: LIS_baseforcingMod.F90) . . . 60

3.7.8 LIS_baseforcing_reset (Source File: LIS_baseforcingMod.F90) 60
3.7.9 LIS_baseforcing_finalize (Source File: LIS_baseforcingMod.F90) 61
3.7.10 allocate_forcing_mem (Source File: LIS_baseforcingMod.F90) 61
3.7.11 get_forcingvar_attributes (Source File: LIS_baseforcingMod.F90) 61
3.7.12 add_forcing_fields (Source File: LIS_baseforcingMod.F90) 62
3.7.13 diagnoseForcingOutput (Source File: LIS_baseforcingMod.F90) 62
3.8 Fortran: Module Interface LIS_constantsMod (Source File: LIS_constantsMod.F90) 63
3.9 Fortran: Module Interface LIS_coreMod (Source File: LIS_coreMod.F90) 64
3.9.1 Overview 64
3.9.2 LIS_config_init (Source File: LIS_coreMod.F90) 66
3.9.3 lisconfig_offline (Source File: LIS_coreMod.F90) 66
3.9.4 lisconfig_coupled1 (Source File: LIS_coreMod.F90) 67
3.9.5 lisconfig_coupled2 (Source File: LIS_coreMod.F90) 67
3.9.6 LIS_ticktime (Source File: LIS_coreMod.F90) 68
3.9.7 LIS_endofrun (Source File: LIS_coreMod.F90) 68
3.9.8 LIS_timeToRunNest (Source File: LIS_coreMod.F90) 69
3.9.9 LIS_resetTimeMgr (Source File: LIS_coreMod.F90) 69
3.9.10 LIS_finalize (Source File: LIS_coreMod.F90) 69
3.9.11 spmd_init_offline (Source File: LIS_coreMod.F90) 70
3.9.12 spmd_init_coupled (Source File: LIS_coreMod.F90) 70
3.9.13 spmd_setup (Source File: LIS_coreMod.F90) 70
3.9.14 spmd_finalize (Source File: LIS_coreMod.F90) 70
3.10 Fortran: Module Interface LIS_dataAssimMod (Source File: LIS_dataAssimMod.F90) 71
3.10.1 Overview 71
3.10.2 LIS_dataassim_init (Source File: LIS_dataAssimMod.F90) 71
3.10.3 LIS_dataassim_run (Source File: LIS_dataAssimMod.F90) 72
3.10.4 LIS_dataassim_output (Source File: LIS_dataAssimMod.F90) 72
3.10.5 LIS_dataassim_finalize (Source File: LIS_dataAssimMod.F90) 73
3.11 Fortran: Module Interface LIS_domainMod (Source File: LIS_domainMod.F90) 73
3.11.1 Overview 73
3.11.2 LIS_domain_init (Source File: LIS_domainMod.F90) 74
3.11.3 LIS_domain_setup (Source File: LIS_domainMod.F90) 74
3.11.4 LIS_quilt_domain (Source File: LIS_domainMod.F90) 75
3.11.5 LIS_setParameterDomainSpecs (Source File: LIS_domainMod.F90) 75
3.11.6 LIS_domain_finalize (Source File: LIS_domainMod.F90) 76
3.12 Fortran: Module Interface LIS_fileIOMod (Source File: LIS_fileIOMod.F90) 76
3.12.1 LIS_readData (Source File: LIS_fileIOMod.F90) 77
3.12.2 LIS_putget (Source File: LIS_fileIOMod.F90) 77
3.12.3 LIS_create_output_directory (Source File: LIS_fileIOMod.F90) 77
3.12.4 LIS_create_output_filename (Source File: LIS_fileIOMod.F90) 78
3.12.5 LIS_create_restart_filename (Source File: LIS_fileIOMod.F90) 79
3.12.6 LIS_create_stats_filename (Source File: LIS_fileIOMod.F90) 80
3.12.7 LIS_create_innov_filename (Source File: LIS_fileIOMod.F90) 80
3.12.8 LIS_create_obs_filename (Source File: LIS_fileIOMod.F90) 81
3.12.9 LIS_create_daspread_filename (Source File: LIS_fileIOMod.F90) 81
3.12.10 putget_int (Source File: LIS_fileIOMod.F90) 82
3.12.11 putget_real (Source File: LIS_fileIOMod.F90) 83
3.12.12 LIS_create_gain_filename (Source File: LIS_fileIOMod.F90) 83
3.12.13 read2Ddata (Source File: LIS_fileIOMod.F90) 84
3.12.14 readLCdata (Source File: LIS_fileIOMod.F90) 84
3.12.15 LIS_readDomainConfigSpecs (Source File: LIS_fileIOMod.F90) 85

3.12.16 LIS_checkDomainExtents (Source File: LIS_fileIOMod.F90) . . . 85
3.13 Fortran: Module Interface LIS_gfracMod (Source File: LIS_gfracMod.F90) . . . 86
3.13.1 Overview . . . 86
3.13.2 LIS_greenness_setup (Source File: LIS_gfracMod.F90) . . . 86
3.13.3 LIS_read_greenness (Source File: LIS_gfracMod.F90) . . . 87
3.13.4 LIS_greenness_finalize (Source File: LIS_gfracMod.F90) . . . 87
3.13.5 LIS_diagnosegfrac (Source File: LIS_gfracMod.F90) . . . 87
3.13.6 LIS_greenness_reset (Source File: LIS_gfracMod.F90) . . . 88
3.14 Fortran: Module Interface LIS_gribMod (Source File: LIS_gribMod.F90) . . . 88
3.14.1 LIS_grib_init (Source File: LIS_gribMod.F90) . . . 89
3.14.2 LIS_grib2_init (Source File: LIS_gribMod.F90) . . . 89
3.14.3 LIS_grib1_finalize (Source File: LIS_gribMod.F90) . . . 89
3.14.4 LIS_grib1_setup1 (Source File: LIS_gribMod.F90) . . . 90
3.14.5 LIS_grib1_setup2 (Source File: LIS_gribMod.F90) . . . 90
3.14.6 LIS_grib1_setup3 (Source File: LIS_gribMod.F90) . . . 90
3.14.7 LIS_gribdat (Source File: LIS_gribMod.F90) . . . 91
3.14.8 LIS_grib2_setup (Source File: LIS_gribMod.F90) . . . 92
3.14.9 LIS_grib2_finalize (Source File: LIS_gribMod.F90) . . . 92
3.14.10 gr2_convert_lon (Source File: LIS_gribMod.F90) . . . 92
3.15 Fortran: Module Interface LIS_gswpMod (Source File: LIS_gswpMod.F90) . . . 93
3.15.1 getgswp_monindex (Source File: LIS_gswpMod.F90) . . . 93
3.15.2 getgswp_timeindex (Source File: LIS_gswpMod.F90) . . . 93
3.16 Fortran: Module Interface LIS_histDataMod (Source File: LIS_histDataMod.F90) . . . 94
3.16.1 LIS_histDataInit (Source File: LIS_histDataMod.F90) . . . 96
3.16.2 get_moc_attributes (Source File: LIS_histDataMod.F90) . . . 97
3.16.3 allocate_dataEntry (Source File: LIS_histDataMod.F90) . . . 97
3.16.4 LIS_diagnoseOutputVar (Source File: LIS_histDataMod.F90) . . . 97
3.16.5 diagnoseDataEntry (Source File: LIS_histDataMod.F90) . . . 98
3.16.6 LIS_resetOutputVars (Source File: LIS_histDataMod.F90) . . . 99
3.16.7 resetOutputVar (Source File: LIS_histDataMod.F90) . . . 99
3.16.8 resetOutputVar (Source File: LIS_histDataMod.F90) . . . 99
3.17 Fortran: Module Interface LIS_historyMod (Source File: LIS_historyMod.F90) . . . 100
3.17.1 Overview . . . 100
3.17.2 LIS_writevar_bin (Source File: LIS_historyMod.F90) . . . 101
3.17.3 LIS_writevar_restart (Source File: LIS_historyMod.F90) . . . 102
3.17.4 LIS_writevar_reduced_tilespace (Source File: LIS_historyMod.F90) . . . 102
3.17.5 LIS_readvar_reduced_tilespace (Source File: LIS_historyMod.F90) . . . 103
3.17.6 LIS_readvar_restart (Source File: LIS_historyMod.F90) . . . 103
3.17.7 LIS_readvar_gridded (Source File: LIS_historyMod.F90) . . . 103
3.17.8 LIS_writevar_gridded (Source File: LIS_historyMod.F90) . . . 104
3.17.9 LIS_writevar_grib1 (Source File: LIS_historyMod.F90) . . . 104
3.17.10 LIS_writevar_grib2 (Source File: LIS_historyMod.F90) . . . 104
3.17.11 LIS_writevar_netcdf (Source File: LIS_historyMod.F90) . . . 105
3.17.12 LIS_writeModelOutput (Source File: LIS_historyMod.F90) . . . 105
3.17.13 writeBinaryOutput (Source File: LIS_historyMod.F90) . . . 106
3.17.14 writeSingleBinaryVar (Source File: LIS_historyMod.F90) . . . 107
3.17.15 writeGrib1Output (Source File: LIS_historyMod.F90) . . . 108
3.17.16 writeSingleGrib1Var (Source File: LIS_historyMod.F90) . . . 108
3.17.17 writeNetcdfOutput (Source File: LIS_historyMod.F90) . . . 109
3.17.18 defineNETCDFheaderVar (Source File: LIS_historyMod.F90) . . . 110
3.17.19 writeSingleNETCDFvar (Source File: LIS_historyMod.F90) . . . 111

3.17.20 writeGrib2Output (Source File: LIS_historyMod.F90) . . . 111
3.17.21 writevar_bin_withstats_real (Source File: LIS_historyMod.F90) . . . 112
3.17.22 writevar_bin_real (Source File: LIS_historyMod.F90) . . . 113
3.17.23 writevar_bin_real_direct (Source File: LIS_historyMod.F90) . . . 114
3.17.24 writevar_restart_int (Source File: LIS_historyMod.F90) . . . 114
3.17.25 writevar_restart_real (Source File: LIS_historyMod.F90) . . . 115
3.17.26 writevar_tile_noensemble_real (Source File: LIS_historyMod.F90) . . . 115
3.17.27 readvar_tile_noensemble_real (Source File: LIS_historyMod.F90) . . . 116
3.17.28 writevar_restart_char (Source File: LIS_historyMod.F90) . . . 117
3.17.29 readvar_restart_int (Source File: LIS_historyMod.F90) . . . 117
3.17.30 readvar_restart_real (Source File: LIS_historyMod.F90) . . . 118
3.17.31 readvar_restart_char (Source File: LIS_historyMod.F90) . . . 118
3.17.32 readvar_1dgridded_real (Source File: LIS_historyMod.F90) . . . 119
3.17.33 readvar_1dgridded_fromvector_real (Source File: LIS_historyMod.F90) . . . 119
3.17.34 readvar_2dgridded_real (Source File: LIS_historyMod.F90) . . . 120
3.17.35 writevar_gridded_real (Source File: LIS_historyMod.F90) . . . 120
3.17.36 writevar_gridded_real_withstats (Source File: LIS_historyMod.F90) . . . 121
3.17.37 writevar_netcdf_withstats_real (Source File: LIS_historyMod.F90) . . . 122
3.17.38 writevar_grib1_withstats_real (Source File: LIS_historyMod.F90) . . . 123
3.17.39 writevar_grib2_withstats_real (Source File: LIS_historyMod.F90) . . . 124
3.17.40 writevar_grib1_withstats_real3d (Source File: LIS_historyMod.F90) . . . 125
3.17.41 LIS_tile2grid (Source File: LIS_historyMod.F90) . . . 127
3.17.42 LIS_write_innov (Source File: LIS_historyMod.F90) . . . 127
3.17.43 LIS_write_dastats (Source File: LIS_historyMod.F90) . . . 128
3.17.44 makepdsn (Source File: LIS_historyMod.F90) . . . 128
3.17.45 stats (Source File: LIS_historyMod.F90) . . . 129
3.17.46 stats_da (Source File: LIS_historyMod.F90) . . . 129
3.17.47 write_stats (Source File: LIS_historyMod.F90) . . . 130
3.17.48 gather_tiled_vector_output (Source File: LIS_historyMod.F90) . . . 130
3.17.49 gather_gridded_output (Source File: LIS_historyMod.F90) . . . 131
3.17.50 gather_gridded_vector_output (Source File: LIS_historyMod.F90) . . . 131
3.18 Fortran: Module Interface LIS_laiMod (Source File: LIS_laiMod.F90) . . . 132
3.18.1 Overview . . . 132
3.18.2 LIS_lai_setup (Source File: LIS_laiMod.F90) . . . 133
3.18.3 LIS_read_lai (Source File: LIS_laiMod.F90) . . . 133
3.18.4 LIS_lai_finalize (Source File: LIS_laiMod.F90) . . . 134
3.18.5 LIS_diagnoseLAI (Source File: LIS_laiMod.F90) . . . 134
3.18.6 LIS_lai_reset (Source File: LIS_laiMod.F90) . . . 135
3.18.7 LIS_lapseRateCorrection (Source File: LIS_lapseRateCorrection.F90) . . . 135
3.19 Fortran: Module Interface LIS_logMod (Source File: LIS_logMod.F90) . . . 136
3.19.1 LIS_verify (Source File: LIS_logMod.F90) . . . 136
3.19.2 LIS_log_init (Source File: LIS_logMod.F90) . . . 137
3.19.3 LIS_log_msg (Source File: LIS_logMod.F90) . . . 137
3.19.4 LIS_abort (Source File: LIS_logMod.F90) . . . 137
3.19.5 LIS_alert (Source File: LIS_logMod.F90) . . . 138
3.19.6 LIS_flush (Source File: LIS_logMod.F90) . . . 138
3.19.7 check_error (Source File: LIS_logMod.F90) . . . 139
3.19.8 verify (Source File: LIS_logMod.F90) . . . 139
3.19.9 LIS_getNextUnitNumber (Source File: LIS_logMod.F90) . . . 139
3.19.10 LIS_releaseUnitNumber (Source File: LIS_logMod.F90) . . . 140
3.19.11 LIS_endrun (Source File: LIS_logMod.F90) . . . 140

3.20 Fortran: Module Interface LIS_lsmMod (Source File: LIS_lsmMod.F90) . . . 140
3.20.1 Overview . . . 140
3.20.2 LIS_lsm_setexport (Source File: LIS_lsmMod.F90) . . . 141
3.20.3 LIS_lsm_init (Source File: LIS_lsmMod.F90) . . . 141
3.20.4 LIS_setuplsm (Source File: LIS_lsmMod.F90) . . . 142
3.20.5 LIS_lsm_main (Source File: LIS_lsmMod.F90) . . . 143
3.20.6 LIS_perturb_states (Source File: LIS_lsmMod.F90) . . . 143
3.20.7 LIS_readrestart (Source File: LIS_lsmMod.F90) . . . 144
3.20.8 LIS_lsm_output (Source File: LIS_lsmMod.F90) . . . 144
3.20.9 LIS_setLSMDynparams (Source File: LIS_lsmMod.F90) . . . 144
3.20.10 LIS_force2tile (Source File: LIS_lsmMod.F90) . . . 145
3.20.11 LIS_writerestart (Source File: LIS_lsmMod.F90) . . . 145
3.20.12 lsm_setexport_esmf (Source File: LIS_lsmMod.F90) . . . 146
3.20.13 lsm_setexport_noesmf (Source File: LIS_lsmMod.F90) . . . 146
3.20.14 LIS_lsm_finalize (Source File: LIS_lsmMod.F90) . . . 147
3.20.15 applyLSMPert (Source File: LIS_lsmMod.F90) . . . 147
3.20.16 LIS_mapSoilType (Source File: LIS_mapSoilType.F90) . . . 148
3.20.17 LIS_microMetCorrection (Source File: LIS_microMetCorrection.F90) . . . 148
3.21 Fortran: Module Interface LIS_mpiMod (Source File: LIS_mpiMod.F90) . . . 149
3.22 Fortran: Module Interface LIS_numerRecipesMod (Source File: LIS_numerRecipesMod.F90) . 149
3.22.1 ran2 (Source File: LIS_numerRecipesMod.F90) . . . 150
3.23 Fortran: Module Interface LIS_optUEMod (Source File: LIS_optUEMod.F90) . . . 150
3.23.1 Overview . . . 150
3.23.2 LIS_optUE_init (Source File: LIS_optUEMod.F90) . . . 151
3.23.3 LIS_optUE_setup (Source File: LIS_optUEMod.F90) . . . 151
3.23.4 LIS_isOptStopCriterionTrue (Source File: LIS_optUEMod.F90) . . . 152
3.23.5 LIS_runoptUE (Source File: LIS_optUEMod.F90) . . . 152
3.24 Fortran: Module Interface LIS_paramsMod (Source File: LIS_paramsMod.F90) . . . 152
3.24.1 Overview . . . 153
3.24.2 LIS_param_init (Source File: LIS_paramsMod.F90) . . . 154
3.24.3 LIS_setDynparams (Source File: LIS_paramsMod.F90) . . . 154
3.24.4 diagnoseOutputparams (Source File: LIS_paramsMod.F90) . . . 155
3.24.5 LIS_param_finalize (Source File: LIS_paramsMod.F90) . . . 156
3.24.6 LIS_param_reset (Source File: LIS_paramsMod.F90) . . . 156
3.25 Fortran: Module Interface LIS_perturbMod (Source File: LIS_perturbMod.F90) . . . 157
3.25.1 LIS_perturb_init (Source File: LIS_perturbMod.F90) . . . 157
3.25.2 LIS_perturb_readrestart (Source File: LIS_perturbMod.F90) . . . 158
3.25.3 LIS_perturb_writerestart (Source File: LIS_perturbMod.F90) . . . 158
3.26 Fortran: Module Interface LIS_precisionMod (Source File: LIS_precisionMod.F90) . . . 158
3.26.1 LIS_process_cmd_args (Source File: LIS_process_cmd_args.F90) . . . 159
3.26.2 LIS_readConfig (Source File: LIS_readConfig.F90) . . . 159
3.26.3 LIS_readObsAttributes (Source File: LIS_readObsAttributes.F90) . . . 161
3.26.4 LIS_readObsPertAttributes (Source File: LIS_readObsPertAttributes.F90) . . . 162
3.27 Fortran: Module Interface LIS_saiMod (Source File: LIS_saiMod.F90) . . . 163
3.27.1 Overview . . . 163
3.27.2 LIS_sai_setup (Source File: LIS_saiMod.F90) . . . 163
3.27.3 LIS_read_sai (Source File: LIS_saiMod.F90) . . . 164
3.27.4 LIS_sai_finalize (Source File: LIS_saiMod.F90) . . . 164
3.27.5 LIS_diagnoseSAI (Source File: LIS_saiMod.F90) . . . 164
3.27.6 LIS_sai_reset (Source File: LIS_saiMod.F90) . . . 165
3.27.7 LIS_slopeAspectCorrection (Source File: LIS_slopeAspectCorrection.F90) . . . 165

3.28 Fortran: Module Interface LIS_snowMod (Source File: LIS_snowMod.F90) . . . 166
3.28.1 LIS_snow_setup (Source File: LIS_snowMod.F90) . . . 166
3.28.2 LIS_snow_finalize (Source File: LIS_snowMod.F90) . . . 167
3.29 Fortran: Module Interface LIS_soilsMod (Source File: LIS_soilsMod.F90) . . . 167
3.29.1 Overview . . . 167
3.29.2 LIS_soils_setup (Source File: LIS_soilsMod.F90) . . . 168
3.29.3 LIS_read_soils (Source File: LIS_soilsMod.F90) . . . 168
3.29.4 LIS_soils_finalize (Source File: LIS_soilsMod.F90) . . . 169
3.29.5 LIS_diagnosesoils (Source File: LIS_soilsMod.F90) . . . 169
3.30 Fortran: Module Interface LIS_suppforcingMod (Source File: LIS_suppforcingMod.F90) . . . 170
3.30.1 Overview . . . 170
3.30.2 LIS_suppforcing_init (Source File: LIS_suppforcingMod.F90) . . . 170
3.30.3 LIS_get_supp_forcing (Source File: LIS_suppforcingMod.F90) . . . 171
3.30.4 LIS_suppforcing_finalize (Source File: LIS_suppforcingMod.F90) . . . 172
3.31 Fortran: Module Interface LIS_timeMgrMod (Source File: LIS_timeMgrMod.F90) . . . 172
3.31.1 isMonthlyAlarmRinging (Source File: LIS_timeMgrMod.F90) . . . 172
3.31.2 LIS_computeTemporalWeights (Source File: LIS_timeMgrMod.F90) . . . 172
3.31.3 LIS_resetclock (Source File: LIS_timeMgrMod.F90) . . . 173
3.31.4 timemgr_set (Source File: LIS_timeMgrMod.F90) . . . 174
3.31.5 LIS_timemgr_print (Source File: LIS_timeMgrMod.F90) . . . 175
3.31.6 LIS_advance_timestep (Source File: LIS_timeMgrMod.F90) . . . 175
3.31.7 LIS_get_step_size (Source File: LIS_timeMgrMod.F90) . . . 175
3.31.8 LIS_get_nstep (Source File: LIS_timeMgrMod.F90) . . . 176
3.31.9 LIS_get_curr_day (Source File: LIS_timeMgrMod.F90) . . . 176
3.31.10 LIS_get_julhr (Source File: LIS_timeMgrMod.F90) . . . 177
3.31.11 LIS_get_curr_calday (Source File: LIS_timeMgrMod.F90) . . . 177
3.31.12 LIS_is_last_step (Source File: LIS_timeMgrMod.F90) . . . 178
3.31.13 LIS_setAlarm (Source File: LIS_timeMgrMod.F90) . . . 178
3.31.14 LIS_setMonthlyAlarm (Source File: LIS_timeMgrMod.F90) . . . 179
3.31.15 isMonthlyAlarmRinging1 (Source File: LIS_timeMgrMod.F90) . . . 179
3.31.16 isMonthlyAlarmRinging2 (Source File: LIS_timeMgrMod.F90) . . . 180
3.31.17 LIS_setHourlyAlarm (Source File: LIS_timeMgrMod.F90) . . . 180
3.31.18 LIS_isHourlyAlarmRinging (Source File: LIS_timeMgrMod.F90) . . . 180
3.31.19 LIS_computeTimeBookEnds (Source File: LIS_timeMgrMod.F90) . . . 181
3.31.20 LIS_computeTemporalWeights (Source File: LIS_timeMgrMod.F90) . . . 181
3.31.21 LIS_computeTemporalWeights (Source File: LIS_timeMgrMod.F90) . . . 182
3.31.22 LIS_date2time (Source File: LIS_timeMgrMod.F90) . . . 183
3.31.23 LIS_time2date (Source File: LIS_timeMgrMod.F90) . . . 183
3.31.24 LIS_doy2date (Source File: LIS_timeMgrMod.F90) . . . 184
3.31.25 LIS_tick (Source File: LIS_timeMgrMod.F90) . . . 184
3.31.26 LIS_julhr_date (Source File: LIS_timeMgrMod.F90) . . . 185
3.31.27 Method . . . 185
3.31.28 LIS_tmjul4 (Source File: LIS_timeMgrMod.F90) . . . 186
3.31.29 Method . . . 186
3.32 Fortran: Module Interface LIS_topoMod (Source File: LIS_topoMod.F90) . . . 187
3.32.1 Overview . . . 187
3.32.2 LIS_topo_init (Source File: LIS_topoMod.F90) . . . 187
3.32.3 LIS_diagnoseTopography (Source File: LIS_topoMod.F90) . . . 187
3.32.4 ft_check_index (Source File: FTable.c) . . . 188
3.33 Fortran: Module Interface LIS_RTM_FTable (Source File: LIS_RTM_FTable.c) . . . 188
3.33.1 registerrtminit (Source File: LIS_RTM_FTable.c) . . . 188

3.33.2 rtminit (Source File: LIS_RTM_FTable.c) . . . 189
3.33.3 registerrtmf2t (Source File: LIS_RTM_FTable.c) . . . 189
3.33.4 rtmf2t (Source File: LIS_RTM_FTable.c) . . . 189
3.33.5 registersfc2rtm (Source File: LIS_RTM_FTable.c) . . . 189
3.33.6 sfc2rtm (Source File: LIS_RTM_FTable.c) . . . 190
3.33.7 registergeometry2rtm (Source File: LIS_RTM_FTable.c) . . . 190
3.33.8 geometry2rtm (Source File: LIS_RTM_FTable.c) . . . 190
3.33.9 registerrtmrun (Source File: LIS_RTM_FTable.c) . . . 190
3.33.10 rtmrun (Source File: LIS_RTM_FTable.c) . . . 191
3.33.11 registerrtmoutput (Source File: LIS_RTM_FTable.c) . . . 191
3.33.12 rtmoutput (Source File: LIS_RTM_FTable.c) . . . 191
3.34 Fortran: Module Interface LIS_alb_FTable (Source File: LIS_alb_FTable.c) . . . 192
3.34.1 registerreadmxsnoalb (Source File: LIS_alb_FTable.c) . . . 192
3.34.2 readmxsnoalb (Source File: LIS_alb_FTable.c) . . . 192
3.34.3 registeralbedosetup (Source File: LIS_alb_FTable.c) . . . 192
3.34.4 albedosetup (Source File: LIS_alb_FTable.c) . . . 193
3.34.5 registerreadalbedo (Source File: LIS_alb_FTable.c) . . . 193
3.34.6 readalbedo (Source File: LIS_alb_FTable.c) . . . 193
3.35 Fortran: Module Interface LIS_biasestim_FTable (Source File: LIS_biasestim_FTable.c) . . . 193
3.35.1 registerbiasestimationinit (Source File: LIS_biasestim_FTable.c) . . . 194
3.35.2 biasestimationinit (Source File: LIS_biasestim_FTable.c) . . . 194
3.35.3 registerbiasestimationsetup (Source File: LIS_biasestim_FTable.c) . . . 194
3.35.4 biasestimationsetup (Source File: LIS_biasestim_FTable.c) . . . 194
3.35.5 registerbiasestimationcompute (Source File: LIS_biasestim_FTable.c) . . . 195
3.35.6 computebiascorrection (Source File: LIS_biasestim_FTable.c) . . . 195
3.35.7 registerbiasestimationupdate (Source File: LIS_biasestim_FTable.c) . . . 195
3.35.8 applybiascorrection (Source File: LIS_biasestim_FTable.c) . . . 196
3.35.9 registerbiasestimationrestart (Source File: LIS_biasestim_FTable.c) . . . 196
3.35.10 writebiasrestart (Source File: LIS_biasestim_FTable.c) . . . 196
3.35.11 registerbiasestimationfinalize (Source File: LIS_biasestim_FTable.c) . . . 197
3.35.12 biasestimationfinalize (Source File: LIS_biasestim_FTable.c) . . . 197
3.36 Fortran: Module Interface LIS_dataassim_FTable (Source File: LIS_dataassim_FTable.c) . . . 197
3.36.1 registerdaobssetup (Source File: LIS_dataassim_FTable.c) . . . 197
3.36.2 readobsdataconfig (Source File: LIS_dataassim_FTable.c) . . . 198
3.36.3 registerdaobs (Source File: LIS_dataassim_FTable.c) . . . 198
3.36.4 readdaobs (Source File: LIS_dataassim_FTable.c) . . . 198
3.36.5 registerwritedaobs (Source File: LIS_dataassim_FTable.c) . . . 199
3.36.6 writedaobs (Source File: LIS_dataassim_FTable.c) . . . 199
3.36.7 registergetnso (Source File: LIS_dataassim_FTable.c) . . . 199
3.36.8 getselectedobsnumber (Source File: LIS_dataassim_FTable.c) . . . 200
3.36.9 registerdainit (Source File: LIS_dataassim_FTable.c) . . . 200
3.36.10 dataassiminit (Source File: LIS_dataassim_FTable.c) . . . 200
3.36.11 registerdasetup (Source File: LIS_dataassim_FTable.c) . . . 201
3.36.12 dataassimsetup (Source File: LIS_dataassim_FTable.c) . . . 201
3.36.13 registerapplyincrements (Source File: LIS_dataassim_FTable.c) . . . 201
3.36.14 applyincrements (Source File: LIS_dataassim_FTable.c) . . . 201
3.36.15 registercomputeincrements (Source File: LIS_dataassim_FTable.c) . . . 202
3.36.16 computeincrements (Source File: LIS_dataassim_FTable.c) . . . 202
3.36.17 registerdafinalize (Source File: LIS_dataassim_FTable.c) . . . 202
3.36.18 dafinalize (Source File: LIS_dataassim_FTable.c) . . . 203
3.36.19 registerdaoutput (Source File: LIS_dataassim_FTable.c) . . . 203

3.36.20 daoutput (Source File: LIS_dataassim_FTable.c) . . . 203
3.37 Fortran: Module Interface LIS_domain_FTable (Source File: LIS_domain_FTable.c) . . . 203
3.37.1 registerdomain (Source File: LIS_domain_FTable.c) . . . 204
3.37.2 makedomain (Source File: LIS_domain_FTable.c) . . . 204
3.37.3 registerinput (Source File: LIS_domain_FTable.c) . . . 204
3.37.4 readinput (Source File: LIS_domain_FTable.c) . . . 204
3.38 Fortran: Module Interface LIS_forcing_FTable (Source File: LIS_forcing_FTable.c) . . . 205
3.38.1 registerget (Source File: LIS_forcing_FTable.c) . . . 205
3.38.2 retrieveforcing (Source File: LIS_forcing_FTable.c) . . . 205
3.38.3 registertimeinterp (Source File: LIS_forcing_FTable.c) . . . 205
3.38.4 timeinterp (Source File: LIS_forcing_FTable.c) . . . 206
3.38.5 registerdefinenative (Source File: LIS_forcing_FTable.c) . . . 206
3.38.6 definenative (Source File: LIS_forcing_FTable.c) . . . 206
3.38.7 registerforcingfinal (Source File: LIS_forcing_FTable.c) . . . 206
3.38.8 forcingfinalize (Source File: LIS_forcing_FTable.c) . . . 207
3.38.9 registerforcingreset (Source File: LIS_forcing_FTable.c) . . . 207
3.38.10 forcingresetize (Source File: LIS_forcing_FTable.c) . . . 207
3.39 Fortran: Module Interface LIS_gfrac_FTable (Source File: LIS_gfrac_FTable.c) . . . 208
3.39.1 registergfracsetup (Source File: LIS_gfrac_FTable.c) . . . 208
3.39.2 gfracsetup (Source File: LIS_gfrac_FTable.c) . . . 208
3.39.3 registerreadgfrac (Source File: LIS_gfrac_FTable.c) . . . 208
3.39.4 readgfrac (Source File: LIS_gfrac_FTable.c) . . . 209
3.40 Fortran: Module Interface LIS_laisai_FTable (Source File: LIS_laisai_FTable.c) . . . 209
3.40.1 registerreadlai (Source File: LIS_laisai_FTable.c) . . . 209
3.40.2 readlai (Source File: LIS_laisai_FTable.c) . . . 209
3.40.3 registerreadsai (Source File: LIS_laisai_FTable.c) . . . 210
3.40.4 readsai (Source File: LIS_laisai_FTable.c) . . . 210
3.41 Fortran: Module Interface LIS_landcover_FTable (Source File: LIS_landcover_FTable.c) . . . 210
3.41.1 registerrreadlc (Source File: LIS_landcover_FTable.c) . . . 210
3.41.2 readlandcover (Source File: LIS_landcover_FTable.c) . . . 211
3.41.3 registerrreadmask (Source File: LIS_landcover_FTable.c) . . . 211
3.41.4 readlandmask (Source File: LIS_landcover_FTable.c) . . . 211
3.41.5 lis_log_msgC.c (Source File: LIS_log_msgC.c) . . . 211
3.41.6 lis_log_msgC (Source File: LIS_log_msgC.c) . . . 212
3.42 Fortran: Module Interface LIS_lsm_FTable (Source File: LIS_lsm_FTable.c) . . . 212
3.42.1 registerlsmini (Source File: LIS_lsm_FTable.c) . . . 212
3.42.2 lsmini (Source File: LIS_lsm_FTable.c) . . . 212
3.42.3 registerlsmrun (Source File: LIS_lsm_FTable.c) . . . 212
3.42.4 lsmrun (Source File: LIS_lsm_FTable.c) . . . 213
3.42.5 registerlsmfinalize (Source File: LIS_lsm_FTable.c) . . . 213
3.42.6 lsmfinalize (Source File: LIS_lsm_FTable.c) . . . 213
3.42.7 registerlsmsetup (Source File: LIS_lsm_FTable.c) . . . 213
3.42.8 lsmsetup (Source File: LIS_lsm_FTable.c) . . . 214
3.42.9 registerlsmrestart (Source File: LIS_lsm_FTable.c) . . . 214
3.42.10 lsmrestart (Source File: LIS_lsm_FTable.c) . . . 214
3.42.11 registerlsmoutput (Source File: LIS_lsm_FTable.c) . . . 214
3.42.12 lsmoutput (Source File: LIS_lsm_FTable.c) . . . 215
3.42.13 registerlsmf2t (Source File: LIS_lsm_FTable.c) . . . 215
3.42.14 lsmf2t (Source File: LIS_lsm_FTable.c) . . . 215
3.42.15 registerlsmwrst (Source File: LIS_lsm_FTable.c) . . . 216
3.42.16 lsmwrst (Source File: LIS_lsm_FTable.c) . . . 216

3.42.17 registergetstatevar (Source File: LIS_lsm_FTable.c) . . . 216
3.42.18 getlsmstatevar (Source File: LIS_lsm_FTable.c) . . . 217
3.42.19 registersetstatevar (Source File: LIS_lsm_FTable.c) . . . 217
3.42.20 setlsmstatevar (Source File: LIS_lsm_FTable.c) . . . 217
3.42.21 registerobstransform (Source File: LIS_lsm_FTable.c) . . . 218
3.42.22 obstransform (Source File: LIS_lsm_FTable.c) . . . 218
3.42.23 registergetobspred (Source File: LIS_lsm_FTable.c) . . . 218
3.42.24 getobspred (Source File: LIS_lsm_FTable.c) . . . 219
3.42.25 registermapobstolsm (Source File: LIS_lsm_FTable.c) . . . 219
3.42.26 mapobstolsm (Source File: LIS_lsm_FTable.c) . . . 219
3.42.27 registerlsmexport (Source File: LIS_lsm_FTable.c) . . . 220
3.42.28 lsmsetexport (Source File: LIS_lsm_FTable.c) . . . 220
3.42.29 registerlsmdynsetup (Source File: LIS_lsm_FTable.c) . . . 220
3.42.30 lsmdynsetup (Source File: LIS_lsm_FTable.c) . . . 221
3.42.31 registerqclsmstate (Source File: LIS_lsm_FTable.c) . . . 221
3.42.32 qclsmstate (Source File: LIS_lsm_FTable.c) . . . 221
3.42.33 registerqcobsstate (Source File: LIS_lsm_FTable.c) . . . 222
3.42.34 qcobsstate (Source File: LIS_lsm_FTable.c) . . . 222
3.42.35 registerscalelsmstatevar (Source File: LIS_lsm_FTable.c) . . . 222
3.42.36 scalelsmstatevar (Source File: LIS_lsm_FTable.c) . . . 223
3.42.37 registerdescalelsmstatevar (Source File: LIS_lsm_FTable.c) . . . 223
3.42.38 descalelsmstatevar (Source File: LIS_lsm_FTable.c) . . . 223
3.42.39 registerupdatelsmstate (Source File: LIS_lsm_FTable.c) . . . 224
3.42.40 updatelsmstate (Source File: LIS_lsm_FTable.c) . . . 224
3.42.41 registerwritestatevar (Source File: LIS_lsm_FTable.c) . . . 224
3.42.42 writelsmstatevar (Source File: LIS_lsm_FTable.c) . . . 225
3.42.43 registersetlsmdecisionspace (Source File: LIS_lsm_FTable.c) . . . 225
3.42.44 setlsmdecisionspace (Source File: LIS_lsm_FTable.c) . . . 225
3.42.45 registerqclsmdecisionspace (Source File: LIS_lsm_FTable.c) . . . 226
3.42.46 qclsmdecisionspace (Source File: LIS_lsm_FTable.c) . . . 226
3.42.47 registersetuppeobspred (Source File: LIS_lsm_FTable.c) . . . 226
3.42.48 setuppeobspredspace (Source File: LIS_lsm_FTable.c) . . . 227
3.42.49 registergetpeobspred (Source File: LIS_lsm_FTable.c) . . . 227
3.42.50 getpeobspred (Source File: LIS_lsm_FTable.c) . . . 227
3.42.51 LIS_memory_managementC (Source File: LIS_memory_managementC.c) . . . 228
3.42.52 lis_calloc (Source File: LIS_memory_managementC.c) . . . 228
3.42.53 lis_malloc (Source File: LIS_memory_managementC.c) . . . 228
3.42.54 LIS_mpDecomp (Source File: LIS_mpDecomp.c) . . . 228
3.43 Fortran: Module Interface LIS_optUE_FTable (Source File: LIS_optUE_FTable.c) . . . 228
3.43.1 registeroptuealginit (Source File: LIS_optUE_FTable.c) . . . 228
3.43.2 optuealginit (Source File: LIS_optUE_FTable.c) . . . 229
3.43.3 registeroptuealgsetup (Source File: LIS_optUE_FTable.c) . . . 229
3.43.4 optuealgsetup (Source File: LIS_optUE_FTable.c) . . . 229
3.43.5 registeroptuetypeinit (Source File: LIS_optUE_FTable.c) . . . 229
3.43.6 optuetypeinit (Source File: LIS_optUE_FTable.c) . . . 230
3.43.7 registeroptuetypesetdecspace (Source File: LIS_optUE_FTable.c) . . . 230
3.43.8 setoptuetypedecspace (Source File: LIS_optUE_FTable.c) . . . 230
3.43.9 registeroptueconvergencecheck (Source File: LIS_optUE_FTable.c) . . . 230
3.43.10 checkconvergence (Source File: LIS_optUE_FTable.c) . . . 231
3.43.11 registeroptuealgrun (Source File: LIS_optUE_FTable.c) . . . 231
3.43.12 runoptue (Source File: LIS_optUE_FTable.c) . . . 231

3.43.13 registerpeobssetup (Source File: LIS_optUE_FTable.c) . . . 231
3.43.14 setuppeobsspace (Source File: LIS_optUE_FTable.c) . . . 232
3.43.15 registergetpeobs (Source File: LIS_optUE_FTable.c) . . . 232
3.43.16 getpeobs (Source File: LIS_optUE_FTable.c) . . . 232
3.43.17 registerwritepeobs (Source File: LIS_optUE_FTable.c) . . . 233
3.43.18 writepeobs (Source File: LIS_optUE_FTable.c) . . . 233
3.43.19 registerinitobjfunctype (Source File: LIS_optUE_FTable.c) . . . 233
3.43.20 objectivefunctypeinit (Source File: LIS_optUE_FTable.c) . . . 233
3.43.21 registercomputeobjfunctype (Source File: LIS_optUE_FTable.c) . . . 234
3.43.22 computeobjectivefunctype (Source File: LIS_optUE_FTable.c) . . . 234
3.43.23 registerupdateobjfunctype (Source File: LIS_optUE_FTable.c) . . . 234
3.43.24 updateobjectivefunctype (Source File: LIS_optUE_FTable.c) . . . 235
3.43.25 registerresetobjfunctype (Source File: LIS_optUE_FTable.c) . . . 235
3.43.26 resetobjectivefunctype (Source File: LIS_optUE_FTable.c) . . . 235
3.43.27 registeroptuetypeobjfunceval (Source File: LIS_optUE_FTable.c) . . . 235
3.43.28 evaluateobjfunction (Source File: LIS_optUE_FTable.c) . . . 236
3.43.29 registergetdecisionspace (Source File: LIS_optUE_FTable.c) . . . 236
3.43.30 (Source File: LIS_optUE_FTable.c) . . . 236
3.43.31 registeroptuesetdecisionspace (Source File: LIS_optUE_FTable.c) . . . 237
3.43.32 (Source File: LIS_optUE_FTable.c) . . . 237
3.43.33 registeroptuegetnparam (Source File: LIS_optUE_FTable.c) . . . 237
3.43.34 (Source File: LIS_optUE_FTable.c) . . . 237
3.44 Fortran: Module Interface LIS_perturb_FTable (Source File: LIS_perturb_FTable.c) . . . 238
3.44.1 registerperturbinit (Source File: LIS_perturb_FTable.c) . . . 238
3.44.2 perturbinit (Source File: LIS_perturb_FTable.c) . . . 238
3.44.3 registerperturbsetup (Source File: LIS_perturb_FTable.c) . . . 238
3.44.4 perturbsetup (Source File: LIS_perturb_FTable.c) . . . 239
3.44.5 registerperturbmethod (Source File: LIS_perturb_FTable.c) . . . 239
3.44.6 perturbmethod (Source File: LIS_perturb_FTable.c) . . . 239
3.44.7 registerperturbwriterst (Source File: LIS_perturb_FTable.c) . . . 240
3.44.8 writepertrestart (Source File: LIS_perturb_FTable.c) . . . 240
3.44.9 registerperturbreadrst (Source File: LIS_perturb_FTable.c) . . . 240
3.44.10 readpertrestart (Source File: LIS_perturb_FTable.c) . . . 241
3.45 Fortran: Module Interface LIS_runmode_FTable (Source File: LIS_runmode_FTable.c) . . . 241
3.45.1 registerlisinit (Source File: LIS_runmode_FTable.c) . . . 241
3.45.2 lisinit (Source File: LIS_runmode_FTable.c) . . . 241
3.45.3 registerlisrun (Source File: LIS_runmode_FTable.c) . . . 242
3.45.4 lisrun (Source File: LIS_runmode_FTable.c) . . . 242
3.45.5 registerlisfinalize (Source File: LIS_runmode_FTable.c) . . . 242
3.45.6 lisfinalize (Source File: LIS_runmode_FTable.c) . . . 242
3.46 Fortran: Module Interface LIS_soils_FTable (Source File: LIS_soils_FTable.c) . . . 243
3.46.1 registerreadsand (Source File: LIS_soils_FTable.c) . . . 243
3.46.2 readsand (Source File: LIS_soils_FTable.c) . . . 243
3.46.3 registerreadclay (Source File: LIS_soils_FTable.c) . . . 243
3.46.4 readclay (Source File: LIS_soils_FTable.c) . . . 244
3.46.5 registerreadsilt (Source File: LIS_soils_FTable.c) . . . 244
3.46.6 readsilt (Source File: LIS_soils_FTable.c) . . . 244
3.46.7 registerreadtexture (Source File: LIS_soils_FTable.c) . . . 245
3.46.8 readsoiltexture (Source File: LIS_soils_FTable.c) . . . 245
3.46.9 registerreadporosity (Source File: LIS_soils_FTable.c) . . . 245
3.46.10 readporosity (Source File: LIS_soils_FTable.c) . . . 245

3.46.11 registerreadpsisat (Source File: LIS_soils_FTable.c) . . . 246
3.46.12 readpsisat (Source File: LIS_soils_FTable.c) . . . 246
3.46.13 registerreadksat (Source File: LIS_soils_FTable.c) . . . 246
3.46.14 readksat (Source File: LIS_soils_FTable.c) . . . 247
3.46.15 registerreadbexp (Source File: LIS_soils_FTable.c) . . . 247
3.46.16 readbexp (Source File: LIS_soils_FTable.c) . . . 247
3.46.17 registerreadquartz (Source File: LIS_soils_FTable.c) . . . 248
3.46.18 readquartz (Source File: LIS_soils_FTable.c) . . . 248
3.46.19 registerreadcolor (Source File: LIS_soils_FTable.c) . . . 248
3.46.20 readcolor (Source File: LIS_soils_FTable.c) . . . 249
3.47 Fortran: Module Interface LIS_suppforcing_FTable (Source File: LIS_suppforcing_FTable.c) . 249
3.47.1 registerdefinenativesupp (Source File: LIS_suppforcing_FTable.c) . . . 249
3.47.2 definenativesupp (Source File: LIS_suppforcing_FTable.c) . . . 249
3.47.3 registerreadsupp (Source File: LIS_suppforcing_FTable.c) . . . 250
3.47.4 getsuppforc (Source File: LIS_suppforcing_FTable.c) . . . 250
3.47.5 registersuppti (Source File: LIS_suppforcing_FTable.c) . . . 250
3.47.6 timeinterpsupp (Source File: LIS_suppforcing_FTable.c) . . . 250
3.47.7 registersuppforcingfinal (Source File: LIS_suppforcing_FTable.c) . . . 251
3.47.8 suppforcingfinalize (Source File: LIS_suppforcing_FTable.c) . . . 251
3.48 Fortran: Module Interface LIS_tbot_FTable (Source File: LIS_tbot_FTable.c) . . . 251
3.48.1 registerreadtbot (Source File: LIS_tbot_FTable.c) . . . 251
3.48.2 readtbot (Source File: LIS_tbot_FTable.c) . . . 252
3.49 Fortran: Module Interface LIS_topo_FTable (Source File: LIS_topo_FTable.c) . . . 252
3.49.1 registerreadelev (Source File: LIS_topo_FTable.c) . . . 252
3.49.2 readelev (Source File: LIS_topo_FTable.c) . . . 252
3.49.3 registerreadslope (Source File: LIS_topo_FTable.c) . . . 253
3.49.4 readslope (Source File: LIS_topo_FTable.c) . . . 253
3.49.5 registerreadaspect (Source File: LIS_topo_FTable.c) . . . 253
3.49.6 readaspect (Source File: LIS_topo_FTable.c) . . . 254
3.49.7 registerreadcurv (Source File: LIS_topo_FTable.c) . . . 254
3.49.8 readcurv (Source File: LIS_topo_FTable.c) . . . 254

IV Plugin Interfaces in LIS 255

4 Plugin interfaces in LIS 256
4.1 Fortran: Module Interface LIS_DAobs_pluginMod (Source File: LIS_DAobs_pluginMod.F90) . 256
4.1.1 LIS_DAobs_plugin (Source File: LIS_DAobs_pluginMod.F90) . . . 256
4.2 Fortran: Module Interface LIS_ObjFunc_pluginMod (Source File: LIS_ObjFunc_pluginMod.F90) 257
4.2.1 LIS_ObjFunc_plugin (Source File: LIS_ObjFunc_pluginMod.F90) . . . 257
4.3 Fortran: Module Interface LIS_PEobs_pluginMod (Source File: LIS_PEobs_pluginMod.F90) . 257
4.3.1 LIS_PEobs_plugin (Source File: LIS_PEobs_pluginMod.F90) . . . 257
4.4 Fortran: Module Interface LIS_RTM_pluginMod (Source File: LIS_RTM_pluginMod.F90) . . . 258
4.4.1 LIS_RTM_plugin (Source File: LIS_RTM_pluginMod.F90) . . . 258
4.5 Fortran: Module Interface LIS_baseforcing_pluginMod (Source File: LIS_baseforcing_pluginMod.F90) 258
4.5.1 LIS_baseforcing_plugin (Source File: LIS_baseforcing_pluginMod.F90) . . . 258
4.6 Fortran: Module Interface LIS_biasEstimation_pluginMod (Source File: LIS_biasEstimation_pluginMod.F90) 259
4.6.1 LIS_biasEstimation_plugin (Source File: LIS_biasEstimation_pluginMod.F90) . . . 259
4.7 Fortran: Module Interface LIS_dataassim_pluginMod.F90 (Source File: LIS_dataassim_pluginMod.F90) 260
4.7.1 LIS_dataassim_plugin (Source File: LIS_dataassim_pluginMod.F90) . . . 260
4.8 Fortran: Module Interface LIS_domain_pluginMod (Source File: LIS_domain_pluginMod.F90) 261

4.8.1 LIS_domain_plugin (Source File: LIS_domain_pluginMod.F90) 261
4.9 Fortran: Module Interface LIS_lsm_pluginMod (Source File: LIS_lsm_pluginMod.F90) 262
4.9.1 LIS_lsm_plugin (Source File: LIS_lsm_pluginMod.F90) 262
4.10 Fortran: Module Interface LIS_lsmcpl_pluginMod (Source File: LIS_lsmcpl_pluginMod.F90) . 264
4.10.1 LIS_lsmcpl_plugin (Source File: LIS_lsmcpl_pluginMod.F90) 264
4.11 Fortran: Module Interface LIS_lsmda_pluginMod (Source File: LIS_lsmda_pluginMod.F90) . . 264
4.11.1 LIS_lsmda_plugin (Source File: LIS_lsmda_pluginMod.F90) 265
4.12 Fortran: Module Interface LIS_lsmopt_pluginMod (Source File: LIS_lsmopt_pluginMod.F90) . 266
4.12.1 LIS_lsmopt_plugin (Source File: LIS_lsmopt_pluginMod.F90) 266
4.13 Fortran: Module Interface LIS_lsmrtm_pluginMod (Source File: LIS_lsmrtm_pluginMod.F90) 266
4.13.1 LIS_lsmtm_plugin (Source File: LIS_lsmrtm_pluginMod.F90) 267
4.14 Fortran: Module Interface LIS_optUEAlgorithm_pluginMod (Source File: LIS_optUEAlgorithm_pluginMod.F90) 267
4.14.1 LIS_optUEAlgorithm_plugin (Source File: LIS_optUEAlgorithm_pluginMod.F90) . . . 267
4.15 Fortran: Module Interface LIS_optUEType_pluginMod (Source File: LIS_optUEType_pluginMod.F90) 267
4.15.1 LIS_optUEType_plugin (Source File: LIS_optUEType_pluginMod.F90) 267
4.16 Fortran: Module Interface LIS_param_pluginMod (Source File: LIS_param_pluginMod.F90) . 268
4.16.1 LIS_topo_plugin (Source File: LIS_param_pluginMod.F90) 268
4.16.2 LIS_soils_plugin (Source File: LIS_param_pluginMod.F90) 269
4.16.3 LIS_laisai_plugin (Source File: LIS_param_pluginMod.F90) 270
4.16.4 LIS_landcover_plugin (Source File: LIS_param_pluginMod.F90) 270
4.16.5 LIS_tbot_plugin (Source File: LIS_param_pluginMod.F90) 271
4.16.6 LIS_gfrac_plugin (Source File: LIS_param_pluginMod.F90) 272
4.16.7 LIS_alb_plugin (Source File: LIS_param_pluginMod.F90) 272
4.17 Fortran: Module Interface LIS_perturb_pluginMod (Source File: LIS_perturb_pluginMod.F90) 273
4.17.1 LIS_perturb_plugin (Source File: LIS_perturb_pluginMod.F90) 273
4.18 Fortran: Module Interface LIS_pluginIndices (Source File: LIS_pluginIndices.F90) 274
4.19 Fortran: Module Interface LIS_runmode_pluginMod (Source File: LIS_runmode_pluginMod.F90) 278
4.19.1 LIS_runmode_plugin (Source File: LIS_runmode_pluginMod.F90) 278
4.20 Fortran: Module Interface LIS_suppforcing_pluginMod (Source File: LIS_suppforcing_pluginMod.F90) 279
4.20.1 LIS_suppforcing_plugin (Source File: LIS_suppforcing_pluginMod.F90) 279

V Interpolation Tools in LIS 281

5 Interpolation tools in LIS 282
5.1 Fortran: Module Interface UTM_utils (Source File: UTM_utils.F90) 282
5.1.1 bilinear_interp (Source File: bilinear_interp.F90) 282
5.1.2 bilinear_interp_input (Source File: bilinear_interp_input.F90) 284
5.1.3 compute_earth_coord (Source File: compute_earth_coord.F90) 285
5.1.4 compute_earth_coord_UTM (Source File: compute_earth_coord_UTM.F90) 286
5.1.5 compute_earth_coord_ease (Source File: compute_earth_coord_ease.F90) 287
5.1.6 compute_earth_coord_gauss (Source File: compute_earth_coord_gauss.F90) 287
5.1.7 compute_earth_coord_hrap (Source File: compute_earth_coord_hrap.F90) 288
5.1.8 compute_earth_coord_lambert (Source File: compute_earth_coord_lambert.F90) 289
5.1.9 compute_earth_coord_latlon (Source File: compute_earth_coord_latlon.F90) 290
5.1.10 compute_earth_coord_latlon_ll (Source File: compute_earth_coord_latlon_ll.F90) 291
5.1.11 compute_earth_coord_latlon_ur (Source File: compute_earth_coord_latlon_ur.F90) . . . 292
5.1.12 compute_earth_coord_ll (Source File: compute_earth_coord_ll.F90) 292
5.1.13 compute_earth_coord_merc (Source File: compute_earth_coord_merc.F90) 293
5.1.14 compute_earth_coord_polar (Source File: compute_earth_coord_polar.F90) 294
5.1.15 compute_earth_coord_ur (Source File: compute_earth_coord_ur.F90) 295

5.1.16 compute_grid_coord (Source File: compute_grid_coord.F90) . . . 296
5.1.17 compute_grid_coord_ease (Source File: compute_grid_coord_ease.F90) . . . 297
5.1.18 compute_grid_coord_gauss (Source File: compute_grid_coord_gauss.F90) . . . 298
5.1.19 compute_grid_coord_hrap (Source File: compute_grid_coord_hrap.F90) . . . 299
5.1.20 compute_grid_coord_lambert (Source File: compute_grid_coord_lambert.F90) . . . 300
5.1.21 compute_grid_coord_latlon (Source File: compute_grid_coord_latlon.F90) . . . 301
5.1.22 compute_grid_coord_merc (Source File: compute_grid_coord_merc.F90) . . . 301
5.1.23 compute_grid_coord_polar (Source File: compute_grid_coord_polar.F90) . . . 302
5.1.24 compute_stnwts (Source File: compute_stnwts.F90) . . . 303
5.1.25 conserv_interp (Source File: conserv_interp.F90) . . . 304
5.1.26 conserv_interp_input (Source File: conserv_interp_input.F90) . . . 306
5.1.27 defineLocalDomain (Source File: defineLocalDomain.F90) . . . 307
5.1.28 ezlh_convert (Source File: ezlh_convert.F90) . . . 307
5.1.29 ezlh_inverse (Source File: ezlh_inverse.F90) . . . 308
5.2 Fortran: Module Interface gaussian_mod (Source File: gaussian_mod.F90) . . . 309
5.2.1 gaussian_find_row (Source File: gaussian_mod.F90) . . . 309
5.2.2 gaussian_find_col (Source File: gaussian_mod.F90) . . . 310
5.2.3 gaussian_comp_lats (Source File: gaussian_mod.F90) . . . 310
5.2.4 gaussian_comp_lons (Source File: gaussian_mod.F90) . . . 310
5.2.5 gaussian_read_grid (Source File: gaussian_mod.F90) . . . 311
5.2.6 get_field_pos (Source File: get_fieldpos.F90) . . . 312
5.2.7 hrapToLatLon (Source File: hrapToLatLon.F90) . . . 313
5.2.8 interp_stndata (Source File: interp_stndata.F90) . . . 313
5.2.9 latlonTohrap (Source File: latlonTohrap.F90) . . . 314
5.2.10 latlonTopolar (Source File: latlonTopolar.F90) . . . 315
5.2.11 lltops (Source File: lltops.F90) . . . 315
5.3 Fortran: Module Interface map_utils (Source File: map_utils.F90) . . . 316
5.3.1 Supported Projections . . . 316
5.3.2 Remarks . . . 317
5.3.3 Assumptions . . . 317
5.3.4 Data Definitions: . . . 317
5.3.5 Usage . . . 317
5.3.6 map_init (Source File: map_utils.F90) . . . 319
5.3.7 map_set (Source File: map_utils.F90) . . . 319
5.3.8 latlon_to_ij (Source File: map_utils.F90) . . . 320
5.3.9 ij_to_latlon (Source File: map_utils.F90) . . . 321
5.3.10 neighbor_interp (Source File: neighbor_interp.F90) . . . 321
5.3.11 neighbor_interp_input (Source File: neighbor_interp_input.F90) . . . 323
5.3.12 normalize_stnwts (Source File: normalize_stnwts.F90) . . . 324
5.3.13 polarToLatLon (Source File: polarToLatLon.F90) . . . 324
5.3.14 polfixs (Source File: polfixs.F90) . . . 325
5.3.15 pstoll (Source File: pstoll.F90) . . . 326
5.3.16 upscaleByAveraging (Source File: upscaleByAveraging.F90) . . . 327
5.3.17 upscaleByAveraging_input (Source File: upscaleByAveraging_input.F90) . . . 328
5.3.18 zterp (Source File: zterp.F90) . . . 329
5.3.19 coszenith (Source File: zterp.F90) . . . 330
5.3.20 LIS_localtime (Source File: zterp.F90) . . . 331

VI User-defined extensible components in LIS 332

6 User-defined extensible components in LIS **333**

VII Running Modes in LIS 334

7 Running Modes in LIS **335**

8 GCE Coupled Running Mode **335**
8.1 Fortran: Module Interface lisgceexport_module.F90 (Source File: lisgceexport_module.F90) . 335
8.2 Fortran: Module Interface lisgceimport_module.F90 (Source File: lisgceimport_module.F90) . 335

9 optUE Running Mode **336**
9.1 Fortran: Module Interface optUE_runMod (Source File: optUE_runMod.F90) 336
9.1.1 lis_init_optUE (Source File: optUE_runMod.F90) 336
9.1.2 lis_run_optUE (Source File: optUE_runMod.F90) 337
9.1.3 lis_final_optUE (Source File: optUE_runMod.F90) 338

10 Retrospective Running Mode **338**
10.1 Fortran: Module Interface retrospective_runMod (Source File: retrospective_runMod.F90) . . 338
10.1.1 LIS_init_retrospective (Source File: retrospective_runMod.F90) 339
10.1.2 lis_run_retrospective (Source File: retrospective_runMod.F90) 340
10.1.3 lis_final_retrospective (Source File: retrospective_runMod.F90) 341

11 WRF Coupled ESMF Running Mode **341**
11.1 Fortran: Module Interface LIS_WRF_interfaceMod (Source File: LIS_WRF_interfaceMod.F90) 342
11.1.1 LIS_init_ImpExpFields (Source File: LIS_WRF_interfaceMod.F90) 342
11.1.2 LIS_translateImportState (Source File: LIS_WRF_interfaceMod.F90) 343
11.2 Fortran: Module Interface module_lis_component (Source File: module_lis_component.F90) . . 343
11.2.1 LIS_register (Source File: module_lis_component.F90) 343
11.2.2 LIS_component_init1 (Source File: module_lis_component.F90) 344
11.2.3 LIS_component_init2 (Source File: module_lis_component.F90) 344
11.2.4 LIS_component_run (Source File: module_lis_component.F90) 345
11.2.5 LIS_component_final (Source File: module_lis_component.F90) 345

12 WRF Coupled No ESMF Running Mode **346**
12.1 Fortran: Module Interface LISWRFGridCompMod (Source File: LISWRFGridCompMod.F90) 346
12.2 Fortran: Module Interface LISWRFexport_module (Source File: LISWRFexport_module.F90) 346
12.3 Fortran: Module Interface LISWRFimport_module (Source File: LISWRFimport_module.F90) 347
12.3.1 liswrfinit_coupled (Source File: liswrfinit_coupled.F90) 347
12.3.2 liswrfrun_coupled (Source File: liswrfrun_coupled.F90) 347

VIII Domain implementations in LIS 349

13 Domain implementations in LIS **350**

14 UTM domain **350**
14.0.3 createtiles_UTM (Source File: createtiles_UTM.F90) 350
14.0.4 readinput_UTM (Source File: readinput_UTM.F90) 350

15 Catchment domain **351**
15.0.5 createtiles_catch (Source File: createtiles_catch.F90) 351
15.0.6 readinput_catch (Source File: readinput_catch.F90) 352
15.1 Fortran: Module Interface tile_corrd_routines (Source File: tile_coord_routines.F90) 352
15.1.1 read_tile_coord (Source File: tile_coord_routines.F90) 352
15.1.2 read_tile_coord (Source File: tile_coord_routines.F90) 353
15.1.3 get_tile_num_in_atm_ij (Source File: tile_coord_routines.F90) 354
15.1.4 extract_land_tiles (Source File: tile_coord_routines.F90) 354
15.1.5 get_land_tile_info (Source File: tile_coord_routines.F90) 355
15.1.6 read_catchment_def (Source File: tile_coord_routines.F90) 355
15.1.7 get_tile_num_from_latlon (Source File: tile_coord_routines.F90) 356
15.1.8 is_in_box (Source File: tile_coord_routines.F90) 357
15.1.9 get_ij_atm_from_latlon (Source File: tile_coord_routines.F90) 357
15.1.10 tile2grid (Source File: tile_coord_routines.F90) 358
15.1.11 grid2tile (Source File: tile_coord_routines.F90) 358
15.2 Fortran: Module Interface tile_coord_types (Source File: tile_coord_types.F90) 359

16 Gaussian domain **359**
16.0.1 create_vegtilespace_gaussian (Source File: create_vegtilespace_gaussian.F90) 359
16.0.2 createtiles_gaussian (Source File: createtiles_gaussian.F90) 360
16.0.3 readinput_gaussian (Source File: readinput_gaussian.F90) 361

17 GSWP domain **362**
17.0.4 createtiles_gswp (Source File: createtiles_gswp.F90) 362
17.0.5 readinput_gswp (Source File: readinput_gswp.F90) 362

18 Lambert Conformal domain **363**
18.0.6 create_vegtilespace_lambert (Source File: create_vegtilespace_lambert.F90) 363
18.0.7 createtiles_lambert (Source File: createtiles_lambert.F90) 364
18.0.8 readinput_lambert (Source File: readinput_lambert.F90) 365

19 Lat/Lon domain **365**
19.0.9 calculate_domveg (Source File: calculate_domveg.F90) 365
19.0.10 create_vegtilespace_latlon (Source File: create_vegtilespace_latlon.F90) 366
19.0.11 createtiles_latlon (Source File: createtiles_latlon.F90) 367
19.0.12 readinput_latlon (Source File: readinput_latlon.F90) 367

20 Mercator domain **368**
20.0.13 create_vegtilespace_merc (Source File: create_vegtilespace_merc.F90) 368
20.0.14 createtiles_merc (Source File: createtiles_merc.F90) 369
20.0.15 readinput_merc (Source File: readinput_merc.F90) 370

21 Polar stereographic domain **370**
21.0.16 create_vegtilespace_polar (Source File: create_vegtilespace_polar.F90) 370
21.0.17 createtiles_polar (Source File: createtiles_polar.F90) 371
21.0.18 readinput_polar (Source File: readinput_polar.F90) 372

IX Land Surface Parameters in LIS **373**

22 Land Surface Parameters in LIS **374**

23 Albedo Data **374**
23.0.19 read_Briegleb_albedo (Source File: read_Briegleb_albedo.F90) . . . 374
23.0.20 read_GFS_albedo (Source File: read_GFS_albedo.F90) . . . 375
23.0.21 read_GFS_mxsnoalb (Source File: read_GFS_mxsnoalb.F90) . . . 375
23.0.22 read_RobinsonKukla_mxsnoalb (Source File: read_RobinsonKukla_mxsnoalb.F90) . . . 376
23.0.23 setup_Briegleb_albedo (Source File: setup_Briegleb_albedo.F90) . . . 376
23.0.24 setup_GFS_albedo (Source File: setup_GFS_albedo.F90) . . . 376

24 Greenness Fraction Data **377**
24.0.25 read_AVHRRgfrac (Source File: read_AVHRRgfrac.F90) . . . 377
24.0.26 setup_AVHRRgfrac (Source File: setup_AVHRRgfrac.F90) . . . 378
24.0.27 read_GFSfrac (Source File: read_GFSgfrac.F90) . . . 378
24.0.28 setup_GFSgfrac (Source File: setup_GFSgfrac.F90) . . . 379
24.0.29 read_NESDISgfrac (Source File: read_NESDISgfrac.F90) . . . 379

25 Leaf Area Index Data **380**
25.0.30 read_AVHRRlai (Source File: read_AVHRRlai.F90) . . . 380
25.0.31 read_AVHRRsai (Source File: read_AVHRRsai.F90) . . . 381
25.0.32 read_MODISlai (Source File: read_MODISlai.F90) . . . 382
25.0.33 read_MODISsai (Source File: read_MODISsai.F90) . . . 382
25.0.34 read_tiled_AVHRRlai (Source File: read_tiled_AVHRRlai.F90) . . . 383
25.0.35 read_tiled_AVHRRsai (Source File: read_tiled_AVHRRsai.F90) . . . 384

26 Landmask/Landcover Data **384**
26.0.36 read_GFSlc (Source File: read_GFSlc.F90) . . . 384
26.0.37 read_GFSmask (Source File: read_GFSmask.F90) . . . 385
26.0.38 read_UMDlc (Source File: read_UMDlc.F90) . . . 386
26.0.39 read_UMDmask (Source File: read_UMDmask.F90) . . . 386
26.0.40 read_USGSlc (Source File: read_USGSlc.F90) . . . 387
26.0.41 read_USGSmask (Source File: read_USGSmask.F90) . . . 387

27 Soils **388**
27.0.42 read_GFStexture (Source File: read_GFStexture.F90) . . . 388
27.0.43 read_ecmwftexture (Source File: read_ecmwftexture.F90) . . . 389
27.0.44 read_faoclay (Source File: read_faoclay.F90) . . . 389
27.0.45 read_faocolor (Source File: read_faocolor.F90) . . . 390
27.0.46 read_faoporosity (Source File: read_faoporosity.F90) . . . 390
27.0.47 read_faosand (Source File: read_faosand.F90) . . . 391
27.0.48 read_faosilt (Source File: read_faosilt.F90) . . . 392
27.0.49 read_faotexture (Source File: read_faotexture.F90) . . . 392
27.0.50 read_statsgoclay (Source File: read_statsgoclay.F90) . . . 393
27.0.51 read_statsgocolor (Source File: read_statsgocolor.F90) . . . 393
27.0.52 read_statsgososand (Source File: read_statsgosand.F90) . . . 394
27.0.53 read_statsgosilt (Source File: read_statsgosilt.F90) . . . 395
27.0.54 read_statsgotexture (Source File: read_statsgotexture.F90) . . . 395

28 Bottom Temperature **396**
28.0.55 read_NCEPtbot (Source File: read_NCEPtbot.F90) . . . 396

29 Topography **396**
29.0.56 read_aspect_gtopo30 (Source File: read_aspect_gtopo30.F90) 396
29.0.57 read_curv_gtopo30 (Source File: read_curv_gtopo30.F90) 397
29.0.58 read_elev_GFS (Source File: read_elev_GFS.F90) 397
29.0.59 read_elev_gtopo30 (Source File: read_elev_gtopo30.F90) 398
29.0.60 read_slope_gtopo30 (Source File: read_slope_gtopo30.F90) 399

X Base forcing analyses in LIS **400**

30 Overview **401**

31 ECMWF **401**
31.1 Fortran: Module Interface ecmwf_forcingMod (Source File: ecmwf_forcingMod.F90) 401
31.1.1 defineNativeECMWF (Source File: ecmwf_forcingMod.F90) 402
31.1.2 ecmwfforcing_finalize (Source File: ecmwfforcing_finalize.F90) 403
31.1.3 getecmwf (Source File: getecmwf.F90) 403
31.1.4 read_ecmwf_elev (Source File: read_ecmwf_elev.F90) 404
31.1.5 readecmwfcrd (Source File: readecmwfcrd.F90) 405
31.1.6 retecmwf (Source File: retecmwf.F90) 405
31.1.7 ret_inst3 (Source File: retecmwf.F90) 406
31.1.8 ret_accum (Source File: retecmwf.F90) 407
31.1.9 interp_iv (Source File: retecmwf.F90) 408
31.1.10 time_interp_ecmwf (Source File: time_interp_ecmwf.F90) 409

32 ECMWF Reanalysis **410**
32.1 Fortran: Module Interface ecmwfreanal_forcingMod (Source File: ecmwfreanal_forcingMod.F90) 410
32.1.1 defineNativeECMWFREANAL (Source File: ecmwfreanal_forcingMod.F90) 411
32.1.2 ecmwfreanalforcing_finalize (Source File: ecmwfreanalforcing_finalize.F90) 412
32.1.3 geogfill2 (Source File: geogfill2.F90) 412
32.1.4 getecmwfreanal (Source File: getecmwfreanal.F90) 413
32.1.5 read_ecmwfreanal_elev (Source File: read_ecmwfreanal_elev.F90) 414
32.1.6 readecmwfreanalcrd (Source File: readecmwfreanalcrd.F90) 415
32.1.7 readecmwfreanalmask (Source File: readecmwfreanalmask.F90) 415
32.1.8 retecmwfreanal (Source File: retecmwfreanal.F90) 416
32.1.9 ecmwfreanalgrid_2_lisgrid (Source File: retecmwfreanal.F90) 417
32.1.10 fillgaps (Source File: retecmwfreanal.F90) 418
32.1.11 time_interp_ecmwfreanal (Source File: time_interp_ecmwfreanal.F90) 418

33 GDAS **419**
33.0.12 create_gdasf9_filename (Source File: create_gdasf9_filename.F90) 419
33.0.13 create_gdasfilename (Source File: create_gdasfilename.F90) 420
33.1 Fortran: Module Interface gdas_forcingMod (Source File: gdas_forcingMod.F90) 420
33.1.1 defineNativeGDAS (Source File: gdas_forcingMod.F90) 422
33.2 Fortran: Module Interface gdasforcing_finalize (Source File: gdasforcing_finalize.F90) 422
33.3 Fortran: Module Interface gdasforcing_reset (Source File: gdasforcing_reset.F90) 423
33.3.1 getgdas (Source File: getgdas.F90) 423
33.3.2 read_gdas_elev (Source File: read_gdas_elev.F90) 424
33.3.3 readgdascrd (Source File: readgdascrd.F90) 425
33.3.4 retgdas (Source File: retgdas.F90) 425
33.3.5 interp_gdas (Source File: retgdas.F90) 427
33.3.6 time_interp_gdas (Source File: time_interp_gdas.F90) 428

34 GEOS **428**
34.1 Fortran: Module Interface geos_forcingMod (Source File: geos_forcingMod.F90) 429
34.1.1 defineNativeGEOS (Source File: geos_forcingMod.F90) 430
34.1.2 geosforcing_finalize (Source File: geosforcing_finalize.F90) 430
34.1.3 getgeos (Source File: getgeos.F90) 431
34.1.4 geosfile (Source File: getgeos.F90) 432
34.1.5 readgeos (Source File: readgeos.F90) 432
34.1.6 readgeoscrd (Source File: readgeoscrd.F90) 433
34.1.7 time_interp_geos (Source File: time_interp_geos.F90) 434

35 GFS **435**
35.0.8 create_gfs_f0backup_filename (Source File: create_gfs_f0backup_filename.F90) 435
35.0.9 create_gfsfilename (Source File: create_gfsfilename.F90) 435
35.0.10 getgfs (Source File: getgfs.F90) 436
35.1 Fortran: Module Interface gfs_forcingMod (Source File: gfs_forcingMod.F90) 437
35.1.1 defineNativeGFS (Source File: gfs_forcingMod.F90) 439
35.2 Fortran: Module Interface gfsforcing_finalize (Source File: gfsforcing_finalize.F90) 439
35.2.1 read_gfs_elev (Source File: read_gfs_elev.F90) 440
35.2.2 readgfscrd (Source File: readgfscrd.F90) 440
35.2.3 retgfs (Source File: retgfs.F90) 441
35.2.4 interp_gfs (Source File: retgfs.F90) 442
35.2.5 time_interp_gfs (Source File: time_interp_gfs.F90) 443

36 GSWP **444**
36.0.6 finterp (Source File: finterp.F90) 444
36.0.7 getgswp (Source File: getgswp.F90) 445
36.0.8 gswp_fillgaps (Source File: gswp_fillgaps.F90) 446
36.1 Fortran: Module Interface gswp_forcingMod (Source File: gswp_forcingMod.F90) 446
36.1.1 defineNativeGSWP (Source File: gswp_forcingMod.F90) 447
36.2 Fortran: Module Interface gswpforcing_finalize (Source File: gswpforcing_finalize.F90) 448
36.2.1 readgswp (Source File: readgswp.F90) 448
36.2.2 gswp_process_forcing (Source File: readgswp.F90) 449
36.2.3 interp_gswp (Source File: readgswp.F90) 450
36.2.4 gswp_expand_to_grid (Source File: readgswp.F90) 451
36.2.5 gswp_2d_to_1d (Source File: readgswp.F90) 451
36.2.6 gswp_mask (Source File: readgswp.F90) 452
36.2.7 readgswpcrd (Source File: readgswpcrd.F90) 452
36.2.8 time_interp_gswp (Source File: time_interp_gswp.F90) 452

37 GSWP1 **453**
37.0.9 getgswp1 (Source File: getgswp1.F90) 453
37.0.10 gswp1file (Source File: getgswp1.F90) 454
37.1 Fortran: Module Interface gswp1_forcingMod (Source File: gswp1_forcingMod.F90) 454
37.1.1 defineNativeGSWP1 (Source File: gswp1_forcingMod.F90) 455
37.2 Fortran: Module Interface gswp1forcing_finalize (Source File: gswp1forcing_finalize.F90) . . . 456
37.2.1 readgswp1 (Source File: readgswp1.F90) 456
37.3 Core Functions of readgswp1 456
37.3.1 readgswp1crd (Source File: readgswp1crd.F90) 457
37.3.2 time_interp_gswp1 (Source File: time_interp_gswp1.F90) 457

38 MERRA **458**
38.0.3 getmerra (Source File: getmerra.F90) . . . 458
38.0.4 merrafile (Source File: getmerra.F90) . . . 459
38.1 Fortran: Module Interface merra_forcingMod (Source File: merra_forcingMod.F90) . . . 459
38.1.1 defineNativeMERRA (Source File: merra_forcingMod.F90) . . . 460
38.1.2 merraforcing_finalize (Source File: merraforcing_finalize.F90) . . . 461
38.1.3 readmerra (Source File: readmerra.F90) . . . 461
38.1.4 readmerracrd (Source File: readmerracrd.F90) . . . 462
38.1.5 time_interp_merra (Source File: time_interp_merra.F90) . . . 463

39 Princeton **463**
39.0.6 getprinceton (Source File: getprinceton.F90) . . . 464
39.1 Fortran: Module Interface princeton_forcingMod (Source File: princeton_forcingMod.F90) . . . 464
39.1.1 defineNativePRINCETON (Source File: princeton_forcingMod.F90) . . . 466
39.1.2 princetonforcing_finalize (Source File: princetonforcing_finalize.F90) . . . 466
39.1.3 read_princeton_elev (Source File: read_princeton_elev.F90) . . . 467
39.1.4 readprincetoncrd (Source File: readprincetoncrd.F90) . . . 467
39.1.5 retprinceton (Source File: retprinceton.F90) . . . 468
39.1.6 princetongrid_2_lisgrid (Source File: retprinceton.F90) . . . 469
39.1.7 time_interp_princeton (Source File: time_interp_princeton.F90) . . . 469

40 Rhone-AGG **470**
40.0.8 getrhone (Source File: getrhone.F90) . . . 470
40.0.9 rhonefile (Source File: getrhone.F90) . . . 471
40.0.10 readrhone (Source File: readrhone.F90) . . . 471
40.1 Core Functions of readrhone . . . 472
40.1.1 readrhonecrd (Source File: readrhonecrd.F90) . . . 472
40.2 Fortran: Module Interface rhone_forcingMod (Source File: rhone_forcingMod.F90) . . . 472
40.2.1 defineNativeRHONE (Source File: rhone_forcingMod.F90) . . . 473
40.2.2 rhoneforcing_finalize (Source File: rhoneforcing_finalize.F90) . . . 473
40.2.3 time_interp_rhone (Source File: time_interp_rhone.F90) . . . 473
40.3 Core Functions of time_interp_rhone . . . 474

41 Template **474**
41.0.1 gettemplate (Source File: gettemplate.F90) . . . 474
41.1 Fortran: Module Interface template_forcingMod (Source File: template_forcingMod.F90) . . . 475
41.1.1 defineNativetemplate (Source File: template_forcingMod.F90) . . . 475
41.2 Fortran: Module Interface templateforcing_finalize (Source File: templateforcing_finalize.F90) 475
41.3 Fortran: Module Interface templateforcing_reset (Source File: templateforcing_reset.F90) . . . 475
41.3.1 time_interp_template (Source File: time_interp_template.F90) . . . 476

XI Supplemental forcing analyses in LIS **477**

42 Overview **478**

43 TRMM 3B42RT **478**
43.1 Fortran: Module Interface TRMM3B42RT_forcingMod (Source File: TRMM3B42RT_forcingMod.F90) 478
43.1.1 defineNativeTRMM3B42RT (Source File: TRMM3B42RT_forcingMod.F90) . . . 479
43.1.2 TRMM3B42RTforcing_finalize (Source File: TRMM3B42RTforcing_finalize.F90) . . . 479
43.1.3 getTRMM3B42RT (Source File: getTRMM3B42RT.F90) . . . 480
43.1.4 TRMM3B42RTfile (Source File: getTRMM3B42RT.F90) . . . 481

43.1.5 glbprecip_TRMM3B42RT (Source File: glbprecip_TRMM3B42RT.F90) 481
43.1.6 interp_TRMM3B42RT (Source File: interp_TRMM3B42RT.F90) 482
43.1.7 readTRMM3B42RTcrd (Source File: readTRMM3B42RTcrd.F90) 483
43.1.8 time_interp_TRMM3B42RT (Source File: time_interp_3B42RT.F90) 483

44 TRMM 3B42V6 **484**
44.0.9 defineNativeTRMM3B42V6 (Source File: TRMM3B42V6_forcingMod.F90) 484
44.0.10 TRMM3B42V6forcing_finalize (Source File: TRMM3B42V6forcing_finalize.F90) 484
44.0.11 getTRMM3B42V6 (Source File: getTRMM3B42V6.F90) 485
44.0.12 TRMM3B42V6file (Source File: getTRMM3B42V6.F90) 488
44.0.13 glbprecip_TRMM3B42V6.F90 (Source File: glbprecip_TRMM3B42V6.F90) 488
44.0.14 interp_TRMM3B42V6.F90 (Source File: interp_TRMM3B42V6.F90) 491
44.0.15 readTRMM3B42V6crd (Source File: readTRMM3B42V6crd.F90) 491
44.0.16 time_interp_TRMM3B42V6 (Source File: time_interp_TRMM3B42V6.F90) 492

45 AGRMET radiation **492**
45.0.17 agrrad_finalize (Source File: agrrad_finalize.F90) 493
45.1 Fortran: Module Interface agrrad_forcingMod (Source File: agrrad_forcingMod.F90) 493
45.1.1 defineNativeAGRRAD (Source File: agrrad_forcingMod.F90) 494
45.1.2 getagrrad (Source File: getagrrad.F90) 495
45.1.3 readagrrad (Source File: readagrrad.F90) 495
45.1.4 agrrad_filename (Source File: readagrrad.F90) 496
45.1.5 agrrad_fillgaps (Source File: readagrrad.F90) 497
45.1.6 readagrradcrd (Source File: readagrradcrd.F90) 498
45.1.7 time_interp_agrrad (Source File: time_interp_agrrad.F90) 498

46 AGRMET polar stereographic radiation **499**
46.0.8 agrradps_filename_sw (Source File: agrradps_filenames.F90) 499
46.0.9 agrradps_filename_cloud (Source File: agrradps_filenames.F90) 499
46.1 Fortran: Module Interface agrradps_finalize (Source File: agrradps_finalize.F90) 500
46.2 Fortran: Module Interface agrradps_forcingMod (Source File: agrradps_forcingMod.F90) . . . 500
46.2.1 definenativeAGRRADPS (Source File: agrradps_forcingMod.F90) 502
46.2.2 getagrradps (Source File: getagrradps.F90) 503
46.2.3 interp_agrradpsvar1 (Source File: interp_agrradpsvar1.F90) 503
46.2.4 readagrradps1 (Source File: readagrradps1.F90) 504
46.2.5 agrradps_fillgaps1 (Source File: readagrradps1.F90) 505
46.2.6 readagrradpscrd (Source File: readagrradpscrd.F90) 506
46.2.7 time_interp_agrradps (Source File: time_interp_agrradps.F90) 506

47 ARMS forcing **507**
47.1 Fortran: Module Interface arms_forcingMod (Source File: arms_forcingMod.F90) 507
47.1.1 defineNativeARMS (Source File: arms_forcingMod.F90) 507
47.1.2 armsforcing_finalize (Source File: armsforcing_finalize.F90) 508
47.1.3 getarms (Source File: getarms.F90) 508
47.1.4 readarms (Source File: readarms.F90) 509
47.1.5 doytoCalday (Source File: readarms.F90) 510
47.1.6 RHtoq (Source File: readarms.F90) 510
47.1.7 readarmscrd (Source File: readarmscrd.F90) 511
47.1.8 time_interp_arms (Source File: time_interp_arms.F90) 511

48 CEOP forcing **512**
48.1 Fortran: Module Interface ceop_forcingMod (Source File: ceop_forcingMod.F90) 512
48.1.1 defineNativeCEOP (Source File: ceop_forcingMod.F90) 512
48.1.2 ceopforcing_finalize (Source File: ceopforcing_finalize.F90) 513
48.1.3 getceop (Source File: getceop.F90) 513
48.1.4 readbon (Source File: readbon.F90) 514
48.1.5 readceop (Source File: readceop.F90) 515
48.1.6 readceopcrd (Source File: readceopcrd.F90) 516
48.1.7 readfpk (Source File: readfpk.F90) 516
48.1.8 readsgp (Source File: readsgp.F90) 518
48.1.9 time_interp_ceop (Source File: time_interp_ceop.F90) 519

49 CMAP **519**
49.1 Fortran: Module Interface cmap_forcingMod (Source File: cmap_forcingMod.F90) 519
49.1.1 defineNativeCMAP (Source File: cmap_forcingMod.F90) 520
49.1.2 cmapforcing_finalize (Source File: cmapforcing_finalize.F90) 521
49.1.3 getcmap (Source File: getcmap.F90) 521
49.1.4 cmapfile (Source File: getcmap.F90) 522
49.1.5 cmapfile_old (Source File: getcmap.F90) 523
49.1.6 glbprecip_cmap (Source File: glbprecip_cmap.F90) 524
49.1.7 interp_cmap (Source File: interp_cmap.F90) 525
49.1.8 readcmapcrd (Source File: readcmapcrd.F90) 526
49.1.9 time_interp_cmap (Source File: time_interp_cmap.F90) 526

50 CMORPH **527**
50.0.10 cmorforcing_finalize (Source File: cmorforcing_finalize.F90) 527
50.1 Fortran: Module Interface cmorph_forcingMod (Source File: cmorph_forcingMod.F90) 527
50.1.1 defineNativeCMORPH (Source File: cmorph_forcingMod.F90) 528
50.1.2 getcmor (Source File: getcmor.F90) 529
50.1.3 cmorfile (Source File: getcmor.F90) 530
50.1.4 glbprecip_cmor (Source File: glbprecip_cmor.F90) 530
50.1.5 interp_cmor (Source File: interp_cmor.F90) 531
50.1.6 readcmorcrd (Source File: readcmorcrd.F90) 532
50.1.7 time_interp_cmor (Source File: time_interp_cmor.F90) 532

51 GDAS LSWG forcing **533**
51.1 Fortran: Module Interface gdasLSWG_forcingMod (Source File: gdasLSWG_forcingMod.F90) 533
51.1.1 defineNativeGdasLSWG (Source File: gdasLSWG_forcingMod.F90) 533
51.2 Fortran: Module Interface gdasLSWGforcing_finalize (Source File: gdasLSWGforcing_finalize.F90) 534
51.2.1 getgdasLSWG (Source File: getgdasLSWG.F90) 534
51.2.2 readgdasLSWGcrd (Source File: readgdasLSWGcrd.F90) 535
51.2.3 retgdasLSWG (Source File: retgdasLSWG.F90) 535
51.2.4 time_interp_gdasLSWG (Source File: time_interp_gdasLSWG.F90) 536

52 NLDAS **536**
52.0.5 apply_conusmask (Source File: apply_conusmask.F90) 536
52.0.6 getnldas (Source File: getnldas.F90) 537
52.0.7 nldasfile (Source File: getnldas.F90) 538
52.0.8 interp_nldas (Source File: interp_nldas.F90) 539
52.0.9 nldas_ec_removal (Source File: nldas_ec_removal.F90) 540
52.1 Fortran: Module Interface nldas_forcingMod (Source File: nldas_forcingMod.F90) 540
52.1.1 defineNativeNLDAS (Source File: nldas_forcingMod.F90) 542

52.2 Fortran: Module Interface nldasforcing_finalize (Source File: nldasforcing_finalize.F90) 543
52.2.1 orig_elevdiff (Source File: orig_elevdiff.F90) 543
52.2.2 read_edas_elev (Source File: read_edas_elev.F90) 544
52.2.3 readnldascrd (Source File: readnldascrd.F90) 544
52.2.4 retnldas (Source File: retnldas.F90) 545
52.2.5 time_interp_nldas (Source File: time_interp_nldas.F90) 546

53 NLDAS2 **547**
53.0.6 getnldas2 (Source File: getnldas2.F90) 547
53.0.7 nldas2filea (Source File: getnldas2.F90) 548
53.0.8 nldas2fileb (Source File: getnldas2.F90) 549
53.1 Fortran: Module Interface nldas2_forcingMod (Source File: nldas2_forcingMod.F90) 549
53.1.1 defineNativeNLDAS2 (Source File: nldas2_forcingMod.F90) 551
53.2 Fortran: Module Interface nldas2forcing_finalize (Source File: nldas2forcing_finalize.F90) . . . 551
53.2.1 read_nldas2_elev (Source File: read_nldas2_elev.F90) 552
53.2.2 readnldas2crd (Source File: readnldas2crd.F90) 552
53.2.3 retnldas2 (Source File: retnldas2a.F90) 553
53.2.4 interp_nldas2 (Source File: retnldas2a.F90) 554
53.2.5 retnldas2 (Source File: retnldas2b.F90) 555
53.2.6 time_interp_nldas2 (Source File: time_interp_nldas2.F90) 556

54 SALDAS **557**
54.0.7 getsaldas (Source File: getsaldas.F90) 557
54.0.8 saldasfile (Source File: getsaldas.F90) 558
54.0.9 read_saldas_elev (Source File: read_saldas_elev.F90) 559
54.0.10 readsaldascrd (Source File: readsaldascrd.F90) 559
54.0.11 retsaldas (Source File: retsaldas.F90) 560
54.0.12 interp_saldas (Source File: retsaldas.F90) 561
54.1 Fortran: Module Interface saldas_forcingMod (Source File: saldas_forcingMod.F90) 562
54.1.1 defineNativeSALDAS (Source File: saldas_forcingMod.F90) 563
54.1.2 saldasforcing_finalize (Source File: saldasforcing_finalize.F90) 564
54.1.3 time_interp_saldas (Source File: time_interp_saldas.F90) 564

55 SCAN forcing **565**
55.0.4 getscan (Source File: getscan.F90) 565
55.0.5 readscan (Source File: readscan.F90) 566
55.0.6 readscancrd (Source File: readscancrd.F90) 566
55.1 Fortran: Module Interface scan_forcingMod (Source File: scan_forcingMod.F90) 567
55.1.1 defineNativeSCAN (Source File: scan_forcingMod.F90) 567
55.1.2 scanforcing_finalize (Source File: scanforcing_finalize.F90) 568
55.1.3 time_interp_scan (Source File: time_interp_scan.F90) 568

56 STAGEII **569**
56.0.4 getstg2 (Source File: getstg2.F90) 569
56.0.5 interp_stg2 (Source File: interp_stg2.F90) 570
56.0.6 precip_stg2 (Source File: precip_stg2.F90) 571
56.0.7 readstg2crd (Source File: readstg2crd.F90) 572
56.1 Fortran: Module Interface stg2_finalize (Source File: stg2_finalize.F90) 572
56.2 Fortran: Module Interface stg2_forcingMod (Source File: stg2_forcingMod.F90) 572
56.2.1 defineNativeSTG2 (Source File: stg2_forcingMod.F90) 573
56.2.2 stg2file (Source File: stg2file.F90) 574
56.2.3 time_interp_stg2 (Source File: time_interp_stg2.F90) 575

57 STAGEIV **575**
57.0.4 getstg4 (Source File: getstg4.F90) . . . 575
57.0.5 interp_stg4 (Source File: interp_stg4.F90) . . . 576
57.0.6 precip_stg4 (Source File: precip_stg4.F90) . . . 577
57.0.7 readstg4crd (Source File: readstg4crd.F90) . . . 578
57.1 Fortran: Module Interface stg4_finalize (Source File: stg4_finalize.F90) . . . 579
57.2 Fortran: Module Interface stg4_forcingMod (Source File: stg4_forcingMod.F90) . . . 579
57.2.1 defineNativeSTG4 (Source File: stg4_forcingMod.F90) . . . 580
57.2.2 stg4file (Source File: stg4file.F90) . . . 581
57.2.3 time_interp_stg4 (Source File: time_interp_stg4.F90) . . . 581

XII Data Assimilation in LIS **582**

58 Data Assimilation Algorithms in LIS **583**

59 Direct Insertion **583**
59.1 Fortran: Module Interface directInsertion_Mod (Source File: directInsertion_Mod.F90) . . . 583
59.1.1 di_init (Source File: directInsertion_Mod.F90) . . . 583
59.1.2 di_increments (Source File: directInsertion_Mod.F90) . . . 584
59.1.3 di_update (Source File: directInsertion_Mod.F90) . . . 584
59.1.4 di_diagnostics (Source File: directInsertion_Mod.F90) . . . 585
59.1.5 di_final (Source File: directInsertion_Mod.F90) . . . 585

60 GMAO Ensemble Kalman Filter **585**
60.1 Fortran: Module Interface enkf_general (Source File: enkf_general.F90) . . . 585
60.1.1 enkf_increments (Source File: enkf_general.F90) . . . 585
60.1.2 hadamard_for_repr_matrix (Source File: enkf_general.F90) . . . 586
60.1.3 gaspari_cohn (Source File: enkf_general.F90) . . . 587
60.1.4 get_gaspari_cohn (Source File: enkf_general.F90) . . . 587
60.2 Fortran: Module Interface enkf_types (Source File: enkf_types.F90) . . . 587
60.3 Fortran: Module Interface gmaoenkf_Mod (Source File: gmaoenkf_Mod.F90) . . . 588
60.3.1 gmaoenkf_init (Source File: gmaoenkf_Mod.F90) . . . 588
60.3.2 gmaoenkf_setup (Source File: gmaoenkf_Mod.F90) . . . 589
60.3.3 gmaoenkf_increments (Source File: gmaoenkf_Mod.F90) . . . 589
60.3.4 gmaoenkf_update (Source File: gmaoenkf_Mod.F90) . . . 591
60.3.5 gmaoenkf_diagnostics (Source File: gmaoenkf_Mod.F90) . . . 591
60.3.6 pruneVarname (Source File: gmaoenkf_Mod.F90) . . . 592
60.3.7 assemble_obs_cov (Source File: gmaoenkf_Mod.F90) . . . 592
60.3.8 gmaoenkf_final (Source File: gmaoenkf_Mod.F90) . . . 593
60.3.9 getLSMvarnames (Source File: gmaoenkf_Mod.F90) . . . 593
60.3.10 getObsPert (Source File: gmaoenkf_Mod.F90) . . . 594
60.3.11 generateObservations (Source File: gmaoenkf_Mod.F90) . . . 594
60.3.12 generateObsparam (Source File: gmaoenkf_Mod.F90) . . . 595
60.3.13 getLSMData (Source File: gmaoenkf_Mod.F90) . . . 595
60.3.14 ludcmp (Source File: my_lu_decomp.F90) . . . 596
60.3.15 lubksb (Source File: my_lu_decomp.F90) . . . 596
60.4 Fortran: Module Interface my_matrix_functions (Source File: my_matrix_functions.F90) . . . 596
60.4.1 (Source File: my_matrix_functions.F90) . . . 596
60.4.2 row_variance (Source File: my_matrix_functions.F90) . . . 596
60.4.3 row_third_moment (Source File: my_matrix_functions.F90) . . . 597

60.4.4 five_number_summary (Source File: my_matrix_functions.F90) . . . 597
60.4.5 adjust_mean (Source File: my_matrix_functions.F90) . . . 598
60.4.6 adjust_std (Source File: my_matrix_functions.F90) . . . 598
60.4.7 matrix_std (Source File: my_matrix_functions.F90) . . . 599
60.4.8 nr_sort (Source File: nr_sort.F90) . . . 599

61 Perturbation Algorithms in LIS **599**

62 GMAO Perturbation Algorithm **599**
62.1 Fortran: Module Interface gmaopert_Mod (Source File: gmaopert_Mod.F90) . . . 599
62.1.1 gmaopert_init (Source File: gmaopert_Mod.F90) . . . 600
62.1.2 gmaoperturb (Source File: gmaopert_Mod.F90) . . . 600
62.1.3 gmaopert_writerestart (Source File: gmaopert_Mod.F90) . . . 601
62.1.4 gmaopert_readrestart (Source File: gmaopert_Mod.F90) . . . 601
62.2 Fortran: Module Interface landpert_module.F90 (Source File: landpert_module.F90) . . . 602
62.2.1 allocate_pert_param (Source File: landpert_module.F90) . . . 602
62.3 Fortran: Module Interface landpert_routines.F90 (Source File: landpert_routines.F90) . . . 602
62.3.1 assemble_pert_param (Source File: landpert_routines.F90) . . . 603
62.3.2 get_sqrt_corr_matrix (Source File: landpert_routines.F90) . . . 603
62.3.3 get_init_Pert_rseed (Source File: landpert_routines.F90) . . . 604
62.3.4 get_pert (Source File: landpert_routines.F90) . . . 604
62.3.5 propagate_pert (Source File: landpert_routines.F90) . . . 606
62.3.6 truncate_std_normal (Source File: landpert_routines.F90) . . . 607
62.3.7 adjust_mean (Source File: landpert_routines.F90) . . . 607
62.4 Fortran: Module Interface nr_jacobi.F90 (Source File: nr_jacobi.F90) . . . 607
62.4.1 jacobi (Source File: nr_jacobi.F90) . . . 608
62.5 Fortran: Module Interface nr_ran2_gasdev.F90 (Source File: nr_ran2_gasdev.F90) . . . 608
62.6 Fortran: Module Interface random_fields.F90 (Source File: random_fields.F90) . . . 608
62.6.1 sqrt_gauss_spectrum_2d (Source File: random_fields.F90) . . . 609
62.6.2 rfg2d_fft (Source File: random_fields.F90) . . . 609
62.6.3 get_fft_grid (Source File: random_fields.F90) . . . 610
62.6.4 generate_white_field (Source File: random_fields.F90) . . . 611

63 Static Perturbation Algorithm **611**
63.1 Fortran: Module Interface staticPerturbMod (Source File: staticPerturbMod.F90) . . . 611
63.1.1 staticPerturb_init (Source File: staticPerturbMod.F90) . . . 611
63.1.2 staticPerturburb (Source File: staticPerturbMod.F90) . . . 612

XIII Observation data handling for assimilation in LIS **613**

64 Observation data handling **614**

65 AMSR-E Soil Moisture **614**
65.1 Fortran: Module Interface AMSREsmobs_module (Source File: AMSREsmobs_module.F90) . 614
65.1.1 AMSREsmobs_setup (Source File: AMSREsmobs_module.F90) . . . 614
65.1.2 getNSO_AMSREsm.F90 (Source File: getNSO_AMSREsm.F90) . . . 615
65.1.3 read_AMSREsmobs (Source File: read_AMSREsmobs.F90) . . . 615

66 NASA AMSR-E Soil Moisture **616**
66.1 Fortran: Module Interface NASA_AMSREsm_Mod (Source File: NASA_AMSREsm_Mod.F90) 616
66.1.1 NASA_AMSREsm_setup (Source File: NASA_AMSREsm_Mod.F90) 616
66.1.2 getNSO_NASA_AMSREsm (Source File: getNSO_NASA_AMSREsm.F90) 617
66.1.3 read_NASA_AMSREsm (Source File: read_NASA_AMSREsm.F90) 617
66.1.4 NASA_AMSREsm_filename (Source File: read_NASA_AMSREsm.F90) 618
66.1.5 write_NASA_AMSREsmobs (Source File: write_NASA_AMSREsmobs.F90) 618
66.1.6 NASA_AMSRE_smobsname (Source File: write_NASA_AMSREsmobs.F90) 619

67 NESDIS AMSR-E Soil Moisture **619**
67.1 Fortran: Module Interface NESDIS_AMSREsm_Mod (Source File: NESDIS_AMSREsm_Mod.F90) 619
67.1.1 NESDIS_AMSREsm_setup (Source File: NESDIS_AMSREsm_Mod.F90) 620
67.1.2 getNSO_NESDIS_AMSREsm (Source File: getNSO_NESDIS_AMSREsm.F90) 620
67.1.3 read_NESDIS_AMSREsm (Source File: read_NESDIS_AMSREsm.F90) 621
67.1.4 NESDIS_AMSREsm_filename (Source File: read_NESDIS_AMSREsm.F90) 622
67.1.5 write_NESDIS_AMSREsmobs (Source File: write_NESDIS_AMSREsmobs.F90) 622
67.1.6 NESDIS_AMSRE_smobsname (Source File: write_NESDIS_AMSREsmobs.F90) 623

68 Synthetic Soil Moisture from Noah **623**
68.0.7 getNSO_synsm1.F90 (Source File: getNSO_synsm1.F90) 623
68.0.8 read_syntheticsm1obs (Source File: read_syntheticsm1obs.F90) 624
68.1 Fortran: Module Interface syntheticsm1obs_module (Source File: syntheticsm1obs_module.F90) 624
68.1.1 syntheticsm1obs_setup (Source File: syntheticsm1obs_module.F90) 624

69 Synthetic Soil Moisture from Catchment **625**
69.0.2 getNSO_synsm2.F90 (Source File: getNSO_synsm2.F90) 625
69.0.3 read_syntheticsm2obs (Source File: read_syntheticsm2obs.F90) 626
69.1 Fortran: Module Interface syntheticsm2obs_module (Source File: syntheticsm2obs_module.F90) 626
69.1.1 syntheticsm2obs_setup (Source File: syntheticsm2obs_module.F90) 626

70 Synthetic Snow Water Equivalent **627**
70.0.2 getNSO_synswe1 (Source File: getNSO_synswe1.F90) 627
70.0.3 read_syntheticswe1obs (Source File: read_syntheticswe1obs.F90) 628
70.1 Fortran: Module Interface syntheticswe1obs_module (Source File: syntheticswe1obs_module.F90) 628
70.1.1 syntheticswe1obs_setup (Source File: syntheticswe1obs_module.F90) 628

XIV Parameter Estimation in LIS **630**

71 Parameter Estimation in LIS **631**

XV Land Surface Models in LIS **632**

72 Land Surface Models in LIS **633**

73 SiB2 Land Surface Model **633**
73.0.2 readcrdsib2 (Source File: readcrdsib2.F90) 633
73.0.3 sib2_coldstart (Source File: sib2_coldstart.F90) 633
73.0.4 sib2_dynsetup (Source File: sib2_dynsetup.F90) 634
73.0.5 sib2_f2t (Source File: sib2_f2t.F90) 634
73.0.6 sib2_finalize (Source File: sib2_finalize.F90) 635

73.1 Fortran: Module Interface sib2_lsmMod (Source File: sib2_lsmMod.F90) 635
73.1.1 sib2_lsm_ini (Source File: sib2_lsmMod.F90) 636
73.1.2 sib2_main (Source File: sib2_main.F90) 636
73.1.3 sib2_mapvegc (Source File: sib2_mapvegc.F90) 637
73.2 Fortran: Module Interface sib2_module (Source File: sib2_module.F90) 637
73.2.1 sib2_monthly (Source File: sib2_monthly.F90) 637
73.2.2 sib2_output (Source File: sib2_output.F90) 638
73.2.3 sib2_readrst (Source File: sib2_readrst.F90) 639
73.2.4 sib2_setup (Source File: sib2_setup.F90) 639
73.2.5 sib2_setvegparms (Source File: sib2_setvegparms.F90) 639
73.2.6 sib2_writerst (Source File: sib2_writerst.F90) 640
73.2.7 sib2_dump_restart (Source File: sib2_writerst.F90) 640
73.3 Fortran: Module Interface sib2alb_module (Source File: sib2alb_module.F90) 641
73.3.1 sib2alb_ini (Source File: sib2alb_module.F90) 642

74 Catchment Land Surface Model 642
74.0.2 cat_dump_param (Source File: cat_dump_param.F90) 642
74.0.3 cat_dynsetup.F90 (Source File: cat_dynsetup.F90) 643
74.0.4 cat_f2t (Source File: cat_f2t.F90) 643
74.0.5 cat_finalize (Source File: cat_finalize.F90) 644
74.0.6 cat_lsmMod.F90 (Source File: cat_lsmMod.F90) 644
74.0.7 cat_varalloc (Source File: cat_lsmMod.F90) 645
74.0.8 cat_main (Source File: cat_main.F90) 645
74.0.9 cat_output (Source File: cat_output.F90) 646
74.0.10 cat_readrestart (Source File: cat_readrestart.F90) 647
74.0.11 cat_setup (Source File: cat_setup.F90) 647
74.0.12 cat_writerestart (Source File: cat_writerestart.F90) 648
74.0.13 cat_dump_restart (Source File: cat_writerestart.F90) 649
74.1 Fortran: Module Interface cat_diagn_routines (Source File: catch_diagn_routines.F90) 650
74.1.1 calc_tsurf (Source File: catch_diagn_routines.F90) 650
74.1.2 calc_arX (Source File: catch_diagn_routines.F90) 650
74.1.3 calc_asnow (Source File: catch_diagn_routines.F90) 651
74.2 Fortran: Module Interface catch_types (Source File: catch_types.F90) 651
74.3 Fortran: Module Interface clsm_para_mapping (Source File: clsm_para_mapping.F90) 651
74.3.1 read_catchment_parameters (Source File: clsm_para_mapping.F90) 651
74.3.2 read_catchment_parameters_final (Source File: clsm_para_mapping.F90) 652
74.3.3 read_rst (Source File: clsm_para_mapping.F90) 652
74.3.4 get_tile_index (Source File: clsm_para_mapping.F90) 652
74.3.5 get_param (Source File: clsm_para_mapping.F90) 653
74.4 Fortran: Module Interface drv_catch_types (Source File: drv_catch_types.F90) 653
74.5 Fortran: Module Interface esat_qsat (Source File: esat_qsat.F90) 653
74.5.1 pmonth (Source File: pmonth_stage4.F90) 654
74.5.2 read_land_parameters1 (Source File: read_land_parameters.F90) 654
74.5.3 read_land_parameters2 (Source File: read_land_parameters.F90) 655
74.5.4 read_land_parameters3 (Source File: read_land_parameters.F90) 655
74.5.5 d2g (Source File: read_land_parameters.F90) 656
74.5.6 read_modis_scale_param (Source File: read_modis_scale_param.F90) 656
74.5.7 readcatchcrd (Source File: readcatchcrd.F90) 657
74.5.8 turb (Source File: turb.F90) 657

75 Community Land Model version 2.0 **658**
75.0.9 clm_dynsetup (Source File: clm_dynsetup.F90) . . . 658
75.0.10 clm_finalize (Source File: clm_finalize.F90) . . . 659
75.1 Fortran: Module Interface clm_lsmMod (Source File: clm_lsmMod.F90) . . . 659
75.1.1 clm_lsm_ini (Source File: clm_lsmMod.F90) . . . 660
75.1.2 clm_lsm_init (Source File: clm_lsmMod.F90) . . . 660
75.1.3 clm_main (Source File: clm_main.F90) . . . 661
75.1.4 clm_output (Source File: clm_output.F90) . . . 662
75.1.5 clm_readrestart (Source File: clm_readrestart.F90) . . . 663
75.1.6 clm_setup (Source File: clm_setup.F90) . . . 663
75.1.7 clm_writerestart (Source File: clm_writerestart.F90) . . . 664
75.1.8 clm_dump_restart (Source File: clm_writerestart.F90) . . . 665
75.1.9 clmlairead.F90: (Source File: clmlairead.F90) . . . 665
75.2 Fortran: Module Interface clmtype (Source File: clmtype.F90) . . . 666
75.2.1 clm2_wrf_f2t (Source File: clm2_wrf_f2t.F90) . . . 671
75.2.2 clm_setwrfexport.F90 (Source File: clm_setwrfexport.F90) . . . 671
75.2.3 iniTimeConst (Source File: iniTimeConst.F90) . . . 672
75.2.4 iniTimeVar (Source File: iniTimeVar.F90) . . . 672
75.3 Fortran: Module Interface atmdrvMod (Source File: atmdrvMod.F90) . . . 673
75.3.1 atmdrv (Source File: atmdrvMod.F90) . . . 673
75.3.2 bufferunpack3d . . . 675
75.3.3 bufferunpack4d . . . 676
75.4 Fortran: Module Interface debugutilitiesmodule (Source File: debugutilitiesmodule.F90) . . . 677
75.4.1 DumAssert . . . 678
75.4.2 DumEnter . . . 679
75.4.3 DumLeave . . . 680
75.5 Fortran: Module Interface decompmodule (Source File: decompmodule.F90) . . . 681
75.5.1 DecompFree . . . 683
75.5.2 DecompCopy . . . 684
75.5.3 DecompPermute . . . 685
75.5.4 DecompRegular1D . . . 687
75.5.5 DecompRegular2D . . . 688
75.5.6 DecompRegular3D . . . 690
75.5.7 DecompRegular3Dorder . . . 693
75.5.8 DecompCreateIrr . . . 696
75.5.9 DecompCreateTags . . . 698
75.5.10 DecompGlobalToLocal . . . 700
75.5.11 DecompLocalToGlobal . . . 701
75.5.12 DecompInfo . . . 703
75.6 Fortran: Module Interface ghostmodule (Source File: ghostmodule.F90) . . . 703
75.6.1 GhostFree . . . 705
75.6.2 GhostCopy . . . 705
75.6.3 GhostIrregular . . . 706
75.6.4 GhostRegular1D . . . 708
75.6.5 GhostRegular2D . . . 710
75.6.6 GhostRegular3D . . . 713
75.6.7 GhostInfo . . . 716
75.7 Fortran: Module Interface parutilitiesmodule (Source File: parutilitiesmodule.F90) . . . 717
75.7.1 ParInit . . . 720
75.7.2 ParExit . . . 722
75.7.3 ParSplit . . . 723

75.7.4 ParFree . . . 724
75.7.5 ParPatternGhost . . . 725
75.7.6 ParPatternDecompToDecomp . . . 728
75.7.7 ParPatternDecompToGhost . . . 732
75.7.8 ParPatternFree . . . 736
75.7.9 ParScatterReal . . . 738
75.7.10 ParScatterInt . . . 741
75.7.11 ParGatherReal . . . 744
75.7.12 ParGatherInt . . . 747
75.7.13 ParBeginTransferReal . . . 750
75.7.14 ParBeginTransferPattern . . . 753
75.7.15 ParEndTransferReal . . . 755
75.7.16 ParEndTransferPattern . . . 757
75.7.17 ParExchangeVectorReal . . . 758
75.7.18 ParExchangeVectorInt . . . 761
75.7.19 ParCollectiveBarrier — Barrier: Simplest collective op. (Source File: parutilitiesmodule.F90) . . . 764
75.7.20 ParCollective0D — Perform global Collective of a scalar (Source File: parutilitiesmodule.F90) . . . 764
75.7.21 ParCollective1D — Perform component-wise global Collective of a vector (Source File: parutilitiesmodule.F90) . . . 766
75.7.22 ParCollective2D — Perform component-wise collective operation (Source File: parutilitiesmodule.F90) . . . 767
75.7.23 ParCollective3D — Perform component-wise global Collective of a vector (Source File: parutilitiesmodule.F90) . . . 770
75.7.24 ParCollective0DInt — Perform global Collective of a scalar (Source File: parutilitiesmodule.F90) . . . 772
75.7.25 ParCollective1DInt — Perform component-wise global (Source File: parutilitiesmodule.F90) . . . 773
75.7.26 scatter — wrapper for PILGRIM utility (Source File: puminterfaces.F90) . . . 775
75.7.27 gather — wrapper for PILGRIM utility (Source File: puminterfaces.F90) . . . 776
75.7.28 begintransfer — wrapper for PILGRIM utility (Source File: puminterfaces.F90) . . . 777
75.7.29 endtransfer — wrapper for PILGRIM utility (Source File: puminterfaces.F90) . . . 777
75.7.30 exchangevector — wrapper for PILGRIM utility (Source File: puminterfaces.F90) . . . 778
75.8 Fortran: Module Interface redistributemodule (Source File: redistributemodule.F90) . . . 779
75.8.1 RedistributeCreate . . . 780
75.8.2 RedistributePerform . . . 782
75.8.3 RedistributeFree . . . 784
75.8.4 RedistributeStart . . . 785
75.8.5 RedistributeFinish . . . 786

76 HySSIB Land Surface Model **788**
76.0.6 hyssib_coldstart (Source File: hyssib_coldstart.F90) . . . 788
76.0.7 hyssib_dynsetup (Source File: hyssib_dynsetup.F90) . . . 788
76.0.8 hyssib_f2t (Source File: hyssib_f2t.F90) . . . 788
76.0.9 hyssib_finalize (Source File: hyssib_finalize.F90) . . . 789
76.0.10 hyssib_gfrac.F90 (Source File: hyssib_gfrac.F90) . . . 789
76.1 Fortran: Module Interface hyssib_lsmMod (Source File: hyssib_lsmMod.F90) . . . 790
76.1.1 hyssib_lsm_ini (Source File: hyssib_lsmMod.F90) . . . 791
76.1.2 hyssib_main (Source File: hyssib_main.F90) . . . 791
76.1.3 hyssib_mapvegc.F90 (Source File: hyssib_mapvegc.F90) . . . 792

76.2 Fortran: Module Interface hyssib_module (Source File: hyssib_module.F90) 793
76.2.1 hyssib_output (Source File: hyssib_output.F90) 796
76.2.2 hyssib_readcrd (Source File: hyssib_readcrd.F90) 797
76.2.3 hyssib_readrst (Source File: hyssib_readrst.F90) 798
76.2.4 hyssib_settbot (Source File: hyssib_settbot.F90) 798
76.2.5 hyssib_settopostd (Source File: hyssib_settopostd.F90) 799
76.2.6 hyssib_setup (Source File: hyssib_setup.F90) 799
76.2.7 hyssib_setvegparms (Source File: hyssib_setvegparms.F90) 800
76.2.8 hyssib_totinit (Source File: hyssib_totinit.F90) 801
76.2.9 hyssib_writerst (Source File: hyssib_writerst.F90) 801
76.2.10 hyssib_dump_restart (Source File: hyssib_writerst.F90) 802
76.3 Fortran: Module Interface hyssibalb_module (Source File: hyssibalb_module.F90) 803
76.3.1 hyssibalb_ini (Source File: hyssibalb_module.F90) 804
76.4 Fortran: Module Interface hyssibveg_module (Source File: hyssibveg_module.F90) 804
76.4.1 hyssibveg_ini (Source File: hyssibveg_module.F90) 805

77 Mosaic Land Surface Model 806
77.0.2 mapsib2umd (Source File: mapsib2umd.F90) 806
77.0.3 mos_coldstart (Source File: mos_coldstart.F90) 806
77.0.4 mos_f2t (Source File: mos_f2t.F90) 807
77.0.5 mos_finalize (Source File: mos_finalize.F90) 807
77.0.6 mos_lsmMod (Source File: mos_lsmMod.F90) 808
77.0.7 mos_lsm_ini (Source File: mos_lsmMod.F90) 809
77.0.8 mos_main (Source File: mos_main.F90) 809
77.1 Fortran: Module Interface mos_module (Source File: mos_module.F90) 810
77.1.1 mos_output (Source File: mos_output.F90) 812
77.1.2 mos_readrestart (Source File: mos_readrestart.F90) 813
77.1.3 mos_setup (Source File: mos_setup.F90) 814
77.1.4 mos_writerst (Source File: mos_writerst.F90) 814
77.1.5 mos_dump_restart (Source File: mos_writerst.F90) 815
77.1.6 mosdynp (Source File: mosdynp.F90) 816
77.1.7 paramb (Source File: paramb.F90) 816
77.1.8 readmoscrd (Source File: readmoscrd.F90) 817
77.1.9 setmosp (Source File: setmosp.F90) 817
77.2 Fortran: Module Interface sibalb_module (Source File: sibalb_module.F90) 818
77.2.1 texture (Source File: texture.F90) 818

78 Noah land surface model version 2.7.1 819
78.0.2 noah271_gce_f2t (Source File: noah271_gce_f2t.F90) 819
78.0.3 noah271_setgceexport.F90 (Source File: noah271_setgceexport.F90) 820
78.0.4 noah271_setwrfesmfexport.F90 (Source File: noah271_setwrfesmfexport.F90) 820
78.0.5 noah271_wrf_esmff2t (Source File: noah271_wrf_esmff2t.F90) 821
78.0.6 noah271_setwrfexport.F90 (Source File: noah271_setwrfexport.F90) 821
78.0.7 noah271_descale_soilm (Source File: noah271_descale_soilm.F90) 824
78.0.8 noah271_getsoilm (Source File: noah271_getsoilm.F90) 825
78.0.9 noah271_getsynsmpred (Source File: noah271_getsynsmpred.F90) 826
78.0.10 noah271_map_sm (Source File: noah271_map_sm.F90) 826
78.0.11 noah271_qc_soilmobs (Source File: noah271_qc_soilmobs.F90) 827
78.0.12 noah271_qcsoilm (Source File: noah271_qcsoilm.F90) 827
78.0.13 noah271_scale_soilm (Source File: noah271_scale_soilm.F90) 828
78.0.14 noah271_setsoilm (Source File: noah271_setsoilm.F90) 829

78.0.15 noah271_transform_synsm (Source File: noah271_transform_synsm.F90) 829
78.0.16 noah271_updatesoilm (Source File: noah271_updatesoilm.F90) 830
78.0.17 noah271_getsoilm (Source File: noah271_write_soilm.F90) 830
78.0.18 noah271_calc_tskin (Source File: noah271_calc_tskin.F90) 832
78.0.19 noah271_coldstart (Source File: noah271_coldstart.F90) 832
78.0.20 noah271_dynsetup (Source File: noah271_dynsetup.F90) 833
78.0.21 noah271_f2t (Source File: noah271_f2t.F90) 833
78.0.22 noah271_finalize (Source File: noah271_finalize.F90) 834
78.1 Fortran: Module Interface noah271_lsmMod (Source File: noah271_lsmMod.F90) 834
78.1.1 noah271_lsm_ini (Source File: noah271_lsmMod.F90) 835
78.1.2 noah271_main (Source File: noah271_main.F90) 836
78.1.3 sh2o_init.F90 (Source File: noah271_main.F90) 836
78.2 Fortran: Module Interface noah271_module (Source File: noah271_module.F90) 838
78.2.1 noah271_output (Source File: noah271_output.F90) 839
78.2.2 noah271_setsoils (Source File: noah271_setsoils.F90) 840
78.2.3 noah271_settbot (Source File: noah271_settbot.F90) 841
78.2.4 noah271_setup (Source File: noah271_setup.F90) 841
78.2.5 noah271_setvegparms (Source File: noah271_setvegparms.F90) 842
78.2.6 noah271_writerst (Source File: noah271_writerst.F90) 842
78.2.7 noah271_dump_restart (Source File: noah271_writerst.F90) 843
78.2.8 noah271rst (Source File: noah271rst.F90) 844
78.2.9 readnoah271crd (Source File: readnoah271crd.F90) 845

79 Template **845**
79.0.10 readtemplatecrd (Source File: readtemplatecrd.F90) 845
79.0.11 template_dynsetup (Source File: template_dynsetup.F90) 846
79.0.12 template_f2t (Source File: template_f2t.F90) 846
79.0.13 template_finalize (Source File: template_finalize.F90) 847
79.1 Fortran: Module Interface template_lsmMod (Source File: template_lsmMod.F90) 847
79.1.1 template_lsm_ini (Source File: template_lsmMod.F90) 848
79.1.2 template_main (Source File: template_main.F90) 848
79.1.3 template_main.F90 (Source File: template_main.F90) 848
79.2 Fortran: Module Interface template_module.F90 (Source File: template_module.F90) 849
79.2.1 template_output (Source File: template_output.F90) 849
79.2.2 template_setup (Source File: template_setup.F90) 850
79.2.3 template_writerst (Source File: template_writerst.F90) 850
79.2.4 templaterst (Source File: templaterst.F90) 851

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 (3.9.2)
call to initialize configuration tool and read model independent options

lisinit (3.45.2)
call to initialize lis based on the runmode

lisrun (3.45.4)
call to run lis based on the runmode

lisfinalize (3.45.6)
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(LIS_rc%runmode)
  call lisrun(LIS_rc%runmode)
  call lisfinalize(LIS_rc%runmode)
```

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 implements routines to read various sources of observations to be used for data assimilation.

REVISION HISTORY:

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

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), pointer :: LIS_OBS_State(:,:)
 type(ESMF_State), pointer :: 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 an empty ESMF State that will be used to incorporte 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 (3.36.2)
invokes the generic method in the registry to set up the structures to read the observation data, for assimilation

perturbinit (3.44.2)
invokes the generic method in the registry to perturb the prognostic variables using the specified algorithm.

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 (3.36.4)
method 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 (3.44.6)
invokes the specified algorithm to perturb the observations.

REVISION HISTORY:

```
 14 Nov 2002    Sujay Kumar  Initial Specification

use ESMF_Mod

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
```

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
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_Mod
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:

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 (3.16.4)
This routine maps a variable 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 (3.16.4)
This routine maps a variable to the history writing routines

3.3 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

3.3.1 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

REVISION HISTORY:

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

3.4 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)

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

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

glbnch_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.

domain Choice of "LIS domain" (Please refer to `domain_module` for the definition of a domain in LIS.

lsm Choice of the land surface model in LIS.

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)

gfracsrc Choice of the greenness 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)

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

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

mina 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 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) = 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)

force Choice of forcing data

base_ecor Choice of topographical downscaling method for base forcing

use_prism option to specify the use of PRISM downscaling (0-do not use, 1-use)

nf Number of forcing variables

nbf Number of base forcing variables

nsf Number of supplemental forcing variables for each instance.

supp_ecor Choice of topographical downscaling method for supplemental forcing

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.

base_interp Spatial interpolation option for base forcing (1-bilinear, 2-conservative, 3-neighbor)

supp_interp Spatial interpolation option for supplemental forcing

base_tinterp Temporal interpolation option for base forcing (1-next, 2-uber-next)

supp_tinterp Temporal interpolation option for supplemental forcing

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

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

findtime1, findtime2 boolean flags to indicate which time is to be read for temporal interpolation, for each nest

nsuppforc Number of supplemental forcing instances

suppforc Choice of supplemental forcing

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)

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

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

laifile Name of the LAI data source, for each nest

saifile Name of the SAI data source, for each nest

gfracfile Name of the greenness 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

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

albInterval Frequency of albedo climatology in months

gfracInterval Frequency of greenness 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

wfor Option to output forcing data

wopt Output methodology (0=no output, 1-tiled, 2-gridded)

wout Output data format (1-binary, 2-grib, 3-netcdf)

wparam Output parameters (0-do not write, 1-write)

wsingle Option to write each variable to a separate file

wstyle Output file naming style (1 - hierarchical, 4 - AFWA/WMO)

grib_table Grib Table version number

grib_center_id Grib center id

grib_process_id Grib process id

grib_grid_id Grib grid id

expcode Three character experiment code

startcode Start mode (1-restart, 2-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 Choice of optimization/uncertianty estimation set (obs + LSM)

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

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
```

3.4.1 LIS_PRIV_tileMod (Source File: LIS_PRIV_tileMod.F90)

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

3.4.2 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
```

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

REVISION HISTORY:

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

3.5.1 LIS_RTM_init (Source File: LIS_RTMMod.F90)

INTERFACE:

```
subroutine LIS_RTM_init
```

DESCRIPTION:

USES:

```
use ESMF_Mod
use LIS_coreMod,      only : LIS_rc, LIS_config
use LIS_logMod,       only : LIS_verify
use LIS_timeMgrMod,   only : LIS_clock, LIS_seconds2time, LIS_calendar
```

3.5.2 LIS_RTM_run (Source File: LIS_RTMMod.F90)

INTERFACE:

```
subroutine LIS_RTM_run(n)
```

USES:

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

DESCRIPTION:

USES:

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

3.5.3 LIS_RTM_output (Source File: LIS_RTMMod.F90)

INTERFACE:

```
subroutine LIS_RTM_output(n)
```

USES:

```
use LIS_coreMod,  only : LIS_rc

implicit none

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

3.5.4 LIS_RTM_final (Source File: LIS_RTMMod.F90)

INTERFACE:

```
subroutine LIS_RTM_final
```

DESCRIPTION:

USES:

3.6 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.

3.6.1 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_Mod

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
```

3.6.2 LIS_albedo_setup (Source File: LIS_albedoMod.F90)

INTERFACE:

```
subroutine LIS_albedo_setup
```

USES:

```
  use LIS_coreMod,    only : LIS_rc, LIS_domain, LIS_config
  use LIS_timeMgrMod, only : LIS_setAlarm, LIS_clock, &
       LIS_computeTemporalWeights, LIS_calendar
  use LIS_logMod,     only : LIS_verify
```

DESCRIPTION:

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

LIS_read_mxsnalb (3.6.3)
method to read the max snow albedo

LIS_setAlarm (3.31.13)
sets the monthly alarm to read the albedo climatology

readalbedo (3.34.6)
invokes the generic method in the registry to read the albedo climatology data

albedosetup (3.34.4)
calls the registry to invoke the albedo setup method.

3.6.3 LIS_read_mxsnalb (Source File: LIS_albedoMod.F90)

INTERFACE:

```
subroutine LIS_read_mxsnalb(n)
```

USES:

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

ARGUMENTS:

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

DESCRIPTION:

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

n index of the domain or nest.

The routines invoked are:

readmxsnoalb (3.34.2)
Calls the generic method in the registry to read the max snow albedo

---

3.6.4 LIS_read_albedo (Source File: LIS_albedoMod.F90)

INTERFACE:

```
subroutine LIS_read_albedo(n)
```

USES:

```
use LIS_coreMod, only    : LIS_rc, LIS_domain
use LIS_timeMgrMod, only : LIS_computeTemporalWeights
```

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 (3.31.21)
computes the temporal interpolation weights

readalbedo (3.34.6)
invokes the generic method in the registry to read the albedo climatology data

---

3.6.5 LIS_albedo_finalize (Source File: LIS_albedoMod.F90)

INTERFACE:

```
subroutine LIS_albedo_finalize
```

USES:

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

DESCRIPTION:

Deallocates objects created in this module

3.6.6 LIS_diagnosealbedo (Source File: LIS_albedoMod.F90)

INTERFACE:

```
subroutine LIS_diagnosealbedo(n)
```

USES:

```
use LIS_coreMod, only    : LIS_rc, LIS_domain
use LIS_histDataMod, only : LIS_diagnoseOutputVar, &
     LIS_MOC_SNFRALBEDO, LIS_MOC_MXSNALBEDO
```

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 (3.16.4)
generic routine to map a single variable to the LIS history writer

---

3.6.7 LIS_albedo_reset (Source File: LIS_albedoMod.F90)

INTERFACE:

```
subroutine LIS_albedo_reset
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain, LIS_config
use LIS_timeMgrMod, only : LIS_setAlarm, LIS_clock, &
     LIS_computeTemporalWeights
use LIS_logMod,     only : LIS_verify
```

DESCRIPTION:

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

LIS_read_mxsnalb (3.6.3)
method to read the max snow albedo

LIS_setAlarm (3.31.13)
sets the monthly alarm to read the albedo climatology

albedosetup (3.34.4)
calls the registry to invoke the albedo setup method.

readalbedo (3.34.6)
invokes the generic method in the registry to read the albedo climatology data

---

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

The code in this file provides interfaces to manage different meteorological forcing analyses

3.7.1 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)

REVISION HISTORY:

```
 14 Nov 2002    Sujay Kumar  Initial Specification
 10 Oct 2006:   Sujay Kumar: Switched to using ESMF_State for storing
                             forcing data.

use ESMF_Mod
use LIS_FORC_AttributesMod

implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: LIS_baseforcing_init      !initialize base forcing setup
public :: LIS_get_base_forcing    !retrieve data and interpolate
                                            !spatially and temporally
public :: LIS_perturb_forcing       !perturbs the met forcing variables
public :: LIS_baseforcing_reset     !resets required forcing variables
public :: LIS_baseforcing_finalize  !cleanup allocated structures
public :: diagnoseForcingOutput ! prepares forcing variables for output
```

PUBLIC TYPES:

```
public   :: LIS_forc
public   :: LIS_FORC_State
public   :: LIS_FORC_Pert_State
```

3.7.2 LIS_baseforcing_init (Source File: LIS_baseforcingMod.F90)

INTERFACE:

```
interface LIS_baseforcing_init
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure baseforcing_init_offline
module procedure baseforcing_init_coupled
```

DESCRIPTION:

Initialization routine for baseforcing structures. There are two private methods based on whether LIS is run in an offline mode or coupled mode.

3.7.3 baseforcing_init_offline (Source File: LIS_baseforcingMod.F90)

INTERFACE:

```
!Private name: call using LIS_baseforcing_init
subroutine baseforcing_init_offline
```

USES:

```
use LIS_coreMod,               only : LIS_rc
use LIS_baseforcing_pluginMod, only : LIS_baseforcing_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_baseforcing_plugin (4.5.1)
sets up function table registries for implemented base forcing analyses

allocate_forcing_mem (3.7.10)
allocate memory for required structures

defineNative (3.38.6)
invokes the generic method in the registry to define the native domain of the base forcing scheme

forcingPerturbSetup (3.7.5)
setup and allocate structures required for forcing perturbations.

3.7.4 baseforcing_init_coupled (Source File: LIS_baseforcingMod.F90)

INTERFACE:

```
subroutine baseforcing_init_coupled(flag)
```

USES:

```
use LIS_coreMod,   only : LIS_rc

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:

allocate_forcing_mem (3.7.10)
allocate memory for required structures

forcingPerturbSetup (3.7.5)
setup and allocate structures required for forcing perturbations.

3.7.5 forcingPerturbSetup (Source File: LIS_baseforcingMod.F90)

INTERFACE:

```
subroutine forcingPerturbSetup
```

USES:

```
use LIS_coreMod,   only : LIS_rc, LIS_vecTile
use LIS_logMod,    only : LIS_logunit, LIS_verify, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber
```

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.

3.7.6 LIS_perturb_forcing (Source File: LIS_baseforcingMod.F90)

INTERFACE:

```
subroutine LIS_perturb_forcing(n)
```

USES:

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

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 (3.7.13)
routine to map a forcing fields to the LIS history writer

3.7.7 LIS_get_base_forcing (Source File: LIS_baseforcingMod.F90)

INTERFACE:

```
subroutine LIS_get_base_forcing(n)
```

USES:

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

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:

retrieveforcing (3.38.2)
invokes the generic method in the registry to retrieve the base forcing data

timeinterp (3.38.4)
invokes the generic method in the registry to perform temporal interpolation

LIS_lapseRateCorrection (3.18.7)
method to apply lapse-rate based topographical corrections

LIS_microMetCorrection (3.20.17)
method to apply topographical corrections (not currently supported)

3.7.8 LIS_baseforcing_reset (Source File: LIS_baseforcingMod.F90)

INTERFACE:

```
subroutine LIS_baseforcing_reset
```

USES:

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

DESCRIPTION:

This subroutine issues the invocation to resetvariables for specific forcing scheme.

3.7.9 LIS_baseforcing_finalize (Source File: LIS_baseforcingMod.F90)

INTERFACE:

```
subroutine LIS_baseforcing_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 the forcing scheme.
The methods invoked are:

forcingfinalize (3.38.8)
invokes the generic method in the registry to cleanup the allocated structures for the base forcing scheme.

3.7.10 allocate_forcing_mem (Source File: LIS_baseforcingMod.F90)

INTERFACE:

```
subroutine allocate_forcing_mem()
```

USES:

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

DESCRIPTION:

This subroutine 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

---

3.7.11 get_forcingvar_attributes (Source File: LIS_baseforcingMod.F90)

INTERFACE:

```
subroutine get_forcingvar_attributes(forcConfig, forc_attrib, forcname, varcount, &
     checkentry)
```

USES:

```
use LIS_logMod,   only : LIS_logunit

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

---

3.7.12 add_forcing_fields (Source File: LIS_baseforcingMod.F90)

INTERFACE:

```
subroutine add_forcing_fields(n, FORC_State, forc_attrib)
```

USES:

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

implicit none
```

ARGUMENTS:

```
integer                   :: n
type(ESMF_State)          :: FORC_State
type(forc_attrib_type)    :: forc_attrib
```

DESCRIPTION:

This subroutine creates fields for the specified forcing variables and initializes them to be undefined values.

3.7.13 diagnoseForcingOutput (Source File: LIS_baseforcingMod.F90)

INTERFACE:

```
subroutine diagnoseForcingOutput(n)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_logMod,         only : LIS_verify
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 (3.16.4)
generic routine to map a single variable to the LIS history writer

---

3.8 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
```

3.9 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

3.9.1 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_nchs 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_Mod
  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_ticktime
  public :: LIS_endofrun
  public :: LIS_TimeToRunNest
  public :: LIS_resetTimeMgr
  public :: LIS_finalize

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

PUBLIC TYPES:

```
 ------------------------------------------------------------------------------
  public :: LIS_rc
  public :: LIS_domain
  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_nchs
public :: LIS_ngrids
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_vecGrid
public :: LIS_ensOnGrid
```

3.9.2 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.

3.9.3 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 (3.9.11)
performs SPMD initializations

LIS_log_init (3.19.2)
initializes the LIS log handler

LIS_readConfig (3.26.2)
reads the model independent options

spmd_setup (3.9.13)
allocates memory for SPMD variables

LIS_timemgr_init (**??**)
initializes the time manager

3.9.4 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 (**??**)
performs SPMD initializations

lis_log_init (**??**)
initializes the LIS log handler

LIS_readConfig (3.26.2)
reads the model independent options

spmd_setup (3.9.13)
allocates memory for SPMD variables

LIS_timemgr_init (**??**)
initializes the time manager

3.9.5 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 (**??**)
performs SPMD initializations

lis_log_init (**??**)
initializes the LIS log handler

LIS_readConfig (3.26.2)
reads the model independent options

spmd_setup (3.9.13)
allocates memory for SPMD variables

LIS_timemgr_init (**??**)
initializes the time manager

3.9.6 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 (3.31.6)
advances the clock

3.9.7 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:

is_last_step (**??**)
check if the clock has reached the stop time

3.9.8 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

---

3.9.9 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:

resetclock (**??**)

3.9.10 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 (3.9.14)
cleanup SPMD structures

3.9.11 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.

3.9.12 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.

3.9.13 spmd_setup (Source File: LIS_coreMod.F90)

INTERFACE:

```
subroutine spmd_setup(nnest, ndas)

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

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

nnest Number of nests or domains

3.9.14 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.

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

The module controls the management of different data assimilation implementations.

3.10.1 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

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
```

3.10.2 LIS_dataassim_init (Source File: LIS_dataAssimMod.F90)

INTERFACE:

```
subroutine LIS_dataassim_init
```

USES:

```
  use ESMF_Mod
  use LIS_coreMod,                  only : LIS_rc, LIS_config
  use LIS_dataassim_pluginMod,      only : LIS_dataassim_plugin
  use LIS_biasEstimation_pluginMod, only : LIS_biasEstimation_plugin
```

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 (**??**)
sets up function table registries for implemented bias estimation algorithms.

dataassiminit (3.36.10)
invokes the generic method in the registry to initialize the data assimilation algorithm- structures

dataassimsetup (3.36.12)
invokes the generic method in the registry to set up the data assimilation algorithm- structures

biasestimationsetup (3.35.4)
invokes the generic method in the registry to set up the bias estimation algorithm structures.

3.10.3 LIS_dataassim_run (Source File: LIS_dataAssimMod.F90)

INTERFACE:

```
subroutine LIS_dataassim_run(n)
```

USES:

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

DESCRIPTION:

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

computeIncrements (3.36.16)
computes the increments from the analysis

applyBiasCorrection (**??**)
applies the bias correction increments

applyIncrements (3.36.14)
applies the increments from the analysis

3.10.4 LIS_dataassim_output (Source File: LIS_dataAssimMod.F90)

INTERFACE:

```
subroutine LIS_dataassim_output(n)
```

USES:

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

DESCRIPTION:

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

daoutput (3.36.20)
invokes the generic method in the registry to write data assimilation specific output

---

3.10.5 LIS_dataassim_finalize (Source File: LIS_dataAssimMod.F90)

INTERFACE:

```
subroutine LIS_dataassim_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 the data assimilation implementations.
The calling sequence is:

dafinalize (3.36.18)
invokes the generic method in the registry to cleanup the allocated structures for the data assimilation algorithm

---

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

The code in this file provides interfaces to manages different running domain implementations

3.11.1 Overview

The definition of a domain in LIS is includes two dimensions. A LIS domain is defined by the projection of the running grid and the input data ordering rules. For example, a domain using latlon projection with a SW to NE data ordering is considered different from a domain using latlon projection with NW to SE data ordering. Each input data used with a domain definition is expected to be transformed to the specified projection and data ordering rules through the interfaces defined in the `param_module`.

REVISION HISTORY:

```
 17 Feb 2004    Sujay Kumar  Initial Specification
 24 Aug 2008    Sujay Kumar  Implemented halo support

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
```

3.11.2 LIS_domain_init (Source File: LIS_domainMod.F90)

INTERFACE:

```
subroutine LIS_domain_init
```

USES:

```
use ESMF_Mod
use LIS_coreMod,      only : LIS_rc, LIS_vecTile, LIS_vecGrid, &
     LIS_ensOnGrid, LIS_localPet, LIS_masterproc, LIS_npes, LIS_goffsets, &
     LIS_toffsets, LIS_ooffsets, LIS_nchs, LIS_ngrids, LIS_tdeltas, &
     LIS_gdeltas, LIS_odeltas
use LIS_logMod,       only : LIS_verify
use LIS_LMLCMod,      only : LIS_LMLC_init
use LIS_topoMod,      only : LIS_topo_init
use LIS_domain_pluginMod, only : LIS_domain_plugin
use LIS_gribMod,      only : LIS_grib_init, LIS_grib2_init
use LIS_histDataMod,  only : LIS_histDataInit
use LIS_mpiMod
```

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, 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 (3.37.4)
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 (3.32.2)
initialize topography data structures

makedomain (3.37.2)
invokes the generic method in the registry to create the grid and tile spaces.

3.11.3 LIS_domain_setup (Source File: LIS_domainMod.F90)

INTERFACE:

```
subroutine LIS_domain_setup(n)
```

USES:

```
use LIS_coreMod,   only : LIS_rc, LIS_domain, LIS_localPet, LIS_npes, &
     LIS_deltas, LIS_offsets, LIS_ews_ind, LIS_ewe_ind, &
     LIS_nss_ind, LIS_nse_ind, LIS_npes, LIS_masterproc, &
     LIS_nss_halo_ind, LIS_nse_halo_ind, LIS_ews_halo_ind, LIS_ewe_halo_ind
use LIS_mpiMod
```

ARGUMENTS:

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

DESCRIPTION:

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

3.11.4 LIS_quilt_domain (Source File: LIS_domainMod.F90)

INTERFACE:

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

USES:

```
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, LIS_deltas, LIS_offsets, &
     LIS_masterproc, LIS_npes
use LIS_logMod,   only  : LIS_logunit
use LIS_mpiMod
```

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.

3.11.5 LIS_setParameterDomainSpecs (Source File: LIS_domainMod.F90)

INTERFACE:

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

USES:

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

ARGUMENTS:

```
real,   intent(in)    :: param_dd(6)
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

---

3.11.6 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 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_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
```

3.12.1 LIS_readData (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
interface LIS_readData
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure read2Ddata
module procedure readLCdata
```

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.

3.12.2 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
```

3.12.3 LIS_create_output_directory (Source File: LIS_fileIOMod.F90)

INTERFACE:

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

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
integer, intent(IN), optional :: style
```

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.
style option 1:

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

style option 2:

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

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

3.12.4 LIS_create_output_filename (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine LIS_create_output_filename(n, fname, model_name, writeint, wout, style)
```

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
integer, intent(in), optional          :: writeint ! output writing interval
integer, intent(in), optional          :: wout ! output format
integer, intent(in), optional          :: style ! output directory style
```

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 the following default format (style==1) ¡output directory¿/EXP¡expno¿/¡model name¿/¡yr¿/¡yrmoda¿/¡yrmodahrmn¿.¡extension¿ Style option ==2 corresponds to the following style: ¡output directory¿/EXP¡expno¿/¡model name¿.¡yrmodahrmn¿.¡extension¿ Style option ==3 : ¡output directory¿/EXP¡expno¿/¡model name¿.¡extension¿ Style option ==4 : ¡output directory¿/¡AFWA Weather product style¿ Style option 4: 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 Style option 5: ¡output directory¿/EXP¡expno¿/¡model name¿/¡yr¿/¡yrmoda¿/¡yrmodahrmnss¿.¡extension¿
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

wout output writing format

style style option as described above

3.12.5 LIS_create_restart_filename (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine LIS_create_restart_filename(n, fname,model_name,extn, style)
```

USES:

```
use LIS_coreMod,  only : LIS_rc

implicit none
```

ARGUMENTS:

```
integer, intent(in)           :: n
character(len=*), intent(in)  :: model_name ! needed for gswp run
character(len=*), intent(out) :: fname
character(len=*), intent(in)  :: extn
integer, intent(in), optional :: style ! output directory style
```

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¿/¡model 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

extn string extension to be used in the filename

style output directory style

3.12.6 LIS_create_stats_filename (Source File: LIS_fileIOMod.F90)

INTERFACE:

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

USES:

```
use LIS_coreMod,  only : LIS_rc

implicit none
```

ARGUMENTS:

```
integer, intent(in)           :: n
character(len=*), intent(out) :: fname
character(len=*), intent(in)  :: mname
integer, intent(in), optional :: style ! output directory style
```

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

3.12.7 LIS_create_innov_filename (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine LIS_create_innov_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 innovations files. The convention used in LIS creates a restart filename in the following format.
¡output directory¿/EXP¡expno¿/¡model name¿¡timestamp¿.innov
The arguments are:

n index of the domain or nest

fname the created file name.

mname string describing the name of the model

3.12.8 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

3.12.9 LIS_create_daspread_filename (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine LIS_create_daspread_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 data assimilation diagnostic - the ensemble spread. The convention used in LIS creates a restart filename in the following format.
¡output directory¿/EXP¡expno¿/¡model name¿¡timestamp¿.spread
The arguments are:

n index of the domain or nest

fname the created file name.

mname string describing the name of the model

3.12.10 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 (**??**)
abort the program in case of a fatal error

3.12.11 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 (**??**)
abort the program in case of a fatal error

3.12.12 LIS_create_gain_filename (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine LIS_create_gain_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 gainations files. The convention used in LIS creates a restart filename in the following format.
¡output directory¿/EXP¡expno¿/¡model name¿¡timestamp¿.gain
The arguments are:

n index of the domain or nest

fname the created file name.

mname string describing the name of the model

3.12.13 read2Ddata (Source File: LIS_fileIOMod.F90)

INTERFACE:

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

USES:

```
use LIS_coreMod,  only : LIS_rc, LIS_domain, LIS_localPet
use gaussian_mod, only : gaussian_find_row, gaussian_find_col
use map_utils,    only : ij_to_latlon
use LIS_logMod,   only : LIS_abort, LIS_endrun, LIS_logunit

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.

3.12.14 readLCdata (Source File: LIS_fileIOMod.F90)

INTERFACE:

```
subroutine readLCdata(n, ftn, gridDesc, array, domain)
```

USES:

```
use LIS_coreMod,  only : LIS_rc, LIS_domain, LIS_localPet
use gaussian_mod, only : gaussian_find_row, gaussian_find_col
use map_utils,    only : ij_to_latlon

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine reads a 3-d vegetation distribution data from a binary sequential access file.

3.12.15 LIS_readDomainConfigSpecs (Source File: LIS_fileIOMod.F90)

INTERFACE:

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

USES:

```
use ESMF_Mod
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.

3.12.16 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.

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

The code in this file implements routines to read greenness fraction data.

3.13.1 Overview

This routines in this module provides routines to read the greenness fraction 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_Mod

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 TYPES:

```
-------------------------------------------------------------------------------
 public :: LIS_gfrac !data structure containing greenness fraction data.
```

3.13.2 LIS_greenness_setup (Source File: LIS_gfracMod.F90)

INTERFACE:

```
subroutine LIS_greenness_setup
```

USES:

```
 use LIS_coreMod,    only : LIS_rc, LIS_config
 use LIS_logMod,     only : LIS_verify
 use LIS_timeMgrMod, only : LIS_clock, LIS_calendar, &
      LIS_setAlarm, LIS_computeTemporalWeights
```

DESCRIPTION:

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

gfracsetup (3.39.2)
calls the registry to invoke the gfrac setup methods.

3.13.3 LIS_read_greenness (Source File: LIS_gfracMod.F90)

INTERFACE:

```
subroutine LIS_read_greenness(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_timeMgrMod, only : LIS_computeTemporalWeights

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 (3.31.21)
computes the interpolation weights

readgfrac (3.39.4)
invokes the generic method in the registry to read the greenness climatology data

---

3.13.4 LIS_greenness_finalize (Source File: LIS_gfracMod.F90)

INTERFACE:

```
subroutine LIS_greenness_finalize
```

USES:

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

DESCRIPTION:

Deallocates objects created in this module

3.13.5 LIS_diagnosegfrac (Source File: LIS_gfracMod.F90)

INTERFACE:

```
subroutine LIS_diagnosegfrac(n)
```

USES:

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

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 (3.16.4)
generic routine to map a single variable to the LIS history writer

---

3.13.6 LIS_greenness_reset (Source File: LIS_gfracMod.F90)

INTERFACE:

```
subroutine LIS_greenness_reset
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_config
use LIS_logMod,     only : LIS_verify
use LIS_timeMgrMod, only : LIS_clock, &
     LIS_setAlarm, LIS_computeTemporalWeights
```

DESCRIPTION:

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

gfracsetup (3.39.2)
calls the registry to invoke the gfrac data reading methods.

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

The code in this file sets up various grib related variables used for writing grib output.

REVISION HISTORY:

```
15 May 2006: James Geiger; Initial code
15 Dec 2006: Charles Alonge; Updated parameter ids, added
             support for multiple grid output, and code cleanup
 4 Apr 2007: Sujay Kumar; Added a derived data type and support
             for parallel runs
14 Jul 2008: Sujay Kumar; Added support for GRIB2 format
```

3.14.1 LIS_grib_init (Source File: LIS_gribMod.F90)

INTERFACE:

```
subroutine LIS_grib_init()
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, &
     LIS_masterproc, LIS_gdeltas, LIS_goffsets,&
     LIS_ews_ind,LIS_nss_ind,LIS_ewe_ind,LIS_nse_ind,LIS_npes
use LIS_histDataMod
```

DESCRIPTION:

Initializes the grib_pds_vals array. This array contains the variable-dependent parameters for Section 1 (PDS) of the GRIB record. This routine also initializes the landmask. The landmask is then used create a logical bitmap for use in the GRIB records.

3.14.2 LIS_grib2_init (Source File: LIS_gribMod.F90)

INTERFACE:

```
subroutine LIS_grib2_init
```

USES:

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

DESCRIPTION:

Initializes the grib2_pds_vals array. This array contains the variable-dependent parameters for Section 1 (PDS) of the GRIB record.

3.14.3 LIS_grib1_finalize (Source File: LIS_gribMod.F90)

INTERFACE:

```
subroutine LIS_grib1_finalize(index, sect1, time_unit, time_P1, time_P2, &
     time_range)
```

DESCRIPTION:

This routine completes the definition of the sect1 array. It sets the variable-dependent parameters for Section 1 (PDS) of the GRIB record.
The arguments are:

index Index into the grib_pds_vals array corresponding to the variable being processed

sect1 Parameters for Section 1 (PDS) of the GRIB record

time_unit Units in time for output contained in GRIB record (See GRIB Table 4)

time_P1 Time1 for GRIB record descriptor (See GRIB Table 5)

time_P2 Time2 for GRIB record descriptor (See GRIB Table 5)

time_range Time range descriptor for P1 and P2 (See GRIB Table 5)

3.14.4 LIS_grib1_setup1 (Source File: LIS_gribMod.F90)

INTERFACE:

```
subroutine LIS_grib1_setup1 (sect1, info, bitmap, date10_12, &
                             table_id, center_id, subcenter_id, process_id, grid_id)
```

DESCRIPTION:

This routine initializes the sect1 array. It sets the variable-independent parameters for Section 1 (PDS) of the GRIB record.
The arguments are:

sect1 Parameters for Section 1 (PDS) of the GRIB record

info Sizes of grib sections

bitmap Logical varible indicating if a bitmap is used in grib

date10_12 10 digit date/time group of daily cycle – yyyymmddhh

---

3.14.5 LIS_grib1_setup2 (Source File: LIS_gribMod.F90)

INTERFACE:

```
subroutine LIS_grib1_setup2 (sect2, info, imax, jmax, n)
```

DESCRIPTION:

This routine initializes the sect2 array. It sets the parameters for Section 2 (GDS) of the GRIB record.
The arguments are:

sect2 Parameters for Section 2 (GDS) of the GRIB record

info Sizes of grib sections

imax Number of points in the i-direction

jmax Number of points in the j-direction

n Number of LIS nest

3.14.6 LIS_grib1_setup3 (Source File: LIS_gribMod.F90)

INTERFACE:

```
subroutine LIS_grib1_setup3 (sect3, imax, jmax, numpts, &
     info, bitmap, size_3, numoct_3, &
     land )
```

DESCRIPTION:

This routine initializes the sect3 array. It sets the parameters for Section 3 (BMS) of the GRIB record.
The arguments are:

sect3 Parameters for Section 3 (BMS) of the GRIB record

imax Number of points in the i-direction

jmax Number of points in the j-direction

numpts Number of land points

info Sizes of grib sections

bitmap Logical varible indicating if a bitmap is used in grib

size_3 Number of elements in the sect3 array.

numoct_3 Number of octets in the sect3 array.

land Input land mask (1-Land, 0-Water) to create bitmap.

3.14.7 LIS_gribdat (Source File: LIS_gribMod.F90)

INTERFACE:

```
subroutine LIS_gribdat (info, sect1, s1opt, sect2, sect3, dum3d, &
     toplev, botlev, idim, jdim, kdim, size_3)
```

DESCRIPTION:

This routine calls the low-level grib routines, contained in the grib library, to pack the grib record corresponding to the variable contained in dum3d. Then it writes the grib record.
The arguments are:

info Sizes of grib sections

sect1 Parameters for Section 1 (PDS) of the GRIB record

s1opt Optional parameters for Section 1 (PDS) of the GRIB record

sect2 Parameters for Section 2 (GDS) of the GRIB record

sect3 Parameters for Section 3 (BMS) of the GRIB record

dum3d Array of data to be processed

toplev Array of depths for the top of each layer

botlev Array of depths for the bottom of each layer

idim Number of points in the i-direction

jdim Number of points in the j-direction

kdim Number of layers

size_3 Number of elements in the sect3 array.

3.14.8 LIS_grib2_setup (Source File: LIS_gribMod.F90)

INTERFACE:

```
subroutine LIS_grib2_setup(n, igds,igdstmplen, igdstmpl)
```

USES:

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

DESCRIPTION:

This routine the igdstmpl array.

3.14.9 LIS_grib2_finalize (Source File: LIS_gribMod.F90)

INTERFACE:

```
subroutine LIS_grib2_finalize(n, var_index, ipdsnum, ipdstmplen, ipdstmpl, &
     idrsnum, idrstmplen, idrstmpl, numcoord, ibmap)
```

USES:

```
none
```

ARGUMENTS:

```
integer,   intent(in)    :: n
integer,   intent(in)    :: var_index
integer,   intent(inout) :: ipdsnum
integer,   intent(in)    :: ipdstmplen
integer,   intent(inout) :: ipdstmpl(ipdstmplen)
integer,   intent(inout) :: idrsnum
integer,   intent(in)    :: idrstmplen
integer,   intent(inout) :: idrstmpl(idrstmplen)
integer,   intent(inout) :: numcoord
integer,   intent(inout) :: ibmap
```

DESCRIPTION:

This routine initializes the ipdstmpl array.

3.14.10 gr2_convert_lon (Source File: LIS_gribMod.F90)

INTERFACE:

```
subroutine gr2_convert_lon(value)

IMPLICIT NONE
```

ARGUMENTS:

```
integer, intent(inout) :: value
```

DESCRIPTION:

Set longitude to be in the range of 0-360 degrees.

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 July 2008   Sujay Kumar Redesigned the I/O to enable generic I/O for
                   all LSMs.

 real*8  :: time        ! time to begin writing output
 integer :: year        ! year to begin writing output
 integer :: month       ! month to begin writing output
 integer :: day         ! day to begin writing output
 integer :: hour        ! hour to begin writing output
 integer :: min         ! minutes to begin writing output
 integer :: sec         ! seconds to begin writing output

 type(LIS_metadataEntry) :: swnet        ! Net shortwave radiation (surface) (W/m2)
 type(LIS_metadataEntry) :: lwnet        ! Net longwave radiation (surface) (W/m2)
 type(LIS_metadataEntry) :: qle          ! Latent Heat Flux (W/m2)
 type(LIS_metadataEntry) :: qh           ! Sensible Heat Flux (W/m2)
 type(LIS_metadataEntry) :: qg           ! Ground Heat Flux (W/m2)
 type(LIS_metadataEntry) :: qf           ! Energy of fusion (W/m2)
 type(LIS_metadataEntry) :: qv           ! Energy of sublimation (W/m2)
 type(LIS_metadataEntry) :: qtau         ! Momentum flux (N/m2)
 type(LIS_metadataEntry) :: qa           ! Advective flux (W/m2)
 type(LIS_metadataEntry) :: delsurfheat  ! Change in surface heat storage (J/m2)
 type(LIS_metadataEntry) :: delcoldcont  ! Change in snow water content (J/m2)
 type(LIS_metadataEntry) :: br           ! Bowen Ratio
 type(LIS_metadataEntry) :: ef           ! Evaporative Fraction

 type(LIS_metadataEntry) :: snowf        ! Snowfall rate (kg/m2s)
 type(LIS_metadataEntry) :: rainf        ! Rainfall rate (kg/m2s)
 type(LIS_metadataEntry) :: evap         ! Evapotranspiration (kg/m2s)
 type(LIS_metadataEntry) :: qs           ! Surface Runoff(kg/m2s)
 type(LIS_metadataEntry) :: qrec         ! Recharge from river to the floodplain (kg/m2s)
 type(LIS_metadataEntry) :: qsb          ! Subsurface Runoff (kg/m2s)
 type(LIS_metadataEntry) :: qsm          ! Snowmelt (kg/m2s)
 type(LIS_metadataEntry) :: qfz          ! Refreezing of water in the snowpack (kg/m2s)
 type(LIS_metadataEntry) :: qst          ! Snow throughfall (kg/m2s)
 type(LIS_metadataEntry) :: delsoilmoist ! DelSoilMoist
 type(LIS_metadataEntry) :: delswe       ! DelSWE
 type(LIS_metadataEntry) :: delsurfstor  ! Change in surface water storage (kg/m2)
 type(LIS_metadataEntry) :: delintercept ! Change in interception storage (kg/m2)

 type(LIS_metadataEntry) :: snowt        ! Snow surface temperature (K)
 type(LIS_metadataEntry) :: vegt         ! Vegetation canopy temperature (K)
 type(LIS_metadataEntry) :: baresoilt    ! Temperature of bare soil (K)
 type(LIS_metadataEntry) :: avgsurft     ! Average Surface Temperature (K)
 type(LIS_metadataEntry) :: radt         ! Surface Radiative Tempearture (K)
 type(LIS_metadataEntry) :: albedo       ! Surface Albedo (-)
```

```
type(LIS_metadataEntry) :: swe          ! Snow water equivalent (kg/m2)
type(LIS_metadataEntry) :: sweveg       ! SWE intercepted by vegetation (kg/m2)
type(LIS_metadataEntry) :: snowfrac     ! Grid cell snow covered fraction
type(LIS_metadataEntry) :: snowdepth    ! Snow Depth(m)
type(LIS_metadataEntry) :: snowcover    ! Snow cover
type(LIS_metadataEntry) :: surfstor     ! Surface water storage (kg/m2)

type(LIS_metadataEntry) :: soilmoist
type(LIS_metadataEntry) :: soiltemp
type(LIS_metadataEntry) :: sliqfrac   ! fraction of SWE which is in the liquid phase
type(LIS_metadataEntry) :: smliqfrac  ! Average layer fraction of liquid
                                     ! moisture
type(LIS_metadataEntry) :: smfrozfrac ! Average layer fraction of liquid
                                     ! moisture

type(LIS_metadataEntry) :: soilwet      ! Total Soil Wetness (-)

type(LIS_metadataEntry) :: potevap      ! Potential Evapotranspiration (kg/m2s)
type(LIS_metadataEntry) :: ecanop       ! Interception evaporation (kg/m2s)
type(LIS_metadataEntry) :: tveg         ! Vegetation transpiration (kg/m2s)
type(LIS_metadataEntry) :: esoil        ! Bare soil evaporation (kg/m2s)
type(LIS_metadataEntry) :: ewater       ! Open water evaporation (kg/m2s)
type(LIS_metadataEntry) :: rootmoist    ! Root zone soil moisture (kg/m2)
type(LIS_metadataEntry) :: canopint     ! Total canopy water storage (kg/m2s)
type(LIS_metadataEntry) :: evapsnow     ! Snow evaporation (kg/m2s)
type(LIS_metadataEntry) :: subsnow      ! Snow sublimation (kg/m2s)
type(LIS_metadataEntry) :: subsurf      ! Sublimation of the snow free area (kg/m2s)
type(LIS_metadataEntry) :: acond        ! Aerodynamic conductance (m/s)

type(LIS_metadataEntry) :: etpndx       ! Ponded water evaporation (kg/m2s)
type(LIS_metadataEntry) :: infxsrt      ! Infiltration excess (kg/m2)
type(LIS_metadataEntry) :: sfcheadrt    ! Surface overland flow head (kg/m2)
type(LIS_metadataEntry) :: totalprecip  ! Total precipitation rate (kg/m2/s)
type(LIS_metadataEntry) :: rainfconv  ! Convective Rainfall rate (kg/m2/s)

type(LIS_metadataEntry) ::  windforc
type(LIS_metadataEntry) ::  rainfforc
type(LIS_metadataEntry) ::  snowfforc
type(LIS_metadataEntry) ::  tairforc
type(LIS_metadataEntry) ::  qairforc
type(LIS_metadataEntry) ::  psurfforc
type(LIS_metadataEntry) ::  swdownforc
type(LIS_metadataEntry) ::  lwdownforc
type(LIS_metadataEntry) ::  directswforc
type(LIS_metadataEntry) ::  diffuseswforc
type(LIS_metadataEntry) ::  nwindforc
type(LIS_metadataEntry) ::  ewindforc
type(LIS_metadataEntry) ::  fheightforc
type(LIS_metadataEntry) ::  chforc
type(LIS_metadataEntry) ::  cmforc
type(LIS_metadataEntry) ::  emissforc
type(LIS_metadataEntry) ::  mixratioforc
type(LIS_metadataEntry) ::  coszenforc
type(LIS_metadataEntry) ::  albedoforc
```

```
type(LIS_metadataEntry) :: landmask
type(LIS_metadataEntry) :: landcover
type(LIS_metadataEntry) :: soiltype
type(LIS_metadataEntry) :: sandfrac
type(LIS_metadataEntry) :: clayfrac
type(LIS_metadataEntry) :: siltfrac
type(LIS_metadataEntry) :: porosity
type(LIS_metadataEntry) :: soilcolor
type(LIS_metadataEntry) :: elevation
type(LIS_metadataEntry) :: slope
type(LIS_metadataEntry) :: lai
type(LIS_metadataEntry) :: sai
type(LIS_metadataEntry) :: snfralbedo
type(LIS_metadataEntry) :: mxsnalbedo
type(LIS_metadataEntry) :: greenness
type(LIS_metadataEntry) :: tempbot
type(LIS_metadataEntry) :: roottemp

type(LIS_metadataEntry) :: ccond

type(LIS_metadataEntry) :: relsmc
type(LIS_metadataEntry) :: rhmin
```

3.16.1 LIS_histDataInit (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine LIS_histDataInit(n, wout, nch)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod,  only : LIS_verify, LIS_logunit

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

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

n index of the nest

wout flag to indicate if output is turned on or off (1-on, 0-off)

nch size of the tilespace

The routines invoked are:

register_dataEntry (**??**)
initializes the master table of pointers to the data structures and allocates the data structures related to the variable being output

LIS_resetOutputVars (3.16.6)
resets the arrays storing the variable values.

3.16.2 get_moc_attributes (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine get_moc_attributes(modelSpecConfig, dataEntry, standard_name,status)
```

DESCRIPTION:

This routine reads the model output configuration attributes for the given dataEntry output data structure.

3.16.3 allocate_dataEntry (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine allocate_dataEntry(dataEntry,nunits,nch,unittypes)

  implicit none
```

ARGUMENTS:

```
  type(LIS_metadataEntry) :: dataEntry
  integer                 :: nunits
  integer                 :: nch
  character(len=*)        :: unittypes(nunits)
```

DESCRIPTION:

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

3.16.4 LIS_diagnoseOutputVar (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine LIS_diagnoseOutputVar(n, t, index, vlevel, value, unit,&
                                 minValue, maxValue)
  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
real   , intent(in), optional :: minValue, maxValue
```

DESCRIPTION:

This routine maps the LSM specific variable to the appropriate variable in the generic list and also enables time averaging of the variable, if specified.
The arguments are:

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

minvalue if min value should be diagnosed

maxvalue if max value should be diagnosed

3.16.5 diagnoseDataEntry (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine diagnoseDataEntry(dataentry, t, vlevel,value, unit, &
                             minValue, maxValue)
```

USES:

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

ARGUMENTS:

```
type(LIS_metadataEntry) :: dataEntry
integer                 :: t
integer                 :: vlevel
real                    :: value
character(len=*)        :: unit
real, intent(in), optional :: minValue, maxValue
```

DESCRIPTION:

This routine maps a single output variable to the appropriate variable in the generic list.

3.16.6 LIS_resetOutputVars (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine LIS_resetOutputVars(n)
```

ARGUMENTS:

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

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

3.16.7 resetOutputVar (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine resetOutputVar(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.
implicit none

ARGUMENTS:

```
 \begin{description}
 \item[dataEntry] data structure to reset
 \end{description}
type(LIS_metadataEntry), intent(out) :: dataEntry
```

3.16.8 resetOutputVar (Source File: LIS_histDataMod.F90)

INTERFACE:

```
subroutine register_dataEntry(lis_moc_index,dataEntry,nunits,n,nch,&
                              unittypes,form,gribsfc,griblvl)
implicit none
```

ARGUMENTS:

```
integer                 :: lis_moc_index
type(LIS_metadataEntry),target :: dataEntry
integer                 :: n
integer                 :: nunits
integer                 :: nch
character(len=*)        :: unittypes(nunits)
integer                 :: form
integer                 :: gribsfc
integer                 :: griblvl
```

DESCRIPTION:

This routine builds the master table of pointers that maps the MOC enumerated list to their corresponding data structures in the LIS_histData data structure.
The arguments are:

dataEntry data structure to map

lis_moc_index corresponding index value

n nest

nunits number of units associated with this variable

nch number of tiles

unittypes types of units 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

---

3.17 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 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 format
- *rst: restart files. Currently all restart files are written as binary format

This module also manages any data aggregatiion from computing nodes when parallel processing is employed.

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
    13 Jul 2008: Sujay Kumar; Added support for GRIB2
```

USES:

```
 #if ( defined USE_NETCDF )
   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_grib2   ! write a variable into a grib2 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_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
```

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_int
module procedure writevar_restart_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_int
module procedure readvar_restart_real
 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
module procedure writevar_grib1_withstats_real3d
```

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_writevar_grib2 (Source File: LIS_historyMod.F90)

INTERFACE:

```
interface LIS_writevar_grib2
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure writevar_grib2_withstats_real
```

DESCRIPTION:

This interface provides routines for writing variables (real) in a grib2 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.

3.17.11 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_writeModelOutput (Source File: LIS_historyMod.F90)

INTERFACE:

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

USES:

```
    use LIS_coreMod,     only : LIS_masterproc, LIS_rc
    use LIS_histDataMod, only : LIS_resetOutputVars
#if (defined USE_NETCDF)
    use netcdf
#endif
```

ARGUMENTS:

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

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

wout history format option

outInterval history output frequency

nsoillayers Number of soil layers

lyrthk Thickness of soil layers

The routines invoked are:

writeBinaryOutput (3.17.13)
writes the history files in binary format

writeGrib1Output (3.17.15)
writes the history files in GRIB1 format

writeNetcdfOutput (3.17.17)
writes the history files in NETCDF format

writeGrib2Output (3.17.20)
writes the history files in GRIB2 format

LIS_resetOutputVars (3.16.6)
resets the time averaged varibles for the next output.

3.17.13 writeBinaryOutput (Source File: LIS_historyMod.F90)

INTERFACE:

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

USES:

```
use LIS_coreMod,     only : LIS_rc
use LIS_histDataMod, only : LIS_histDataPtr, LIS_MOC_COUNT
```

ARGUMENTS:

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

ftn file unit for the output file

ftn_stats file unit for the output statistics file

The routines invoked are:

writeSingleBinaryVar (3.17.14)
writes a single variable into a flat binary file.

3.17.14 writeSingleBinaryVar (Source File: LIS_historyMod.F90)

INTERFACE:

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

USES:

```
use LIS_coreMod, only     : LIS_rc, LIS_domain
use LIS_logMod,  only     : LIS_logunit, LIS_endrun
use LIS_histDataMod

implicit none
```

ARGUMENTS:

```
integer :: ftn
integer :: ftn_stats
integer :: n
type(LIS_metadataEntry) :: 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 (3.19.11) call to abort program when a fatal error is detected.

3.17.15 writeGrib1Output (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine writeGrib1Output(n,ftn,ftn_stats, outInterval, nsoillayers, lyrthk)
```

USES:

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

ARGUMENTS:

```
implicit none
integer,   intent(in)   :: n
integer,   intent(in)   :: ftn
integer,   intent(in)   :: ftn_stats
integer,   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 (**??**)
writes a single variable into a grib1 formatted file.

3.17.16 writeSingleGrib1Var (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine writeSingleGrib1Var(ftn,ftn_stats,n,time_unit, time_past, &
     nsoillayers,toplev, botlev,var_index,dataEntry)
```

USES:

```
use LIS_coreMod, only     : LIS_rc
use LIS_logMod,  only     : LIS_logunit, LIS_endrun
use LIS_gribMod
use LIS_histDataMod

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                  :: nsoillayers
real, dimension(nsoillayers) :: toplev, botlev
type(LIS_metadataEntry)  :: 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 (3.19.11) call to abort program when a fatal error is detected.

3.17.17 writeNetcdfOutput (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine writeNetcdfOutput(n, ftn, ftn_stats, outInterval, &
     nsoillayers, lyrthk)
```

USES:

```
use LIS_coreMod, only     : LIS_rc, LIS_domain, LIS_masterproc
use LIS_logMod,  only     : LIS_logunit, LIS_endrun, LIS_verify
use LIS_histDataMod
```

ARGUMENTS:

```
integer,   intent(in)   :: n
integer,   intent(in)   :: ftn
integer,   intent(in)   :: ftn_stats
integer,   intent(in)   :: outInterval
integer,   intent(in)   :: nsoillayers
real,      intent(in)   :: lyrthk(nsoillayers)
```

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

The routines invoked are:

defineNETCDFheadervar (3.17.18) writes the required headers for a single variable

writeSingleNETCDFVar (**??**) writes a single variable into a netcdf formatted file.

LIS_verify (3.19.1) call to check if the return value is valid or not.

3.17.18 defineNETCDFheaderVar (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine defineNETCDFheaderVar(ftn,dimID, dataEntry)
```

USES:

```
use LIS_logMod,  only : LIS_logunit,LIS_verify, LIS_endrun
use LIS_histDataMod
```

ARGUMENTS:

```
integer                 :: ftn
type(LIS_metadataEntry) :: dataEntry
integer      :: dimID(3)
```

DESCRIPTION:

This routine writes the required NETCDF header for a single variable
The routines invoked are:

LIS_endrun (3.19.11) call to abort program when a fatal error is detected.

LIS_verify (3.19.1) call to check if the return value is valid or not.

3.17.19 writeSingleNETCDFvar (Source File: LIS_historyMod.F90)

INTERFACE:

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

USES:

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

implicit none

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

DESCRIPTION:

This routine writes a single variable to a NETCDF 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

outInterval history output frequency

nsoillayers Number of soil layers

lyrthk Thickness of soil layers

The routines invoked are:

LIS_writevar_netcdf (3.17.11) writes a variable into a netcdf formatted file.

3.17.20 writeGrib2Output (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine writeGrib2Output(n,ftn,ftn_stats, outInterval, nsoillayers, lyrthk)
```

USES:

```
use LIS_coreMod, only     : LIS_rc
use LIS_logMod,  only     : LIS_logunit, LIS_verify, LIS_endrun
use LIS_histDataMod
use LIS_gribMod
```

ARGUMENTS:

```
integer,   intent(in)   :: n
integer,   intent(in)   :: ftn
integer,   intent(in)   :: ftn_stats
integer,   intent(in)   :: outInterval
integer,   intent(in)   :: nsoillayers
real,      intent(in)   :: lyrthk(nsoillayers)
```

DESCRIPTION:

This routine writes an output file in the GRIB2 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 routines invoked are:

LIS_endrun (3.19.11) call to abort program when a fatal error is detected.

3.17.21 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:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_localPet, LIS_masterproc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: ftn
integer, intent(in) :: ftn_stats
real, intent(in)    :: var(LIS_rc%nch(n))
character (len=*)   :: mvar
integer, intent(in) :: form
```

DESCRIPTION:

Write a real variable to a binary 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)

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:

gather_tiled_vector_output (3.17.48) call to gather the 1d tiled output variables in tile space

gather_gridded_output (3.17.49) call to gather the 1d tiled output variable into a 2d gridded array

gather_gridded_vector_output (3.17.50) call to gather the 1d tiled output variable into a 1d gridded array

write_stats (3.17.47) call to compute the diagnostic statistics

---

3.17.22 writevar_bin_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_writevar_bin
subroutine writevar_bin_real(ftn, n, var)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_localPet, LIS_masterproc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: ftn
real, intent(in)    :: var(LIS_rc%nch(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

3.17.23 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:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, &
     LIS_localPet, LIS_masterproc,LIS_gdeltas,LIS_goffsets,&
     LIS_nss_ind,LIS_ews_ind,LIS_nse_ind,LIS_ewe_ind,&
     LIS_nss_halo_ind,LIS_ews_halo_ind,LIS_nse_halo_ind,LIS_ewe_halo_ind,&
     LIS_npes

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 mode
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

direct dummy argument for direct access output

3.17.24 writevar_restart_int (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_writevar_restart
subroutine writevar_restart_int(ftn, n, var)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, &
     LIS_masterproc, LIS_localPet, LIS_tdeltas,LIS_toffsets,&
     LIS_nss_halo_ind,LIS_nse_halo_ind,LIS_ews_halo_ind,&
     LIS_nss_ind,LIS_nse_ind,LIS_ews_ind,LIS_ewe_ind,&
     LIS_ewe_halo_ind,LIS_npes

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: ftn
integer, intent(in) :: n
integer, intent(in) :: var(LIS_rc%nch(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

3.17.25 writevar_restart_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_writevar_restart
subroutine writevar_restart_real(ftn, n, var)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, &
     LIS_masterproc,LIS_localPet, LIS_toffsets,LIS_tdeltas,&
     LIS_nss_halo_ind,LIS_nse_halo_ind,LIS_ews_halo_ind,&
     LIS_nss_ind,LIS_nse_ind,LIS_ews_ind,LIS_ewe_ind,&
     LIS_ewe_halo_ind,LIS_npes

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: ftn
integer, intent(in) :: n
real, intent(in)    :: var(LIS_rc%nch(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

3.17.26 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:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, &
     LIS_masterproc,LIS_localPet, LIS_toffsets,LIS_tdeltas,&
     LIS_nss_halo_ind,LIS_nse_halo_ind,LIS_ews_halo_ind,&
     LIS_nss_ind,LIS_nse_ind,LIS_ews_ind,LIS_ewe_ind,&
     LIS_ewe_halo_ind,LIS_npes

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: ftn
integer, intent(in) :: n
real, intent(in)    :: var(LIS_rc%nch(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

3.17.27 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:

```
use LIS_coreMod, only : LIS_rc,LIS_domain,                 &
                        LIS_nss_halo_ind,LIS_nse_halo_ind, &
                        LIS_ews_halo_ind,LIS_ewe_halo_ind, &
                        LIS_localPet

implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: ftn
integer, intent(in)   :: n
real, intent(inout)   :: var(LIS_rc%nch(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

3.17.28 writevar_restart_char (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_writevar_restart
subroutine writevar_restart_char(ftn, n, var)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain
use spmdMod,       only : LIS_masterproc,LIS_localPet, LIS_toffsets,LIS_tdeltas,&
     LIS_nss_ind,LIS_nse_ind,LIS_ews_ind,LIS_ewe_ind,LIS_npes

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: ftn
integer, intent(in) :: n
character*1, intent(in)    :: var(LIS_rc%nch(n))
```

DESCRIPTION:

Writes a character 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 readvar_restart_int (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_readvar_restart
subroutine readvar_restart_int(ftn, n, var)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, &
     LIS_nss_halo_ind,LIS_nse_halo_ind,LIS_ews_halo_ind,LIS_ewe_halo_ind,LIS_localPet

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: ftn
integer, intent(in)    :: n
integer, intent(inout) :: var(LIS_rc%nch(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

3.17.30 readvar_restart_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call LIS_readvar_restart
subroutine readvar_restart_real(ftn, n, var)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain,                 &
                        LIS_nss_halo_ind,LIS_nse_halo_ind, &
                        LIS_ews_halo_ind,LIS_ewe_halo_ind, &
                        LIS_localPet

implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: ftn
integer, intent(in)   :: n
real, intent(inout)   :: var(LIS_rc%nch(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

3.17.31 readvar_restart_char (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_readvar_restart
subroutine readvar_restart_char(ftn, n, var)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain
use spmdMod,       only : LIS_nss_ind,LIS_nse_ind,LIS_ews_ind,LIS_ewe_ind,LIS_localPet

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: ftn
integer, intent(in)    :: n
character*1, intent(inout) :: var(LIS_rc%nch(n))
```

DESCRIPTION:

Reads an character 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

3.17.32 readvar_1dgridded_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call LIS_readvar_gridded
subroutine readvar_1dgridded_real(ftn, n, var)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, &
     LIS_localPet, LIS_nss_halo_ind,LIS_nse_halo_ind,&
     LIS_ews_halo_ind,LIS_ewe_halo_ind

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

3.17.33 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:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, &
     LIS_localPet,LIS_nss_halo_ind,LIS_nse_halo_ind,&
     LIS_ews_halo_ind,LIS_ewe_halo_ind

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.

3.17.34 readvar_2dgridded_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call LIS_readvar_gridded
subroutine readvar_2dgridded_real(ftn, n, var)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, &
     LIS_localPet, LIS_nss_halo_ind,LIS_nse_halo_ind,&
     LIS_ews_halo_ind,LIS_ewe_halo_ind

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

3.17.35 writevar_gridded_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call LIS_writevar_gridded
subroutine writevar_gridded_real(ftn, n, var)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, &
     LIS_localPet, LIS_masterproc, LIS_npes, &
     LIS_nss_ind,LIS_nse_ind,LIS_ews_ind,LIS_ewe_ind, &
     LIS_nss_halo_ind,LIS_nse_halo_ind,LIS_ews_halo_ind,LIS_ewe_halo_ind, &
     LIS_gdeltas, LIS_goffsets

implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: ftn
integer, intent(in)   :: n
real, intent(inout)   :: var(LIS_rc%ngrid(n))
```

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

3.17.36 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:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, &
     LIS_localPet, LIS_masterproc, LIS_npes, &
     LIS_nss_ind,LIS_nse_ind,LIS_ews_ind,LIS_ewe_ind, &
     LIS_nss_halo_ind,LIS_nse_halo_ind,LIS_ews_halo_ind,LIS_ewe_halo_ind, &
     LIS_gdeltas, LIS_goffsets

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

3.17.37 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:

```
use LIS_coreMod, only : LIS_rc,LIS_domain,LIS_masterproc,LIS_npes, &
     LIS_localPet, LIS_nss_ind,LIS_nse_ind,LIS_ews_ind,LIS_ewe_ind, &
     LIS_nss_halo_ind,LIS_nse_halo_ind,LIS_ews_halo_ind,LIS_ewe_halo_ind, &
     LIS_goffsets, LIS_gdeltas, LIS_tdeltas, LIS_toffsets

use LIS_logMod,  only : LIS_endrun, LIS_flush

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: ftn
integer, intent(in) :: ftn_stats
integer             :: varid
real, intent(in)    :: var(LIS_rc%nch(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 (3.17.45)
call to compute the diagnostic statistics

3.17.38 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, form,     &
                                          toplev, botlev, kdim)
```

USES:

```
     use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_localPet, LIS_masterproc
     use LIS_gribMod
 use LIS_logMod

     implicit none
```

ARGUMENTS:

```
     integer, intent(in)         :: ftn
     integer, intent(in)         :: ftn_stats
     integer, intent(in)         :: n
     real, intent(in)            :: var(LIS_rc%nch(n))
     character(len=*),intent(in) :: mvar
     integer, intent(in)         :: form

     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_gribdat (3.14.7) calls the low-level grib routines contained in the grib library

gather_gridded_output (3.17.49) call to gather the 1d tiled output variable into a 2d gridded array

write_stats (3.17.47) call to compute diagnostic statistics of a variable

---

3.17.39 writevar_grib2_withstats_real (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_writevar_grib2
subroutine writevar_grib2_withstats_real(ftn,ftn_stats, n, &
     var, mvar, form, cgrib, lcgrib, ipdsnum, ipdstmplen, ipdstmpl, &
     coordlist, numcoord, idrsnum, idrstmplen, idrstmpl, &
     ibmap, bmap, ierr)

 subroutine writevar_grib2_withstats_real(ftn, ftn_stats, n, &
                                          var, mvar, form,    &
                                          cgrib,lcgrib, ipdsnum, &
                                          ipdstmplen, ipdstmpl, &
                                          coordlist, numcoord, idrsnum, &
                                          idrstmplen, idrstmpl,&
                                          ibmap, bmap, ierr)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, &
     LIS_npes, LIS_localPet, LIS_masterproc, LIS_goffsets, LIS_gdeltas,&
     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_gribMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)         :: ftn
integer, intent(in)         :: ftn_stats
integer, intent(in)         :: n
real, intent(in)            :: var(LIS_rc%nch(n))
character(len=*),intent(in) :: mvar
integer, intent(in)         :: form
integer                     :: lcgrib
character (lcgrib)          :: cgrib
integer                     :: ipdsnum
integer                     :: ipdstmplen
integer                     :: ipdstmpl(ipdstmplen)
```

```
integer                    :: coordlist(1)
integer                    :: numcoord
integer                    :: idrsnum
integer                    :: idrstmplen
integer                    :: idrstmpl(idrstmplen)
integer                    :: ibmap
integer                    :: bmap(1)
integer                    :: ierr
```

DESCRIPTION:

Write a real variable to a grib2 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.

size_3 Size of the sect3 array

The routines invoked are:

stats (3.17.45)
call to compute diagnostic statistics of a variable

3.17.40 writevar_grib1_withstats_real3d (Source File: LIS_historyMod.F90)

INTERFACE:

```
Private name: call using LIS_writevar_grib1
subroutine writevar_grib1_withstats_real3d(ftn, ftn_stats, n,                &
                                           var, mvar, form,            &
                                           toplev, botlev, kdim)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, &
     LIS_localPet, LIS_masterproc, LIS_gdeltas, LIS_goffsets,&
     LIS_ews_ind,LIS_ewe_ind,LIS_nss_ind,LIS_nse_ind,LIS_npes, &
     LIS_ews_halo_ind,LIS_ewe_halo_ind,LIS_nss_halo_ind,LIS_nse_halo_ind
use LIS_gribMod

implicit none
```

ARGUMENTS:

```
integer                     :: kdim

integer, intent(in)         :: ftn
integer, intent(in)         :: ftn_stats
integer, intent(in)         :: n
real, intent(in)            :: var(LIS_rc%nch(n),kdim)
character(len=*),intent(in) :: mvar
integer, intent(in)         :: form
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_gribdat (3.14.7)
calls the low-level grib routines contained in the grib library

stats (3.17.45)
call to compute diagnostic statistics of a variable

3.17.41 LIS_tile2grid (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine LIS_tile2grid(n,gvar,tvar)
```

USES:

```
use LIS_coreMod,      only : LIS_rc,LIS_domain

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
real              :: gvar(LIS_rc%lnc(n),LIS_rc%lnr(n))
real, intent(in)  :: tvar(LIS_rc%nch(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

3.17.42 LIS_write_innov (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine LIS_write_innov(ftn, n, k, var)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, &
     LIS_localPet, LIS_masterproc, LIS_odeltas, LIS_ooffsets, &
     LIS_ews_ind,LIS_ewe_ind,LIS_nss_ind,LIS_nse_ind,LIS_npes, &
     LIS_nss_halo_ind, LIS_nse_halo_ind, LIS_ews_halo_ind, &
     LIS_ewe_halo_ind

implicit none
```

ARGUMENTS:

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

---

3.17.43 LIS_write_dastats (Source File: LIS_historyMod.F90)

INTERFACE:

```
subroutine LIS_write_dastats(ftn,n,var)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, &
     LIS_localPet, LIS_masterproc, LIS_tdeltas, LIS_toffsets,&
     LIS_ews_ind,LIS_ewe_ind,LIS_nss_ind,LIS_nse_ind,LIS_npes, &
     LIS_ews_halo_ind, LIS_ewe_halo_ind, LIS_nss_halo_ind, &
     LIS_nse_halo_ind

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: ftn
real, intent(in)    :: var(LIS_rc%nch(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 (3.17.46)
call to compute the diagnostic statistics

3.17.44 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

3.17.45 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,nch,mean,stdev,min,max)
```

ARGUMENTS:

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

3.17.46 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,nch,mean,spread, stdev,min,max)
```

ARGUMENTS:

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

3.17.47 write_stats (Source File: LIS_historyMod.F90)

Top level call to write statistics to the STATS file.

REVISION HISTORY:

```
29 Oct 2008:  James Geiger; Initial code
```

INTERFACE:

```
subroutine write_stats(var, size, mvar, ftn_stats, form)
```

USES:

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

ARGUMENTS:

```
\begin{description}
 \item [var]
   output data to process
 \item [size]
   size of array var
 \item[mvar]
   name of the variable being written
 \item [ftn\_stats]
   unit number of the ASCII text statistics file
 \item[form]
   format to be used in the stats file (1-decimal format,
   2-scientific format)
\end{description}
```

3.17.48 gather_tiled_vector_output (Source File: LIS_historyMod.F90)

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.

REVISION HISTORY:

```
30 Jan 2009:  Sujay Kumar; Initial code
```

INTERFACE:

```
subroutine gather_tiled_vector_output(n, gtmp, var)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_localPet, LIS_masterproc, &
                        LIS_gdeltas, LIS_goffsets,&
                        LIS_nss_ind,LIS_ews_ind,LIS_nse_ind,&
                        LIS_ewe_ind, LIS_npes, &
                        LIS_nss_halo_ind,LIS_ews_halo_ind,&
                        LIS_nse_halo_ind,LIS_ewe_halo_ind,&
                        LIS_tdeltas, LIS_toffsets

implicit none
```

ARGUMENTS:

```
\begin{description}
 \item [n]
   index of the current nest
 \item [gtmp]
   return array for the tiled output data
 \item [var]
   output data to process
\end{description}
```

3.17.49 gather_gridded_output (Source File: LIS_historyMod.F90)

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.

REVISION HISTORY:

```
29 Oct 2008:  James Geiger; Initial code
```

INTERFACE:

```
subroutine gather_gridded_output(n, gtmp, var)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_localPet, LIS_masterproc, &
                        LIS_gdeltas, LIS_goffsets,&
                        LIS_nss_ind,LIS_ews_ind,LIS_nse_ind,&
                        LIS_ewe_ind, LIS_npes, &
                        LIS_nss_halo_ind,LIS_ews_halo_ind,&
                        LIS_nse_halo_ind,LIS_ewe_halo_ind,&
                        LIS_tdeltas, LIS_toffsets

implicit none
```

ARGUMENTS:

```
\begin{description}
 \item [n]
   index of the current nest
 \item [gtmp]
   return array for the gridded output data
 \item [var]
   output data to process
\end{description}
```

3.17.50 gather_gridded_vector_output (Source File: LIS_historyMod.F90)

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.

REVISION HISTORY:

```
30 Jan 2009: Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine gather_gridded_vector_output(n, gtmp, var)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_localPet, LIS_masterproc, &
                        LIS_gdeltas, LIS_goffsets,&
                        LIS_nss_ind,LIS_ews_ind,LIS_nse_ind,&
                        LIS_ewe_ind, LIS_npes, &
                        LIS_nss_halo_ind,LIS_ews_halo_ind,&
                        LIS_nse_halo_ind,LIS_ewe_halo_ind,&
                        LIS_tdeltas, LIS_toffsets

implicit none
```

ARGUMENTS:

```
\begin{description}
 \item [n]
   index of the current nest
 \item [gtmp]
   return array for the gridded output data
 \item [var]
   output data to process
\end{description}
```

3.18 Fortran: Module Interface LIS_laiMod (Source File: LIS_laiMod.F90)

The code in this file provides interfaces to manage different sources of Leaf Area Index (LAI) data.

3.18.1 Overview

This routines in this module provides routines to read the LAI climatology data. The climatological data is temporally interpolated between months to the current simulation date.
Note that currently the module assumes the frequency of LAI climatology to be monthly.

REVISION HISTORY:

```
08 Aug 2005: Sujay Kumar; Initial implementation
```

USES:

```
use ESMF_Mod
use LIS_precisionMod

implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
 ------------------------------------------------------------------------------
  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 TYPES:

```
 ------------------------------------------------------------------------------
  public :: LIS_lai ! data structure containing LAI data.
```

3.18.2 LIS_lai_setup (Source File: LIS_laiMod.F90)

INTERFACE:

```
subroutine LIS_lai_setup
```

USES:

```
  use LIS_coreMod,    only : LIS_rc, LIS_Config
  use LIS_timeMgrMod, only : LIS_setAlarm, LIS_computeTemporalWeights, &
       LIS_Clock
  use LIS_logMod,     only : LIS_verify
```

DESCRIPTION:

Allocates memory and other structures for reading LAI datasets
The routines invoked are:

LIS_setAlarm (3.31.13)
sets the monthly alarm to read the LAI climatology

3.18.3 LIS_read_lai (Source File: LIS_laiMod.F90)

INTERFACE:

```
subroutine LIS_read_lai(n)
```

USES:

```
  use LIS_coreMod,    only : LIS_rc, LIS_domain
  use LIS_timeMgrMod, only : LIS_computeTemporalWeights

  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 (3.40.2)
invokes the generic method in the registry to read the LAI climatology data

---

3.18.4 LIS_lai_finalize (Source File: LIS_laiMod.F90)

INTERFACE:

```
subroutine LIS_lai_finalize
```

USES:

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

DESCRIPTION:

Deallocates objects created in this module

3.18.5 LIS_diagnoseLAI (Source File: LIS_laiMod.F90)

INTERFACE:

```
subroutine LIS_diagnoseLAI(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_histDataMod, only : LIS_diagnoseOutputVar, 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 (3.16.4)
This routine maps a variable to the history writing routines

3.18.6 LIS_lai_reset (Source File: LIS_laiMod.F90)

INTERFACE:

```
subroutine LIS_lai_reset
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_Config
use LIS_timeMgrMod, only : LIS_setAlarm, LIS_computeTemporalWeights, &
     LIS_Clock
use LIS_logMod,     only : LIS_verify
```

DESCRIPTION:

Resets data structures for reading LAI datasets
The routines invoked are:

LIS_setAlarm (3.31.13)
sets the monthly alarm to read the LAI climatology

3.18.7 LIS_lapseRateCorrection (Source File: LIS_lapseRateCorrection.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)
```

USES:

```
use ESMF_Mod
use LIS_logMod,         only : LIS_verify
use LIS_constantsMod,   only : LIS_CONST_G, LIS_CONST_TKFRZ
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_baseforcingMod, only : LIS_forc, LIS_FORC_State
use LIS_FORC_AttributesMod

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: nest
real,    intent(in) :: modelelev(LIS_rc%nch(nest))
```

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.

3.19 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 TYPES:

```
-----------------------------------------------------------------------------
 public :: LIS_logunit ! file unit number used for diagnostic logging
```

3.19.1 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.

3.19.2 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.

3.19.3 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

3.19.4 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

3.19.5 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 )
```

3.19.6 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

3.19.7 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.

3.19.8 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.

3.19.9 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

---

3.19.10 LIS_releaseUnitNumber (Source File: LIS_logMod.F90)

INTERFACE:

```
subroutine LIS_releaseUnitNumber(iunit)
```

DESCRIPTION:

close and release Fortran unit no longer in use

3.19.11 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 (**??**)
flushes the output streams

3.20 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

3.20.1 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_Mod

implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: LIS_lsm_init       ! initialize lsm variables, memory
public :: LIS_setuplsm       ! set land surface parameters
public :: LIS_readrestart    ! read the restart file
public :: LIS_force2tile     ! transfer forcing to model tiles
public :: LIS_lsm_main       ! execute the land model
public :: LIS_lsm_output     ! write model output
public :: LIS_setLSMDynParams    ! set the time dependent parameters
public :: LIS_writerestart   ! write the restart file
public :: LIS_lsm_setexport  ! set the variables to be exported to the
                             ! atmospheric component
public :: LIS_perturb_states ! perturbs the prognostic variables
public :: LIS_lsm_finalize   ! cleanup allocated structures
```

PUBLIC TYPES:

```
public :: LIS_LSM_State
public :: LIS_LSM_Pert_State
public :: LIS_LSM_Incr_State
```

3.20.2 LIS_lsm_setexport (Source File: LIS_lsmMod.F90)

INTERFACE:

```
interface LIS_lsm_setexport
```

PRIVATE MEMBER FUNCTIONS:

```
    module procedure lsm_setexport_esmf
    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.

3.20.3 LIS_lsm_init (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_lsm_init()
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_vecTile
use LIS_lsm_pluginMod,    only : LIS_lsm_plugin
use LIS_lsmcpl_pluginMod, only : LIS_lsmcpl_plugin
use LIS_lsmda_pluginMod,  only : LIS_lsmda_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 LSM specific variables and datastructures and specification of any runtime specific options.
The calling sequence is:

LIS_lsm_plugin (4.9.1)
sets up function table registries for implemented land surface models

LIS_lsmcpl_plugin (4.10.1)
sets up function table registries for implemented land surface models used in coupled modes

LIS_lsmda_plugin (4.11.1)
sets up function table registries for land surface models used in data assimilation

lsmini (3.42.2)
invokes the generic method in the registry to initialize the land surface model

perturbinit (3.44.2)
invokes the generic method in the registry to perturb the prognostic variables using the specified algorithm.

3.20.4 LIS_setuplsm (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine  LIS_setuplsm
```

USES:

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

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 (3.42.8)
invokes the generic method in the registry to set up the land surface model

---

3.20.5 LIS_lsm_main (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_lsm_main(n)
```

USES:

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

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 (3.42.4)
invokes the generic method in the registry to run the land surface model

---

3.20.6 LIS_perturb_states (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_perturb_states(n)
```

USES:

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

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 (3.44.6)
invokes the abstract method to invoke the perturbation algorithm to perturb LSM prognostic variables

getLSMstateVar (3.42.18)
obtains the list of prognostic variables

applyLSMPert (3.20.15)
applies the specified perturbations to the LSM state

qcLSMState (3.42.32)
performs the QC of the perturbed LSM state

setLSMstateVar (3.42.20)
assigns the prognostic variables back to the model states

3.20.7 LIS_readrestart (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_readrestart
```

USES:

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

DESCRIPTION:

This interface provides the entry point to read LSM specific model restart files.
The calling sequence is:

lsmrestart (3.42.10)
invokes the generic method in the registry to read restart files for the land surface model

---

3.20.8 LIS_lsm_output (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_lsm_output(n)
```

USES:

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

ARGUMENTS:

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

DESCRIPTION:

This interface provides the entry point to write LSM specific model output files.
The arguments are:

n index of the nest or domain

The calling sequence is:

lsmoutput (3.42.12)
invokes the generic method in the registry to write the land surface model output

---

3.20.9 LIS_setLSMDynparams (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_setLSMDynparams(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 LSM.
The arguments are:

n index of the nest or domain

The calling sequence is:

lsmdynsetup (3.42.30)
invokes the generic method in the registry to set up time dependent parameters for the land surface model

---

3.20.10 LIS_force2tile (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_force2tile(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:

lsmf2t (3.42.14)
invokes the generic method in the registry to transfer the forcing to the land surface model tiles

---

3.20.11 LIS_writerestart (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_writerestart(n)
```

USES:

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

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 (3.42.16)
invokes the generic method in the registry to write restart files for the land surface model

---

3.20.12 lsm_setexport_esmf (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine lsm_setexport_esmf(n, expState)
```

USES:

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

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: expState
```

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:

lsmsetexport (3.42.28)
invokes the generic method in the registry to set the export state from the land surface model

---

3.20.13 lsm_setexport_noesmf (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine lsm_setexport_noesmf(n)
```

USES:

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

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:

lsmsetexport (3.42.28)
invokes the generic method in the registry to set the export state from the land surface model

---

3.20.14 LIS_lsm_finalize (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine LIS_lsm_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:

lsmfinalize (3.42.6)
invokes the generic method in the registry to cleanup the LSM related datastructures

---

3.20.15 applyLSMPert (Source File: LIS_lsmMod.F90)

INTERFACE:

```
subroutine applyLSMPert(n, k, LIS_LSM_State, LIS_LSM_Pert_State)
```

USES:

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

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

---

3.20.16 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

INTERFACE:

```
subroutine LIS_mapSoilType(class,nc,nr,sand,clay,silt,soiltyp)
```

USES:

```
use LIS_logMod, only : LIS_logunit

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: class
integer, intent(in)    :: nc,nr
real, intent(in)       :: sand(nc,nr),clay(nc,nr),silt(nc,nr)
integer, intent(inout) :: soiltyp(nc,nr)
```

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)

nc dimension of the grid along the east-west direction

nr dimension of the grid along the north-south direction

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

3.20.17 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

3.21 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

3.22 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

contains
```

```
%//////////////////////////////////////////////////////////

\mbox{}\hrulefill\


   \subsubsection{LIS\_gasdev (Source File: LIS\_numerRecipesMod.F90)}


   \label{LIS_gasdev}


   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

\bigskip{\sf REVISION HISTORY:}
\begin{verbatim}    27 Feb 2005: Sujay Kumar : Specification in LIS.
```

INTERFACE:

```
     FUNCTION LIS_gasdev(idum)
```

3.22.1 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)
```

3.23 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

3.23.1 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
```

3.23.2 LIS_optUE_init (Source File: LIS_optUEMod.F90)

INTERFACE:

```
subroutine LIS_optUE_init
```

USES:

```
use ESMF_Mod
use LIS_coreMod,  only : LIS_rc, LIS_config, LIS_vecTile
use LIS_logMod,   only : LIS_verify
```

DESCRIPTION:

This routine intializes the optimization/uncertainty estimation algorithm, the objective function and decision space objects.
The methods invoked are:

optueinit (**??**)
invokes the initialization routine for the specified optimization or uncertainty estimation algorithm

objectivefuncinit (**??**)
invokes the initialization routine for the specified objection function metric/method

---

3.23.3 LIS_optUE_setup (Source File: LIS_optUEMod.F90)

INTERFACE:

```
subroutine LIS_optUE_setup
```

USES:

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

DESCRIPTION:

This routine intializes the optimization/uncertainty estimation algorithm, the objective function and decision space objects.
The methods invoked are:

optueinit (**??**)
invokes the initialization routine for the specified optimization or uncertainty estimation algorithm

objectivefuncinit (**??**)
invokes the initialization routine for the specified objection function metric/method

---

3.23.4 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 (3.43.10)
invokes the method from the registry that specifies the stopping criteria of the algorithm

---

3.23.5 LIS_runoptUE (Source File: LIS_optUEMod.F90)

INTERFACE:

```
subroutine LIS_runOptUE()
```

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 (3.43.12)
invokes the specific optue algorithm related operations to solve for potential solutions

resetobjectivefunctype (3.43.26)
invokes the call to reset the objective function objects based on the specific method

---

3.24 Fortran: Module Interface LIS_paramsMod (Source File: LIS_paramsMod.F90)

The code in this file provides interfaces to manage different sources of parameter datasets.

3.24.1 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_gfracMod
use LIS_albedoMod
use LIS_soilsMod
use LIS_laiMod
use LIS_saiMod

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:

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

3.24.2 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_soils_setup (3.29.2)
call to set up the soil parameters options

LIS_read_soils (3.29.3)
call to read the chosen soil parameters

LIS_albedo_setup (3.6.2)
call to setup albedo parameter options

LIS_greenness_setup (3.13.2)
call to setup greenness parameter options

LIS_lai_setup (3.18.2)
call to setup LAI parameter options

LIS_sai_setup (3.27.2)
call to setup SAI parameter options

3.24.3 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 (3.13.3)
call to read the greenness data source

LIS_read_albedo (3.6.4)
call to read the albedo data source

LIS_read_lai (3.18.3)
call to read the LAI data source

LIS_read_sai (3.27.3)
call to read the SAI data source

3.24.4 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 (**??**)
maps the landcover data to the LIS history writer

LIS_diagnosetopography (**??**)
maps the topography data to the LIS history writer

LIS_diagnosesoils (3.29.5)
maps the soils data to the LIS history writer

LIS_diagnosegfrac (3.13.5)
maps the greenness data to the LIS history writer

LIS_diagnoseLAI (3.18.5)
maps the LAI data to the LIS history writer

LIS_diagnoseSAI (3.27.5)
maps the SAI data to the LIS history writer

LIS_diagnosealbedo (3.6.6)
maps the albedo data to the LIS history writer

3.24.5 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 (3.6.5)
call to cleanup albedo related structures

LIS_greenness_finalize (3.13.4)
call to cleanup greenness related structures

LIS_lai_finalize (3.18.4)
call to LAI related structures

LIS_sai_finalize (3.27.4)
call to SAI related structures

3.24.6 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 (3.13.6)
resets the greenness data structures

LIS_albedo_reset (3.6.7)
resets the albedo data structures

LIS_lai_reset (3.18.6)
resets the lai data structures

LIS_sai_reset (3.27.6)
resets the sai data structures

3.25 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
```

3.25.1 LIS_perturb_init (Source File: LIS_perturbMod.F90)

INTERFACE:

```
subroutine LIS_perturb_init
```

DESCRIPTION:

The routines invoked are:

LIS_perturb_plugin (4.17.1)
sets up the function table registries for implemented perturbation algorithms

USES:

```
use LIS_coreMod,            only : LIS_rc
use LIS_timeMgrMod,         only : LIS_calendar, LIS_clock
use LIS_logMod,             only : LIS_verify
use LIS_perturb_pluginMod
```

3.25.2 LIS_perturb_readrestart (Source File: LIS_perturbMod.F90)

INTERFACE:

```
subroutine LIS_perturb_readrestart
```

USES:

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

ARGUMENTS:

DESCRIPTION:

3.25.3 LIS_perturb_writerestart (Source File: LIS_perturbMod.F90)

INTERFACE:

```
subroutine LIS_perturb_writerestart(n)
```

USES:

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

ARGUMENTS:

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

DESCRIPTION:

3.26 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.

3.26.1 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. Currently, it only supports "–version" and "-V" options, both of which cause LIS to dump information about itself and then end.

3.26.2 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
```

INTERFACE:

```
subroutine LIS_readConfig()
```

USES:

```
use ESMF_Mod
use LIS_coreMod,    only : LIS_rc, LIS_config, &
     LIS_localPet, LIS_npes, LIS_masterproc
use LIS_histDataMod, only : LIS_histData, LIS_histDataPtr, LIS_MOC_COUNT
use LIS_timeMgrMod, only : LIS_date2time
use LIS_logMod,  only : LIS_logunit, LIS_verify, LIS_endrun
use LIS_param_pluginMod, only : LIS_soils_plugin, LIS_laisai_plugin,&
     LIS_tbot_plugin, LIS_alb_plugin, LIS_gfrac_plugin, &
     LIS_landcover_plugin,LIS_topo_plugin
use LIS_runmode_pluginMod, only : LIS_runmode_plugin

use LIS_optUEAlgorithm_pluginMod, only : LIS_optUEAlgorithm_plugin
use LIS_optUEType_pluginMod,      only : LIS_optUEType_plugin
use LIS_lsmopt_pluginMod,     only : LIS_lsmopt_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_lsmrtm_pluginMod,     only : LIS_lsmrtm_plugin
```

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, and calls some of the registries that set up the component plugin definitions.
The routines invoked are:

LIS_date2time (3.31.22)
convert the date format to a floating point time format

LIS_runmode_plugin (4.19.1)
sets up function table registries for implemented runmodes

LIS_soils_plugin (4.16.2)
sets up function table registries for implemented soil data sources

LIS_laisai_plugin (4.16.3)
sets up function table registries for implemented LAI/SAI data sources

LIS_tbot_plugin (4.16.5)
sets up function table registries for implemented bottom temperature data sources

LIS_alb_plugin (4.16.7)
sets up function table registries for implemented albedo data sources

LIS_gfrac_plugin (4.16.6)
sets up function table registries for implemented greenness fraction data sources

LIS_landcover_plugin (4.16.4)
sets up function table registries for implemented landcover data sources

LIS_topo_plugin (4.16.1)
sets up function table registries for implemented topography data sources

LIS_optimization_plugin (**??**)
sets up function table registries for implemented parameter estimation algorithms

LIS_lsmopt_plugin (4.12.1)
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

get_moc_attributes (3.16.2)
subroutine reads the specification for a single variable in the model output attributes file.

3.26.3 LIS_readObsAttributes (Source File: LIS_readObsAttributes.F90)

REVISION HISTORY:

```
14 Oct 2006: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine LIS_readObsAttributes(k, vname,&
    ssdev,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*40       :: vname(LIS_rc%nobtypes(k))
real               :: ssdev(LIS_rc%nobtypes(k))
real               :: varmin(LIS_rc%nobtypes(k))
real               :: varmax(LIS_rc%nobtypes(k))
integer            :: ftn
```

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

3.26.4 LIS_readObsPertAttributes (Source File: LIS_readObsPertAttributes.F90)

REVISION HISTORY:

```
14 Oct 2006: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine LIS_readObsPertAttributes(k, vname,&
     perttype,ssdev,stdmax,zeromean,tcorr,&
     xcorr, ycorr, ccorr)
```

USES:

```
use LIS_coreMod,   only : LIS_rc
use LIS_logMod,    only : LIS_logunit, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber

implicit none
```

ARGUMENTS:

```
integer            :: k
character*40       :: vname(LIS_rc%nobtypes(k))
integer            :: perttype(LIS_rc%nobtypes(k))
real               :: ssdev(LIS_rc%nobtypes(k))
real               :: stdmax(LIS_rc%nobtypes(k))
integer            :: zeromean(LIS_rc%nobtypes(k))
real               :: tcorr(LIS_rc%nobtypes(k))
real               :: xcorr(LIS_rc%nobtypes(k))
real               :: ycorr(LIS_rc%nobtypes(k))
real               :: ccorr(LIS_rc%nobtypes(k),LIS_rc%nobtypes(k))
```

DESCRIPTION:

This routine reads the observation perturbation attributes for each observation type. The format of the perturbation attributes file is:
¡Variable name¿ ¡perturbation type¿ ¡standard deviation¿ ¡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

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

xcorr horizontal x-correlation scale

ycorr horizontal y-correlation scale

ccorr cross correlations

3.27 Fortran: Module Interface LIS_saiMod (Source File: LIS_saiMod.F90)

The code in this file provides interfaces to manage different sources of Leaf Area Index (SAI) data.

3.27.1 Overview

This routines in this module provides routines to read the SAI climatology data. The climatological data is temporally interpolated between months to the current simulation date.
Note that currently the module assumes the frequency of SAI climatology to be monthly.

REVISION HISTORY:

```
08 Aug 2005: Sujay Kumar; Initial implementation
```

USES:

```
use ESMF_Mod
use LIS_precisionMod
implicit none

PRIVATE

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

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
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_sai ! data structure containing SAI data.
```

3.27.2 LIS_sai_setup (Source File: LIS_saiMod.F90)

INTERFACE:

```
subroutine LIS_sai_setup
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_setAlarm, LIS_computeTemporalWeights, &
     LIS_Clock
use LIS_logMod,     only : LIS_verify
```

DESCRIPTION:

Allocates memory and other structures for reading SAI datasets
The routines invoked are:

LIS_setAlarm (3.31.13)
sets the monthly alarm to read the SAI climatology

3.27.3 LIS_read_sai (Source File: LIS_saiMod.F90)

INTERFACE:

```
subroutine LIS_read_sai(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_timeMgrMod, only : LIS_computeTemporalWeights

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

---

3.27.4 LIS_sai_finalize (Source File: LIS_saiMod.F90)

INTERFACE:

```
subroutine LIS_sai_finalize
```

USES:

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

DESCRIPTION:

Deallocates objects created in this module

3.27.5 LIS_diagnoseSAI (Source File: LIS_saiMod.F90)

INTERFACE:

```
subroutine LIS_diagnoseSAI(n)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_histDataMod, only : LIS_diagnoseOutputVar, 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 (3.16.4)
This routine maps a variable to the history writing routines

3.27.6 LIS_sai_reset (Source File: LIS_saiMod.F90)

INTERFACE:

```
subroutine LIS_sai_reset
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_setAlarm, LIS_computeTemporalWeights, &
     LIS_Clock
use LIS_logMod,     only : LIS_verify
```

DESCRIPTION:

resets the structures that hold SAI datasets
The routines invoked are:

LIS_setAlarm (3.31.13)
sets the monthly alarm to read the SAI climatology

3.27.7 LIS_slopeAspectCorrection (Source File: LIS_slopeAspectCorrection.F90)

REVISION HISTORY:

```
8 Mar 2008: Sujay Kumar; Adopted in LIS
```

INTERFACE:

```
subroutine LIS_slopeAspectCorrection(nest)
```

USES:

```
use ESMF_Mod
use LIS_logMod,         only : LIS_verify
use LIS_constantsMod,   only : LIS_CONST_PI, LIS_CONST_ANGULAR_VELOCITY
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_baseforcingMod, only : LIS_forc, LIS_FORC_State
use LIS_FORC_AttributesMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Adapted from Dingman, S.L., Physical Hydrology - 2nd ed.- Appendix E, Prentice Hall, 2002,
The arguments are:

nest index of the domain or nest.

3.28 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
```

3.28.1 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_Mod
use LIS_coreMod,    only : LIS_rc
```

3.28.2 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
```

3.29 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.

3.29.1 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
implicit none

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

PUBLIC MEMBER FUNCTIONS:

```
 ------------------------------------------------------------------------------
  public :: LIS_soils_setup  ! initializes data structures and memory
  public :: LIS_read_soils   ! read soil parameters
  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
```

3.29.2 LIS_soils_setup (Source File: LIS_soilsMod.F90)

INTERFACE:

```
subroutine LIS_soils_setup()
```

USES:

```
  use ESMF_Mod
  use LIS_coreMod,  only : LIS_rc, LIS_config
  use LIS_fileIOMod, only : LIS_readDomainConfigSpecs
```

DESCRIPTION:

Allocates memory for data structures for reading soils datasets

---

3.29.3 LIS_read_soils (Source File: LIS_soilsMod.F90)

INTERFACE:

```
subroutine LIS_read_soils()
```

USES:

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

DESCRIPTION:

Reads the soils data based on the choice of options specified in the lis configuration.
The routines invoked are:

readsoiltexture (3.46.8)
invokes the generic method in the registry to read the soil texture data

readsand (3.46.2)
invokes the generic method in the registry to read the sand fraction data

readclay (3.46.4)
invokes the generic method in the registry to read the clay fraction data

readsilt (3.46.6)
invokes the generic method in the registry to read the silt fraction data

readcolor (3.46.20)
invokes the generic method in the registry to read the soil color data

readporosity (3.46.10)
invokes the generic method in the registry to read the porosity data

readpsisat (3.46.12)
invokes the generic method in the registry to read the saturated matric potential data

readksat (3.46.14)
invokes the generic method in the registry to read the saturated hydraulic conductivity data

readbexp (3.46.16)
invokes the generic method in the registry to read the b parameter data

readquartz (3.46.18)
invokes the generic method in the registry to read the quartz data

---

3.29.4 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.

3.29.5 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_diagnoseOutputVar, LIS_MOC_SOILTYPE, &
     LIS_MOC_SOILCOLOR, LIS_MOC_SANDFRAC, LIS_MOC_CLAYFRAC, LIS_MOC_SILTFRAC, &
     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 (3.16.4)
This routine maps a variable to the history writing routines

3.30 Fortran: Module Interface LIS_suppforcingMod (Source File: LIS_suppforcingMod.F90

The code in this file provides interfaces to manage supplemental forcing analyses

3.30.1 Overview

The module provides interfaces for incorporating supplemental forcing analyses into LIS. Supplemental forcing is defined as any analysis that includes a subset of the meteorological forcing variables defined in `baseforcing_module`. Examples include the incorporating precipitation analyses from various sources such as CMAP, CMORPH, etc. If a supplemental forcing is provided, the correspoinding variables set by the baseforcing is overwritten.

REVISION HISTORY:

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

USES:

```
implicit none

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

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: LIS_suppforcing_init ! initializes supplemental forcing
public :: LIS_get_supp_forcing ! retrieve supplemental forcing data
                               ! and interpolate spatially and temporally
public :: LIS_suppforcing_finalize !cleanup allocated structures

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

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: LIS_suppforc   ! LIS object to store supplemental forcing data
```

3.30.2 LIS_suppforcing_init (Source File: LIS_suppforcingMod.F90)

INTERFACE:

```
subroutine LIS_suppforcing_init
```

USES:

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

DESCRIPTION:

This routine sets up the structures to include supplemental forcing analyses. The registry that defines the implemented schemes are invoked, followed by the routines to initialize the specific instance of the forcing scheme.
The methods invoked are:

LIS_suppforcing_plugin (4.20.1)
sets up function table registries for implemented supplemental forcing analyses

defineNativeSupp (3.47.2)
invokes the generic method in the registry to define the native domain of the supplemental forcing scheme

---

3.30.3 LIS_get_supp_forcing (Source File: LIS_suppforcingMod.F90)

INTERFACE:

```
subroutine LIS_get_supp_forcing(n)
```

USES:

```
use LIS_coreMod,           only : LIS_rc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine issues the call to retrieve data from the specified supplemental forcing analysis. 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 details are similar to the description in `LIS_baseforcingMod`.
The arguments are:

n index of the domain or nest.

The methods invoked are:

getsuppforc (3.47.4)
invokes the generic method in the registry to retrieve the supplemental forcing data

timeinterpsupp (3.47.6)
invokes the generic method in the registry to perform temporal interpolation

LIS_lapseRateCorrection (3.18.7)
method to apply lapse-rate based topographical corrections

3.30.4 LIS_suppforcing_finalize (Source File: LIS_suppforcingMod.F90)

INTERFACE:

```
subroutine LIS_suppforcing_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 the supplemental forcing scheme.
The methods invoked are:

suppforcingfinalize (3.47.8)
invokes the generic method in the registry to cleanup the allocated structures for the supplemental forcing scheme.

3.31 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_mod
use LIS_PRIV_rcMod
```

3.31.1 isMonthlyAlarmRinging (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
interface LIS_isMonthlyAlarmRinging
```

PRIVATE MEMBER FUNCTIONS:

```
module procedure isMonthlyAlarmRinging1
module procedure isMonthlyAlarmRinging2
```

DESCRIPTION:

checks if the monthly alarm is ringing. The private functions have different arguments based on the input options specified.

3.31.2 LIS_computeTemporalWeights (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
    interface LIS_computeTemporalWeights
       module procedure computeTemporalWeights1
       module procedure computeTemporalWeights2
    end interface
 contains
%//////////////////////////////////////////////////////////

\mbox{}\hrulefill\

\subsubsection{LIS\_timemgr\_init (Source File: LIS\_timeMgrMod.F90)}


   \label{LIS_timemgr_init}

\bigskip{\sf INTERFACE:}
\begin{verbatim}   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 (3.31.22)
to convert current date to a floating point format

LIS_timemgr_print (3.31.5)
display the contents of the time manager

3.31.3 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 (3.31.22)
to convert current date to a floating point format

LIS_timemgr_print (3.31.5)
display the contents of the time manager

3.31.4 timemgr_set (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_timemgr_set(LIS_rc,yr,mo,da,hr,mn,ss)
```

USES:

```
use LIS_logMod, only : LIS_logunit, LIS_verify
implicit none
```

ARGUMENTS:

```
type(lisrcdec) :: LIS_rc
integer :: yr,mo,da,hr,mn,ss
```

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 (3.31.22)
convert date to a flotating point format

3.31.5 LIS_timemgr_print (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_timemgr_print(LIS_rc)
```

3.31.6 LIS_advance_timestep (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_advance_timestep(LIS_rc)
```

USES:

```
use LIS_logMod, only : LIS_logunit
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 (3.31.22)
convert date to a flotating point format

3.31.7 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

3.31.8 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

3.31.9 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.

3.31.10 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

3.31.11 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

3.31.12 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

3.31.13 LIS_setAlarm (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_setAlarm(LIS_rc, intervalType, alarm)
```

USES:

```
    use LIS_logMod,   only : LIS_verify
```

ARGUMENTS:

```
    type(lisrcdec)    :: LIS_rc
    integer           :: intervalType
    type(ESMF_Alarm)  :: alarm
```

DESCRIPTION:

Creates the alarms (monthly/weekly/daily) and attaches them to the time manager clock.

3.31.14 LIS_setMonthlyAlarm (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_setMonthlyAlarm(alarmTime)

  implicit none
```

ARGUMENTS:

```
  real*8 :: alarmTime
```

DESCRIPTION:

Initializes the monthly alarm
The arguments are:

alarmTime time of the monthly alarm

3.31.15 isMonthlyAlarmRinging1 (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
!Private name: call using isMonthlyAlarmRinging
subroutine isMonthlyAlarmRinging1(LIS_rc, alarmTime, interval, &
     midmonth, ringFlag)

  implicit none
```

ARGUMENTS:

```
  type(lisrcdec), intent(in) :: LIS_rc
  real*8, intent(inout) :: alarmTime
  logical, intent(inout):: ringFlag
  integer, intent(in) :: interval
  logical, intent(in) :: midmonth
```

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

interval 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:

date2time (**??**)
convert date to a flotating point format

3.31.16 isMonthlyAlarmRinging2 (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
!Private name: call using isMonthlyAlarmRinging
subroutine isMonthlyAlarmRinging2(LIS_rc, alarmTime, interval, ringFlag)
```

3.31.17 LIS_setHourlyAlarm (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_setHourlyAlarm(alarmTime)

  implicit none
```

ARGUMENTS:

```
  real*8 :: alarmTime
```

DESCRIPTION:

Initializes the hourly alarm
The arguments are:

alarmTime time of the monthly alarm

3.31.18 LIS_isHourlyAlarmRinging (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine LIS_isHourlyAlarmRinging(LIS_rc, alarmTime, interval, ringFlag)

  implicit none
```

ARGUMENTS:

```
  type(lisrcdec), intent(in) :: LIS_rc
  real*8, intent(inout) :: alarmTime
  integer, intent(in)   :: interval
  logical, intent(inout) :: ringFlag
```

DESCRIPTION:

checks if the hourly alarm is ringing. The function returns true when the elapsed alarm time is greater than the number of hours specified in the alarm's interval.
The arguments are:

LIS_rc instance of the `lis_module`

alarmTime the elapsed alarm time

interval the alarm's frequency in hours

ringflag flag indicating the status of the call

The calling sequence is:

tick (**??**)
advance the time by the specified increment

3.31.19 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 (3.31.25)
advance the time by the specified increment

3.31.20 LIS_computeTemporalWeights (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine computeTemporalWeights1(LIS_rc,alarm,intervalType,&
     time1, time2,wt1,wt2)

  implicit none
```

ARGUMENTS:

```
  type(lisrcdec)   :: LIS_rc
  type(ESMF_Alarm) :: alarm
  integer          :: intervalType
  type(ESMF_Time)  :: time1
  type(ESMF_Time)  :: time2
  real    :: wt1,wt2
```

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 of 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:

date2time (**??**)
converts date to a floating point format

3.31.21 LIS_computeTemporalWeights (Source File: LIS_timeMgrMod.F90)

INTERFACE:

```
subroutine computeTemporalWeights2(LIS_rc,intervalType,mo1,mo2,wt1,wt2)

  implicit none
```

ARGUMENTS:

```
  type(lisrcdec) :: LIS_rc
  integer :: intervalType
  integer :: mo1,mo2
  real    :: wt1,wt2
```

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 of 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 (3.31.22)
converts date to a floating point format

3.31.22 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
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

3.31.23 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

3.31.24 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

---

3.31.25 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,ts,doy
  real    :: 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

3.31.26 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)

3.31.27 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 (3.31.28)
convert julian hour to hour, day, month, and year

lis_abort (**??**)
abort the code in case of error

3.31.28 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.

3.31.29 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

3.32 Fortran: Module Interface LIS_topoMod (Source File: LIS_topoMod.F90)

The code in this file implements routines to read various sources of topography data

3.32.1 Overview

This module provides routines for reading and modifying topography data.

REVISION HISTORY:

```
 18 Jul 2008: Sujay Kumar; Initial implementation
implicit none

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

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
 public :: LIS_topo_init  ! initializes data structures and memory
 public :: LIS_diagnosetopography
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
 public :: LIS_topo
```

3.32.2 LIS_topo_init (Source File: LIS_topoMod.F90)

INTERFACE:

```
subroutine LIS_topo_init()
```

USES:

```
  use ESMF_Mod
  use LIS_coreMod,  only : LIS_rc, LIS_config
  use LIS_fileIOMod, only : LIS_readDomainConfigSpecs
```

DESCRIPTION:

Allocates memory for data structures for reading topo datasets

---

3.32.3 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 3.16.4

3.32.4 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.

3.33 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

---

3.33.1 registerrtminit (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(registerrtminit)(int *i, void (*func)(int*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to initialize the radiative transfer model
The arguments are:

i index of the RTM .

3.33.2 rtminit (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(rtminitialize)(int *i, int *index)
```

DESCRIPTION:

Invokes the routine from the registry to initialize the RTM
The arguments are:

i index of the RTM

3.33.3 registerrtmf2t (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(registerrtmf2t)(int *i, void (*func)(int*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to transfer atmospheric profiles to the RTM.
The arguments are:

i index of the RTM .

3.33.4 rtmf2t (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(rtmf2t)(int *i, int *index)
```

DESCRIPTION:

Invokes the routine from the registry to initialize the RTM
The arguments are:

i index of the RTM

3.33.5 registersfc2rtm (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(registersfc2rtm)(int *i, int *j, void (*func)(int*, void*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to transfer surface properties to the RTM.
The arguments are:

i index of the RTM .

3.33.6 sfc2rtm (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(sfc2rtm)(int *i, int *j, int *index, void *sfc)
```

DESCRIPTION:

Invokes the routine from the registry to map surface properties to the RTM
The arguments are:

i index of the RTM

j index of the lsm

3.33.7 registergeometry2rtm (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(registergeometry2rtm)(int *i, void (*func)(int*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to specify the sensor geometries in the RTM
The arguments are:

i index of the RTM .

3.33.8 geometry2rtm (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(geometry2rtm)(int *i, int *index)
```

DESCRIPTION:

Invokes the routine from the registry to map sensor geometry information to RTM
The arguments are:

i index of the RTM

3.33.9 registerrtmrun (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(registerrtmrun)(int *i, void (*func)(int*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to specify the forward model integration step of the RTM
The arguments are:

i index of the RTM .

3.33.10 rtmrun (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(rtmrun)(int *i, int *index)
```

DESCRIPTION:

Invokes the routine from the registry to call the forward model integration step of the RTM
The arguments are:

i index of the RTM

3.33.11 registerrtmoutput (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(registerrtmoutput)(int *i, void (*func)(int*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to specify the output writing step of the RTM
The arguments are:

i index of the RTM .

3.33.12 rtmoutput (Source File: LIS_RTM_FTable.c)

INTERFACE:

```
void FTN(rtmoutput)(int *i, int *index)
```

DESCRIPTION:

Invokes the routine from the registry to call the output writing step of the the RTM
The arguments are:

i index of the RTM

3.34 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

---

3.34.1 registerreadmxsnoalb (Source File: LIS_alb_FTable.c)

INTERFACE:

```
void FTN(registerreadmxsnoalb)(int *j,void (*func)(int*, float*))
```

DESCRIPTION:

Creates a entry in the registry for the routine to open and read albedo upper bound over deep snow
The arguments are:

j index of the albedo source

3.34.2 readmxsnoalb (Source File: LIS_alb_FTable.c)

INTERFACE:

```
void FTN(readmxsnoalb)(int *n, int *j,float *array)
```

DESCRIPTION:

Invokes the routines from the registry for reading albedo upper bound over deep snow data
The arguments are:

n index of the nest

j index of the albedo source

array pointer to the array containing the albedo data.

3.34.3 registeralbedosetup (Source File: LIS_alb_FTable.c)

INTERFACE:

```
void FTN(registeralbedosetup)(int *j,void (*func)(int*))
```

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

3.34.4 albedosetup (Source File: LIS_alb_FTable.c)

INTERFACE:

```
void FTN(albedosetup)(int *n, int *j)
```

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

3.34.5 registerreadalbedo (Source File: LIS_alb_FTable.c)

INTERFACE:

```
void FTN(registerreadalbedo)(int *j,void (*func)(int*, void*, float*))
```

DESCRIPTION:

Creates an entry in the registry for the routine to read albedo climatology data
The arguments are:

j index of the albedo source

3.34.6 readalbedo (Source File: LIS_alb_FTable.c)

INTERFACE:

```
void FTN(readalbedo)(int *n, int *j, void *time, float *array)
```

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.

3.35 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

---

3.35.1 registerbiasestimationinit (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(registerbiasestimationinit)(int *i, void (*func)())
```

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:

i index of the bias estimation algorithm.

3.35.2 biasestimationinit (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(biasestimationinit)(int *i)
```

DESCRIPTION:

Invokes the the selected bias estimation algorithm initialization for use in data assimilation
The arguments are:

i index of the variable being assimilated

3.35.3 registerbiasestimationsetup (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(registerbiasestimationsetup)(int *i, void (*func)(int*))
```

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:

i index of the bias estimation algorithm.

3.35.4 biasestimationsetup (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(biasestimationsetup)(int *i, int *k)
```

DESCRIPTION:

Invokes the the selected bias estimation algorithm setup for use in data assimilation
The arguments are:

i index of the variable being assimilated

k index of the DA instance

3.35.5 registerbiasestimationcompute (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(registerbiasestimationcompute)(int *i, void (*func)(int*, int*))
```

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:

i index of the bias estimation algorithm.

3.35.6 computebiascorrection (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(computebiascorrection)(int *i, int *n, int *k)
```

DESCRIPTION:

Invokes the the selected bias estimation algorithm for computing bias correction prior to model propagation
The arguments are:

i index of the variable being assimilated

n index of the nest

k index of the assimilation instance

3.35.7 registerbiasestimationupdate (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(registerbiasestimationupdate)(int *i, void (*func)(int*, int*))
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for updating bias correction estimation
The arguments are:

i index of the bias estimation algorithm.

3.35.8 applybiascorrection (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(applybiascorrection)(int *i, int *n, int *k)
```

DESCRIPTION:

Invokes the the selected bias estimation algorithm for updating bias correction estimate
The arguments are:

i index of the variable being assimilated

n index of the nest

k index of the assimilation instance

3.35.9 registerbiasestimationrestart (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(registerbiasestimationrestart)(int *i, void (*func)(int*, int*))
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for writing bias restart
The arguments are:

i index of the bias estimation algorithm.

3.35.10 writebiasrestart (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(writebiasrestart)(int *i, int *n, int *k)
```

DESCRIPTION:

Invokes the the selected bias estimation algorithm for writing bias restart
The arguments are:

i index of the variable being assimilated

n index of the nest

k index of the assimilation instance

3.35.11 registerbiasestimationfinalize (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(registerbiasestimationfinalize)(int *i, void (*func)())
```

DESCRIPTION:

Creates an entry in the registry for the routine that finalizes structures related to bias correction algorithm.
The arguments are:

i index of the bias estimation algorithm.

3.35.12 biasestimationfinalize (Source File: LIS_biasestim_FTable.c)

INTERFACE:

```
void FTN(biasestimationfinalize)(int *i)
```

DESCRIPTION:

Finalizes the the selected bias estimation algorithm related structures
The arguments are:

i index of the variable being assimilated

3.36 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.

3.36.1 registerdaobssetup (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(registerdaobssetup)(int *i, void (*func)(int*, void*, void*))
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for handling observation data for data assimilation
The arguments are:

i index of the assimilation set

3.36.2 readobsdataconfig (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(readobsdataconfig)(int *i, int *k, void *obs, void *pert)
```

DESCRIPTION:

Invokes the routine from the registry to set up structures for handling observation data for data assimilation
The arguments are:

i 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

3.36.3 registerdaobs (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(registerdaobs)(int *i, void (*func)(int*, void*))
```

DESCRIPTION:

Creates an entry in the registry for the routine that reads observations for data assimilation.
The arguments are:

i index of the observations variable

3.36.4 readdaobs (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(readdaobs)(int *i,int *n, void *obsstate)
```

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:

i index of the observations variable

n index of the nest

obsstate pointer to the ESMF observation state

3.36.5 registerwritedaobs (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(registerwritedaobs)(int *i, void (*func)(int*, void*))
```

DESCRIPTION:

Creates an entry in the registry for the routine that writes observations used in data assimilation to disk.
The arguments are:

i index of the observations variable

3.36.6 writedaobs (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(writedaobs)(int *i,int *n, void *obsstate)
```

DESCRIPTION:

Invokes the routine from the registry to write observations used in data assimilation to disk.
The arguments are:

i index of the observations variable

n index of the nest

obsstate pointer to the ESMF observation state

3.36.7 registergetnso (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(registergetnso)(int *i, void (*func)(int*, int*, int*, int*))
```

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:

i index of the observations variable

3.36.8 getselectedobsnumber (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(getselectedobsnumber)(int *i, int *n, int *gid, int *sid, int *eid)
```

DESCRIPTION:

Invokes the routine from the registry that computes the number of selected observations for a particular modeling point
The arguments are:

i 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

3.36.9 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:

i index of the assimilation algorithm

INTERFACE:

```
void FTN(registerdainit)(int *i, void (*func)())
```

3.36.10 dataassiminit (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(dataassiminit)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to initialize the specific structures for the data assimilation algorithm used.
The arguments are:

i index of the assimilation algorithm

3.36.11 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:

i index of the assimilation algorithm

INTERFACE:

```
void FTN(registerdasetup)(int *i, void (*func)(int*))
```

3.36.12 dataassimsetup (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(dataassimsetup)(int *i, int *k)
```

DESCRIPTION:

Invokes the routine from the registry to setup the specific structures for the data assimilation algorithm used.
The arguments are:

i index of the assimilation algorithm

k index of the DA instance

3.36.13 registerapplyincrements (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(registerapplyincrements)(int *i, void (*func)(int*, int*))
```

DESCRIPTION:

Creates an entry in the registry for the routine that applies the analysis increments
The arguments are:

i index of the assimilation algorithm

3.36.14 applyincrements (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(applyincrements)(int *i, int *n, int *k)
```

DESCRIPTION:

Invoke the routine from the registry that applies analysis increments
The arguments are:

i index of the assimilation algorithm

n index of the nest

k index of the assimilation instance

3.36.15 registercomputeincrements (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(registercomputeincrements)(int *i, void (*func)(int*, int*))
```

DESCRIPTION:

Creates an entry in the registry for the routine that computes the analysis increments
The arguments are:

i index of the assimilation algorithm

3.36.16 computeincrements (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(computeincrements)(int *i, int *n, int *k)
```

DESCRIPTION:

Invoke the routine from the registry that computes analysis increments
The arguments are:

i index of the assimilation algorithm

n index of the nest

k index of assimilation instance

3.36.17 registerdafinalize (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(registerdafinalize)(int *i, void (*func)())
```

DESCRIPTION:

Cretes an entry in the registry for the routine to cleanup allocated structures
The arguments are:

i index of the assimilation algorithm

3.36.18 dafinalize (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(dafinalize)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry that cleans up the data assimilation specific structures
The arguments are:

i index of the assimilation algorithm

3.36.19 registerdaoutput (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(registerdaoutput)(int *i, void (*func)(int*, int*))
```

DESCRIPTION:

Cretes an entry in the registry for the writing diagnostic output from the data assimilation routines
The arguments are:

i index of the assimilation algorithm

3.36.20 daoutput (Source File: LIS_dataassim_FTable.c)

INTERFACE:

```
void FTN(daoutput)(int *i,int *n, int *k)
```

DESCRIPTION:

Invokes the routine from the registry to write data assimilation specific output
The arguments are:

i index of the assimilation algorithm

n index of the nest

k index of the assimilation instance

3.37 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

---

3.37.1 registerdomain (Source File: LIS_domain_FTable.c)

INTERFACE:

```
void FTN(registerdomain)(int *i,void (*func)())
```

DESCRIPTION:

Creates an entry in the registry for the routine to create grid and tile spaces for a particular domain definition
The arguments are:

i index of the domain

3.37.2 makedomain (Source File: LIS_domain_FTable.c)

INTERFACE:

```
void FTN(makedomain)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to create grid and tile spaces for a particular domain definition
The arguments are:

i index of the domain

3.37.3 registerinput (Source File: LIS_domain_FTable.c)

INTERFACE:

```
void FTN(registerinput)(int *i,void (*func)())
```

DESCRIPTION:

Makes an entry in the registry for the routine to read the runtime domain specifics
The arguments are:

i index of the domain

3.37.4 readinput (Source File: LIS_domain_FTable.c)

INTERFACE:

```
void FTN(readinput)(int *i)
```

DESCRIPTION:

Calls the routine from the registry to read the the runtime domain specifics
The arguments are:

i index of the domain

3.38 Fortran: Module Interface LIS_forcing_FTable (Source File: LIS_forcing_FTable.c)

Function table registries for storing the interface implementations for managing different meteorological analyses implmented as "baseforcings"

3.38.1 registerget (Source File: LIS_forcing_FTable.c)

INTERFACE:

```
void FTN(registerget)(int *i,void (*func)(int*))
```

DESCRIPTION:

Creates an entry in the registry for the routines to open and read base forcing
The arguments are:

i index of the forcing type

3.38.2 retrieveforcing (Source File: LIS_forcing_FTable.c)

INTERFACE:

```
void FTN(retrieveforcing)(int *i, int *n)
```

DESCRIPTION:

Invokes the routine from the registry to open and read the base forcing
The arguments are:

i index of the forcing type

n index of the nest

3.38.3 registertimeinterp (Source File: LIS_forcing_FTable.c)

INTERFACE:

```
void FTN(registertimeinterp)(int *i,void (*func)(int*))
```

DESCRIPTION:

Creats an entry in the registry for the routine to perform temporal interpolation
The arguments are:

i index of the forcing type

3.38.4 timeinterp (Source File: LIS_forcing_FTable.c)

INTERFACE:

```
void FTN(timeinterp)(int *i, int *n)
```

DESCRIPTION:

Invokes the routine from the registry to perform temporal interpolation
The arguments are:

i index of the forcing type

n index of the nest

3.38.5 registerdefinenative (Source File: LIS_forcing_FTable.c)

INTERFACE:

```
void FTN(registerdefinenative)(int *i, void (*func)())
```

DESCRIPTION:

Creates an entry in the registry for the routine that define the native domain of the base forcing scheme.
The arguments are:

i index of the forcing type

3.38.6 definenative (Source File: LIS_forcing_FTable.c)

INTERFACE:

```
void FTN(definenative)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry that defines the native domain of the base forcine scheme.
The arguments are:

i index of the forcing type

3.38.7 registerforcingfinal (Source File: LIS_forcing_FTable.c)

INTERFACE:

```
void FTN(registerforcingfinal)(int *i,void (*func)())
```

DESCRIPTION:

Creates an entry in the registry for the routine to cleanup allocated structures
The arguments are:

i index of the forcing type

3.38.8 forcingfinalize (Source File: LIS_forcing_FTable.c)

INTERFACE:

```
void FTN(forcingfinalize)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to cleanup the allocated structures associated with the base forcing
The arguments are:

i index of the forcing type

3.38.9 registerforcingreset (Source File: LIS_forcing_FTable.c)

INTERFACE:

```
void FTN(registerforcingreset)(int *i,void (*func)())
```

DESCRIPTION:

Creates an entry in the registry for the routine to cleanup allocated structures
The arguments are:

i index of the forcing type

3.38.10 forcingresetize (Source File: LIS_forcing_FTable.c)

INTERFACE:

```
void FTN(forcingreset)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to cleanup the allocated structures associated with the base forcing
The arguments are:

i index of the forcing type

3.39 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

---

3.39.1 registergfracsetup (Source File: LIS_gfrac_FTable.c)

INTERFACE:

```
void FTN(registergfracsetup)(int *j,void (*func)(int*))
```

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

3.39.2 gfracsetup (Source File: LIS_gfrac_FTable.c)

INTERFACE:

```
void FTN(gfracsetup)(int *n, int *j)
```

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

3.39.3 registerreadgfrac (Source File: LIS_gfrac_FTable.c)

INTERFACE:

```
void FTN(registerreadgfrac)(int *j,void (*func)(int*, void*, float*))
```

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

3.39.4 readgfrac (Source File: LIS_gfrac_FTable.c)

INTERFACE:

```
void FTN(readgfrac)(int *n, int *j,void *time, float *array)
```

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

3.40 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

---

3.40.1 registerreadlai (Source File: LIS_laisai_FTable.c)

INTERFACE:

```
void FTN(registerreadlai)(int *j, void (*func)(int*, void*,float*))
```

DESCRIPTION:

Creates an entry in the registry for the routines to read LAI data
The arguments are:

j index of the LAI source

3.40.2 readlai (Source File: LIS_laisai_FTable.c)

INTERFACE:

```
void FTN(readlai)(int *n, int *j, void *time, float* array)
```

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

array pointer to the LAI data

3.40.3 registerreadsai (Source File: LIS_laisai_FTable.c)

INTERFACE:

```
void FTN(registerreadsai)(int *j, void (*func)(int*, void*, float*))
```

DESCRIPTION:

Creates an entry in the registry for the routines to read SAI data
The arguments are:

j index of the SAI source

3.40.4 readsai (Source File: LIS_laisai_FTable.c)

INTERFACE:

```
void FTN(readsai)(int *n, int *j, void *time, float* array)
```

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

3.41 Fortran: Module Interface LIS_landcover_FTable (Source File: LIS_landcover_FTable.c)

Function table registries for storing the interface implementations for managing different sources of landcover data

---

3.41.1 registerreadlc (Source File: LIS_landcover_FTable.c)

INTERFACE:

```
void FTN(registerreadlc)(int *j, void (*func)(int*, float*))
```

DESCRIPTION:

Creates an entry in the registry for the routine to read landcover data
The arguments are:

j index of the landcover source

3.41.2 readlandcover (Source File: LIS_landcover_FTable.c)

Invokes the routine from the registry for reading landcover data.
The arguments are:

n index of the nest

j index of the landcover source

array pointer to the landcover data

INTERFACE:

```
void FTN(readlandcover)(int *n,  int *j,float *array)
```

3.41.3 registerreadmask (Source File: LIS_landcover_FTable.c)

INTERFACE:

```
void FTN(registerreadmask)(int *j, void (*func)(int*, float*))
```

DESCRIPTION:

Creates an entry in the registry for the routine to read landmask data
The arguments are:

j index of the landcover source

3.41.4 readlandmask (Source File: LIS_landcover_FTable.c)

INTERFACE:

```
void FTN(readlandmask)(int *n, int *j,float *array)
```

DESCRIPTION:

Invokes the routine from the registry for reading landmask data.
The arguments are:

n index of the nest

j index of the landcover source

array pointer to the landmask data

3.41.5 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.

3.41.6 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)
```

3.42 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

---

3.42.1 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(registerlsmini)(int *i, void (*func)())
```

3.42.2 lsmini (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmini)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to perform land surface model initialization

i index of the LSM

3.42.3 registerlsmrun (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmrun)(int *i, void (*func)(int*))
```

DESCRIPTION:

Creates an entry in the registry for the routine to run the land surface model

i index of the LSM

3.42.4 lsmrun (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmrun)(int *i,int *n)
```

DESCRIPTION:

Invokes the routine from the registry to run the land surface model

i index of the LSM

n index of the nest

3.42.5 registerlsmfinalize (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmfinalize)(int *i, void (*func)())
```

DESCRIPTION:

Creates an entry in the registry for the routine to cleanup allocated structures specific to the land surface model

i index of the LSM

3.42.6 lsmfinalize (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmfinalize)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry for cleaning up allocated structures specific to the land surface model

i index of the LSM

3.42.7 registerlsmsetup (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmsetup)(int *i, void (*func)())
```

DESCRIPTION:

Makes an entry in the registry for the routine to set up land surface model parameters

i index of the LSM

3.42.8 lsmsetup (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmsetup)(int *i)
```

DESCRIPTION:

Invokes the routine in the registry to set up land surface model parameters

i index of the LSM

3.42.9 registerlsmrestart (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmrestart)(int *i, void (*func)())
```

DESCRIPTION:

Makes an entry in the registry for the routine to restart the land surface model from a previously saved state

i index of the LSM

3.42.10 lsmrestart (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmrestart)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to restart the land surface model from a previously saved state

i index of the LSM

3.42.11 registerlsmoutput (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmoutput)(int *i, void (*func)(int*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to perform land surface model output

i index of the LSM

3.42.12 lsmoutput (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmoutput)(int *i, int *n)
```

DESCRIPTION:

Invokes the routine from the registry to perform land surface model output

i index of the LSM

n index of the nest

3.42.13 registerlsmf2t (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmf2t)(int *i, int *j, void (*func)(int*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to transfer forcing to model tiles

i index of the LSM

j index of the runmode

3.42.14 lsmf2t (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmf2t)(int *i, int *j, int*n)
```

DESCRIPTION:

Invokes the routine from the registry to transfer forcing to model tiles

i index of the LSM

j index of the runmode

n index of the nest

3.42.15 registerlsmwrst (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmwrst)(int *i, void (*func)(int*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to write restart files for a land surface model

i index of the LSM

3.42.16 lsmwrst (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmwrst)(int *i, int *n)
```

DESCRIPTION:

Invokes the routine from the registry to write restart files for a land surface model

i index of the LSM

n index of the nest

3.42.17 registergetstatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registergetstatevar)(int *i, int *j, void (*func)(int*, void*))
```

DESCRIPTION:

Makes an entry in the registry for the routine for obtaining the specified prognostic variables from the land surface model (for data assimilation)

i index of the LSM

j index of the variable

3.42.18 getlsmstatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(getlsmstatevar)(int *i, int *j, int *n, void *state)
```

DESCRIPTION:

Invokes the routine from the registry for obtaining the specified prognostic variables from the land surface model (for data assimilation)

i index of the LSM

j index of the variable

n index of the nest

state pointer to the prognostic variable state

3.42.19 registersetstatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registersetstatevar)(int *i, int *j, void (*func)(int*, void*))
```

DESCRIPTION:

Makes an entry in the registry for updating the specified state variable in a land surface model (for data assimilation)

i index of the LSM

j index of the variable

3.42.20 setlsmstatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(setlsmstatevar)(int *i, int *j, int *n, void *statevar)
```

DESCRIPTION:

Invokes the routine from the registry for updating the specified state variable in a land surface model (for data assimilation)

i index of the LSM

j index of the variable

n index of the nest

statevars pointer to the prognostic variable state

3.42.21 registerobstransform (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerobstransform)(int *i, int *j, void (*func)(int*, void*))
```

DESCRIPTION:

Makes an entry in the registry to perform the translation of observations to state variables (for data assimilation)

i index of the LSM

j index of the variable being assimilated

k index of the observation data

3.42.22 obstransform (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(obstransform)(int *i, int *j, int *n, void *obs)
```

DESCRIPTION:

Invokes the routine from the registry to translate the observations to the prognostic variable space (for data assimilation)

i index of the LSM

j index of the variable being assimilated

n index of the nest

obs transformed variable

3.42.23 registergetobspred (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registergetobspred)(int *i, int *j, void (*func)(int*,float*))
```

DESCRIPTION:

Makes an entry in the registry for the routine that provides an LSM's estimate of the observations. (for data assimilation)

i index of the LSM

j index of the observation data

3.42.24 getobspred (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(getobspred)(int *i, int *j, int *n, float *pred)
```

DESCRIPTION:

Invokes the routine from the registry to translate the observations to state variables (for data assimilation)

i index of the LSM

j index of the observation data

n index of the nest

pred model's estimated observation prediction

3.42.25 registermapobstolsm (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registermapobstolsm)(int *i, int *j, void (*func)(int*, void*, void*))
```

DESCRIPTION:

Makes an entry in the registry to perform the translation of observations to state variables (for data assimilation)

i index of the LSM

j index of the observation data

3.42.26 mapobstolsm (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(mapobstolsm)(int *i, int *j, int *n, void *obs, void *lsm)
```

DESCRIPTION:

Invokes the routine from the registry to translate the observations to state variables (for data assimilation)

i index of the LSM

j index of the observation data

n index of the nest

obs observations to be mapped

lsm updated lsm states

3.42.27 registerlsmexport (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmexport)(int *i, int *j, void (*func)(int*, void*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to set export states for a LSM

i index of the LSM

j index of the run mode

3.42.28 lsmsetexport (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmsetexport)(int *i, int *j, int *n, void *exp)
```

DESCRIPTION:

Invokes the routine in the registry to set the export states for a LSM

i index of the LSM

j index of the run mode

n index of the nest

3.42.29 registerlsmdynsetup (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerlsmdynsetup)(int *i, void (*func)(int*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to set the time dependent land surface parameters

i index of the LSM

3.42.30 lsmdynsetup (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(lsmdynsetup)(int *i, int *n)
```

DESCRIPTION:

Invokes the routine from the registry to set the time dependent land surface parameters

i index of the LSM

n index of the nest

3.42.31 registerqclsmstate (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerqclsmstate)(int *i, int *j, void (*func)(int*, void*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to QC the updated LSM state (for data assimilation)

i index of the LSM

j index of the assimilated variable

3.42.32 qclsmstate (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(qclsmstate)(int *i, int *j, int *n, void *LSM_State)
```

DESCRIPTION:

Invokes the routine from the registry to set the QC the updated LSM variables (for data assimilation)

i index of the LSM

j index of the assimilated variable

n index of the nest

LSM_State The LSM state being qc'd

3.42.33 registerqcobsstate (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerqcobsstate)(int *i, int *j, void (*func)(int*, void*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to QC the OBS state (for data assimilation)

i index of the LSM

j index of the assimilated variable

3.42.34 qcobsstate (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(qcobsstate)(int *i, int *j, int *n, void *LSM_State)
```

DESCRIPTION:

Invokes the routine from the registry to set the QC the updated LSM variables (for data assimilation)

i index of the LSM

j index of the assimilated variable

n index of the nest

LSM_State The LSM state being qc'd

3.42.35 registerscalelsmstatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerscalelsmstatevar)(int *i, int *j, void (*func)(int*, void*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to scale the LSM state variables (for data assimilation)

i index of the LSM

j index of the assimilated variable

3.42.36 scalelsmstatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(scalelsmstatevar)(int *i, int *j, int *n, void *LSM_State)
```

DESCRIPTION:

Invokes the routine from the registry to scale the LSM state variables (for data assimilation)

i index of the LSM

j index of the assimilated variable

n index of the nest

LSM_State The LSM state being scaled

3.42.37 registerdescalelsmstatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerdescalelsmstatevar)(int *i, int *j, void (*func)(int*, void*, void*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to descale the LSM state variables (for data assimilation)

i index of the LSM

j index of the assimilated variable

3.42.38 descalelsmstatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(descalelsmstatevar)(int *i, int *j, int *n, void *LSM_State, void *LSM_Incr_State)
```

DESCRIPTION:

Invokes the routine from the registry descale the LSM state variables (for data assimilation)

i index of the LSM

j index of the assimilated variable

n index of the nest

LSM_State The LSM state being descaled

3.42.39 registerupdatelsmstate (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerupdatelsmstate)(int *i, int *j, void (*func)(int*, void*, void*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to apply the LSM state increments to LSM state (for data assimilation)

i index of the LSM

j index of the assimilation set

3.42.40 updatelsmstate (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(updatelsmstate)(int *i, int *j, int *n, void *LSM_State, void *LSM_Incr_State)
```

DESCRIPTION:

Invokes the routine from the registry to apply the LSM state increments to LSM State (for data assimilation)

i index of the LSM

j index of the assimilated variable

n index of the nest

LSM_State The LSM state being updated

LSM_Incr_State The LSM incr state

3.42.41 registerwritestatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerwritestatevar)(int *i, int *j, void (*func)(int *, int*, void*))
```

DESCRIPTION:

Makes an entry in the registry for the routine for obtaining the specified prognostic variables from the land surface model

i index of the LSM

j index of the variable

3.42.42 writelsmstatevar (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(writelsmstatevar)(int *i, 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

i index of the LSM

j index of the variable

n index of the nest

statevars pointer to the prognostic variable array

3.42.43 registersetlsmdecisionspace (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registersetlsmdecisionspace)(int *i, int *j, void (*func)(void*))
   Makes an entry in the registry for the routine
   to set the decision space for optimization
```

DESCRIPTION:

i index of the LSM

j index of the variable

3.42.44 setlsmdecisionspace (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(setlsmdecisionspace)(int *i, int *j, void *state)
```

DESCRIPTION:

Invokes the routine from the registry to set the decision space for optimization

i index of the LSM

j index of the variable

n index of the nest

statevars pointer to the prognostic variable array

3.42.45 registerqclsmdecisionspace (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registerqclsmdecisionspace)(int *i, int *j, void (*func)(void*))
```

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

3.42.46 qclsmdecisionspace (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(qclsmdecisionspace)(int *i, int *j, void *state)
```

DESCRIPTION:

Method to QC a specified decision space.

i index of the LSM

j index of the optimization set

state decision space object

3.42.47 registersetuppeobspred (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registersetuppeobspred)(int *i, int *j, void (*func)(void*))
```

DESCRIPTION:

registers the method to initialize the obs pred for the PE observation (model simulated value of the observation)

i index of the LSM

j index of the optimization set

3.42.48 setuppeobspredspace (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(setuppeobspredspace)(int *i, int *j,void *pepred)
```

DESCRIPTION:

invokes the method to compute the obs pred for the PE observation (model simulated value of the observation)

i index of the LSM

j index of the variable

pepred object containing the PE obspred

3.42.49 registergetpeobspred (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(registergetpeobspred)(int *i, int *j, void (*func)(void*))
```

DESCRIPTION:

registers the method to compute the obs pred for the PE observation (model simulated value of the observation)

i index of the LSM

j index of the optimization set

3.42.50 getpeobspred (Source File: LIS_lsm_FTable.c)

INTERFACE:

```
void FTN(getpeobspred)(int *i, int *j,void *pepred)
```

DESCRIPTION:

invokes the method to compute the obs pred for the PE observation (model simulated value of the observation)

i index of the LSM

j index of the variable

pepred object containing the PE obspred

3.42.51 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.

3.42.52 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)
```

3.42.53 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)
```

3.42.54 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
```

3.43 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.

3.43.1 registeroptuealginit (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuealginit)(int *i, void (*func)())
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for initializing optimization / uncertainty estimation algorithms
The arguments are:

i index of the optmization/UE algorithm

3.43.2 optuealginit (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(optuealginit)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to set up structures for initializing optimization or uncertainty estimation algorithm
The arguments are:

i index of the optimization/UE algorithm

3.43.3 registeroptuealgsetup (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuealgsetup)(int *i, void (*func)())
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for optimization/ uncertainty estimation algorithms
The arguments are:

i index of the optmization/UE algorithm

3.43.4 optuealgsetup (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(optuealgsetup)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to set up structures for optimization/uncertainty estimation algorithm
The arguments are:

i index of the optimization/uncertainty estimation algorithm

3.43.5 registeroptuetypeinit (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuetypeinit)(int *i, void (*func)())
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for initializing optimization instance
The arguments are:

i index of the optmization type

3.43.6 optuetypeinit (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(optuetypeinit)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to set up structures for initializing optimization instance type
The arguments are:

i index of the optimization type

3.43.7 registeroptuetypesetdecspace (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuetypesetdecspace)(int *i, void (*func)())
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets the decision space for the particular optimization type
The arguments are:

i index of the optmization type

3.43.8 setoptuetypedecspace (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(setoptuetypedecspace)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to set up decision space for the particular optimization type
The arguments are:

i index of the optimization type

3.43.9 registeroptueconvergencecheck (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptueconvergencecheck)(int *i, void (*func)(void*))
```

DESCRIPTION:

Creates an entry in the registry for the routine that checks for convergence for an optimization algorithm
The arguments are:

i index of the optimization algorithm

3.43.10 checkconvergence (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(checkconvergence)(int *i, void* check)
```

DESCRIPTION:

Invokes the routine from the registry to check for convergence of an optimization algorithm.
The arguments are:

i index of the optimization algorithm

3.43.11 registeroptuealgrun (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuealgrun)(int *i, void (*func)())
```

DESCRIPTION:

Creates an entry in the registry for the routine that executes the optimization algorithm run method
The arguments are:

i index of the optimization algorithm.

3.43.12 runoptue (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(runoptue)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to execute the optimization algorithm run method
The arguments are:

i index of the optimization algorithm

3.43.13 registerpeobssetup (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registerpeobssetup)(int *i, void (*func)(void*))
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for handling observation data for optimization.
The arguments are:

i index of the optimization set

3.43.14 setuppeobsspace (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(setuppeobsspace)(int *i, void *obs)
```

DESCRIPTION:

Invokes the routine from the registry to set up structures for handling observation data for optimization
The arguments are:

i index of the optimization set

obs ESMF state that contain the observations

3.43.15 registergetpeobs (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registergetpeobs)(int *i, void (*func)(void*))
```

DESCRIPTION:

Creates an entry in the registry for the routine that sets up structures for reading observation data for optimization
The arguments are:

i index of the optimization set

3.43.16 getpeobs (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(getpeobs)(int *i, void *peobs)
```

DESCRIPTION:

Invokes the routine from the registry to set up structures for reading observation data for optimization
The arguments are:

i index of the variable being assimilated

peobs ESMF state that contain the observations

3.43.17 registerwritepeobs (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registerwritepeobs)(int *i, void (*func)(void*))
```

DESCRIPTION:

Creates an entry in the registry for the routine that writes the processed observations (objective space) to disk
The arguments are:

i index of the optimization set

3.43.18 writepeobs (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(writepeobs)(int *i, void *peobs)
```

DESCRIPTION:

Invokes the routine from the registry to set up structures for writing observation data for optimization
The arguments are:

i index of the variable being assimilated

peobs ESMF state that contain the observations

3.43.19 registerinitobjfunctype (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registerinitobjfunctype)(int *i, void (*func)())
```

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:

i index of the objective function method

3.43.20 objectivefunctypeinit (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(objectivefunctypeinit)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to set up structures for initializing the method of objective function evaluation
The arguments are:

i index of the objective function method

3.43.21 registercomputeobjfunctype (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registercomputeobjfunctype)(int *i, void (*func)())
```

DESCRIPTION:

Creates an entry in the registry for the routine that computes objective function based on the desired method
The arguments are:

i index of the objective function method

3.43.22 computeobjectivefunctype (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(computeobjectivefunctype)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to invoke an objective function method
The arguments are:

i index of the objective function method

3.43.23 registerupdateobjfunctype (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registerupdateobjfunctype)(int *i, void (*func)())
```

DESCRIPTION:

Creates an entry in the registry for the routine that updates objective function based on the desired method
The arguments are:

i index of the objective function method

3.43.24 updateobjectivefunctype (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(updateobjectivefunc)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to invoke an objective function method
The arguments are:

i index of the objective function method

3.43.25 registerresetobjfunctype (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registerresetobjfunctype)(int *i, void (*func)())
```

DESCRIPTION:

Creates an entry in the registry for the routine that resets objective function
The arguments are:

i index of the objective function method

3.43.26 resetobjectivefunctype (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(resetobjectivefunctype)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to invoke an objective function method
The arguments are:

i index of the objective function method

3.43.27 registeroptuetypeobjfunceval (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuetypeobjfunceval)(int *i, void (*func)())
```

DESCRIPTION:

Creates an entry in the registry for the routine that evaluates objective function
The arguments are:

i index of the runmode (param estimation, for e.g)

3.43.28 evaluateobjfunction (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(evaluateobjfunction)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to invoke an objective function evaluation
The arguments are:

i index of the objective function evaluation

3.43.29 registergetdecisionspace (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuegetdecisionspace)(int *i, void (*func)(int*, void*))
```

DESCRIPTION:

Creates an entry in the registry for the routine that retrieves decision space values from the optimization algorithm datastructures
The arguments are:

i index of the optimization algorithm

3.43.30 (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(getoptuealgdecspace)(int *i, int *n, void *decspace)
```

DESCRIPTION:

Invokes the routine from the registry to invoke an objective function method
The arguments are:

i index of the optimization algorithm

n index of the nest

decspace decision space object

3.43.31 registeroptuesetdecisionspace (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuesetdecisionspace)(int *i, void (*func)(int*, void*))
```

DESCRIPTION:

Creates an entry in the registry for the routine that assigns decision space values to the optimization algorithm datastructures
The arguments are:

i index of the optimization algorithm

3.43.32 (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(setoptuealgdecspace)(int *i, int *n, void *decspace)
```

DESCRIPTION:

Invokes the routine from the registry to set a given decision space to the optimization algorithm
The arguments are:

i index of the optimization algorithm

n index of the nest

decspace decision space object

3.43.33 registeroptuegetnparam (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(registeroptuegetnparam)(int *i, void (*func)(void*))
```

DESCRIPTION:

Creates an entry in the registry for the routine that retrieves the size of the decision space
The arguments are:

i index of the optimization algorithm

3.43.34 (Source File: LIS_optUE_FTable.c)

INTERFACE:

```
void FTN(getoptuealgnparam)(int *i, void *nparam)
```

DESCRIPTION:

Invokes the routine from the registry to retrieve the size of the decision space
The arguments are:

i index of the optimization algorithm

nparam size of the decision space

3.44 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

---

3.44.1 registerperturbinit (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(registerperturbinit)(int *i, void (*func)(int*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to initialize the pertubation scheme.
The arguments are:

i index of the perturbation algorithm .

3.44.2 perturbinit (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(perturbinit)(int *i, int *index)
```

DESCRIPTION:

Invokes the routine from the registry to initialize the pertubation scheme
The arguments are:

i index of the perturbation algorithm

3.44.3 registerperturbsetup (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(registerperturbsetup)(int *i, void (*func)(int*, int*, void*, void*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to initialize the pertubation scheme.
The arguments are:

i index of the perturbation algorithm .

3.44.4 perturbsetup (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(perturbsetup)(int *i, int *index, int *k,  void *base, void *pert)
```

DESCRIPTION:

Invokes the routine from the registry to initialize the pertubation scheme
The arguments are:

i index 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

3.44.5 registerperturbmethod (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(registerperturbmethod)(int *i, void (*func)(int*, int*, int*, void*, void*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to invoke the pertubation scheme.
The arguments are:

i index of the perturbation algorithm .

3.44.6 perturbmethod (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(perturbmethod)(int *i, int *index, int *n, int *k, void *base, void *pert)
```

DESCRIPTION:

Invokes the routine from the registry to call the pertubation scheme
The arguments are:

i index 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

3.44.7 registerperturbwriterst (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(registerperturbwriterst)(int *i, void (*func)(int*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to write restart files for the perturbations scheme.
The arguments are:

i index of the perturbation algorithm .

3.44.8 writepertrestart (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(writepertrestart)(int *i, int *nest)
```

DESCRIPTION:

Invokes the routine from the registry to write the restart file for the pertubation scheme
The arguments are:

nest nest index

3.44.9 registerperturbreadrst (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(registerperturbreadrst)(int *i, void (*func)())
```

DESCRIPTION:

Makes an entry in the registry for the routine to read restart files for the perturbations scheme.
The arguments are:

i index of the perturbation algorithm .

3.44.10 readpertrestart (Source File: LIS_perturb_FTable.c)

INTERFACE:

```
void FTN(readpertrestart)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to read the restart file for the pertubation scheme
The arguments are:

nest nest index

3.45 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

3.45.1 registerlisinit (Source File: LIS_runmode_FTable.c)

INTERFACE:

```
void FTN(registerlisinit)(int *i, void (*func)())
```

DESCRIPTION:

Makes an entry in the registry for the LIS initialization method for a certain running mode
The arguments are:

i index of the running mode

3.45.2 lisinit (Source File: LIS_runmode_FTable.c)

INTERFACE:

```
void FTN(lisinit)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to perform the LIS initialization for the specified running mode
The arguments are:

i index of the running mode

3.45.3 registerlisrun (Source File: LIS_runmode_FTable.c)

INTERFACE:

```
void FTN(registerlisrun)(int *i, void (*func)())
```

DESCRIPTION:

Makes an entry in the registry for the LIS run method for a certain running mode.
The arguments are:

i index of the running mode

3.45.4 lisrun (Source File: LIS_runmode_FTable.c)

INTERFACE:

```
void FTN(lisrun)(int *i)
```

DESCRIPTION:

Invokes the LIS run method from the registry for the specified running mode
The arguments are:

i index of the running mode

3.45.5 registerlisfinalize (Source File: LIS_runmode_FTable.c)

INTERFACE:

```
void FTN(registerlisfinalize)(int *i, void (*func)())
```

DESCRIPTION:

Makes an entry in the registry for the routine to perform LIS finalization for the specified running mode
The arguments are:

i index of the running mode

3.45.6 lisfinalize (Source File: LIS_runmode_FTable.c)

INTERFACE:

```
void FTN(lisfinalize)(int *i)
```

DESCRIPTION:

Invokes the routine from the registry to perform LIS finalization call for the specified running mode
The arguments are:

i index of the running mode

3.46 Fortran: Module Interface LIS_soils_FTable (Source File: LIS_soils_FTable.c)

Function table registries for storing the interface implementations for managing different sources of soil parameters data

---

3.46.1 registerreadsand (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(registerreadsand)(int *j, void (*func)(int *, float*))
```

DESCRIPTION:

Creates an entry in the registry for the routine to read the sand fraction data
The arguments are:

j index of the soils source

3.46.2 readsand (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(readsand)(int *n, int *j,float *array)
```

DESCRIPTION:

Invokes the routine from the registry to read the sand fraction data
The arguments are:

n index of the nest

j index of the soils source

array pointer to the sand array

3.46.3 registerreadclay (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(registerreadclay)(int *j,void (*func)(int*, float*))
```

DESCRIPTION:

Creates an entry in the registry for the routine to read the clay fraction data
The arguments are:

j index of the soils source

3.46.4 readclay (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(readclay)(int *n, int *j,float *array)
```

DESCRIPTION:

Invokes the routine from the registry to read the clay fraction data
The arguments are:

n index of the nest

j index of the soils source

array pointer to the clay data

3.46.5 registerreadsilt (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(registerreadsilt)(int *j,void (*func)(int*, float*))
```

DESCRIPTION:

Creates an entry in the registry for the routine to read the silt fraction data
The arguments are:

j index of the soils source

3.46.6 readsilt (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(readsilt)(int *n, int *j,float *array)
```

DESCRIPTION:

Invokes the routine from the registry to read the silt fraction data
The arguments are:

n index of the nest

j index of the soils source

array pointer to the silt data

3.46.7 registerreadtexture (Source File: LIS_soils_FTable.c)

Creates an entry in the registry for the routine to read the soil texture data
The arguments are:

j index of the soils source

INTERFACE:

```
void FTN(registerreadsoiltexture)(int *j,void (*func)(int*, int*))
```

3.46.8 readsoiltexture (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(readsoiltexture)(int *n, int *j,int *array)
```

DESCRIPTION:

Invokes the routine from the registry to read the soil texture data
The arguments are:

n index of the nest

j index of the soils source

array pointer to the texture array

3.46.9 registerreadporosity (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(registerreadporosity)(int *j,void (*func)(int*, float*))
```

DESCRIPTION:

Creates an entry in the registry for the routine to read the porosity data
The arguments are:

j index of the soils source

3.46.10 readporosity (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(readporosity)(int *n, int *j,float *array)
```

DESCRIPTION:

Invokes the routine from the registry to read the porosity data
The arguments are:

n index of the nest

j index of the soils source

array pointer to the porosity array

3.46.11 registerreadpsisat (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(registerreadpsisat)(int *j,void (*func)(int*, float*))
```

DESCRIPTION:

Creates an entry in the registry for the routine to read the saturated matric potential data
The arguments are:

j index of the soils source

3.46.12 readpsisat (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(readpsisat)(int *n, int *j,float *array)
```

DESCRIPTION:

Invokes the routine from the registry to read the saturated matric potential data
The arguments are:

n index of the nest

j index of the soils source

array pointer to the saturated matric potential data

3.46.13 registerreadksat (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(registerreadksat)(int *j,void (*func)(int*, float*))
```

DESCRIPTION:

Creates an entry in the registry for the routine to read the saturated hydraulic conductivity data
The arguments are:

j index of the soils source

3.46.14 readksat (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(readksat)(int *n, int *j,float *array)
```

DESCRIPTION:

Invokes the routine from the registry to read the saturated hydraulic conductivity data
The arguments are:

n index of the nest

j index of the soils source

array pointer to the ksat data

3.46.15 registerreadbexp (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(registerreadbexp)(int *j,void (*func)(int*, float*))
```

DESCRIPTION:

Creates an entry in the registry for the routine to read the b parameter data
The arguments are:

j index of the soils source

3.46.16 readbexp (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(readbexp)(int *n, int *j,float *array)
```

DESCRIPTION:

Invokes the routine from the registry to read the b parameter data
The arguments are:

n index of the nest

j index of the soils source

array pointer to the b parameter data

3.46.17 registerreadquartz (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(registerreadquartz)(int *j,void (*func)(int*, float*))
```

DESCRIPTION:

Creates an entry in the registry for the routine to read the quartz fraction data
The arguments are:

j index of the soils source

3.46.18 readquartz (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(readquartz)(int *n, int *j,float *array)
```

DESCRIPTION:

Invokes the routine from the registry to read the quartz fraction data
The arguments are:

n index of the nest

j index of the soils source

array pointer to the quartz data

3.46.19 registerreadcolor (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(registerreadcolor)(int *j, void (*func)(int*, float*))
```

DESCRIPTION:

Creates an entry in the registry for the routine to read the soil color data
The arguments are:

j index of the soils source

3.46.20 readcolor (Source File: LIS_soils_FTable.c)

INTERFACE:

```
void FTN(readcolor)(int *n, int *j,float *array)
```

DESCRIPTION:

Invokes the routine from the registry to read the soil color data
The arguments are:

n index of the nest

j index of the soils source

array pointer to the color array

3.47 Fortran: Module Interface LIS_suppforcing_FTable (Source File: LIS_suppforcing_FTab

Function table registries for storing the interface implementations for managing different supplemental forcing analyses

---

3.47.1 registerdefinenativesupp (Source File: LIS_suppforcing_FTable.c)

INTERFACE:

```
void FTN(registerdefinenativesupp)(int *i, void (*func)(int*))
```

DESCRIPTION:

Creates an entry in the registry for the routine that define the native domain of the supplemental forcing scheme.
The arguments are:

i index of the forcing type

3.47.2 definenativesupp (Source File: LIS_suppforcing_FTable.c)

INTERFACE:

```
void FTN(definenativesupp)(int *i, int *findex)
```

DESCRIPTION:

Invokes the routine from the registry that defines the native domain of the supplemental forcing scheme.
The arguments are:

i index of the forcing type

findex forcing index

3.47.3 registerreadsupp (Source File: LIS_suppforcing_FTable.c)

INTERFACE:

```
void FTN(registerreadsupp)(int *i,void (*func)(int*, int*, float*, float*))
```

DESCRIPTION:

Creates an entry in the registry for the routine to open and read the supplemental forcing
The arguments are:

i index of the forcing type

3.47.4 getsuppforc (Source File: LIS_suppforcing_FTable.c)

INTERFACE:

```
void FTN(getsuppforc)(int *n, int *i, int *m, float *data1, float *data2)
```

DESCRIPTION:

Invokes the routine from the registry to open and read the supplemental forcing
The arguments are:

n index of the nest

i index of the forcing type

3.47.5 registersuppti (Source File: LIS_suppforcing_FTable.c)

INTERFACE:

```
void FTN(registersuppti)(int *i,void (*func)(int*, int*, float*, float*))
```

DESCRIPTION:

Creats an entry in the registry for the routine to perform temporal interpolation of the supplemental forcing
The arguments are:

i index of the forcing type

3.47.6 timeinterpsupp (Source File: LIS_suppforcing_FTable.c)

INTERFACE:

```
void FTN(timeinterpsupp)(int *n, int *i, int *m, float *data1, float *data2)
```

DESCRIPTION:

Invokes the routine from the registry to perform temporal interpolation of the supplemental forcing
The arguments are:

n index of the nest

i index of the forcing type

3.47.7 registersuppforcingfinal (Source File: LIS_suppforcing_FTable.c)

INTERFACE:

```
void FTN(registersuppforcingfinal)(int *i,void (*func)(int*))
```

DESCRIPTION:

Creates an entry in the registry for the routine to cleanup allocated structures
The arguments are:

i index of the forcing type

3.47.8 suppforcingfinalize (Source File: LIS_suppforcing_FTable.c)

INTERFACE:

```
void FTN(suppforcingfinalize)(int *i, int *m)
```

DESCRIPTION:

Invokes the routine from the registry to cleanup the allocated structures associated with the supplemental forcing
The arguments are:

i index of the forcing type

3.48 Fortran: Module Interface LIS_tbot_FTable (Source File: LIS_tbot_FTable.c)

Function table registries for storing the interface implementations for managing different sources of bottom temperature data

---

3.48.1 registerreadtbot (Source File: LIS_tbot_FTable.c)

INTERFACE:

```
void FTN(registerreadtbot)(int *i,void (*func)(int*, float*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to read the bottom temperature data
The arguments are:

i index of the TBOT source

3.48.2 readtbot (Source File: LIS_tbot_FTable.c)

INTERFACE:

```
void FTN(readtbot)(int *n, int *i, float *array)
```

DESCRIPTION:

Invokes the routine from the registry to read the bottom temperature data
The arguments are:

n index of the nest

i index of the TBOT source

array array containing bottom temperature data.

3.49 Fortran: Module Interface LIS_topo_FTable (Source File: LIS_topo_FTable.c)

Function table registries for storing the interface implementations for managing different sources of topography data

---

3.49.1 registerreadelev (Source File: LIS_topo_FTable.c)

INTERFACE:

```
void FTN(registerreadelev)(int *j, void (*func)(int*, float*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to read elevation data
The arguments are:

j index of the topography source

3.49.2 readelev (Source File: LIS_topo_FTable.c)

INTERFACE:

```
void FTN(readelev)(int *n, int *j, float *array)
```

DESCRIPTION:

Invokes the routine from the registry to read the elevation data
The arguments are:

n index of the nest

j index of the topography source

array pointer to the elevation array

3.49.3 registerreadslope (Source File: LIS_topo_FTable.c)

INTERFACE:

```
void FTN(registerreadslope)(int *j, void (*func)(int*, float*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to read slope data
The arguments are:

j index of the topography source

3.49.4 readslope (Source File: LIS_topo_FTable.c)

INTERFACE:

```
void FTN(readslope)(int *n, int *j, float *array)
```

DESCRIPTION:

Invokes the routine from the registry to read the slope data
The arguments are:

n index of the nest

j index of the topography source

array pointer to the slope array

3.49.5 registerreadaspect (Source File: LIS_topo_FTable.c)

INTERFACE:

```
void FTN(registerreadaspect)(int*j, void (*func)(int*, float*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to read aspect data
The arguments are:

j index of the topography source

3.49.6 readaspect (Source File: LIS_topo_FTable.c)

INTERFACE:

```
void FTN(readaspect)(int *n, int *j, float *array)
```

DESCRIPTION:

Invokes the routine from the registry to read the aspect data
The arguments are:

n index of the nest

j index of the topography source

array pointer to the aspect array

3.49.7 registerreadcurv (Source File: LIS_topo_FTable.c)

INTERFACE:

```
void FTN(registerreadcurv)(int *j, void (*func)(int*, float*))
```

DESCRIPTION:

Makes an entry in the registry for the routine to read curvature data
The arguments are:

j index of the topography source

3.49.8 readcurv (Source File: LIS_topo_FTable.c)

INTERFACE:

```
void FTN(readcurv)(int *n, int *j, float *array)
```

DESCRIPTION:

Invokes the routine from the registry to read the curvature data
The arguments are:

n index of the nest

j index of the topography source

array pointer to the curvature array

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 `registerdaobssetup` and later called through `daobssetup`)

read observations Routines to read the observation data and perform any spatial transformation. (to be registered using `registerdaobs` 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 registerdaobs(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
```

4.4.1 LIS_RTM_plugin (Source File: LIS_RTM_pluginMod.F90)

INTERFACE:

```
subroutine LIS_RTM_plugin
```

4.5 Fortran: Module Interface LIS_baseforcing_pluginMod (Source File: LIS_baseforcing_plu

This module contains the definition of the functions used for incorporating a new base 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
```

4.5.1 LIS_baseforcing_plugin (Source File: LIS_baseforcing_pluginMod.F90)

INTERFACE:

```
subroutine LIS_baseforcing_plugin
```

DESCRIPTION:

This is a plugin point for introducing a new base forcing scheme. The interface mandates that the following interfaces be implemented and registered for each base 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 registerdefinenative and later invoked through `defineNative` 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 base forcing scheme. For example, assume that the GDAS forcing scheme is incorporated in the registry by the following calls.

```
call registerget(1,getgdas)
call registerdefinenative(1,defineNativeGDAS)
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 defineNative(1)  - calls defineNativeGDAS
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 defineNative(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 base forcing schemes, at runtime.

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
```

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_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
```

4.9.1 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 `registerlsmini` and later called using `lsmini`)

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`)

Output Routines to write output (to be registered using `registerlsmoutput` and later called using `lsmoutput`)

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 registerlsmini(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 registerlsmoutput(1,noah_output)
call registerlsmwrst(1,noah_writerst)
call registerlsmfinalize(1,noah_finalize)
```

The functions registered above are invoked using generic calls as follows:

```
call lsmini(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 lsmoutput(1)   - calls noah_output
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 lsmini(lis%lsm)
call lsmsetup(lis%lsm)
call lsmf2t(lis%lsm)
call lsmrun(lis%lsm)
call lsmdynsetup(lis%lsm)
call lsmoutput(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
```

4.10 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
```

4.10.1 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
```

4.11 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.
```

4.11.1 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 `registergetstatevar` and later called through `getstatevar` method)

Set the state prognostic variables Routine that sets the specified state prognostic variable(s) (to be registered using `registersetstatevar` and later called through `setstatevar` method)

Retrieve "Obspred" Routine that retrieves the model's estimate of observations. (to be registered using `registergetobspred` and later called through `getobspred` method)

QC the LSM state Routine that QCs the given LSM state for physical consistency. (to be registered using `registerqclsmstate` and later called through `qclsmstate` 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 `registerdescalelsmstat` and later called through `descalelsmstate` 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 `registergetstatevar` and later called through `getstatevar` method)

Set the state prognostic variables Routine that sets the specified state prognostic variable(s) (to be registered using `registersetstatevar` and later called through `setstatevar` method)

Transform Observations Routine that transforms observations to the LSM state space (could be as simple as a unit conversion). (to be registered using `registerobstransform` and later called through `obstransform` method)

Map observations to LSM space Routine that maps observations to the LSM state variables. (to be registered using `registermapobstolsm` and later called through `mapobstolsm` 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 TMI)

```
call registergetstatevar(1,1,noah271_getsoilm)
call registersetstatevar(1,1,noah271_setsoilm)
```

The functions registered above are invoked using generic calls as follows:

```
call getstatevar(1,1) - calls noah271_getsoilm
call setstatevar(1,1) - calls noah271_setsoilm
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call getstatevar(lis%lsm, lis%daset)
call setstatevar(lis%lsm, lis%daset)
call obstransform(lis%lsm, lis%daset)
```

where *lis%lsm* and *lis%daset* are set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_lsmda_plugin
```

4.12 Fortran: Module Interface LIS_lsmopt_pluginMod (Source File: LIS_lsmopt_pluginMod

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_lsmopt_plugin
```

4.12.1 LIS_lsmopt_plugin (Source File: LIS_lsmopt_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_lsmopt_plugin
```

4.13 Fortran: Module Interface LIS_lsmrtm_pluginMod (Source File: LIS_lsmrtm_pluginMo

REVISION HISTORY:

```
 24 Mar 09    Sujay Kumar  Initial Specification
```

4.13.1 LIS_lsmtm_plugin (Source File: LIS_lsmrtm_pluginMod.F90)

INTERFACE:

```
subroutine LIS_lsmrtm_plugin
```

4.14 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
```

4.14.1 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
```

4.15 Fortran: Module Interface LIS_optUEType_pluginMod (Source File: LIS_optUEType_p

This module contains the definition of the functions used for defining new types of optimization instances. 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
```

4.15.1 LIS_optUEType_plugin (Source File: LIS_optUEType_pluginMod.F90)

This is a plugin point for introducing a new optimization type

INTERFACE:

```
subroutine LIS_optUEType_plugin
```

4.16 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
```

4.16.1 LIS_topo_plugin (Source File: LIS_param_pluginMod.F90)

This is a plugin point for introducing new topography datasets. The interface mandates that the following routines be implemented and registered for each parameter data source.

read the elevation data Routines to retrieve the elevation data (to be registered using `registerreadelev` and later called using the generic `readelev` method)

read the slope data Routines to retrieve the slope data (to be registered using `registerreadslope` and later called using the generic `readslope` method)

read the aspect data Routines to retrieve the aspect data (to be registered using `registerreadaspect` and later called using the generic `readaspect` method)

read the curvature data Routines to retrieve the curvature data (to be registered using `registerreadcurv` and later called using the generic `readcurv` method)

The user-defined functions are included in the registry using two indices. For example, consider the incorporation of the topography datasets from the GTOPO30 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 registerreadelev(1,1,read_elev_gtopo30)
call registerreadslope(1,1,read_slope_gtopo30)
call registerreadaspect(1,1,read_aspect_gtopo30)
call registerreadcurv(1,1,read_curv_gtopo30)
```

The functions registered above are invoked using generic calls as follows:

```
call readelev(1,1)   - calls read_elev_gtopo30
call readslope(1,1)  - calls read_slope_gtopo30
call readaspect(1,1)  - calls read_aspect_gtopo30
call readcurv(1,1)  - calls read_curv_gtopo30
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call readelev(lis%domain, lis%toposrc)
call readslope(lis%domain, lis%toposrc)
call readaspect(lis%domain, lis%toposrc)
call readcurv(lis%domain, lis%toposrc)
```

where *lis%domain* and *lis%toposrc* are set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_topo_plugin
```

4.16.2 LIS_soils_plugin (Source File: LIS_param_pluginMod.F90)

This is a plugin point for introducing new soil parameter datasets. The interface mandates that the following routines be implemented and registered for each parameter data source.

read the texture data Routines to retrieve the texture data (to be registered using `registerreadsoiltexture` and later called using the generic `readtexture` method)

read the sand fraction data Routines to retrieve the sand fraction data (to be registered using `registerreadsand` and later called using the generic `readsand` method)

read the clay fraction data Routines to retrieve the clay fraction data (to be registered using `registerreadclay` and later called using the generic `readclay` method)

read the silt fraction data Routines to retrieve the silt fraction data (to be registered using `registerreadsilt` and later called using the generic `readsilt` method)

read the soil color data Routines to retrieve the soil color data (to be registered using `registerreadcolor` and later called using the generic `readcolor` method)

read the soil porosity data Routines to retrieve the soil porosity data (to be registered using `registerreadporosity` and later called using the generic `readporosity` method)

read the saturated matric potential data Routines to retrieve the saturated matric potential data (to be registered using `registerreadpsisat` and later called using the generic `readpsisat` method)

read the hydraulic conductivity data Routines to retrieve the hydraulic conductivity data (to be registered using `registerreadksat` and later called using the generic `readksat` method)

read the b parameter data Routines to retrieve the b parameter data (to be registered using `registerreadbexp` and later called using the generic `readbexp` method)

read the quartz data Routines to retrieve the quartz data (to be registered using `registerreadquartz` and later called using the generic `readquartz` method)

The user-defined functions are included in the registry using two indices. For example, consider the incorporation of the soil datasets from the FAO source with LIS using a latlon projection. The methods should be defined in the registry as follows (only a subset of the parameters is shown below, others can be defined in the same mannter), if the index of the source is defined to 1 and latlon projection in LIS uses and index of 1

```
call registerreadsand(1,1,read_faosand)
call registerreadclay(1,1,read_faoclay)
call registerreadsilt(1,1,read_faosilt)
```

The functions registered above are invoked using generic calls as follows:

```
call readsand(1,1)  - calls read_faosand
call readclay(1,1)  - calls read_faoclay
call readsilt(1,1)  - calls read_faosilt
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call readsand(lis%domain, lis%soilsrc)
call readclay(lis%domain, lis%soilsrc)
call readsilt(lis%domain, lis%soilsrc)
```

where *lis%domain* and *lis%soilsrc* are set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_soils_plugin
```

4.16.3 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
```

4.16.4 LIS_landcover_plugin (Source File: LIS_param_pluginMod.F90)

This is a plugin point for introducing new landcover datasets. The interface mandates that the following routines be implemented and registered for each parameter data source.

read the landmask data Routines to retrieve the landmask data (to be registered using `registerreadmask` and later called using the generic `readmask` method)

read the landcover data Routines to retrieve the landcover data (to be registered using `registerreadlc` and later called using the generic `readlandcover` method)

The user-defined functions are included in the registry using two indices. For example, consider the incorporation of the landcover datasets from the UMD 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 registerreadmask(1,1,read_latlon_umdmask)
call registerreadlc(1,1,read_latlon_umdlc)
```

The functions registered above are invoked using generic calls as follows:

```
call readmask(1,1)  - calls read_latlon_umdmask
call readlc(1,1)  - calls read_latlon_umdlc
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call readmask(lis%domain,lis%vegsrc)
call readlandcover(lis%domain,lis%vegsrc)
```

where *lis%domain* and *lis%vegsrc* are set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_landcover_plugin
```

4.16.5 LIS_tbot_plugin (Source File: LIS_param_pluginMod.F90)

This is a plugin point for introducing new bottom temperature datasets. The interface mandates that the following routines be implemented and registered for each parameter data source.

read the bottome temperature data Routines to retrieve the bottom temperature data (to be registered using `registerreadtbot` and later called using the generic `readtbot` method)

The user-defined functions are included in the registry using a single index. For example, consider the incorporation of the tbot datasets to be used in LIS running lat/lon projection. The methods should be defined in the registry as follows, if the index of the domain is defined to 1.

```
call registerreadtbot(1,read_statictbot)
```

The functions registered above are invoked using generic calls as follows:

```
call readtbot(1)  - calls read_statictbot
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call readtbot(lis%domain)
```

where *lis%domain* is set through the configuration utility, enabling the user make a selection at runtime.

INTERFACE:

```
subroutine LIS_tbot_plugin
```

4.16.6 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
```

4.16.7 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
```

4.17 Fortran: Module Interface LIS_perturb_pluginMod (Source File: LIS_perturb_pluginMo

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:

4.17.1 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
```

4.18 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

!-------------------------------------------------------------------------
! Run modes
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_retroId     = 1
   integer, public,  parameter :: LIS_agrmetrunId = 2
   integer, public,  parameter :: LIS_wrfcpl1Id    = 3
   integer, public,  parameter :: LIS_gcecplId    = 4
   integer, public,  parameter :: LIS_gfscplId    = 5
   integer, public,  parameter :: LIS_optrunId    = 6
   integer, public,  parameter :: LIS_rtmrunId    = 7
   integer, public,  parameter :: LIS_wrfcpl2Id    = 8
!   integer, public,  parameter :: LIS_uerunId    = 8

!-------------------------------------------------------------------------
! Domains
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_latlonId    = 1
   integer, public,  parameter :: LIS_mercId      = 2
   integer, public,  parameter :: LIS_lambertId   = 3
   integer, public,  parameter :: LIS_gaussId     = 4
   integer, public,  parameter :: LIS_polarId     = 5
   integer, public,  parameter :: LIS_afwaId      = 6
   integer, public,  parameter :: LIS_utmId       = 7
   integer, public,  parameter :: LIS_hrapId      = 8
   integer, public,  parameter :: LIS_catdomainId = 10
   integer, public,  parameter :: LIS_gswpdomId   = 11
!   integer, public,  parameter :: LIS_psglobalId  =
!-------------------------------------------------------------------------
! LSMS
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_templateLSMId = 0
   integer, public,  parameter :: LIS_noah271Id  = 1
   integer, public,  parameter :: LIS_clmId      = 2
   integer, public,  parameter :: LIS_vicId      = 3
   integer, public,  parameter :: LIS_mosaicId   = 4
```

```
   integer, public,  parameter :: LIS_hyssibId   = 5
   integer, public,  parameter :: LIS_sib2Id     = 6
   integer, public,  parameter :: LIS_catchId    = 7
   integer, public,  parameter :: LIS_sacId      = 8
   integer, public,  parameter :: LIS_snow17Id   = 9
   integer, public,  parameter :: LIS_sacsnow17Id= 10
   integer, public,  parameter :: LIS_sib3Id     = 11
   integer, public,  parameter :: LIS_mlbcId     = 12
   integer, public,  parameter :: LIS_csuId      = 13
   integer, public,  parameter :: LIS_tessId     = 14
   integer, public,  parameter :: LIS_placeId    = 15
   integer, public,  parameter :: LIS_noah31Id  = 21
!-------------------------------------------------------------------------
! base forcings
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_templateBaseForcId = 0
   integer, public,  parameter :: LIS_gdasId             = 1
   integer, public,  parameter :: LIS_geosId             = 2
   integer, public,  parameter :: LIS_ecmwfId            = 3
   integer, public,  parameter :: LIS_gswpId             = 5
   integer, public,  parameter :: LIS_ecmwfreanalId      = 6
   integer, public,  parameter :: LIS_agrmetId           = 7
   integer, public,  parameter :: LIS_princetonId        = 9
   integer, public,  parameter :: LIS_rhoneId            = 11
   integer, public,  parameter :: LIS_gswp1Id            = 12
   integer, public,  parameter :: LIS_gmaogldasId        = 13
   integer, public,  parameter :: LIS_gfsId              = 14
   integer, public,  parameter :: LIS_merraId             =15
!-------------------------------------------------------------------------
! supplemental forcings
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_agrradId      = 1
   integer, public,  parameter :: LIS_cmapId        = 2
   integer, public,  parameter :: LIS_agrradpsId    = 3
   integer, public,  parameter :: LIS_nldasId       = 4
   integer, public,  parameter :: LIS_nldas2Id      = 5
   integer, public,  parameter :: LIS_ceopId        = 6
   integer, public,  parameter :: LIS_scanId        = 7
   integer, public,  parameter :: LIS_saldasId      = 10
   integer, public,  parameter :: LIS_TRMM3B42RTId  = 11
   integer, public,  parameter :: LIS_TRMM3B42V6Id  = 12
   integer, public,  parameter :: LIS_cmorphId      = 14
   integer, public,  parameter :: LIS_stg2Id        = 15
   integer, public,  parameter :: LIS_stg4Id        = 16
   integer, public,  parameter :: LIS_d2pcpcarId    = 17
   integer, public,  parameter :: LIS_d2pcpoklId    = 18
   integer, public,  parameter :: LIS_gdas3dId      = 19
   integer, public,  parameter :: LIS_narrId        = 20
   integer, public,  parameter :: LIS_armsId        = 21
   integer, public,  parameter :: LIS_gdasLSWGId    = 22
   integer, public,  parameter :: LIS_capaId        = 23
!-------------------------------------------------------------------------
! land surface parameters
!-------------------------------------------------------------------------
!-------------------------------------------------------------------------
```

```
! Topography sources
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_gtopoId = 1
   integer, public,  parameter :: LIS_gfsTopoId = 3
!-------------------------------------------------------------------------
! Soils sources
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_faoSoilId     = 1
   integer, public,  parameter :: LIS_statsgoSoilId = 2
   integer, public,  parameter :: LIS_gfssoilId = 3
   integer, public,  parameter :: LIS_ecmwfsoilId = 4
!-------------------------------------------------------------------------
! LAI/SAI sources
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_avhrrlaiId  = 1
   integer, public,  parameter :: LIS_modislaiId  = 2
   integer, public,  parameter :: LIS_catlaiId    = 4
   integer, public,  parameter :: LIS_tiledavhrrlaiId  = 5
!-------------------------------------------------------------------------
! Landcover sources
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_umdlcId   = 1
   integer, public,  parameter :: LIS_usgslcId  = 2
   integer, public,  parameter :: LIS_gfslcId   = 3
   integer, public,  parameter :: LIS_igbplcId  = 4
!-------------------------------------------------------------------------
! greenness data sources
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_gfracClimoId = 1
   integer, public,  parameter :: LIS_gfraccatId   = 2
   integer, public,  parameter :: LIS_gfsgfracId = 3
   integer, public,  parameter :: LIS_NESDISgfracId =4
!-------------------------------------------------------------------------
! albedo data sources
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_albedoClimoId = 1
   integer, public,  parameter :: LIS_gfsalbId = 3
!-------------------------------------------------------------------------
! tbot data sources
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_nceptbotId = 1
!-------------------------------------------------------------------------
! snow data sources
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_snodepId = 1

!-------------------------------------------------------------------------
! data assimilation algorithms
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_diId         = 1
   integer, public,  parameter :: LIS_gmaoenkfId   = 2
!-------------------------------------------------------------------------
! perturbation algorithms
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_staticId   = 1
```

```
   integer, public,  parameter :: LIS_gmaopertId = 2

!-------------------------------------------------------------------------
! Assimilation set
! DA variable being updated with observations
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_synsm1Id     = 1
   integer, public,  parameter :: LIS_synsm2Id     = 2
   integer, public,  parameter :: LIS_synswe1Id    = 3
   integer, public,  parameter :: LIS_synsca1Id    = 4
   integer, public,  parameter :: LIS_synsm3Id  = 5
   integer, public,  parameter :: LIS_synsm4Id  = 6
   integer, public,  parameter :: LIS_synlst1Id = 8

   integer, public,  parameter :: LIS_isccpTskinId = 7
   integer, public,  parameter :: LIS_amsresmId    = 9
   integer, public,  parameter :: LIS_modisscaId   = 10
   integer, public,  parameter :: LIS_snodepobsId  = 11
   integer, public,  parameter :: LIS_multisynsmobsId = 12
   integer, public,  parameter :: LIS_NESDIS_AMSREsmobsId = 13
   integer, public,  parameter :: LIS_NASA_AMSREsmobsId = 14
   integer, public,  parameter :: LIS_ANSASWEsnowobsId = 15
   integer, public,  parameter :: LIS_ANSASCAsnowobsId = 16
   integer, public,  parameter :: LIS_ANSASNWDsnowobsId = 17
   integer, public,  parameter :: LIS_WindSatsmobsId = 18
!-------------------------------------------------------------------------
! Bias Estimation Algorithms
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_gmaobiasId = 1
!-------------------------------------------------------------------------
! Optimization /Uncertainty estimation Algorithms
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_ESOptId = 0
   integer, public,  parameter :: LIS_LMOptId = 1
   integer, public,  parameter :: LIS_GAOptId = 2
   integer, public,  parameter :: LIS_SCEUAOptId = 3
   integer, public,  parameter :: LIS_GLUEId = 4
   integer, public,  parameter :: LIS_MCMCId = 5
   integer, public,  parameter :: LIS_DREAMId = 6
!-------------------------------------------------------------------------
! Optimization Types
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_paramestId = 1
!-------------------------------------------------------------------------
! Optimization Set definitions
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_wgPBMRsm1Id = 1
   integer, public,  parameter :: LIS_wgPBMRsm2Id = 2
   integer, public,  parameter :: LIS_pesynsm1Id = 3
   integer, public,  parameter :: LIS_pesynsm2Id = 4
   integer, public,  parameter :: LIS_wgPBMRsm3Id = 5
!-------------------------------------------------------------------------
! Objective Function Evaluation Criteria
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_LMestimateId = 0
```

```
   integer, public,  parameter :: LIS_LSestimateId = 1
   integer, public,  parameter :: LIS_LLestimateId = 2

!-------------------------------------------------------------------------
! Objective Function Evaluation Criteria (for Uncertainty Estimation)
!-------------------------------------------------------------------------
   integer, public,  parameter :: LIS_UELSestimateId = 1

!-------------------------------------------------------------------------
! Radiative Transfer Models
!-------------------------------------------------------------------------
   integer, public, parameter :: LIS_crtmId = 1
```

4.19 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
```

4.19.1 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
```

4.20 Fortran: Module Interface LIS_suppforcing_pluginMod (Source File: LIS_suppforcing_p

This module contains the definition of the functions used for incorporating a new supplemental forcing scheme. The user defined functions are incorporated into the appropriate registry to be later invoked through generic calls.

REVISION HISTORY:

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

4.20.1 LIS_suppforcing_plugin (Source File: LIS_suppforcing_pluginMod.F90)

This is a plugin point for introducing a new supplemental forcing scheme. The interface mandates that the following interfaces be implemented and registered for each base forcing scheme.

retrieval of forcing data Routines to retrieve forcing data and to interpolate them. (to be registered using `registerreadsupp` and later invoked through `getsuppforc` method)

definition of native domain Routines to define the native domain (to be registered using `registerdefinenativesupp` and later invoked through `defineNativesupp` method)

temporal interpolation Interpolate forcing data temporally. (to be registered using `registersuppti` and later invoked through `timeinterpsupp` method)

Finalize Cleanup allocated structures (to be registered using `registersuppforcingfinal` and later invoked through `suppforcingfinalize` method)

The index used in the register calls should be used to select the appropriate base forcing scheme. For example, assume that the CMAP precipitation forcing scheme is incorporated in the registry by the following calls.

```
call registerreadsupp(1,getcmap)
call registerdefinenativesupp(1,defineNativeCMAP)
call registersuppti(1,time_interp_cmap)
call registersuppforcingfinal(1,cmapforcing_finalize)
```

The index used here to register these functions is 1. To invoke these methods, the corresponding calls will be:

```
call getsuppforc(1)  - calls getcmap
call defineNativesupp(1)  - calls defineNativeCMAP
call timeinterpsupp(1)  - calls time_interp_cmap
call suppforcingfinalize(1) - calls cmapforcing_finalize
```

In the LIS code, the above calls are typically invoked in the following manner.

```
call getsuppforc(lis%suppforce)
call defineNativeSupp(lis%suppforce)
call timeinterpsupp(lis%suppforce)
call suppforcingfinalize(lis%suppforce)
```

where $lis\%suppforce$ is set through the congiguration utility, enabling the user to select any of the base forcing schemes, at runtime. INTERFACE:

```
subroutine LIS_suppforcing_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,ibi,li,gi,ibo,lo,go,mi,mo, &
      rlat,rlon,w11,w12,w21,w22,n11,n12,n21,n22,udef,iret)
```

USES:

```
   implicit none
```

ARGUMENTS:

```
   real      :: gridDesco(50)
   integer   :: ibi
   integer   :: ibo
   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

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

5.1.2 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 (5.1.16)
Computes the grid coordinates of the input grid, based on the earth coordinates of the output grid.

get_field_pos (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)

  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 (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)

The routines invoked are:

compute_earth_coord_latlon (5.1.9)
computes the earth coordinates of a latlon grid

compute_earth_coord_merc (5.1.13)
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 (5.1.14)
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_latlon_ur (Source File: compute_earth_coord_latlon_ur.F90)

REVISION HISTORY:

```
20 Sept 08 Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine compute_earth_coord_latlon_ur(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 upper right 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

5.1.12 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

5.1.13 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

5.1.14 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

5.1.15 compute_earth_coord_ur (Source File: compute_earth_coord_ur.F90)

REVISION HISTORY:

```
20 Sept 08 Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine compute_earth_coord_ur(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 upper right 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_ur (5.1.11)
computes the upper right earth coordinates of a latlon grid

5.1.16 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 (5.1.21
computes the grid coordinates of a latlon grid

compute_grid_coord_merc (5.1.22
computes the grid coordinates of a mercator grid

compute_grid_coord_lambert (5.1.20
computes the grid coordinates of a lambert conformal grid

compute_grid_coord_gauss (5.1.18
computes the grid coordinates of a gaussian cylindrical grid

compute_grid_coord_polar (5.1.23
computes the grid coordinates of a polar stereographic grid

5.1.17 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

5.1.18 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

5.1.19 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
```

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.

5.1.20 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.

5.1.21 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

5.1.22 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.

5.1.23 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.

5.1.24 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

5.1.25 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,ibi,li,gi,ibo,lo,go,mi,mo,&
     rlat,rlon,w11,w12,w21,w22,n11,n12,n21,n22,udef,iret)
  implicit none
```

ARGUMENTS:

```
integer, parameter :: nb3 = 25, nb4 = 25
real      :: gridDesco(50)
integer   :: ibi
integer   :: ibo
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

5.1.26 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 (5.1.16
Computes the grid coordinates of the input grid, based on the earth coordinates of the output grid.

get_field_pos (5.2.6)
computes the field position for a given point

5.1.27 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.

5.1.28 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)
```

5.1.29 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,pointer,dimension(:)   :: gaussian_lat_array
real,pointer,dimension(:)   :: gaussian_lon_array
 real,pointer,dimension(:,:) :: gaussian_latitudes
 real,pointer,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_field_pos (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 pro- jection 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 pro- jection 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 -¿ 90
- Any arguments that are a longitude value are expressed in degrees east with a valid range of -180 -¿ 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)
  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
```

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)
  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
```

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,ibi,li,gi,ibo,lo,go,mi,mo, &
     rlat,rlon,n11,udef, iret)
  implicit none
```

ARGUMENTS:

```
  real      :: gridDesco(50)
  integer   :: ibi
  integer   :: ibo
  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 (5.1.16)
Computes the grid coordinates of the input grid, based on the earth coordinates of the output grid.

get_field_pos (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 upscaleByAveraging (Source File: upscaleByAveraging.F90)

INTERFACE:

```
subroutine upscaleByAveraging(nc,nr,mi1,mi2,&
     stc,str,enc,enr, udef, li, gi, lo, go)

  implicit none
```

ARGUMENTS:

```
  integer             :: nc
  integer             :: nr
  integer             :: mi1
  integer             :: mi2
  integer             :: stc(nc,nr)
  integer             :: enc(nc,nr)
  integer             :: str(nc,nr)
  integer             :: enr(nc,nr)
  real                :: udef
  logical*1           :: li(mi1,mi2)
  real                :: gi(mi1,mi2)
  logical*1           :: lo(nc,nr)
  real                :: go(nc,nr)
```

DESCRIPTION:

This subprogram upscales scalar data from a finer grid to a coarser grid. 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:

nc number of points along the east-west direction in the output grid (coarser grid)

nr number of points along the north-south direction in the output grid (coarser grid)

mi1 number of points along the east-west direction in the input grid (finer grid)

mi2 number of points along the north-south direction in the input grid (finer 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

udef undefine value

li input bit mask

gi input data (finer grid)

lo output bit mask

go output data (coarser grid)

5.3.17 upscaleByAveraging_input (Source File: upscaleByAveraging_input.F90)

INTERFACE:

```
subroutine upscaleByAveraging_input (gridDesci,gridDesco,mi, nc,nr,&
     stc,str,enc,enr)

  implicit none
```

ARGUMENTS:

```
  real, intent(in)    :: gridDesci(50)
  real                :: gridDesco(50)
  integer             :: mi
  integer             :: nc
  integer             :: nr
  integer             :: stc(nc,nr)
  integer             :: enc(nc,nr)
  integer             :: str(nc,nr)
  integer             :: enr(nc,nr)
```

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 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

nc number of points along the east-west direction in the output grid

nr number of points along the north-south direction in the output 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

---

5.3.18 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 (5.3.20)
compute the LIS_localtime based on the GMT

coszenith (5.3.19)
compute the cosine of zenith angle

5.3.19 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)

  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)
```

5.3.20 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

Supplemental forcing products

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 GCE Coupled Running Mode

This section describes the GCE coupled running mode.

8.1 Fortran: Module Interface lisgceexport_module.F90 (Source File: lisgceexport_module.F90)

Defines the export state variables into LIS Variable names conform to the ALMA convention

REVISION HISTORY:

```
09 Nov 2005: Sujay Kumar; Initial specification
```

INTERFACE:

```
module lisgceexport_module
  implicit none
  public lisgceexport
```

ARGUMENTS:

```
  type lisgceexport
     real*8, pointer :: swnet(:,:)
     real*8, pointer :: lwnet(:,:)
     real*8, pointer :: qle(:,:)
     real*8, pointer :: qh(:,:)
     real*8, pointer :: qg(:,:)
     real*8, pointer :: avgsurft(:,:)
     real*8, pointer :: albedo(:,:)
     real*8, pointer :: tauu(:,:)
     real*8, pointer :: tauv(:,:)
     real*8, pointer :: tau(:,:)
  end type lisgceexport
```

8.2 Fortran: Module Interface lisgceimport_module.F90 (Source File: lisgceimport_module.F90)

Defines the import state variables into LIS

REVISION HISTORY:

```
09 Nov 2005: Sujay Kumar; Initial specification
```

INTERFACE:

```
module lisgceimport_module
  implicit none
  public lisgceimport
```

ARGUMENTS:

```
type lisgceimport
Time information
   integer       :: yr
   integer       :: mo
   integer       :: da
   integer       :: hr
   integer       :: mn
   integer       :: ss
   real*8, pointer :: data_tair(:,:)
   real*8, pointer :: data_qair(:,:)
   real*8, pointer :: data_swd(:,:)
   real*8, pointer :: data_lwd(:,:)
   real*8, pointer :: data_psurf(:,:)
   real*8, pointer :: data_prcp(:,:)
   real*8, pointer :: data_u(:,:)
   real*8, pointer :: data_v(:,:)
end type lisgceimport
```

---

9 optUE Running Mode

This section describes the optUE running mode.

9.1 Fortran: Module Interface optUE_runMod (Source File: optUE_runMod.F90)

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

implicit none

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

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
 public :: lis_init_optUE  !init method for optimization-uncertainty modeling mode
 public :: lis_run_optUE   !run method for optimization-uncertainty modeling mode
 public :: lis_final_optUE !finalize method for optimization-uncertainty modeling mode
```

9.1.1 lis_init_optUE (Source File: optUE_runMod.F90)

INTERFACE:

```
subroutine lis_init_optUE
```

USES:

```
use LIS_domainMod,       only : LIS_domain_init
use LIS_lsmMod,          only : LIS_lsm_init, LIS_setuplsm, &
     LIS_readrestart
use LIS_baseforcingMod,  only : LIS_baseforcing_init
use LIS_suppforcingMod,  only : LIS_suppforcing_init
use LIS_optUEMod, only : LIS_optUE_init, &
     LIS_optUE_setup
use LIS_paramsMod,        only : LIS_param_init
```

DESCRIPTION:

This is the initialize method for LIS in a optUE running mode. The following calls are invoked from this method.

LIS_domain_init (3.11.2)
initialize the LIS domains

LIS_param_init (3.24.2)
initialize parameters

LIS_lsm_init (3.20.3)
initialize the land surface model.

LIS_baseforcing_init (3.7.2)
initialize the base forcing

LIS_suppforcing_init (3.30.2)
initialize the supplemental forcing

LIS_initObservations (**??**)
initialize structures needed to read observations for data assimilation

LIS_setuplsm (3.20.4)
complete the LSM setups

LIS_readrestart (3.20.7)
read the restart files

9.1.2 lis_run_optUE (Source File: optUE_runMod.F90)

INTERFACE:

```
subroutine lis_run_optUE
```

USES:

```
use LIS_coreMod,         only : LIS_rc
use LIS_optUEMod, only : LIS_isOptStopCriterionTrue,&
     LIS_runOptUE
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.

9.1.3 lis_final_optUE (Source File: optUE_runMod.F90)

INTERFACE:

```
subroutine lis_final_optUE
```

USES:

```
use LIS_coreMod,       only : lis_finalize
use LIS_lsmMod,          only : LIS_lsm_finalize
use LIS_paramsMod,        only : LIS_param_finalize
use LIS_baseforcingMod,  only : LIS_baseforcing_finalize
use LIS_suppforcingMod,  only : LIS_suppforcing_finalize
```

DESCRIPTION:

This is the finalize method for LIS in a optUE running mode. The following calls are invoked from this method.

LIS_finalize (3.9.10)
cleanup LIS generic structures

LIS_lsm_finalize (3.20.14)
cleanup land surface model specific structures

LIS_param_finalize (3.24.5)
cleanup parameter specific structures

LIS_baseforcing_finalize (3.7.9)
cleanup baseforcing specific structures

LIS_suppforcing_finalize (3.30.4)
cleanup supplemental forcing specific structures

LIS_dataassim_finalize (3.10.5)
cleanup data assimilation specific structures

---

10 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.

10.1 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
```

10.1.1 LIS_init_retrospective (Source File: retrospective_runMod.F90)

INTERFACE:

```
subroutine LIS_init_retrospective
```

USES:

```
use LIS_domainMod,       only : LIS_domain_init
use LIS_lsmMod,          only : LIS_lsm_init, LIS_setuplsm, &
     LIS_readrestart
use LIS_baseforcingMod,  only : LIS_baseforcing_init
use LIS_suppforcingMod,  only : LIS_suppforcing_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
```

DESCRIPTION:

This is the initialize method for LIS in a retrospective running mode. The following calls are invoked from this method.

LIS_domain_init (3.11.2)
initialize the LIS domains

LIS_param_init (3.24.2)
initialize parameters

LIS_lsm_init (3.20.3)
initialize the land surface model.

LIS_baseforcing_init (3.7.2)
initialize the base forcing

LIS_suppforcing_init (3.30.2)
initialize the supplemental forcing

LIS_initDAObservations (**??**)
initialize structures needed to read observations for data assimilation

LIS_setuplsm (3.20.4)
complete the LSM setups

LIS_readrestart (3.20.7)
read the restart files

10.1.2 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_lsmMod,          only : LIS_force2tile, LIS_perturb_states, &
     LIS_lsm_main,LIS_lsm_output, LIS_writerestart
use LIS_paramsMod,        only : LIS_setDynparams
use LIS_baseforcingMod,  only : LIS_get_base_forcing, LIS_perturb_forcing
use LIS_suppforcingMod,  only : LIS_get_supp_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_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 (3.9.7)
check to see if the end of simulation has reached

LIS_ticktime (3.9.6)
advance model clock

LIS_timeToRunNest (3.9.8)
check to see if the current nest needs to be run.

LIS_setDynparams (3.24.3)
set the time dependent parameters

LIS_get_base_forcing (3.7.7)
retrieve the base forcing

LIS_get_supp_forcing (3.30.3)
retrieve the supplemental forcing

LIS_force2tile (3.20.10)
transfer forcing to model tiles

LIS_readDAobservations (3.1.2)
read observations to be used for data assimilation

LIS_lsm_main (3.20.5)
run the land surface model

LIS_lsm_output (3.20.8)
write model output

LIS_writerestart (3.20.11)
write model restart files

10.1.3 lis_final_retrospective (Source File: retrospective_runMod.F90)

INTERFACE:

```
subroutine lis_final_retrospective
```

USES:

```
use LIS_coreMod,         only : LIS_finalize
use LIS_lsmMod,          only : LIS_lsm_finalize
use LIS_paramsMod,        only : LIS_param_finalize
use LIS_baseforcingMod,  only : LIS_baseforcing_finalize
use LIS_suppforcingMod,  only : LIS_suppforcing_finalize
```

DESCRIPTION:

This is the finalize method for LIS in a retrospective running mode. The following calls are invoked from this method.

LIS_finalize (3.9.10)
cleanup LIS generic structures

LIS_lsm_finalize (3.20.14)
cleanup land surface model specific structures

LIS_param_finalize (3.24.5)
cleanup parameter specific structures

LIS_baseforcing_finalize (3.7.9)
cleanup baseforcing specific structures

LIS_suppforcing_finalize (3.30.4)
cleanup supplemental forcing specific structures

LIS_dataassim_finalize (3.10.5)
cleanup data assimilation specific structures

---

11 WRF Coupled ESMF Running Mode

This section describes the WRF coupled ESMF running mode.

11.1 Fortran: Module Interface LIS_WRF_interfaceMod (Source File: LIS_WRF_interfaceM

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 ESMF_Mod
use LIS_logMod,  only : LIS_verify

implicit none
PRIVATE

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

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: LIS_init_ImpExpFields
public :: LIS_translateImportState
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: LISgrid
```

11.1.1 LIS_init_ImpExpFields (Source File: LIS_WRF_interfaceMod.F90)

INTERFACE:

```
subroutine LIS_init_ImpExpFields(importState, exportState)
```

USES:

```
    use LIS_coreMod, only : LIS_rc, LIS_npes, &
         LIS_nss_ind, LIS_nse_ind, LIS_ews_ind, LIS_ewe_ind
    use LIS_baseforcingMod, only : LIS_FORC_State
    use LIS_logMod,  only : LIS_verify

    implicit none
```

ARGUMENTS:

```
    type(ESMF_State)     :: importState
    type(ESMF_State)     :: exportState
```

DESCRIPTION:

This routine creates the placeholders for the import and export fields for the WRF-LIS coupling. The import fields (forcing) fields are created during the standard LIS initialization
The vecGrid is created on the reduced grid that excludes water points. When coupled to WRF, however, the water points are included. Therefore, the LIS_vecGrid object represents a vectorized representation of the 2d grid. We are creating a dummy 2d just to be sure - since the LIS fields are actually 1d

---

11.1.2 LIS_translateImportState (Source File: LIS_WRF_interfaceMod.F90)

INTERFACE:

```
subroutine LIS_translateImportState(n, LISimp)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_baseforcingMod, only : LIS_FORC_State, diagnoseForcingOutput
use LIS_FORC_AttributesMod

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: LISimp
```

DESCRIPTION:

This subroutine extracts fields from WRF export (LIS import) and maps them to the LIS Forcing State objects.

11.2 Fortran: Module Interface module_lis_component (Source File: module_lis_component.

This module encapsulates the LIS system as an ESMF gridded component. It defines the sequence of calls (init, run and finalize steps) when LIS is used coupled to WRF. The LIS initialization step creates the placeholders for generating the import and export states, in addition to standard initialization steps. The LIS run step accepts a WRF export state and translates them into the forcing datastructures. Once the land surface model is run, LIS sends an export state back to WRF.

REVISION HISTORY:

```
 9 Nov 2008    Sujay Kumar  Initial Specification

use ESMF_Mod

implicit none

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

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public LIS_register
```

11.2.1 LIS_register (Source File: module_lis_component.F90)

INTERFACE:

```
SUBROUTINE LIS_register(gcomp, rc)
```

ARGUMENTS:

```
TYPE(ESMF_GridComp), INTENT(INOUT) :: gcomp
INTEGER, INTENT(OUT)   :: rc
INTEGER :: finalrc
```

DESCRIPTION:

LIS_register - Externally visible registration routine
User-supplied SetServices routine for the LIS gridded component. The Register routine sets the subroutines to be called as the init, run, and finalize routines. Note that these are private to the module.
The arguments are: gcomp Component rc Return code; equals ESMF_SUCCESS if there are no errors, otherwise ESMF_FAILURE.

11.2.2 LIS_component_init1 (Source File: module_lis_component.F90)

INTERFACE:

```
subroutine LIS_component_init1(gcomp, importState, exportState, clock, rc)

  implicit none
```

ARGUMENTS:

```
TYPE(ESMF_GridComp)   :: gcomp
TYPE(ESMF_State)      :: importState
TYPE(ESMF_State)      :: exportState
TYPE(ESMF_Clock)      :: clock
INTEGER, INTENT(OUT)  :: rc
```

DESCRIPTION:

LIS component init routine, phase 1. First-phase "init" returns "time" metadata in exportState.
The arguments are: gcomp Component importState Importstate exportState Exportstate clock External clock rc Return code; equals ESMF_SUCCESS if there are no errors, otherwise ESMF_FAILURE.

11.2.3 LIS_component_init2 (Source File: module_lis_component.F90)

INTERFACE:

```
subroutine LIS_component_init2(gcomp, importState, exportState, clock, rc )
```

USES:

```
use LIS_coreMod, only : LIS_config_init
use LIS_logMod,  only : LIS_verify, LIS_logunit, LIS_flush
use LIS_domainMod, only : LIS_domain_init
use LIS_lsmMod,    only : LIS_lsm_init, LIS_setuplsm, LIS_readrestart
use LIS_baseforcingMod, only : LIS_baseforcing_init
use LIS_DAobservationsMod, only : LIS_initDAobservations
use LIS_perturbMod,  only : LIS_perturb_init
use LIS_dataAssimMod, only : LIS_dataassim_init
use LIS_paramsMod, only : LIS_param_init
use LIS_WRF_interfaceMod, only : LIS_init_ImpExpFields
```

ARGUMENTS:

```
TYPE(ESMF_GridComp)  :: gcomp
TYPE(ESMF_State)     :: importState
TYPE(ESMF_State)     :: exportState
TYPE(ESMF_Clock)     :: clock
INTEGER, INTENT(OUT) :: rc
```

DESCRIPTION:

LIS component init routine, phase 2. This phase completes the LIS initialization steps and creates placeholders for import and export states.
The arguments are: gcomp Component importState Importstate exportState Exportstate clock External clock rc Return code; equals ESMF_SUCCESS if there are no errors, otherwise ESMF_FAILURE.

11.2.4 LIS_component_run (Source File: module_lis_component.F90)

INTERFACE:

```
subroutine LIS_component_run(gcomp, importState, exportState, clock, rc )
```

USES:

```
use LIS_coreMod,          only : LIS_ticktime
use LIS_WRF_interfaceMod, only : LIS_translateImportState
use LIS_lsmMod,          only : LIS_force2tile, LIS_perturb_states, &
     LIS_lsm_main,LIS_lsm_output, LIS_writerestart, LIS_lsm_setexport
use LIS_baseforcingMod,  only : LIS_FORC_State, LIS_perturb_forcing
use LIS_paramsMod,        only : LIS_setDynparams
use LIS_logMod,          only : lis_flush, LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

```
TYPE(ESMF_GridComp)  :: gcomp
TYPE(ESMF_State)     :: importState
TYPE(ESMF_State)     :: exportState
TYPE(ESMF_Clock)     :: clock
INTEGER, INTENT(OUT) :: rc
```

DESCRIPTION:

LIS component run routine. This phase translates the LIS import states and uses them to force the land surface model. The LSM outputs are packaged into the LIS export state.
The arguments are: gcomp Component importState Importstate exportState Exportstate clock External clock rc Return code; equals ESMF_SUCCESS if there are no errors, otherwise ESMF_FAILURE.

11.2.5 LIS_component_final (Source File: module_lis_component.F90)

INTERFACE:

```
subroutine LIS_component_final(gcomp, importState, exportState, clock, rc )
```

USES:

```
use LIS_coreMod,         only : LIS_finalize
use LIS_lsmMod,          only : LIS_lsm_finalize
use LIS_paramsMod,        only : LIS_param_finalize
use LIS_baseforcingMod,  only : LIS_baseforcing_finalize
```

ARGUMENTS:

```
TYPE(ESMF_GridComp)  :: gcomp
TYPE(ESMF_State)     :: importState
TYPE(ESMF_State)     :: exportState
TYPE(ESMF_Clock)     :: clock
INTEGER, INTENT(OUT) ::  rc
```

DESCRIPTION:

LIS component finalize routine
The arguments are: gcomp Component importState Importstate exportState Exportstate clock External clock rc Return code; equals ESMF_SUCCESS if there are no errors, otherwise ESMF_FAILURE.

---

12 WRF Coupled No ESMF Running Mode

This section describes the WRF coupled no ESMF running mode.

12.1 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
```

12.2 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:

12.3 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:

12.3.1 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,         only : LIS_initialized, LIS_config_init
use LIS_domainMod,       only : LIS_domain_init
use LIS_lsmMod,          only : LIS_lsm_init, LIS_setuplsm, &
     LIS_readrestart
use LIS_baseforcingMod,  only : LIS_baseforcing_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
  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.

12.3.2 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
```

USES:

```
use ESMF_Mod
use LISWRFGridCompMod, only : LISWRF_import, LISWRF_export
use LIS_coreMod,         only : LIS_rc, LIS_domain, LIS_endofrun, LIS_timetoRunNest, &
     LIS_ticktime
use LIS_baseforcingMod,  only : LIS_perturb_forcing
use LIS_timeMgrMod,      only : LIS_timemgr_set
use LIS_lsmMod,          only : LIS_force2tile, LIS_perturb_states, &
     LIS_lsm_main,LIS_lsm_output, LIS_writerestart, LIS_lsm_setexport
use LIS_paramsMod,        only : LIS_setDynparams
use LIS_baseforcingMod,  only : LIS_FORC_State
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_logMod,          only : LIS_verify, LIS_flush, LIS_logunit
use LIS_FORC_AttributesMod
```

Part VIII
Domain implementations in LIS

13 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.

---

14 UTM domain

This section contains the interface implementations for a LIS instance using a UTM projection.

14.0.3 createtiles_UTM (Source File: createtiles_UTM.F90)

REVISION HISTORY:

```
28 Jun 09, Sujay Kumar : Intial Specification
```

INTERFACE:

```
subroutine createtiles_UTM()
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_masterproc,  &
     LIS_npes, LIS_ews_ind, LIS_ewe_ind, LIS_nss_ind, LIS_nse_ind, &
     LIS_deltas, LIS_offsets
use LIS_mpiMod
use LIS_logMod, only : LIS_logunit, LIS_verify
use LIS_domainMod, only : LIS_domain_setup
use LIS_LMLCMod, only : LIS_LMLC
use LIS_topoMod, only : LIS_topo
```

DESCRIPTION:

The code in this file implements the grid and tile spaces for the UTM domain. The routine first reads the landmask and the landcover data. The size of the grid space is determined by ignoring the water points. The UTM domain does not support subgrid tiling. As a result, the grid and tile spaces are equivalent.

14.0.4 readinput_UTM (Source File: readinput_UTM.F90)

REVISION HISTORY:

```
28 Jun 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine readinput_UTM
```

USES:

```
use ESMF_Mod
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 (3.42.54)
routine to perform domain decomposition

---

15 Catchment domain

This section contains the interface implementations for a LIS instance using a Catchment-based domain.

15.0.5 createtiles_catch (Source File: createtiles_catch.F90)

REVISION HISTORY:

```
10 Jul 2006: James Geiger; Initial version
```

INTERFACE:

```
subroutine createtiles_catch
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_localPet, LIS_masterproc,  &
     LIS_npes, LIS_ews_ind, LIS_ewe_ind, LIS_nss_ind, LIS_nse_ind, &
     LIS_deltas, LIS_offsets
use LIS_mpiMod
use LIS_logMod, only :LIS_logunit, LIS_verify, LIS_endrun
use LIS_domainMod, only : LIS_domain_setup
use tile_coord_types
use tile_coord_routines, only : read_tile_coord, read_tile_coord_header

implicit none
```

ARGUMENTS:

DESCRIPTION:

This routine computes the grid and tile spaces based on the input distribution of the catchment-based tiles. The routine also computes the following structures necessary for domain decomposition and other parallel routines.
`ntiles_pergrid` : Number of tiles per each grid
`s_tid` : Index of the first tile in each grid cell

15.0.6 readinput_catch (Source File: readinput_catch.F90)

REVISION HISTORY:

```
14 Nov 2003: Sujay Kumar; Initial Code
```

INTERFACE:

```
subroutine readinput_catch
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, 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_logMod, only :LIS_logunit, LIS_endrun
use LIS_domainMod, only : LIS_quilt_domain, LIS_setParameterDomainSpecs
use LIS_fileIOMod, only : LIS_readDomainConfigSpecs
```

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 (3.42.54)
routine to perform domain decomposition

15.1 Fortran: Module Interface tile_corrd_routines (Source File: tile_coord_routines.F90)

This file contains types and subroutines for tile coordinates and domain.

REVISION HISTORY:

```
26 Jan 2005 Rolf Reichle, Initial Specification
14 Apr 2006 Rolf Reichle, split tile_coord.F90 into 2 files to avoid
                          having more than one module per file
10 Jul 2006 James Geiger, Implementation in LIS
```

USES:

```
use tile_coord_types
```

15.1.1 read_tile_coord (Source File: tile_coord_routines.F90)

REVISION HISTORY:

```
              Rolf Reichle, Initial Specification
10 Jul 2006 James Geiger, Implementation in LIS
```

INTERFACE:

```
subroutine read_tile_coord_header( ftn, tile_coord_file, atm_grid, N_tile )


  implicit none
```

ARGUMENTS:

```
  integer,             intent(IN)            :: ftn
  character(len=*),    intent(in)            :: tile_coord_file
  type(grid_def_type), intent(out), optional :: atm_grid

  integer,             intent(OUT)           :: N_tile
```

DESCRIPTION:

read tile coordinates from GEOS5 *.til file optionally assemble atmospheric grid parameters
inputs:
tile_coord_file : name with full path of *.til file
outputs:
atm_grid : atmospheric grid parameters
tile_coord : coordinates of tiles (see tile_coord_type), implemented as pointer which is allocated in this subroutine NOTE: number of tiles can be diagnosed with size(tile_coord)

15.1.2 read_tile_coord (Source File: tile_coord_routines.F90)

REVISION HISTORY:

```
              Rolf Reichle, Initial Specification
10 Jul 2006 James Geiger, Implementation in LIS
```

INTERFACE:

```
subroutine read_tile_coord( ftn, N_tile, tile_coord )


  implicit none
```

ARGUMENTS:

```
  integer                 :: ftn
  integer                 :: N_tile
  type(tile_coord_type)   :: tile_coord(N_tile)
```

DESCRIPTION:

read tile coordinates from GEOS5 *.til file optionally assemble atmospheric grid parameters
inputs:
tile_coord_file : name with full path of *.til file
outputs:
atm_grid : atmospheric grid parameters
tile_coord : coordinates of tiles (see tile_coord_type), implemented as pointer which is allocated in this subroutine NOTE: number of tiles can be diagnosed with size(tile_coord)

15.1.3 get_tile_num_in_atm_ij (Source File: tile_coord_routines.F90)

REVISION HISTORY:

```
22 Jul 2005 Rolf Reichle, Initial Specification
10 Jul 2006 James Geiger, Implementation in LIS
```

INTERFACE:

```
subroutine get_tile_num_in_atm_ij( N_tile, tile_coord, atm_grid, &
     d2g_grid_off_i, d2g_grid_off_j,                             &
     N_tile_in_atm_ij, tile_num_in_atm_ij )


  implicit none
```

ARGUMENTS:

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

  type(tile_coord_type), dimension(:), pointer :: tile_coord ! input

  type(grid_def_type), intent(in) :: atm_grid

  integer, intent(in) :: d2g_grid_off_i, d2g_grid_off_j

  integer, dimension(atm_grid%N_lon,atm_grid%N_lat), intent(inout) :: &
       N_tile_in_atm_ij

  ! the pointer is an output arguments that is allocated here

  integer, dimension(:,:,:), pointer, optional :: tile_num_in_atm_ij
```

DESCRIPTION:

find out how many tiles are in a given atm grid cell
When called without optional arguments only counts number of tiles. When called with optional arguments allocates and fills pointer.
d2g_grid_off_i, d2g_grid_off_j : tile_coordatmospheric grid that corresponds to the tile definitions (as obtained from the tile_coord_file) d2g_grid_off_i and d2g_grid_off_j describe the offset between the global "atm_grid_g" and a smaller "atm_grid_d" for the domain of interest. With these offset arguments the subroutine can be used with an atm_grid that is a subgrid of the global atm_grid: atm_grid_datm_grid_d

---

15.1.4 extract_land_tiles (Source File: tile_coord_routines.F90)

REVISION HISTORY:

```
28 Jan 2005 Rolf Reichle, Initial Specification
22 Jul 2005 Rolf Reichle
10 Jul 2006 James Geiger, Implementation in LIS
```

INTERFACE:

```
subroutine extract_land_tiles( N_tile_global, tile_coord_global, &
     N_tile_land, tile_coord_land )

  implicit none
```

ARGUMENTS:

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

type(tile_coord_type), dimension(:), pointer :: tile_coord_global ! input

integer, intent(inout) :: N_tile_land

! the pointer is an output arguments that is allocated here

type(tile_coord_type), dimension(:), pointer, optional :: tile_coord_land
```

DESCRIPTION:

extract land tiles from tile_coord_global
When called without optional arguments only counts number of tiles. When called with optional arguments allocates and fills pointer.

15.1.5 get_land_tile_info (Source File: tile_coord_routines.F90)

REVISION HISTORY:

```
28 Jan 2005 Rolf Reichle, Initial Specification
10 Jul 2006 James Geiger, Implementation in LIS
```

INTERFACE:

```
subroutine get_land_tile_info( tile_coord_file, catchment_def_file, &
     atm_grid,                                                      &
     N_tile, tile_coord )


  implicit none
```

ARGUMENTS:

```
character(300), intent(in) :: tile_coord_file, catchment_def_file

type(grid_def_type), intent(out) :: atm_grid

integer, intent(out) :: N_tile

type(tile_coord_type), dimension(:), pointer :: tile_coord ! out
```

DESCRIPTION:

get land tile coordinates, atmospheric grid parameters, and grid-to-tile mapping from GEOS5 *.til file
note use of optional inout arguments

15.1.6 read_catchment_def (Source File: tile_coord_routines.F90)

REVISION HISTORY:

```
              Rolf Reichle, Initial Specification
  10 Jul 2006 James Geiger, Implementation in LIS
```

INTERFACE:

```
  subroutine read_catchment_def( catchment_def_file, N_tile, tile_coord )

    implicit none
```

ARGUMENTS:

```
    character(300), intent(in) :: catchment_def_file

    integer, intent(in) :: N_tile

    type(tile_coord_type), dimension(:), pointer :: tile_coord ! inout
```

DESCRIPTION:

This routine reads the catchment def file.

15.1.7 get_tile_num_from_latlon (Source File: tile_coord_routines.F90)

REVISION HISTORY:

```
              Rolf Reichle, Initial Specification
  17 Nov 2005 Rolf Reichle, bug fix re. "check that lat/lon is inside atm_grid"
  10 Jul 2006 James Geiger, Implementation in LIS
```

INTERFACE:

```
  subroutine get_tile_num_from_latlon(N_catd, tile_coord,           &
       atm_grid, N_tile_in_atm_ij, tile_num_in_atm_ij, N_latlon, lat, lon,  &
       tile_num)

    implicit none
```

ARGUMENTS:

```
    integer, intent(in) :: N_catd, N_latlon

    type(tile_coord_type), dimension(:), pointer :: tile_coord ! input

    type(grid_def_type), intent(in) :: atm_grid

    integer, dimension(atm_grid%N_lon,atm_grid%N_lat), intent(in) :: &
         N_tile_in_atm_ij

    integer, dimension(:,:,:), pointer :: tile_num_in_atm_ij ! input

    real, dimension(N_latlon), intent(in)  :: lat, lon

    integer, dimension(N_latlon), intent(out) :: tile_num
```

DESCRIPTION:

This routine determines the tile number of a tile specified by its lat/lon.

15.1.8 is_in_box (Source File: tile_coord_routines.F90)

REVISION HISTORY:

```
              Rolf Reichle, Initial Specification
  10 Jul 2006 James Geiger, Implementation in LIS
  determine whether catchment is within bouding box - reichle, 7 May 2003
```

INTERFACE:

```
  logical function is_in_box(                                  &
       this_minlon, this_minlat, this_maxlon, this_maxlat,     &
       minlon, minlat, maxlon, maxlat        )
```

ARGUMENTS:

```
    implicit none

    real :: this_minlon, this_minlat, this_maxlon, this_maxlat
    real :: minlon, minlat, maxlon, maxlat
```

DESCRIPTION:

This routine determines whether a given box is contained within another box.

15.1.9 get_ij_atm_from_latlon (Source File: tile_coord_routines.F90)

REVISION HISTORY:

```
              Rolf Reichle, Initial Specification
  10 Jul 2006 James Geiger, Implementation in LIS
```

INTERFACE:

```
  subroutine get_ij_atm_from_latlon( atm_grid, lat, lon, i_atm, j_atm )


    implicit none
```

ARGUMENTS:

```
    type(grid_def_type), intent(in) :: atm_grid

    real, intent(in) :: lat, lon

    integer, intent(out) :: i_atm, j_atm
```

DESCRIPTION:

This routine determines the column and row indices of the atmospheric grid-cell corresponding to the give lat/lon.
NOTE: latlon is *outside* atm_grid if (i_atm¡0 .or. i_atm¿atm_gridj_atm¡0 .or. j_atm¿atm_grid

---

15.1.10 tile2grid (Source File: tile_coord_routines.F90)

REVISION HISTORY:

```
              Rolf Reichle, Initial Specification
10 Jul 2006 James Geiger, Implementation in LIS
reichle, 28 Jan 2005
reichle, 16 May 2005 - added offset for "domain" grid
```

INTERFACE:

```
subroutine tile2grid( N_tile, tile_coord, atm_grid, tile_data,    &
     grid_data, no_data_value, no_data_tol, echo,                 &
     d2g_grid_off_i, d2g_grid_off_j                            )

  implicit none
```

ARGUMENTS:

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

  type(tile_coord_type), dimension(:), pointer :: tile_coord  ! input

  type(grid_def_type), intent(in) :: atm_grid

  real, dimension(N_tile), intent(in) :: tile_data

  real, dimension(atm_grid%N_lon,atm_grid%N_lat), intent(out) :: grid_data

  real, intent(in), optional :: no_data_value, no_data_tol

  logical, intent(in), optional :: echo

  integer, intent(in), optional :: d2g_grid_off_i, d2g_grid_off_j
```

DESCRIPTION:

map from tile space to atmospheric grid
NOTE: tile_coord must match tile_data
optional inputs: no_data_value : no_data_tol : tolerance when checking tile_data against no_data_value echo : echo no_data_value and tolerance d2g_grid_off_i, d2g_grid_off_j : tile_coordatmospheric grid that corresponds to the tile definitions (as obtained from the tile_coord_file) d2g_grid_off_i and d2g_grid_off_j describe the offset between the global "atm_grid_g" and a smaller "atm_grid_d" for the domain of interest. With these offset arguments tile2grid can be used together with an argument atm_grid=atm_grid_d that is a subgrid of the global atm_grid: atm_grid_datm_grid_d

---

15.1.11 grid2tile (Source File: tile_coord_routines.F90)

REVISION HISTORY:

```
              Rolf Reichle, Initial Specification
10 Jul 2006 James Geiger, Implementation in LIS
reichle, 28 Jan 2005
reichle, 16 May 2005 - added offset for "domain" grid
```

INTERFACE:

```
subroutine grid2tile( atm_grid, N_tile, tile_coord, grid_data, tile_data, &
     d2g_grid_off_i, d2g_grid_off_j                                       )

  implicit none
```

ARGUMENTS:

```
  type(grid_def_type), intent(in) :: atm_grid

  integer, intent(in) :: N_tile

  type(tile_coord_type), dimension(:), pointer :: tile_coord  ! input

  real, dimension(atm_grid%N_lon,atm_grid%N_lat), intent(in) :: grid_data

  real, dimension(N_tile), intent(out) :: tile_data

  integer, intent(in), optional :: d2g_grid_off_i, d2g_grid_off_j
```

DESCRIPTION:

map from atmospheric grid to tile space
NOTE: tile_coord must match tile_data
optional inputs
d2g_grid_off_i, d2g_grid_off_j : tile_coordatmospheric grid that corresponds to the tile definitions (as obtained from the tile_coord_file) d2g_grid_off_i and d2g_grid_off_j describe the offset between the global "atm_grid_g" and a smaller "atm_grid_d" for the domain of interest. With these offset arguments tile2grid can be used together with an argument atm_grid=atm_grid_d that is a subgrid of the global atm_grid: atm_grid_datm_grid_d

---

15.2 Fortran: Module Interface tile_coord_types (Source File: tile_coord_types.F90)

type definitions for tile coordinates and domain

REVISION HISTORY:

```
26 Jan 2005 Rolf Reichle, Initial Specification
14 Apr 2006 Rolf Reichle, split tile_coord.F90 into 2 files to avoid
                          having more than one module per file
10 Jul 2006 James Geiger, Implementation in LIS
=====================================================================
```

---

16 Gaussian domain

This section contains the interface implementations for a LIS instance using a regular Gaussian-based domain.

16.0.1 create_vegtilespace_gaussian (Source File: create_vegtilespace_gaussian.F90)

REVISION HISTORY:

```
13 Nov 2006: James Geiger ; Initial version
```

INTERFACE:

```
subroutine create_vegtilespace_gaussian(n, fgrd, localmask)
      slope, aspect, curv)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_localPet, LIS_masterproc,  &
     LIS_npes, LIS_ews_ind, LIS_ewe_ind, LIS_nss_ind, LIS_nse_ind, &
     LIS_deltas, LIS_offsets
use LIS_domainMod, only : LIS_domain_setup
use LIS_logMod, only :LIS_logunit
use LIS_mpiMod
use gaussian_mod

implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: n
real                  :: fgrd(LIS_rc%lnc(n), LIS_rc%lnr(n), LIS_rc%nt)
real                  :: localmask(LIS_rc%lnc(n), LIS_rc%lnr(n))
```

DESCRIPTION:

This routine computes the grid and tile spaces based on the input distribution of vegetation and landmask. The routine also computes the following structures necessary for domain decomposition and other parallel routines.
`ntiles_pergrid` : Number of tiles per each grid
`s_tid` : Index of the first tile in each grid cell

The arguments are:

n index of the nest

fgrd fraction of grid covered by each vegetation type

localmask landmask for the region of interest

16.0.2 createtiles_gaussian (Source File: createtiles_gaussian.F90)

REVISION HISTORY:

```
13 Nov 2006: James Geiger; Initial specification
```

INTERFACE:

```
subroutine createtiles_gaussian()
```

USES:

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

DESCRIPTION:

The code in this routine creates the grid and tile spaces in a quasi-regular Gaussian lat/lon grid projection, with a SW to NE data ordering. The routine first reads the landmask and landcover data. The topography data is also read in depending on the options specified at runtime. The landmask is used to create the grid space, ignoring the water points. The vegetation distribution from the landcover map is used to create subgrid tiling, based on the specified maximum number of tiles and minimum cutoff percentage.
The routines invoked are:

readlandmask (3.41.4)
calls the generic routine in the registry to read the landmask

readlandcover (3.41.2)
calls the generic routine in the registry to read the landcover data

readelev (3.49.2)
calls the generic routine in the registry to read the elevation data

readslope (3.49.4)
calls the generic routine in the registry to read the slope data

readaspect (3.49.6)
calls the generic routine in the registry to read the aspect data

readcurv (3.49.8)
calls the generic routine in the registry to read the curvature data

calculate_domveg (19.0.9)
computes the dominant vegetation distribution for subgrid tiling

create_vegtilespace_gaussian (16.0.1)
computes the grid and tile spaces based on the landmask, landcover, and topography datasets in a quasi-regular Gaussian latlon projection

---

16.0.3 readinput_gaussian (Source File: readinput_gaussian.F90)

REVISION HISTORY:

```
13 Nov 2006: James Geiger; Initial specification
```

INTERFACE:

```
subroutine readinput_gaussian
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_config, LIS_deltas, LIS_offsets, &
     LIS_localPet, LIS_masterproc, LIS_npes, LIS_ews_halo_ind, &
     LIS_ewe_halo_ind,LIS_nss_halo_ind, LIS_nse_halo_ind
use LIS_logMod, only :LIS_logunit, LIS_endrun
use LIS_domainMod, only : LIS_quilt_domain, LIS_setParameterDomainSpecs
use LIS_fileIOMod, only : LIS_readDomainConfigSpecs
use gaussian_mod
```

DESCRIPTION:

This routine reads the options specified in the LIS configuration file to determine the region of interest in the LIS simulation, in a quasi-regular Gaussian 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 (3.42.54)
routine to perform domain decomposition

---

17 GSWP domain

This section contains the interface implementations for a LIS instance using the domain used in the Global Soil Wetness Project (GSWP).

17.0.4 createtiles_gswp (Source File: createtiles_gswp.F90)

REVISION HISTORY:

```
23 Feb 04, Sujay Kumar : Intial Specification
```

INTERFACE:

```
subroutine createtiles_gswp()
```

USES:

```
  use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_localPet,LIS_masterproc,  &
       LIS_npes, LIS_ews_ind, LIS_ewe_ind, LIS_nss_ind, LIS_nse_ind, &
       LIS_deltas, LIS_offsets
  use LIS_mpiMod
  use LIS_logMod, only : LIS_logunit, LIS_verify
#if ( defined USE_NETCDF )
  use netcdf
#endif
```

DESCRIPTION:

The code in this file implements the grid and tile spaces for the GSWP domain. The routine first reads the landmask and the landcover data. The size of the grid space is determined by ignoring the water points. The GSWP domain does not support subgrid tiling. As a result, the grid and tile spaces are equivalent.

17.0.5 readinput_gswp (Source File: readinput_gswp.F90)

REVISION HISTORY:

```
14 Nov 2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine readinput_gswp
```

USES:

```
use ESMF_Mod
use LIS_logMod, only :LIS_logunit, LIS_endrun
use LIS_coreMod, only : LIS_rc, LIS_config, LIS_localPet, LIS_masterproc, &
     LIS_npes, LIS_ews_ind, LIS_nss_ind, LIS_ewe_ind, &
     LIS_nse_ind, LIS_deltas, LIS_offsets
use LIS_domainMod, only : LIS_setParameterDomainSpecs
use LIS_fileIOMod, only : LIS_readDomainConfigSpecs
use LIS_mpiMod
```

DESCRIPTION:

This routine reads the domain-related runtime options for GSWP. The GSWP grid is a lat/lon 1deg global grid spanning from 90S to 60N. The extents of the running domain are specified in the LIS configuration file, and read by this routine. The routine also reads the domain extents of the paramter maps. In GSWP, all the parameters are also specified in a lat/lon 1deg global grid.

---

18 Lambert Conformal domain

This section contains the interface implementations for a LIS instance using a lambert conformal projection with SW to NE data ordering

---

18.0.6 create_vegtilespace_lambert (Source File: create_vegtilespace_lambert.F90)

REVISION HISTORY:

```
25 Jan 2006: Sujay Kumar ; Initial version
```

INTERFACE:

```
subroutine create_vegtilespace_lambert(n,fgrd, localmask)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_localPet, LIS_masterproc,  &
     LIS_npes, LIS_ews_ind, LIS_ewe_ind, LIS_nss_ind, LIS_nse_ind, &
     LIS_deltas, LIS_offsets
use LIS_mpiMod
use LIS_logMod, only :LIS_logunit
use LIS_domainMod, only : LIS_domain_setup
use map_utils

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
real                   :: fgrd(LIS_rc%lnc(n), LIS_rc%lnr(n), LIS_rc%nt)
real                   :: localmask(LIS_rc%lnc(n), LIS_rc%lnr(n))
```

DESCRIPTION:

This routine computes the grid and tile spaces based on the input distribution of vegetation and landmask. The routine also computes the following structures necessary for domain decomposition and other parallel routines.
`ntiles_pergrid` : Number of tiles per each grid
`s_tid` : Index of the first tile in each grid cell

The arguments are:

n index of the nest

fgrd fraction of grid covered by each vegetation type

localmask landmask for the region of interest

18.0.7 createtiles_lambert (Source File: createtiles_lambert.F90)

REVISION HISTORY:

```
17 Jul 2005: Sujay Kumar ; Initial specification
```

INTERFACE:

```
subroutine createtiles_lambert()
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_logMod,  only : LIS_verify
use LIS_LMLCMod, only : LIS_LMLC
use LIS_topoMod, only : LIS_topo

implicit none
```

DESCRIPTION:

The code in this routine creates the grid and tile spaces in a lambert conformal grid projection, with a SW to NE data ordering. The routine first reads the landmask and landcover data. The topography data is also read in depending on the options specified at runtime. The landmask is used to create the grid space, ignoring the water points. The vegetation distribution from the landcover map is used to create subgrid tiling, based on the specified maximum number of tiles and minimum cutoff percentage.
The routines invoked are:

calculate_domveg (19.0.9)
computes the dominant vegetation distribution for subgrid tiling

create_vegtilespace_lambert (18.0.6)
computes the grid and tile spaces based on the landmask, landcover, and topography datasets for lambert conformal projection.

18.0.8 readinput_lambert (Source File: readinput_lambert.F90)

REVISION HISTORY:

```
15Jul2005: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine readinput_lambert
```

USES:

```
use ESMF_Mod
use LIS_logMod, only : LIS_logunit, LIS_endrun, LIS_verify
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

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 (3.42.54)
routine to perform domain decomposition

---

19 Lat/Lon domain

This section contains the interface implementations for a LIS instance using a lat/lon projection with SW to NE data ordering

---

19.0.9 calculate_domveg (Source File: calculate_domveg.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine calculate_domveg(n, fgrd)
```

USES:

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

implicit none
```

ARGUMENTS:

```
integer, intent(in)  :: n
real :: fgrd(LIS_rc%lnc(n), LIS_rc%lnr(n), LIS_rc%nt)
```

DESCRIPTION:

This primary goal of this routine is to determine the percentages of dominant vegetation for subgrid tiling. The routine uses the distribution of vegetation within a grid cell and imposes the specified cutoff percentage and the maximum number of veg types allowed to create the number of tiles.
The arguments are:

n index of the nest

fgrd fraction of grid covered by each vegetation type

19.0.10 create_vegtilespace_latlon (Source File: create_vegtilespace_latlon.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine create_vegtilespace_latlon(n, fgrd, localmask)
      slope, aspect, curv)
```

USES:

```
use LIS_coreMod,   only : LIS_rc, LIS_domain, LIS_localPet
use LIS_logMod,    only : LIS_logunit
use LIS_domainMod, only : LIS_domain_setup
use LIS_mpiMod

implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: n
real                  :: fgrd(LIS_rc%lnc(n), LIS_rc%lnr(n), LIS_rc%nt)
real                  :: localmask(LIS_rc%lnc(n), LIS_rc%lnr(n))
```

DESCRIPTION:

This routine computes the grid and tile spaces based on the input distribution of vegetation and landmask. The routine also computes the following structures necessary for domain decomposition and other parallel routines.
`ntiles_pergrid` : Number of tiles per each grid
`s_tid` : Index of the first tile in each grid cell

The arguments are:

n index of the nest

fgrd fraction of grid covered by each vegetation type

localmask landmask for the region of interest

19.0.11 createtiles_latlon (Source File: createtiles_latlon.F90)

REVISION HISTORY:

```
04  Feb 2001: Jon Gottschalck; Added option to read and use Koster tile space
17  Oct 2003: Sujay Kumar ; Initial version of subsetting code
18  Feb 2006: Sujay Kumar ; Added nesting options
```

INTERFACE:

```
subroutine createtiles_latlon()
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_logMod,  only : LIS_logunit, LIS_verify
use LIS_LMLCMod, only : LIS_LMLC
use LIS_topoMod, only : LIS_topo
```

DESCRIPTION:

The code in this routine creates the grid and tile spaces in a lat/lon grid projection, with a SW to NE data ordering. The topography data is also read in depending on the options specified at runtime. The landmask is used to create the grid space, ignoring the water points. The vegetation distribution from the landcover map is used to create subgrid tiling, based on the specified maximum number of tiles and minimum cutoff percentage.
The routines invoked are:

calculate_domveg (19.0.9)
computes the dominant vegetation distribution for subgrid tiling

create_vegtilespace_latlon (19.0.10)
computes the grid and tile spaces based on the landmask, landcover, and topography datasets in a latlon projection

---

19.0.12 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_Mod
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
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 (3.42.54)
routine to perform domain decomposition

---

20 Mercator domain

This section contains the interface implementations for a LIS instance using a mercator projection with SW to NE data ordering

---

20.0.13 create_vegtilespace_merc (Source File: create_vegtilespace_merc.F90)

REVISION HISTORY:

```
25 Jan 2006: Sujay Kumar ; Initial version
```

INTERFACE:

```
subroutine create_vegtilespace_merc(n,fgrd, localmask)
      slope, aspect, curv)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_localPet, LIS_masterproc,  &
     LIS_npes, LIS_ews_ind, LIS_ewe_ind, LIS_nss_ind, LIS_nse_ind, &
     LIS_deltas, LIS_offsets
use LIS_mpiMod
use LIS_logMod, only :LIS_logunit
use LIS_domainMod, only : LIS_domain_setup
use map_utils

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
real                :: fgrd(LIS_rc%lnc(n), LIS_rc%lnr(n), LIS_rc%nt)
real                :: localmask(LIS_rc%lnc(n), LIS_rc%lnr(n))
```

DESCRIPTION:

This routine computes the grid and tile spaces based on the input distribution of vegetation and landmask. The routine also computes the following structures necessary for domain decomposition and other parallel routines.
`ntiles_pergrid` : Number of tiles per each grid
`s_tid` : Index of the first tile in each grid cell

The arguments are:

n index of the nest

fgrd fraction of grid covered by each vegetation type

localmask landmask for the region of interest

20.0.14 createtiles_merc (Source File: createtiles_merc.F90)

REVISION HISTORY:

```
17 Jul 2005: Sujay Kumar ; Initial specification
```

INTERFACE:

```
subroutine createtiles_merc()
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_logMod,  only : LIS_verify
use LIS_LMLCMod, only : LIS_LMLC
use LIS_topoMod, only : LIS_topo

implicit none
```

DESCRIPTION:

The code in this routine creates the grid and tile spaces in a mercator grid projection, with a SW to NE data ordering. The routine first reads the landmask and landcover data. The topography data is also read in depending on the options specified at runtime. The landmask is used to create the grid space, ignoring the water points. The vegetation distribution from the landcover map is used to create subgrid tiling, based on the specified maximum number of tiles and minimum cutoff percentage.
The routines invoked are:

calculate_domveg (19.0.9)
computes the dominant vegetation distribution for subgrid tiling

create_vegtilespace_merc (20.0.13)
computes the grid and tile spaces based on the landmask, landcover, and topography datasets for the mercator projection

---

20.0.15 readinput_merc (Source File: readinput_merc.F90)

REVISION HISTORY:

```
15 Jul 2005: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine readinput_merc
```

USES:

```
use ESMF_Mod
use LIS_logMod, only : LIS_logunit, LIS_endrun
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
```

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 (3.42.54)
routine to perform domain decomposition

---

21 Polar stereographic domain

This section contains the interface implementations for a LIS instance using a polar stereographic projection with SW to NE data ordering

---

21.0.16 create_vegtilespace_polar (Source File: create_vegtilespace_polar.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine create_vegtilespace_polar(n, fgrd, localmask)
      slope, aspect, curv)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_localPet, LIS_masterproc,  &
     LIS_npes, LIS_ews_ind, LIS_ewe_ind, LIS_nss_ind, LIS_nse_ind, &
     LIS_deltas, LIS_offsets
use LIS_mpiMod
use LIS_logMod, only :LIS_logunit
use LIS_domainMod, only : LIS_domain_setup
use map_utils

implicit none

integer, intent(in) :: n
real                :: fgrd(LIS_rc%lnc(n), LIS_rc%lnr(n), LIS_rc%nt)
real                :: localmask(LIS_rc%lnc(n), LIS_rc%lnr(n))
```

DESCRIPTION:

This routine computes the grid and tile spaces based on the input distribution of vegetation and landmask. The routine also computes the following structures necessary for domain decomposition and other parallel routines.
`ntiles_pergrid` : Number of tiles per each grid
`s_tid` : Index of the first tile in each grid cell

The arguments are:

n index of the nest

fgrd fraction of grid covered by each vegetation type

localmask landmask for the region of interest

21.0.17 createtiles_polar (Source File: createtiles_polar.F90)

REVISION HISTORY:

```
17Jul2005: Sujay Kumar ; Initial specification
```

INTERFACE:

```
subroutine createtiles_polar()
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_logMod,  only : LIS_verify
use LIS_LMLCMod, only : LIS_LMLC
use LIS_topoMod, only : LIS_topo
```

DESCRIPTION:

The code in this routine creates the grid and tile spaces in a polar stereographic grid projection, with a SW to NE data ordering. The routine first reads the landmask and landcover data. The topography data is also read in depending on the options specified at runtime. The landmask is used to create the grid space, ignoring the water points. The vegetation distribution from the landcover map is used to create subgrid tiling, based on the specified maximum number of tiles and minimum cutoff percentage.
The routines invoked are:

calculate_domveg (19.0.9)
computes the dominant vegetation distribution for subgrid tiling

create_vegtilespace_polar (21.0.16)
computes the grid and tile spaces based on the landmask, landcover, and topography datasets in a polar stereographic projection

---

21.0.18 readinput_polar (Source File: readinput_polar.F90)

REVISION HISTORY:

```
15 Jul 2005: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine readinput_polar
```

USES:

```
use ESMF_Mod
use LIS_logMod, only : LIS_logunit, LIS_endrun
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

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 (3.42.54)
routine to perform domain decomposition

Part IX
Land Surface Parameters in LIS

22 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.

---

23 Albedo Data

This section describes the routines to ingest various sources of albedo data in different LIS domains

---

23.0.19 read_Briegleb_albedo (Source File: read_Briegleb_albedo.F90)

REVISION HISTORY:

```
25 Jul 2005: Sujay Kumar; Initial Specification
27 Nov 2007: Charles Alonge : Added options for monthly files
```

INTERFACE:

```
subroutine read_Briegleb_albedo(n, time1, array)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,        only : LIS_rc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_timeMgrMod,     only : LIS_calendar
use LIS_fileIOMod,      only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_Time)     :: time1
real, intent(inout) :: array(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine retrieves the albedo climatology for the specified month and returns the values in a latlon projection
Ref: Briegleb, B.P., P. Minnis, V. Ramanathan, and E. Harrison, 1986: Comparison of regional clear-sky albedos inferred from satellite observations and model computations. J. Clim. Appl. Meteor., 25, 214-226
The arguments are:

n index of the nest

q time index (month or quarter)

array output field with the retrieved albedo

23.0.20 read_GFS_albedo (Source File: read_GFS_albedo.F90)

REVISION HISTORY:

```
22 Sept 2006: James Geiger; Initial Specification
```

INTERFACE:

```
subroutine read_GFS_albedo(nest, time1, array)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,        only : LIS_rc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,     only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: nest
type(ESMF_Time)     :: time1
real, intent(inout) :: array(LIS_rc%lnc(nest),LIS_rc%lnr(nest))
```

DESCRIPTION:

This subroutine retrieves the albedo climatology for the specified month and returns the values in a quasi-regular Gaussian latlon projection
The arguments are:

nest index of the nest

q time index (month or quarter)

array output field with the retrieved albedo

23.0.21 read_GFS_mxsnoalb (Source File: read_GFS_mxsnoalb.F90)

REVISION HISTORY:

```
22 Sep 2006: James Geiger; Initial Specification
```

INTERFACE:

```
subroutine read_GFS_mxsnoalb(nest,array)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,     only : LIS_readData
```

23.0.22 read_RobinsonKukla_mxsnoalb (Source File: read_RobinsonKukla_mxsnoalb.F90)

REVISION HISTORY:

```
25 Jul 2005: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_RobinsonKukla_mxsnoalb(n,array)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,     only : LIS_readData
```

23.0.23 setup_Briegleb_albedo (Source File: setup_Briegleb_albedo.F90)

REVISION HISTORY:

```
13 Jan 2009: James Geiger; Initial Specification
```

INTERFACE:

```
subroutine setup_Briegleb_albedo(n)
```

USES:

```
use LIS_albedoMod,  only : LIS_alb

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine sets up the variables required for reading the albedo climatology data from Briegleb
The arguments are:

n index of the nest

23.0.24 setup_GFS_albedo (Source File: setup_GFS_albedo.F90)

REVISION HISTORY:

```
13 Jan 2009: James Geiger; Initial Specification
```

INTERFACE:

```
subroutine setup_GFS_albedo(n)
```

USES:

```
use LIS_albedoMod,  only : LIS_alb

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine sets up the variables required for reading the albedo climatology data from GFS
The arguments are:

n index of the nest

---

24 Greenness Fraction Data

This section describes the routines to ingest various sources of greenness fraction data in different LIS domains

---

24.0.25 read_AVHRRgfrac (Source File: read_AVHRRgfrac.F90)

REVISION HISTORY:

```
25 Jul 2005: Sujay Kumar; Initial Specification
20 Feb 2006: Sujay Kumar; Modified to support nesting
```

INTERFACE:

```
subroutine read_AVHRRgfrac(n, time1, array)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_gfracMod,      only : LIS_gfrac
use LIS_fileIOMod,     only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer,          intent(in) :: n
type(ESMF_Time)              :: time1
real,          intent(inout) :: array(LIS_rc%nch(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

24.0.26 setup_AVHRRgfrac (Source File: setup_AVHRRgfrac.F90)

REVISION HISTORY:

```
16 Jul 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine setup_AVHRRgfrac(n)
```

USES:

```
use LIS_gfracMod,  only : LIS_gfrac

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine sets up the variables required for reading the greenness fraction climatology data from AVHRR
The arguments are:

n index of the nest

24.0.27 read_GFSfrac (Source File: read_GFSgfrac.F90)

REVISION HISTORY:

```
22 Sep 2006: James Geiger; Initial Specification
```

INTERFACE:

```
subroutine read_GFSgfrac(nest, time1, array)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_gfracMod,      only : LIS_gfrac
use LIS_fileIOMod,     only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: nest
type(ESMF_Time)     :: time1
real, intent(inout) :: array(LIS_rc%nch(nest))
```

DESCRIPTION:

This subroutine retrieves the greenness fraction climatology for the specified month and returns the values in the quasi-regular Gaussian latlon projection
The arguments are:

nest index of the nest

mo time index (month or quarter)

array output field with the retrieved greenness fraction

24.0.28 setup_GFSgfrac (Source File: setup_GFSgfrac.F90)

REVISION HISTORY:

```
16 Jul 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine setup_GFSgfrac(n)
```

USES:

```
use LIS_gfracMod,  only : LIS_gfrac

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine sets up the variables required for reading the greenness fraction climatology data from GFS
The arguments are:

n index of the nest

24.0.29 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, time1, array)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_logMod,         only : LIS_logunit, LIS_verify, LIS_endrun
use LIS_gfracMod,      only : LIS_gfrac

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_Time)     :: time1
real, intent(inout) :: array(LIS_rc%nch(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

---

25 Leaf Area Index Data

This section describes the routines to ingest various sources of LAI data in different LIS domains

---

25.0.30 read_AVHRRlai (Source File: read_AVHRRlai.F90)

REVISION HISTORY:

```
25 Jul 2005: Sujay Kumar; Initial Specification
20 Feb 2006: Sujay Kumar; Modified to support nesting
```

INTERFACE:

```
subroutine read_AVHRRlai(n, time1, array)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,     only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_Time)     :: time1
real, intent(inout) :: array(LIS_rc%nch(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

25.0.31 read_AVHRRsai (Source File: read_AVHRRsai.F90)

REVISION HISTORY:

```
25 Jul 2005: Sujay Kumar; Initial Specification
20 Feb 2006: Sujay Kumar; Modified to support nesting
```

INTERFACE:

```
subroutine read_AVHRRsai(n, time1, array)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,        only : LIS_rc, LIS_domain, LIS_localPet
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,     only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_Time)     :: time1
real, intent(inout) :: array(LIS_rc%nch(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

25.0.32 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_Mod
use LIS_coreMod,        only : LIS_rc, LIS_domain, LIS_localPet
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,     only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_Time)     :: time1
real, intent(inout) :: array(LIS_rc%nch(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

25.0.33 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_Mod
use LIS_coreMod,        only : LIS_rc, LIS_domain, LIS_localPet
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,      only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_Time)     :: time1
real, intent(inout) :: array(LIS_rc%nch(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

25.0.34 read_tiled_AVHRRlai (Source File: read_tiled_AVHRRlai.F90)

REVISION HISTORY:

```
16 Jun 2008: Charles Alonge; Initial Specification
```

INTERFACE:

```
subroutine read_tiled_AVHRRlai(n, time1, array)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,      only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_Time)     :: time1
real, intent(inout) :: array(LIS_rc%nch(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

25.0.35 read_tiled_AVHRRsai (Source File: read_tiled_AVHRRsai.F90)

REVISION HISTORY:

```
16 Jun 2008: Charles Alonge; Initial Specification
```

INTERFACE:

```
subroutine read_tiled_AVHRRsai(n, time1, array)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,      only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_Time)     :: time1
real, intent(inout) :: array(LIS_rc%nch(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

---

26 Landmask/Landcover Data

This section describes the routines to ingest various sources of landmask/landcover data in different LIS domains

---

26.0.36 read_GFSlc (Source File: read_GFSlc.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_GFSlc(n, fgrd)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_verify
use LIS_fileIOMod,      only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
real  :: fgrd(LIS_rc%lnc(n),LIS_rc%lnr(n),LIS_rc%nt)
```

DESCRIPTION:

This subroutine reads the GFS landcover data and returns the distribution of vegetation in each grid cell, in a lat/lon projection. The data has 13 vegetation types.
The arguments are:

n index of nest

fgrd fraction of grid covered by each vegetation type

26.0.37 read_GFSmask (Source File: read_GFSmask.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_GFSmask(nest, localmask)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_localPet
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,      only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: nest
real                :: localmask(LIS_rc%lnc(nest),LIS_rc%lnr(nest))
```

DESCRIPTION:

This subroutine reads the GFS landmask data and returns the array in a lat/lon projection.
The arguments are:

n index of nest

localmask landmask for the region of interest

26.0.38 read_UMDlc (Source File: read_UMDlc.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_UMDlc(n, fgrd)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_verify, LIS_endrun
use LIS_fileIOMod,      only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
real  :: fgrd(LIS_rc%lnc(n),LIS_rc%lnr(n),LIS_rc%nt)
```

DESCRIPTION:

This subroutine reads the UMD landcover data and returns the distribution of vegetation in each grid cell, in a lat/lon projection. The data has 13 vegetation types.
The arguments are:

n index of nest

fgrd fraction of grid covered by each vegetation type

26.0.39 read_UMDmask (Source File: read_UMDmask.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_UMDmask(nest, localmask)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_localPet
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,      only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: nest
real                :: localmask(LIS_rc%lnc(nest),LIS_rc%lnr(nest))
```

DESCRIPTION:

This subroutine reads the UMD landmask data and returns the array in a lat/lon projection.
The arguments are:

n index of nest

localmask landmask for the region of interest

26.0.40 read_USGSlc (Source File: read_USGSlc.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_USGSlc(n, fgrd)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_verify, LIS_endrun
use LIS_fileIOMod,      only : LIS_readData
use LIS_pluginIndices

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
real  :: fgrd(LIS_rc%lnc(n),LIS_rc%lnr(n),LIS_rc%nt)
```

DESCRIPTION:

This subroutine reads the USGS landcover data and returns the distribution of vegetation in each grid cell, in a lat/lon projection. The data has 24 vegetation types.
The arguments are:

n index of nest

fgrd fraction of grid covered by each vegetation type

26.0.41 read_USGSmask (Source File: read_USGSmask.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_USGSmask(nest, localmask)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,      only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: nest
real                :: localmask(LIS_rc%lnc(nest),LIS_rc%lnr(nest))
```

DESCRIPTION:

This subroutine reads the USGS landmask data and returns the array in a lat/lon projection.
The arguments are:

n index of nest

localmask landmask for the region of interest

---

27 Soils

This section describes the routines to ingest various sources of soil parameter data in different LIS domains

---

27.0.42 read_GFStexture (Source File: read_GFStexture.F90)

REVISION HISTORY:

```
22 Sept 2006: James Geiger; Initial Specification
```

INTERFACE:

```
subroutine read_GFStexture(nest,array)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,     only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer,intent(in)    :: nest
integer,intent(inout) :: array(LIS_rc%lnc(nest),LIS_rc%lnr(nest))
```

DESCRIPTION:

This subroutine retrieves GFS soil texture data on a quasi-regular Gaussian latlon projection.
The arguments are:

nest index of the nest

array output field with the retrieved soil texture

27.0.43 read_ecmwftexture (Source File: read_ecmwftexture.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_ecmwftexture(n, array)
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use LIS_logMod, only      : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,   only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in)  :: n !nest index
integer, intent(inout) :: array(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine retrieves ECMWF texture data and reprojects it to the map projection used by LIS. The arguments are:

n index of the nest

array output field with the retrieved texture data

27.0.44 read_faoclay (Source File: read_faoclay.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_faoclay(n, array)
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use LIS_logMod, only      : LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_logunit, LIS_endrun
use LIS_fileIOMod,     only : LIS_readData

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine retrieves FAO clay fraction data and reprojects it to the latlon projection.
The arguments are:

n index of the nest

array output field with the retrieved clay fraction

27.0.45 read_faocolor (Source File: read_faocolor.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_faocolor(n, array)
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use LIS_logMod, only      : LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,     only : LIS_readData

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine retrieves FAO soil color data and reprojects it to the latlon projection.
The arguments are:

n index of the nest

array output field with the retrieved soil color

27.0.46 read_faoporosity (Source File: read_faoporosity.F90)

REVISION HISTORY:

```
27 Sept 2007: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_faoporosity(n, array)
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use LIS_logMod, only      : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,     only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in)  :: n !nest index
real,          intent(inout) :: array(LIS_rc%lnc(n),LIS_rc%lnr(n),3)
```

DESCRIPTION:

This subroutine retrieves FAO porosity fraction data and reprojects it to the latlon projection.
The arguments are:

n index of the nest

array output field with the retrieved porosity fraction

27.0.47 read_faosand (Source File: read_faosand.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_faosand(n, array)
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use LIS_logMod, only      : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,   only : LIS_readData

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine retrieves FAO sand fraction data and reprojects it to the latlon projection.
The arguments are:

n index of the nest

array output field with the retrieved sand fraction

27.0.48 read_faosilt (Source File: read_faosilt.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_faosilt(n, array)
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use LIS_logMod, only      : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,     only : LIS_readData

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine retrieves FAO silt fraction data and reprojects it to the latlon projection.
The arguments are:

n index of the nest

array output field with the retrieved silt fraction

27.0.49 read_faotexture (Source File: read_faotexture.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_faotexture(n, array)
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use LIS_logMod, only      : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,   only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in)  :: n !nest index
integer, intent(inout) :: array(LIS_rc%lnc(n),LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine retrieves FAO texture data and reprojects it to the map projection used by LIS.
The arguments are:

n index of the nest

array output field with the retrieved texture data

27.0.50 read_statsgoclay (Source File: read_statsgoclay.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_statsgoclay(n, array)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,      only : LIS_readData, LIS_checkDomainExtents

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine retrieves STATSGO clay fraction data and reprojects it to the latlon projection.
The arguments are:

n index of the nest

array output field with the retrieved clay fraction

27.0.51 read_statsgocolor (Source File: read_statsgocolor.F90)

REVISION HISTORY:

```
17 Jan 2007: James Geiger; Initial Specification
```

INTERFACE:

```
subroutine read_statsgocolor(n, array)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,     only  : LIS_readData, LIS_checkDomainExtents

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine retrieves STATSGO color fraction data and reprojects it to the latlon projection. The arguments are:

n index of the nest

array output field with the retrieved color fraction

27.0.52 read_statsgososand (Source File: read_statsgosand.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_statsgosand(n, array)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,     only : LIS_readData, LIS_checkDomainExtents

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine retrieves STATSGO sand fraction data and reprojects it to the latlon projection. The arguments are:

n index of the nest

array output field with the retrieved sand fraction

27.0.53 read_statsgosilt (Source File: read_statsgosilt.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_statsgosilt(n, array)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,      only : LIS_readData, LIS_checkDomainExtents
implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine retrieves STATSGO silt fraction data and reprojects it to the latlon projection.
The arguments are:

n index of the nest

array output field with the retrieved silt fraction

27.0.54 read_statsgotexture (Source File: read_statsgotexture.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_statsgotexture(n,array)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,     only : LIS_readData, LIS_checkDomainExtents
implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine retrieves STATSGO soil texture data and reprojects it to the latlon projection.
The arguments are:

n index of the nest

array output field with the retrieved soil texture

---

28 Bottom Temperature

This section describes the routines to ingest various sources of bottom boundary temperature data in different LIS domains

---

28.0.55 read_NCEPtbot (Source File: read_NCEPtbot.F90)

REVISION HISTORY:

```
25 Jul 2005: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_NCEPtbot(n, array)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,      only : LIS_readData

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine retrieves static, bottom temperature data and reprojects it to the latlon projection. The arguments are:

n index of the nest

array output field with the retrieved bottom temperature

---

29 Topography

This section describes the routines to ingest various sources of topography data in different LIS domains

---

29.0.56 read_aspect_gtopo30 (Source File: read_aspect_gtopo30.F90)

REVISION HISTORY:

```
22 Dec 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_aspect_gtopo30(n,aspect)
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use LIS_logMod,   only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod, only : LIS_readData
```

29.0.57 read_curv_gtopo30 (Source File: read_curv_gtopo30.F90)

REVISION HISTORY:

```
22 Dec 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_curv_gtopo30(n,curv)
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use LIS_logMod, only      : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,   only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
real                :: curv(LIS_rc%lnc(n), LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine retrieves static curvature data from the GTOPO30 source and reprojects it to the latlon projection.
The arguments are:

n index of the nest

curv output field with the retrieved curvature

29.0.58 read_elev_GFS (Source File: read_elev_GFS.F90)

REVISION HISTORY:

```
22 Sep 2006: James Geiger; Initial Specification
```

INTERFACE:

```
subroutine read_elev_GFS(n,elev)
```

USES:

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

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
real                :: elev(LIS_rc%lnc(n), LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine retrieves static elevation data from the GFS source and reprojects it to the quasi-regular GFS latlon projection.
The arguments are:

n index of the nest

elev output field with the retrieved elevation

29.0.59 read_elev_gtopo30 (Source File: read_elev_gtopo30.F90)

REVISION HISTORY:

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

INTERFACE:

```
subroutine read_elev_gtopo30(n,elev)
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use LIS_logMod, only      : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod,   only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
real                :: elev(LIS_rc%lnc(n), LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine retrieves static elevation data from the GTOPO30 source and reprojects it to the latlon projection.
The arguments are:

n index of the nest

elev output field with the retrieved elevation

29.0.60 read_slope_gtopo30 (Source File: read_slope_gtopo30.F90)

REVISION HISTORY:

```
22 Dec 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_slope_gtopo30(n,slope)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber, LIS_endrun
use LIS_fileIOMod, only : LIS_readData

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
real                :: slope(LIS_rc%lnc(n), LIS_rc%lnr(n))
```

DESCRIPTION:

This subroutine retrieves static slope data from the GTOPO30 source and reprojects it to the latlon projection.
The arguments are:

n index of the nest

slope output field with the retrieved slope

Part X
Base forcing analyses in LIS

30 Overview

This section contains various model-based and satellite-based meteorological forcing analyses implemented in LIS. These include the algorithms from the AGRMET system of the Air Force Weather Agency (AFWA), Global Data Assimilation System (GDAS) from NCEP, Goddard Earth Observing System (GEOS) from NASA GSFC, forcing products from the ECMWF, Global Soil Wetness Project (GSWP) and the North American Land Data Assimilation System (NLDAS). The details of each scheme are described in the following sections.

---

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

nmif Number of forcing variables in the ECMWF 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.

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
---------------------------------------------------------------------------
public :: defineNativeECMWF      !defines the native resolution of
                                 !the input data
---------------------------------------------------------------------------
```

PUBLIC TYPES:

```
---------------------------------------------------------------------------
public :: ecmwf_struc
```

31.1.1 defineNativeECMWF (Source File: ecmwf_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defineNativeECMWF
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_date2time

implicit none
```

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 V
The routines invoked are:

readecmwfcrd (31.1.5)
reads the runtime options specified for ECMWF data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

---

31.1.2 ecmwfforcing_finalize (Source File: ecmwfforcing_finalize.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine ecmwfforcing_finalize
```

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 getecmwf (Source File: getecmwf.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 getecmwf(n)
```

USES:

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

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 (3.31.25)
call to advance or retract time

retecmwf (31.1.6)
call to read the ECMWF data and perform spatial interpolation

read_ecmwf_elev (31.1.4)
reads the native elevation of the ECMWF data to be used for topographic adjustments to the forcing

---

31.1.4 read_ecmwf_elev (Source File: read_ecmwf_elev.F90)

REVISION HISTORY:

```
17Dec2004; Sujay Kumar; Initial Specificaton
```

INTERFACE:

```
subroutine read_ecmwf_elev(n, change)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_baseforcingMod, only : LIS_forc
use LIS_logMod,         only : LIS_logunit, LIS_endrun
use ecmwf_forcingMod,   only : ecmwf_struc

implicit none
```

ARGUMENTS:

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

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

31.1.5 readecmwfcrd (Source File: readecmwfcrd.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readecmwfcrd()
```

USES:

```
use ESMF_Mod
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.

31.1.6 retecmwf (Source File: retecmwf.F90)

REVISION HISTORY:

```
18 Jun 2003: Urszula Jambor; original code
```

INTERFACE:

```
subroutine retecmwf( order, n, yr, mon, da, hr, ferror )
```

USES:

```
use LIS_coreMod, only        : LIS_rc, LIS_domain
use LIS_logMod, only         : LIS_logunit, LIS_endrun
use LIS_baseforcingMod, only : LIS_forc
use ecmwf_forcingMod,   only : ecmwf_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n  ! lower(1) or upper(2) time interval bdry
integer, intent(in)    :: order  ! lower(1) or upper(2) time interval bdry
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 1/4 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: –¿ inst3, Instantaneous values, available every 3 hours
–¿ accum, Time integrated values (accumulations), updated every 3 hours

NOTE-1: be aware that ECMWF outputs large-scale and convective precipitation separately. For total precipitation, need to sum the two fields, LSP+CP=TP. Reversed traditional order of model-forcing precip

fields for easier processing – interchange at end of routine to preserve traditional order.
NOTE 2: only time2 SW flux accumulations used in interpolation
NOTE-3: read in of Albedo is currently suppressed. This field is instantaneous and available every 6 hours. At this time, all 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. CP accum, convective precipitation [m]
9. LSP accum, large scale 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:

ret_inst3 (31.1.7)
retrieves the instantaneous variable

ret_accum (31.1.8)
retrieves the accumulated variable

interp_iv (31.1.9)
spatial interpolation from the ECMWF grid to the LIS grid

31.1.7 ret_inst3 (Source File: retecmwf.F90)

INTERFACE:

```
subroutine ret_inst3(dir,yr,mo,da,hr,necmwf,f,id,id7,lb,fret)

  use LIS_logMod, only : LIS_logunit

  implicit none
```

ARGUMENTS:

```
character(len=*), intent(in) :: dir
integer, intent(in)        :: yr,mo,da,hr,necmwf,id,id7
real, intent(out)          :: f(necmwf)
logical*1, intent(out)     :: lb(necmwf)
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.
The arguments are:

dir directory path name for ECMWF data

yr year

mo month

da day of month

hr hour of day

necmwf number of elements of input grid

f 1D input forcing field

id index indicating the forcing variable

lb unpacked input bitmap

fret return code (0-indicates success)

31.1.8 ret_accum (Source File: retecmwf.F90)

INTERFACE:

```
subroutine ret_accum(dir,yr,mo,da,hr,necmwf,f,id,id7,lb,fret)
```

USES:

```
use LIS_timeMgrMod, only : LIS_tick
use LIS_logMod, only : LIS_logunit

implicit none
```

ARGUMENTS:

```
character*200    :: dir
integer          :: yr,mo,da,hr
integer          :: necmwf
real             :: f(necmwf)
integer          :: id,id7
logical*1        :: lb(necmwf)
integer          :: fret
```

DESCRIPTION:

This routine opens the corresponding ECMWF data file to extract the specified variable, which represents an accumulated value. Should be used for radiation and precipitation. Returns field valid for 3-hour interval prior to given time, except for downward sw flux, where interval length varies. Returns forcing in units appropriate to LIS:
W/(m2s) to W/m2
m to mm/s

The arguments are:

dir directory path name for ECMWF data

yr year

mo month

da day of month

hr hour of day

necmwf number of elements of input grid

f 1D input forcing field

id index indicating the forcing variable

lb unpacked input bitmap

fret return code (0-indicates success)

31.1.9 interp_iv (Source File: retecmwf.F90)

INTERFACE:

```
subroutine interp_iv (n, iv,lis_gridDesc,necmwf,nlis,lb,f,lis_1d,mo,iret)
```

USES:

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

implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: n
integer               :: iv
real                  :: lis_gridDesc(50)
integer               :: necmwf
integer               :: nlis
real                  :: f(necmwf)
real, dimension(nlis) :: lis_1d
integer               :: mo
integer               :: iret
```

DESCRIPTION:

This routine interpolates the input forcing to the LIS grid
The arguments are:

n index of the nest

iv field identifier

lis_gridDesc array description of the LIS grid

necmwf number of elements in the input grid

nlis number of elements in the LIS grid

f input data array to be interpolated

lis_1d interpolated array in the LIS grid

mo Number of points in the output grid

iret return code (0 indicates success)

The routines invoked are:

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

---

31.1.10 time_interp_ecmwf (Source File: time_interp_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 time_interp_ecmwf(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,            only : LIS_rc, LIS_domain
use LIS_FORC_AttributesMod
use LIS_baseforcingMod,     only : LIS_forc, LIS_FORC_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
```

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 (3.31.23)
converts the time to a date format

LIS_tick (3.31.25)
advances or retracts time by the specified amount

zterp (5.3.18)
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.

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 :: defineNativeECMWFREANAL      !defines the native resolution of
                                  !the input data
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: ecmwfreanal_struc
```

32.1.1 defineNativeECMWFREANAL (Source File: ecmwfreanal_forcingMod.F90)

REVISION HISTORY:

```
26Jan2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defineNativeECMWFREANAL()
```

USES:

```
use LIS_coreMod, only : LIS_rc

implicit none
```

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 V
The routines invoked are:

readecmwfreanalcrd (32.1.6)
reads the runtime options specified for ECMWFREANAL data

readecmwfreanalmask (32.1.7)
reads the 0.5 degree land sea mask

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

read_ecmwfreanal_elev (32.1.5)
reads the native elevation of the ecmwfreanal data to be used for topographic adjustments to the forcing

---

32.1.2 ecmwfreanalforcing_finalize (Source File: ecmwfreanalforcing_finalize.F90)

REVISION HISTORY:

```
25 Oct 2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine ecmwfreanalforcing_finalize
```

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 (Source File: geogfill2.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(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 getecmwfreanal (Source File: getecmwfreanal.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 getecmwfreanal(n)
```

USES:

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

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 (3.31.25)
call to advance or retract time

retecmwfreanal (32.1.8)
call to read the ECMWF Reanalysis data and perform spatial interpolation

---

32.1.5 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)
```

USES:

```
use LIS_coreMod,            only : LIS_rc, LIS_domain
use LIS_baseforcingMod,     only : LIS_forc
use LIS_logMod,             only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,          only : LIS_readData
use ecmwfreanal_forcingMod, only : ecmwfreanal_struc
```

32.1.6 readecmwfreanalcrd (Source File: readecmwfreanalcrd.F90)

REVISION HISTORY:

```
26Jan2004; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readecmwfreanalcrd()
```

USES:

```
use ESMF_Mod
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.

32.1.7 readecmwfreanalmask (Source File: readecmwfreanalmask.F90)

REVISION HISTORY:

```
16 Apr 2002: Urszula Jambor; Initial Code
24 Nov 2003: Sujay Kumar; Included in LIS
```

INTERFACE:

```
subroutine readecmwfreanalmask(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 (32.1.9)
transform the ECMWF Reanalysis data to the LIS grid

32.1.8 retecmwfreanal (Source File: retecmwfreanal.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.
24 Nov 2003; Sujay Kumar; Included in LIS
15 Mar 2004; Matt Rodell; Fix test for beginning of restart run
```

INTERFACE:

```
subroutine retecmwfreanal(order, n, yr, mon, da, hr, ferror)
  USES:
  use LIS_coreMod,        only : LIS_rc,LIS_domain, LIS_localPet, LIS_masterproc
  use LIS_baseforcingMod, 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)    :: 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 (32.1.9)
transform the ECMWF REANALYSIS data to the LIS grid

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.25)
spatially interpolate the forcing data using conservative interpolation

geogfill2 (32.1.3)
fill in any grid points due to mismatched grids.

32.1.9 ecmwfreanalgrid_2_lisgrid (Source File: retecmwfreanal.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.

32.1.10 fillgaps (Source File: retecmwfreanal.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 )
```

32.1.11 time_interp_ecmwfreanal (Source File: time_interp_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 time_interp_ecmwfreanal(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_FORC_AttributesMod
use LIS_constantsMod, only  : LIS_CONST_SOLAR
use LIS_baseforcingMod, only : LIS_forc, LIS_FORC_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
```

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 (3.31.23)
converts the time to a date format

zterp (5.3.18)
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
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
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 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 onwards : T382 (1152x576) 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 T170

griduptime2 The time to switch GDAS resolution to T254

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 :: defineNativeGDAS      !defines the native resolution of
                                !the input data
```

PUBLIC TYPES:

```
public :: gdas_struc
```

33.1.1 defineNativeGDAS (Source File: gdas_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defineNativeGDAS()
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_date2time
```

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 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 routines invoked are:

readgdascrd (33.3.3)
reads the runtime options specified for GDAS data

date2time (**??**)
converts date to the real time format

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

---

33.2 Fortran: Module Interface gdasforcing_finalize (Source File: gdasforcing_finalize.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine gdasforcing_finalize
```

USES:

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

DESCRIPTION:

Routine to cleanup allocated structures for GDAS forcing.

33.3 Fortran: Module Interface gdasforcing_reset (Source File: gdasforcing_reset.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine gdasforcing_reset
```

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 getgdas (Source File: getgdas.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 getgdas(n)
```

USES:

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

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.
The arguments are:

n index of the nest

The routines invoked are:

LIS_tick (3.31.25)
determines the GDAS data times

create_gdasfilename (33.0.13)
Puts together appropriate file name for 3 hour intervals

gdasfilef06 (**??**)
Puts together appropriate file name for 6 hour intervals

retgdas (33.3.4)
Interpolates GDAS data to LIS grid

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

read_gdas_elev (33.3.2)
reads the native elevation of the GDAS data to be used for topographic adjustments to the forcing

---

33.3.2 read_gdas_elev (Source File: read_gdas_elev.F90)

REVISION HISTORY:

```
17Dec2004; Sujay Kumar; Initial Specificaton
```

INTERFACE:

```
subroutine read_gdas_elev(n, change)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_baseforcingMod, only : LIS_forc
use LIS_fileIOMod,      only : LIS_readData
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) :: change
integer, intent(in) :: n
```

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:

readforcingelev (**??**)
Abstract method to read the elevation of the forcing data in the same map projection used in LIS.

33.3.3 readgdascrd (Source File: readgdascrd.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readgdascrd()
```

USES:

```
use ESMF_Mod
use LIS_coreMod,   only : LIS_rc, LIS_config
use LIS_logMod,    only : LIS_logunit
use gdas_forcingMod, only: gdas_struc
```

DESCRIPTION:

This routine reads the options specific to GDAS forcing from the LIS configuration file.

33.3.4 retgdas (Source File: retgdas.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.
```

INTERFACE:

```
subroutine retgdas( order, n, 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_baseforcingMod, only : LIS_forc
use LIS_logMod,         only : LIS_logunit
use gdas_forcingMod,    only : gdas_struc


implicit none
```

ARGUMENTS:

```
integer, intent(in)           :: order
integer, intent(in)           :: n
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 (33.3.5)
spatially interpolates a GDAS variable

33.3.5 interp_gdas (Source File: retgdas.F90)

INTERFACE:

```
subroutine interp_gdas(n, kpds,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) :: kpds(200)
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

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 (5.1.25)
spatially interpolate the forcing data using conservative interpolation

---

33.3.6 time_interp_gdas (Source File: time_interp_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.
```

INTERFACE:

```
subroutine time_interp_gdas(n)
```

USES:

```
use ESMF_Mod
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_baseforcingMod, only  : LIS_forc, LIS_FORC_State
use LIS_logMod,         only  : LIS_logunit, LIS_verify, LIS_endrun
use gdas_forcingMod,    only  : gdas_struc

implicit none
```

ARGUMENTS:

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

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 (3.31.23)
converts the time to a date format

LIS_tick (3.31.25)
advances or retracts time by the specified amount

zterp (5.3.18)
zenith-angle based interpolation

---

34 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).

34.1 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.

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: defineNativeGEOS      !defines the native resolution of
                                !the input data
```

PUBLIC TYPES:

```
 ---------------------------------------------------------------------------
  public :: geos_struc
```

34.1.1 defineNativeGEOS (Source File: geos_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defineNativeGEOS
```

USES:

```
  use LIS_coreMod,    only:  LIS_rc
  use LIS_timeMgrMod, only : LIS_date2time

  implicit none
```

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 V
The routines invoked are:

readgeoscrd (34.1.6)
reads the runtime options specified for GEOS data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

---

34.1.2 geosforcing_finalize (Source File: geosforcing_finalize.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine geosforcing_finalize
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use geos_forcingMod, only : geos_struc
```

DESCRIPTION:

Routine to cleanup GEOS forcing related memory allocations.

34.1.3 getgeos (Source File: getgeos.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 getgeos(n)
```

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_baseforcingMod, only  : LIS_forc
use geos_forcingMod,    only  : geos_struc

implicit none
```

ARGUMENTS:

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

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 (3.31.25)
call to advance or retract time

geosfile (34.1.4)
Puts together appropriate file name for 3 hour intervals

readgeos (34.1.5)
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 (5.1.26)
computes the neighbor, weights for conservative interpolation

34.1.4 geosfile (Source File: getgeos.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

34.1.5 readgeos (Source File: readgeos.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 readgeos(order,n, name,tscount,ferror)
```

USES:

```
  use LIS_coreMod, only : LIS_rc,LIS_domain, LIS_masterproc
  use LIS_logMod,only : LIS_logunit, LIS_endrun, LIS_verify
  use LIS_baseforcingMod, only: LIS_forc
  use geos_forcingMod, only : geos_struc

  implicit none
```

ARGUMENTS:

```
integer, intent(in)          :: n
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 (5.1.25)
spatially interpolate the forcing data using conservative interpolation

---

34.1.6 readgeoscrd (Source File: readgeoscrd.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readgeoscrd()
```

USES:

```
use ESMF_Mod
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.

34.1.7 time_interp_geos (Source File: time_interp_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 time_interp_geos(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,          only : LIS_rc, LIS_domain, LIS_localPet
use LIS_baseforcingMod,   only : LIS_forc, LIS_FORC_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
```

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 (3.31.23)
converts the time to a date format

zterp (5.3.18)
zenith-angle based interpolation

---

35 GFS

This section describes the implementation of the GFS forcing data.

35.0.8 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

---

35.0.9 create_gfsfilename (Source File: create_gfsfilename.F90)

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

---

35.0.10 getgfs (Source File: getgfs.F90)

REVISION HISTORY:

```
08 Dec 2000: Urszula Jambor; Rewrote geteta.f in fortran90 to
             use GFS 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 GFS forecast files.
```

INTERFACE:

```
subroutine getgfs(n)
```

USES:

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

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 (3.31.25)
determines the GFS data times

create_gfsfilename (35.0.9)
Puts together appropriate file name for 3 hour intervals

create_gfs_f0backup_filename (35.0.8)
Puts together the backup filenames when F00 forecasts are not available.

retgfs (35.2.3)
Interpolates GFS data to LIS grid

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

read_gfs_elev (35.2.1)
reads the native elevation of the GFS data to be used for topographic adjustments to the forcing

---

35.1 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 onwards : T382 (1152x576) 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 T170

griduptime2 The time to switch GFS resolution to T254

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.

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: defineNativeGFS      !defines the native resolution of
                                !the input data
```

PUBLIC TYPES:

```
public :: gfs_struc
```

35.1.1 defineNativeGFS (Source File: gfs_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defineNativeGFS()
```

USES:

```
use LIS_coreMod,    only : LIS_rc
use LIS_timeMgrMod, only : LIS_date2time
```

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 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:

readgfscrd (35.2.2)
reads the runtime options specified for GFS data

date2time (**??**)
converts date to the real time format

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

---

35.2 Fortran: Module Interface gfsforcing_finalize (Source File: gfsforcing_finalize.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine gfsforcing_finalize
```

USES:

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

DESCRIPTION:

Routine to cleanup allocated structures for GFS forcing.

35.2.1 read_gfs_elev (Source File: read_gfs_elev.F90)

REVISION HISTORY:

```
17Dec2004; Sujay Kumar; Initial Specificaton
```

INTERFACE:

```
subroutine read_gfs_elev(n, change)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_baseforcingMod, 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) :: change
integer, intent(in) :: n
```

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:

readforcingelev (**??**)
Abstract method to read the elevation of the forcing data in the same map projection used in LIS.

35.2.2 readgfscrd (Source File: readgfscrd.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readgfscrd()
```

USES:

```
use ESMF_Mod
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.

35.2.3 retgfs (Source File: retgfs.F90)

REVISION HISTORY:

```
 14 Dec 2000: Urszula Jambor; Rewrote geteta.f to use GFS 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
              ungribgfs.  Interpolation now occurs in interp_gfs.
              Using GETGB avoids problems with the Oct2002 GFS
              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.
```

INTERFACE:

```
subroutine retgfs( order, n, 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_baseforcingMod, 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
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 (35.2.4)
spatially interpolates a GFS variable

35.2.4 interp_gfs (Source File: retgfs.F90)

INTERFACE:

```
subroutine interp_gfs(n, kpds,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) :: kpds(200)
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

kpds grib decoding array

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 (5.1.25)
spatially interpolate the forcing data using conservative interpolation

---

35.2.5 time_interp_gfs (Source File: time_interp_gfs.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.
```

INTERFACE:

```
subroutine time_interp_gfs(n)
```

USES:

```
use ESMF_Mod
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_baseforcingMod, only  : LIS_forc, LIS_FORC_State
use LIS_logMod,         only  : LIS_logunit, LIS_verify, LIS_endrun
use gfs_forcingMod,    only  : gfs_struc

implicit none
```

ARGUMENTS:

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

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 (3.31.23)
converts the time to a date format

LIS_tick (3.31.25)
advances or retracts time by the specified amount

zterp (5.3.18)
zenith-angle based interpolation

---

36 GSWP

This section describes the implementation of the forcing data used in the Global Soil Wetness Project Phase II.

36.0.6 finterp (Source File: finterp.F90)

REVISION HISTORY:

```
2002/08/16  15:55:18  guo
```

INTERFACE:

```
subroutine finterp(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

36.0.7 getgswp (Source File: getgswp.F90)

REVISION HISTORY:

```
20Feb2004; Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine getgswp(n)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_baseforcingMod, only : LIS_forc
use LIS_timeMgrMod,     only : LIS_get_nstep, LIS_tick
use LIS_logMod,         only : LIS_logunit
use gswp_forcingMod,    only : gswp_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 3-hrly, GSWP 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 (3.31.25)
determines the GSWP data times

readgswp (36.2.1)
Interpolates GSWP data to LIS grid

36.0.8 gswp_fillgaps (Source File: gswp_fillgaps.F90)

INTERFACE:

```
subroutine gswp_fillgaps(n,ip,varfield)
```

USES:

```
use LIS_coreMod,       only : LIS_rc,LIS_domain
use LIS_logMod,        only : LIS_logunit, LIS_endrun
use gswp_forcingMod,   only : gswp_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 GSWP. 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

36.1 Fortran: Module Interface gswp_forcingMod (Source File: gswp_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

nmif Number of forcing variables in the GSWP 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

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: defineNativeGSWP      !defines the native resolution of
                                !the input data
```

PUBLIC TYPES:

```
public :: gswp_struc
```

36.1.1 defineNativeGSWP (Source File: gswp_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defineNativeGSWP()
```

USES:

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

DESCRIPTION:

Defines the native resolution of the input forcing for GSWP 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:

readgswpcrd (36.2.7)
reads the runtime options specified for GSWP data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

gswp_mask (36.2.6)
reads the GSWP mask

36.2 Fortran: Module Interface gswpforcing_finalize (Source File: gswpforcing_finalize.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine gswpforcing_finalize
```

USES:

```
use gswp_forcingMod, only : gswp_struc
```

DESCRIPTION:

Routine to cleanup allocated structures for GSWP forcing.

36.2.1 readgswp (Source File: readgswp.F90)

REVISION HISTORY:

```
20Feb2004; Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine readgswp(order,n, yr,mo,da,hr,mn,ss)
```

USES:

```
#if ( defined USE_NETCDF )
  use netcdf
#endif
  use LIS_coreMod, only : LIS_rc,LIS_domain
  use LIS_baseforcingMod, only:LIS_forc
  use LIS_logMod, only : LIS_logunit
  use LIS_gswpMod, only : getgswp_timeindex
  use gswp_forcingMod, only : gswp_struc

  implicit none
```

ARGUMENTS:

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

DESCRIPTION:

For the given time, reads parameters from the GSWP 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 GSWP time index

gswp_process_forcing (36.2.2)
reads the GSWP forcing from the native NETCDF file

interp_gswp (36.2.3)
interpolates GSWP forcing to the LIS grid

gswp_fillgaps (36.0.8)
fills in the mismatches due to GSWP and LIS masks

36.2.2 gswp_process_forcing (Source File: readgswp.F90)

INTERFACE:

```
subroutine gswp_process_forcing(n, gswp1d, gswp_filled)
```

USES:

```
use gswp_forcingMod, only : gswp_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)                                        :: n
real, dimension(gswp_struc(n)%vector_len), intent(in)      :: gswp1d
real, dimension(gswp_struc(n)%ncold*gswp_struc(n)%nrold), intent(out) :: gswp_filled

real, dimension(gswp_struc(n)%ncold, gswp_struc(n)%nrold) :: gswp2d
```

DESCRIPTION:

Converts the native GSWP vector data to a gridded data, reordered to the LIS format. The arguments are:

n index of the nest

gswp1d native GSWP data as a 1-d vector.

gswp_filled Filled, 1-d vector GSWP field

gswp2d reordered gridded GSWP field

The routines invoked are:

gswp_expand_to_grid (36.2.4)
expand the gswp native data, change to the LIS ordering style.

gspw_2d_to_1d (36.2.5)
convert the 2d field to a 1d vector

36.2.3 interp_gswp (Source File: readgswp.F90)

INTERFACE:

```
subroutine interp_gswp(n,land_len,f,tempgswp)
```

USES:

```
    use LIS_coreMod,     only : LIS_rc, LIS_domain
    use gswp_forcingMod, only : gswp_struc

    implicit none
```

ARGUMENTS:

```
    integer, intent(in)                        :: n
    integer, intent(in)                        :: land_len
    real, dimension(land_len), intent(in)      :: f
    real, dimension(LIS_rc%ngrid(n)), intent(out) :: tempgswp
```

DESCRIPTION:

Interpolates the GSWP data onto the LIS grid. The arguments are:

n index of the nest

land_len size of the GSWP data

f native GSWP data

tempgswp interpolated GSWP data

The routines invoked are:

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.25)
spatially interpolate the forcing data using conservative interpolation

---

36.2.4 gswp_expand_to_grid (Source File: readgswp.F90)

INTERFACE:

```
subroutine gswp_expand_to_grid(n, array1, array2)
```

USES:

```
use gswp_forcingMod, only : gswp_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)                                          :: n
real, dimension(gswp_struc(n)%vector_len), intent(in)        :: array1
real, dimension(gswp_struc(n)%ncold, gswp_struc(n)%nrold), &
      intent(out) :: array2
```

DESCRIPTION:

changes the 1d native GSWP 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 GSWP data

array2 reordered 2-d GSWP data.

36.2.5 gswp_2d_to_1d (Source File: readgswp.F90)

INTERFACE:

```
subroutine gswp_2d_to_1d(n, array2, array1)
```

USES:

```
use gswp_forcingMod, only : gswp_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)                                          :: n
real, dimension(gswp_struc(n)%ncold, gswp_struc(n)%nrold),          &
      intent(in)                                             :: array2
real, dimension(gswp_struc(n)%ncold*gswp_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

36.2.6 gswp_mask (Source File: readgswp.F90)

INTERFACE:

```
subroutine gswp_mask()
#if ( defined USE_NETCDF )
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use gswp_forcingMod, only : gswp_struc
use netcdf
```

DESCRIPTION:

Reads the GSWP mask and computes the masking index.

36.2.7 readgswpcrd (Source File: readgswpcrd.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readgswpcrd()
```

USES:

```
use LIS_logMod, only : LIS_logunit
use LIS_coreMod, only : LIS_rc, LIS_config
use gswp_forcingMod, only : gswp_struc
use ESMF_Mod
```

DESCRIPTION:

This routine reads the options specific to GSWP forcing from the LIS configuration file.

36.2.8 time_interp_gswp (Source File: time_interp_gswp.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 time_interp_gswp(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,        only : LIS_rc,LIS_domain
use LIS_constantsMod,   only : LIS_CONST_SOLAR
use LIS_baseforcingMod, only : LIS_forc, LIS_FORC_State
use LIS_FORC_AttributesMod
use LIS_timeMgrMod,     only : LIS_time2date
use LIS_logMod,         only : LIS_logunit, LIS_verify, LIS_endrun
use gswp_forcingMod,    only : gswp_struc

implicit none
```

ARGUMENTS:

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

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 (3.31.23)
converts the time to a date format

zterp (5.3.18)
zenith-angle based interpolation

finterp (36.0.6)
routine to compute temporal disaggregation of forcing

---

37 GSWP1

This section describes the implementation of the forcing data used in the Global Soil Wetness Project Phase I.

37.0.9 getgswp1 (Source File: getgswp1.F90)

REVISION HISTORY:

```
11 Dec 2003: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine getgswp1(n)
```

USES:

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

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 (3.31.25)
determines the GSWP data times

gswp1file (37.0.10)
determines the GSWP1 data file

readgswp1 (37.2.1)
reads GSWP1 data to LIS grid

37.0.10 gswp1file (Source File: getgswp1.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

37.1 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

nmif Number of forcing variables in the GSWP 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

37.1.1 defineNativeGSWP1 (Source File: gswp1_forcingMod.F90)

REVISION HISTORY:

```
11 Dec 2003: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine defineNativeGSWP1()
```

USES:

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

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 V
The routines invoked are:

readgswp1crd (37.3.1)
reads the runtime options specified for GSWP1 data

37.2 Fortran: Module Interface gswp1forcing_finalize (Source File: gswp1forcing_finalize.F90

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine gswp1forcing_finalize
```

USES:

```
use gswp1_forcingMod, only : gswp1_struc
```

DESCRIPTION:

Routine to cleanup allocated structures for GSWP1 forcing.

37.2.1 readgswp1 (Source File: readgswp1.F90)

REVISION HISTORY:

```
11 Dec 2003: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine readgswp1(order,n, name,tscount,ferror)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain, LIS_localPet
use LIS_baseforcingMod, 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.

37.3 Core Functions of readgswp1

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. 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)

37.3.1 readgswp1crd (Source File: readgswp1crd.F90)

REVISION HISTORY:

```
11 Dec 2003: Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readgswp1crd()
```

USES:

```
use ESMF_Mod
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.

37.3.2 time_interp_gswp1 (Source File: time_interp_gswp1.F90)

REVISION HISTORY:

```
11 Dec 2003: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine time_interp_gswp1(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_baseforcingMod, only : LIS_forc, LIS_FORC_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
```

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 (3.31.23)
converts the time to a date format

zterp (5.3.18)
zenith-angle based interpolation

---

38 MERRA

This section describes the implementation of the MERRA forcing data.

38.0.3 getmerra (Source File: getmerra.F90)

REVISION HISTORY:

```
12 Oct 2009: Eric Kemp, initial version
```

INTERFACE:

```
subroutine getmerra(n)
```

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_baseforcingMod, only  : LIS_forc
use merra_forcingMod,   only  : merra_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates 6-hrly, 1 degree MERRA 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 (3.31.25)
call to advance or retract time

merrafile (38.0.4)
Puts together appropriate file name for 6 hour intervals

readmerra (38.1.3)
call to read the MERRA data and perform spatial interpolation

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

---

38.0.4 merrafile (Source File: getmerra.F90)

INTERFACE:

```
subroutine merrafile(name,merradir,yr,mo,da,hr,ncold)

  implicit none
```

ARGUMENTS:

```
  character(len=*), intent(in)  :: merradir
  integer, intent(in)           :: yr,mo,da,hr,ncold
  character(len=*), intent(out) :: name
```

DESCRIPTION:

This subroutine puts together MERRA file name for 6 hour file intervals
The arguments are:

merradir Name of the MERRA directory

yr year

mo month

da day of month

hr hour of day

ncold Number of columns (used as to determine the MERRA resolution)

name name of the timestamped MERRA file

38.1 Fortran: Module Interface merra_forcingMod (Source File: merra_forcingMod.F90)

This module contains variables and data structures that are used for the implementation of the MERRA forcing data. The data is global 1 degree dataset in latlon projection, and at 6 hourly intervals. The derived data type `merra_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

merratime1 The nearest, previous 6 hour instance of the incoming data (as a real time).

merratime2 The nearest, next 6 hour instance of the incoming data (as a real time).

merradir 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.

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: defineNativeMERRA      !defines the native resolution of
          !the input data
```

PUBLIC TYPES:

```
public :: merra_struc
```

38.1.1 defineNativeMERRA (Source File: merra_forcingMod.F90)

REVISION HISTORY:

```
12 Oct 2009:  Eric Kemp
```

INTERFACE:

```
subroutine defineNativeMERRA
```

USES:

```
use LIS_coreMod,    only:  LIS_rc
use LIS_timeMgrMod, only : LIS_date2time

implicit none
```

DESCRIPTION:

Defines the native resolution of the input forcing for MERRA 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:

readmerracrd (38.1.4)
reads the runtime options specified for MERRA data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

---

38.1.2 merraforcing_finalize (Source File: merraforcing_finalize.F90)

REVISION HISTORY:

```
12 Oct 2009, Eric Kemp
```

INTERFACE:

```
subroutine merraforcing_finalize
```

USES:

```
use LIS_coreMod,     only : LIS_rc
use merra_forcingMod, only : merra_struc
```

DESCRIPTION:

Routine to cleanup MERRA forcing related memory allocations.

38.1.3 readmerra (Source File: readmerra.F90)

REVISION HISTORY:

```
12 Oct 2009: Eric Kemp, Initial code
```

INTERFACE:

```
subroutine readmerra(order,n, name,tscount,ferror)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, LIS_masterproc
use LIS_logMod,only : LIS_logunit, LIS_endrun, LIS_verify
use LIS_baseforcingMod, only: LIS_forc
use merra_forcingMod, only : merra_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)          :: n
character(len=*), intent(in) :: name
integer, intent(in)          :: order, tscount
integer, intent(out)         :: ferror
```

DESCRIPTION:

For the given time, reads parameters from MERRA data, transforms into 9 LIS forcing parameters and interpolates to the LIS domain.

MERRA 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 6 hourly instance, order=2, read the next 6 hourly instance)

n index of the nest

name name of the 6 hour MERRA 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 (5.1.25)
spatially interpolate the forcing data using conservative interpolation

---

38.1.4 readmerracrd (Source File: readmerracrd.F90)

REVISION HISTORY:

```
12 Oct 2009; Eric Kemp, Initial code
```

INTERFACE:

```
subroutine readmerracrd()
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod, only : LIS_logunit
use merra_forcingMod, only : merra_struc
```

DESCRIPTION:

This routine reads the options specific to MERRA forcing from the LIS configuration file.

38.1.5 time_interp_merra (Source File: time_interp_merra.F90)

REVISION HISTORY:

```
12 Oct 2009: Eric Kemp, initial code
```

INTERFACE:

```
subroutine time_interp_merra(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,          only : LIS_rc, LIS_domain, LIS_localPet
use LIS_baseforcingMod,   only : LIS_forc, LIS_FORC_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 merra_forcingMod,      only : merra_struc

implicit none
```

ARGUMENTS:

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

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 (3.31.23)
converts the time to a date format

zterp (5.3.18)
zenith-angle based interpolation

---

39 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.

39.0.6 getprinceton (Source File: getprinceton.F90)

REVISION HISTORY:

```
26 Jan 2007: Hiroko Kato; Initial Specification adopted from LIS
```

INTERFACE:

```
subroutine getprinceton(n)
```

USES:

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

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 (3.31.25)
call to advance or retract time

retprinceton (39.1.5)
call to read the Princeton data and perform spatial interpolation

read_princeton_elev (39.1.3)
reads the native elevation of the Princeton data to be used for topographic adjustments to the forcing

---

39.1 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.

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 :: defineNativePRINCETON      !defines the native resolution of
                                !the input data
```

PUBLIC TYPES:

```
public :: princeton_struc
```

39.1.1 defineNativePRINCETON (Source File: princeton_forcingMod.F90)

REVISION HISTORY:

```
26Jan2007: Hiroko Kato; Initial Specification
```

INTERFACE:

```
subroutine defineNativePRINCETON()
```

USES:

```
use LIS_coreMod, only : LIS_rc

implicit none
```

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 V
The routines invoked are:

readprincetoncrd (39.1.4)
reads the runtime options specified for PRINCETON data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

read_princeton_elev (39.1.3)
reads the native elevation of the princeton data to be used for topographic adjustments to the forcing

---

39.1.2 princetonforcing_finalize (Source File: princetonforcing_finalize.F90)

REVISION HISTORY:

```
26 Jan 2007; Hiroko Kato, Initial Code
```

INTERFACE:

```
subroutine princetonforcing_finalize
```

USES:

```
use LIS_coreMod, only : LIS_rc
use princeton_forcingMod, only : princeton_struc
```

DESCRIPTION:

Routine to cleanup allocated structures for princeton forcing.

39.1.3 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)
```

USES:

```
use LIS_coreMod,         only : LIS_rc, LIS_domain
use LIS_baseforcingMod,  only : LIS_forc
use LIS_fileIOMod,       only : LIS_readData
use LIS_logMod,          only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use princeton_forcingMod,only : princeton_struc
implicit none
```

ARGUMENTS:

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

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:

readforcingelev (**??**)
call the abstract method to get the elevation data in the LIS projection.

39.1.4 readprincetoncrd (Source File: readprincetoncrd.F90)

REVISION HISTORY:

```
26 Jan 2007; Hiroko Kato, Initial Code adopted from readprincetoncrd.F90
25 Jun 2007; Hiroko Kato, upgraded to LISv5.0
```

INTERFACE:

```
subroutine readprincetoncrd()
```

USES:

```
use ESMF_Mod
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.

39.1.5 retprinceton (Source File: retprinceton.F90)

REVISION HISTORY:

```
26 Jan 2007: Hiroko Kato; Initial Specification adopted from LIS/retberg.F90
```

INTERFACE:

```
subroutine retprinceton( order, n, yr, mon, da, hr, ferror )
```

USES:

```
  use LIS_coreMod,          only : LIS_rc,LIS_domain, LIS_localPet, LIS_masterproc
  use LIS_baseforcingMod,   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_NETCDF)
  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)    :: 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 (39.1.6)
transform the PRINCETON data to the LIS grid

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

conserv_interp (5.1.25)
spatially interpolate the forcing data using conservative interpolation

---

39.1.6 princetongrid_2_lisgrid (Source File: retprinceton.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.

39.1.7 time_interp_princeton (Source File: time_interp_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 time_interp_princeton(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,         only : LIS_rc,LIS_domain
use LIS_constantsMod,    only : LIS_CONST_SOLAR
use LIS_FORC_AttributesMod
use LIS_baseforcingMod,  only : LIS_forc, LIS_FORC_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
```

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 (3.31.23)
converts the time to a date format

zterp (5.3.18)
zenith-angle based interpolation

---

40 Rhone-AGG

This is the forcing data used in the PILPS experiments for the Rhone-AGG domain.

40.0.8 getrhone (Source File: getrhone.F90)

REVISION HISTORY:

```
5 Nov 2003: Dave Mocko, Initial Specification
```

INTERFACE:

```
subroutine getrhone(n)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_baseforcingMod, only : LIS_forc
use LIS_timeMgrMod,     only : LIS_get_nstep, LIS_tick
use LIS_logMod,         only : LIS_logunit, LIS_endrun
use rhone_forcingMod,   only : rhone_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates RHONE 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 (3.31.25)
Determines RHONE data times

rhonefile (40.0.9)
Puts together appropriate file name for 3 hour intervals

readrhone (40.0.10)
Interpolates RHONE data to LIS grid

40.0.9 rhonefile (Source File: getrhone.F90)

This subroutine puts together RHONE file name

INTERFACE:

```
subroutine rhonefile(name,rhonedir,yr,mo,da,hr,ncold)

  implicit none
```

INPUT PARAMETERS:

```
  character*40, intent(in) :: rhonedir
  integer, intent(in)      :: yr,mo,da,hr,ncold
```

OUTPUT PARAMETERS:

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

40.0.10 readrhone (Source File: readrhone.F90)

REVISION HISTORY:

```
5 Nov 2003: Dave Mocko, Initial Specification
```

INTERFACE:

```
subroutine readrhone(n, order,name,tscount,ferror)
```

USES:

```
use LIS_coreMod,        only : LIS_rc,LIS_domain
use LIS_baseforcingMod, only : LIS_forc
use LIS_logMod,         only : LIS_logunit, LIS_verify, LIS_endrun
use rhone_forcingMod,   only : rhone_struc
```

DESCRIPTION:

Reads in RHONE data and performs interpolation to the LIS domain.

40.1 Core Functions of readrhone

bilinear_interp Interpolates RHONE data to LIS grid using bilinear interpolation

RHONE 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)

40.1.1 readrhonecrd (Source File: readrhonecrd.F90)

REVISION HISTORY:

```
11 Dec 2003: Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readrhonecrd()
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod, only    : LIS_logunit
use rhone_forcingMod, only : rhone_struc
```

DESCRIPTION:

Routine to read RHONE specific parameters from the LIS configuration file.

40.2 Fortran: Module Interface rhone_forcingMod (Source File: rhone_forcingMod.F90)

Contains routines and variables that define the native domain for RHONE model forcing.

40.2.1 defineNativeRHONE (Source File: rhone_forcingMod.F90)

REVISION HISTORY:

```
11 Dec 2003: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine defineNativeRHONE()
```

USES:

```
use LIS_coreMod, only: LIS_rc

implicit none
```

DESCRIPTION:

Defines the native resolution of the input forcing for RHONE AGG data.
The routines invoked are:

readrhonecrd (40.1.1)
reads the runtime options specified for RhoneAGG data

40.2.2 rhoneforcing_finalize (Source File: rhoneforcing_finalize.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine rhoneforcing_finalize
```

USES:

```
use LIS_coreMod, only : LIS_rc
use rhone_forcingMod, only : rhone_struc
```

DESCRIPTION:

Routine to cleanup RHONE forcing related memory allocations.

40.2.3 time_interp_rhone (Source File: time_interp_rhone.F90)

REVISION HISTORY:

```
5 Nov 2003: Dave Mocko, Initial Specification
```

INTERFACE:

```
subroutine time_interp_rhone(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only :LIS_rc, LIS_domain, LIS_localPet
use LIS_constantsMod, only  : LIS_CONST_SOLAR
use LIS_FORC_AttributesMod
use LIS_baseforcingMod, only: LIS_forc, LIS_FORC_State
use LIS_timeMgrMod,     only: LIS_time2date
use LIS_logMod, only : LIS_logunit, LIS_verify, LIS_endrun
use rhone_forcingMod, only : rhone_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates RHONE 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.

40.3 Core Functions of time_interp_rhone

zterp Performs zenith angle-based temporal interpolation

---

41 Template

This is a dummy placeholder implementation of a baseforcing, to be used as a template for new implementations.

41.0.1 gettemplate (Source File: gettemplate.F90)

INTERFACE:

```
subroutine gettemplate(n)
```

USES:

```
implicit none
```

ARGUMENTS:

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

DESCRIPTION:

The arguments are:

n index of the nest

41.1 Fortran: Module Interface template_forcingMod (Source File: template_forcingMod.F9

implicit none

PUBLIC MEMBER FUNCTIONS:

```
 ---------------------------------------------------------------------------
  public :: defineNativetemplate      !defines the native resolution of
                                  !the input data
 ---------------------------------------------------------------------------
```

PUBLIC TYPES:

```
 ---------------------------------------------------------------------------
  public :: template_struc
```

41.1.1 defineNativetemplate (Source File: template_forcingMod.F90)

INTERFACE:

```
subroutine defineNativetemplate()
```

USES:

41.2 Fortran: Module Interface templateforcing_finalize (Source File: templateforcing_finalize.F90)

INTERFACE:

```
subroutine templateforcing_finalize
```

USES:

41.3 Fortran: Module Interface templateforcing_reset (Source File: templateforcing_reset.F90)

INTERFACE:

```
subroutine templateforcing_reset
```

USES:

41.3.1 time_interp_template (Source File: time_interp_template.F90)

forcing data. INTERFACE:

```
subroutine time_interp_template(n)
```

USES:

```
implicit none
```

ARGUMENTS:

```
The arguments are:
\begin{description}
\item[n]
 index of the nest
\end{description}
```

Part XI
Supplemental forcing analyses in LIS

42 Overview

This section contains some examples of the supplemental forcing implementations in LIS. These implementations, by definition overwrite the corresponding base forcing analyses.

---

43 TRMM 3B42RT

This section describes the implementation of the precipitation product derived from the TRMM 3B42RT satellite precipitation analysis genereated by George Huffman at NASA GSFC.

43.1 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 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 :: defineNativeTRMM3B42RT      !defines the native resolution of
                                  !the input data
------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
------------------------------------------------------------------------------
 public :: TRMM3B42RT_struc
```

43.1.1 defineNativeTRMM3B42RT (Source File: TRMM3B42RT_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defineNativeTRMM3B42RT(findex)
```

USES:

```
  use LIS_coreMod, only: LIS_rc
  use LIS_timeMgrMod, only : LIS_date2time

  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 V
The routines invoked are:

readTRMM3B42RTcrd (43.1.7)
reads the runtime options specified for TRMM 3B42RT data

LIS_date2time (3.31.22)
converts date to the real time format

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

---

43.1.2 TRMM3B42RTforcing_finalize (Source File: TRMM3B42RTforcing_finalize.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine TRMM3B42RTforcing_finalize(findex)
```

USES:

```
use TRMM3B42RT_forcingMod, only : TRMM3B42RT_struc
```

DESCRIPTION:

Routine to cleanup TRMM 3B42RT forcing related memory allocations.

43.1.3 getTRMM3B42RT (Source File: getTRMM3B42RT.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 getTRMM3B42RT(n,findex,suppdata1, suppdata2)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_timeMgrMod,  only : LIS_time2date, LIS_tick, LIS_get_nstep
use LIS_logMod, only : LIS_logunit
use TRMM3B42RT_forcingMod, only : TRMM3B42RT_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

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 (3.31.25)
: determines the TRMM 3B42RT data times

TRMM3B42RTfile (43.1.4)
: Puts together appropriate file name for 6 hour intervals

glbprecip_TRMM3B42RT (43.1.5)
: Interpolates TRMM 3B42RT data to LIS grid

conserv_interp_input (5.1.26)
: computes the neighbor, weights for conservative interpolation

---

43.1.4 TRMM3B42RTfile (Source File: getTRMM3B42RT.F90)

INTERFACE:

```
subroutine TRMM3B42RTfile( name, TRMM3B42RTdir, yr, mo, da, hr)

  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

43.1.5 glbprecip_TRMM3B42RT (Source File: glbprecip_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 glbprecip_TRMM3B42RT (n, name_TRMM3B42RT, findex, suppdata, ferror_TRMM3B42RT )
```

USES:

```
  use LIS_coreMod, only : LIS_rc, LIS_domain,LIS_masterproc
  use LIS_logMod, only : LIS_logunit, LIS_getNextUnitNumber, &
       LIS_releaseUnitNumber
  use TRMM3B42RT_forcingMod, only : TRMM3B42RT_struc

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: n
  integer, intent(in) :: findex
  real                :: suppdata(LIS_rc%nsf(findex),LIS_rc%ngrid(n))
```

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 (43.1.6)
spatially interpolates the TRMM 3B42RT data

43.1.6 interp_TRMM3B42RT (Source File: interp_TRMM3B42RT.F90)

INTERFACE:

```
subroutine interp_TRMM3B42RT(n, nx, ny, finput, lis_gds, nc, nr, varfield)
```

USES:

```
use TRMM3B42RT_forcingMod, only : TRMM3B42RT_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer             :: nx
integer             :: ny
integer             :: nc
integer             :: nr
real                :: lis_gds(50)
real, dimension(nx,ny) :: finput
```

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 (5.1.25)
spatially interpolate the forcing data using conservative interpolation

---

43.1.7 readTRMM3B42RTcrd (Source File: readTRMM3B42RTcrd.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readTRMM3B42RTcrd()
```

USES:

```
use ESMF_Mod
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.

43.1.8 time_interp_TRMM3B42RT (Source File: time_interp_3B42RT.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 time_interp_TRMM3B42RT(n,findex,suppdata1, suppdata2)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only:LIS_rc,LIS_domain
use LIS_FORC_AttributesMod
use LIS_baseforcingMod, only : LIS_FORC_State
use LIS_logMod,         only : LIS_verify
use TRMM3B42RT_forcingMod, only : TRMM3B42RT_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep.
The arguments are:

n index of the nest

---

44 TRMM 3B42V6

This section describes the implementation of the precipitation product derived from the TRMM 3B42V6 satellite precipitation analysis genereated by George Huffman at NASA GSFC.

44.0.9 defineNativeTRMM3B42V6 (Source File: TRMM3B42V6_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
25Aug2006: Yudong Tian; Implementation for TRMM 3B42 V6
```

INTERFACE:

```
subroutine defineNativeTRMM3B42V6(findex)
```

USES:

```
use LIS_coreMod, only: LIS_rc
use LIS_timeMgrMod, only : LIS_date2time

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 V
The routines invoked are:

readTRMM3B42V6crd (44.0.15)
reads the runtime options specified for TRMM 3B42V6 data

LIS_date2time (3.31.22)
converts date to the real time format

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

---

44.0.10 TRMM3B42V6forcing_finalize (Source File: TRMM3B42V6forcing_finalize.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine TRMM3B42V6forcing_finalize(findex)
```

USES:

```
use TRMM3B42V6_forcingMod, only : TRMM3B42V6_struc
```

DESCRIPTION:

Routine to cleanup TRMM3B42V6 forcing related memory allocations.

44.0.11 getTRMM3B42V6 (Source File: getTRMM3B42V6.F90)

REVISION HISTORY:

```
25Aug2006: Yudong Tian; Modified for LIS 4.2 release
```

INTERFACE:

```
subroutine getTRMM3B42V6(n, findex, suppdata1, suppdata2)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_timeMgrMod, only : LIS_time2date, LIS_tick, LIS_get_nstep
use LIS_logMod, only : LIS_logunit
use TRMM3B42V6_forcingMod, only : TRMM3B42V6_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

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 (3.31.25)
determines the CMAP data times

cmapfile (49.1.4)
Puts together appropriate file name for 6 hour intervals

glbprecip_cmap (49.1.6)
Interpolates CMAP data to LIS grid

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

```
!-----------------------------------------------------------------------
! Set parameter to measure 1.5 hour time offset when using 3B42V6
!-----------------------------------------------------------------------
  gap = 0.0001712328767098370
!-----------------------------------------------------------------------
! Determine required observed precip data times
! (current, accumulation end time)
! Model current time
!-----------------------------------------------------------------------
  yr1 = LIS_rc%yr  !current time
  mo1 = LIS_rc%mo
  da1 = LIS_rc%da
```

```
  hr1 = LIS_rc%hr
  mn1 = LIS_rc%mn
  ss1 = 0
  ts1 = 0
  call LIS_tick( ctime, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )

!-------------------------------------------------------------------------
! 3B42V6 product end time
!-------------------------------------------------------------------------
  yr3 = LIS_rc%yr  !end accumulation time data
  mo3 = LIS_rc%mo
  da3 = LIS_rc%da
  hr3 = 3*(LIS_rc%hr/3)
  mn3 = 0
  ss3 = 0
  ts3 = 3*60*60
  call LIS_tick( ftime_TRMM3B42V6, doy3, gmt3, yr3, mo3, da3, hr3, mn3, ss3, ts3 )
  breaktime = ftime_TRMM3B42V6 - ctime
  datatime  = ftime_TRMM3B42V6
  fnametime = ftime_TRMM3B42V6
     if (breaktime .ge. gap) then
        call LIS_time2date( datatime, kdoy3, kgmt3, kyr3, &
             kmo3, kda3, khr3, kmn3 )
        call LIS_time2date( fnametime, mdoy3, mgmt3, myr3, &
             mmo3, mda3, mhr3, mmn3 )
        flag1 = 1
        if (khr3 == 24) khr3 = 0
        if (mhr3 == 24) mhr3 = 0
        if (kgmt3 .eq. 0.0 .and. flag2 .eq. 2) then
           kts3 = -25.5*60*60
           kss3 = 0
           call LIS_tick( datatime, kdoy3, kgmt3, kyr3, kmo3, &
                kda3, khr3, kmn3, kss3, kts3 )
           mts3 = -27*60*60
           mss3 = 0
           call LIS_tick( fnametime, mdoy3, mgmt3, myr3, mmo3, &
                mda3, mhr3, mmn3, mss3, mts3 )
        else
           kts3 = -1.5*60*60
           kss3 = 0
           call LIS_tick( datatime, kdoy3, kgmt3, kyr3, kmo3, &
                kda3, khr3, kmn3, kss3, kts3 )
           mts3 = -3*60*60
           mss3 = 0
           call LIS_tick( fnametime, mdoy3, mgmt3, myr3, mmo3, &
                mda3, mhr3, mmn3, mss3, mts3 )
        endif
        flag2 = 1
     else
        call LIS_time2date( datatime, kdoy3, kgmt3, kyr3, kmo3, &
             kda3, khr3, kmn3 )
        call LIS_time2date( fnametime, mdoy3, mgmt3, myr3, mmo3, &
             mda3, mhr3, mmn3 )
        if (LIS_get_nstep(LIS_rc, n).eq. 1) then
           if (kgmt3 .eq. 0) then
```

```
            mts3 = -24*60*60
            mss3 = 0
            call LIS_tick( fnametime, mdoy3, mgmt3, myr3, mmo3, &
                 mda3, mhr3, mmn3, mss3, mts3 )
            kts3 = -22.5*60*60
            kss3 = 0
            call LIS_tick( datatime, kdoy3, kgmt3, kyr3, kmo3, &
                 kda3, khr3, kmn3, kss3, kts3 )
         else
            mts3 = 0
            mss3 = 0
            call LIS_tick( fnametime, mdoy3, mgmt3, myr3, mmo3, &
                 mda3, mhr3, mmn3, mss3, mts3 )
            kts3 = 1.5*60*60
            kss3 = 0
            call LIS_tick( datatime, kdoy3, kgmt3, kyr3, kmo3, &
                 kda3, khr3, kmn3, kss3, kts3 )
         endif
      else
         flag1 = 2
         if (flag2 .eq. 1) then
            mts3 = 3*60*60
            mss3 = 0
            call LIS_tick( fnametime, mdoy3, mgmt3, myr3, mmo3, &
                 mda3, mhr3, mmn3, mss3, mts3 )
            kts3 = 3*60*60
            kss3 = 0
            call LIS_tick( datatime, kdoy3, kgmt3, kyr3, kmo3, &
                 kda3, khr3, kmn3, kss3, kts3 )
         endif
         flag2 = 2
      endif
   endif
!-----------------------------------------------------------------------
! Ensure that data is found during first time step
!-----------------------------------------------------------------------
   endtime_TRMM3B42V6 = 0
 if ( LIS_get_nstep(LIS_rc, n).eq. 1 &
 .or.LIS_rc%rstflag(n) .eq. 1) then
   endtime_TRMM3B42V6 = 1
   LIS_rc%rstflag(n) = 0
 endif


!-----------------------------------------------------------------------
! Check for and get 3B42V6 observed Precipitation data
!-----------------------------------------------------------------------
    if ( ctime > TRMM3B42V6_struc(n)%TRMM3B42V6time ) endtime_TRMM3B42V6 = 1
    if ( endtime_TRMM3B42V6 == 1 ) then  !get new time2 data
       ferror_TRMM3B42V6 = 0
       write(LIS_logunit, *) 'myr3, mmo3, mda3, mhr3:', myr3, mmo3, mda3, mhr3
       call TRMM3B42V6file( name, n, myr3, mmo3, mda3, mhr3 )
       write(LIS_logunit, *)'Getting new TRMM 3B42V6 satellite precip data:', name
       call glbprecip_TRMM3B42V6(n, name, findex, suppdata2, ferror_TRMM3B42V6, mhr3)
       TRMM3B42V6_struc(n)%TRMM3B42V6time = datatime
```

```
      endif
!endif
return
```

44.0.12 TRMM3B42V6file (Source File: getTRMM3B42V6.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
```

44.0.13 glbprecip_TRMM3B42V6.F90 (Source File: glbprecip_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
```

INTERFACE:

```
 subroutine glbprecip_TRMM3B42V6 (n, fname, findex, suppdata, ferror_TRMM3B42V6, filehr )
```

USES:

```
  use LIS_coreMod, only : LIS_rc, LIS_domain
  use LIS_logMod, only : LIS_logunit, LIS_getNextUnitNumber, &
       LIS_releaseUnitNumber
  use TRMM3B42V6_forcingMod, only : TRMM3B42V6_struc

   implicit none
```

ARGUMENTS:

```
   integer, intent(in) :: n
   character(len=80)   :: fname
   integer, intent(in) :: findex
   real                :: suppdata(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
   integer             :: ferror_TRMM3B42V6
   integer             :: filehr
```

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 6 hour TRMM 3B42V6 file

ferror_TRMM3B42V6 flag to indicate success of the call (=0 indicates success)

filehr current file hour

The routines invoked are:

interp_TRMM3B42V6 (44.0.14)
spatially interpolates the TRMM 3B42V6 data

```
!-----------------------------------------------------------------------
! Fill necessary arrays to assure not using old 3B42V6 data
!-----------------------------------------------------------------------
 allocate (precip_regrid(LIS_rc%lnc(n),LIS_rc%lnr(n)))
 precip = -1.0
 suppdata = -1.0
!-----------------------------------------------------------------------
! Find 3B42V6 precip data, read it in and assign to forcing precip array.
! Must reverse grid in latitude dimension to be consistent with LDAS grid
!-----------------------------------------------------------------------
 ftn = LIS_getNextUnitNumber()
 open(unit=ftn,file=fname, status='old', &
      &          access='direct',recl=xd*yd*4, &
      &          form='unformatted',iostat=ios)

 if (ios .eq. 0) then
    read (ftn,rec=1) precip
    Do j=1, xd
       Do i=1, yd
          if (precip(j, i) .LT. 0.0 ) precip(j, i) = 0.0    ! reset to 0 for weird values
       End Do
    End Do
!-----------------------------------------------------------------------
! Interpolating to desired LIS_domain and resolution
! Global precip datasets not used currently to force NLDAS
!-----------------------------------------------------------------------
   !print*, "Writing un-interpolated 3B42V6 precipitation out "
   !open(71, file="TRMM3B42V6-ungrid.1gd4r", access="direct", &
   !    recl=xd*yd*4, form="unformatted")
   ! write(71, rec=1) precip
   !close(71)

    nTRMM3B42V6 = TRMM3B42V6_struc(n)%ncold*TRMM3B42V6_struc(n)%nrold
    allocate(lb(nTRMM3B42V6))
    lb = .true.
    call interp_TRMM3B42V6(n, kpds, nTRMM3B42V6, precip, lb, LIS_rc%gridDesc, &
          LIS_rc%lnc(n),LIS_rc%lnr(n),precip_regrid)
    deallocate (lb)

   !print*, "Writing interpolated 3B42V6 precipitation out "
```

```
   !open(73, file="TRMM3B42V6-regrid.1gd4r", access="direct", &
   !    recl=LIS_rc%d%lnr*LIS_rc%d%lnc*4, form="unformatted")
   ! write(73, rec=1) precip_regrid
   !close(73)
   !print*, "Writing interpolated 3B42V6 precipitation out finished"

    do j = 1,LIS_rc%lnr(n)
       do i = 1,LIS_rc%lnc(n)
          if (precip_regrid(i,j) .ne. -1.0) then
             index1 = LIS_domain(n)%gindex(i,j)
             if(index1 .ne. -1) then
                suppdata(1,index1) = precip_regrid(i,j)   !here is mm/h
             endif
          endif
       enddo
    enddo

    ferror_TRMM3B42V6 = 1
    write(LIS_logunit,*) "Obtained 3B42 V6 precipitation data ", fname
 else
    write(LIS_logunit,*) "Missing 3B42 V6 precipitation data ", fname
    ferror_TRMM3B42V6 = 0
 endif
 call LIS_releaseUnitNumber(ftn)

 deallocate (precip_regrid)

end subroutine glbprecip_TRMM3B42V6

\markboth{Left}{Source File: interp\_TRMM3B42V6.F90,  Date: Wed Jun  9 09:46:31 EDT 2010
}

!-----------------------BEGIN NOTICE -- DO NOT EDIT-----------------------
! NASA Goddard Space Flight Center Land Information System (LIS)
!
! See RELEASE_NOTES.txt for more information.
!
! The LIS source code and documentation are not in the public domain
! and may not be freely distributed.  Only qualified entities may receive
! the source code and documentation.
!
! Qualified entities must be covered by a Software Usage Agreement.
! The Software Usage Agreement contains all the terms and conditions
! regarding the release of the LIS software.
!
! NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE
! SOFTWARE FOR ANY PURPOSE.  IT IS PROVIDED AS IS WITHOUT EXPRESS OR
! IMPLIED WARRANTY.  NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE
! LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE.
!
! See the Software Usage Agreement for the full disclaimer of warranty.
!
!-------------------------END NOTICE -- DO NOT EDIT-----------------------
```

44.0.14 interp_TRMM3B42V6.F90 (Source File: interp_TRMM3B42V6.F90)

INTERFACE:

```
subroutine interp_TRMM3B42V6(n, kpds, nTRMM3B42V6, f, lb, lis_gds, nc, nr, varfield)
```

USES:

```
use TRMM3B42V6_forcingMod, only : TRMM3B42V6_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
integer                :: nc
integer                :: nr
integer                :: nTRMM3B42V6
integer                :: kpds(200)
real                   :: lis_gds(50)
real                   :: f(nTRMM3B42V6)
logical*1              :: lb(nTRMM3B42V6)
real, dimension(nc,nr) :: varfield
```

DESCRIPTION:

This subroutine interpolates a given 3B42 V6 field to the LIS grid. The arguments are:

n index of the nest

kpds grib decoding array

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 (5.1.25)
spatially interpolate the forcing data using conservative interpolation

---

44.0.15 readTRMM3B42V6crd (Source File: readTRMM3B42V6crd.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
25 Aug 2006; Yudong Tian, Modification for 3B42, LIS 4.2 release
```

INTERFACE:

```
subroutine readTRMM3B42V6crd()
```

USES:

```
use ESMF_Mod
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.

44.0.16 time_interp_TRMM3B42V6 (Source File: time_interp_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.
```

INTERFACE:

```
subroutine time_interp_TRMM3B42V6(n,findex,suppdata1, suppdata2)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only: LIS_rc, LIS_domain
use LIS_suppforcingMod,  only : LIS_suppforc
use LIS_FORC_AttributesMod
use LIS_baseforcingMod,  only : LIS_FORC_State
use LIS_logMod,         only : LIS_verify
use TRMM3B42V6_forcingMod, only : TRMM3B42V6_struc
implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep.
The arguments are:

n index of the nest

---

45 AGRMET radiation

This section describes the implementation of the AGRMET radiation product.

45.0.17 agrrad_finalize (Source File: agrrad_finalize.F90)

REVISION HISTORY:

```
29 Jul 2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine agrrad_finalize(findex)
```

USES:

```
use agrrad_forcingMod,      only  : agrrad_struc

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

DESCRIPTION:

This routine deallocates and cleans up allocated memory structures for AGRMET radiation forcing implementation.

45.1 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:

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 :: defineNativeAGRRAD      !defines the native resolution of
                                  !the input data
```

PUBLIC TYPES:

```
public :: agrrad_struc
```

45.1.1 defineNativeAGRRAD (Source File: agrrad_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defineNativeAGRRAD(findex)
```

USES:

```
use LIS_coreMod, only: LIS_rc
use LIS_timeMgrMod, only : LIS_date2time

implicit none
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 V
The routines invoked are:

readagrradcrd (45.1.6)
reads the runtime options specified for AGRMET radiation data

LIS_date2time (3.31.22)
converts date to the real time format

bilinear_interp_input (5.1.2)
computes the neighbor, weights for conservative interpolation

---

45.1.2 getagrrad (Source File: getagrrad.F90)

REVISION HISTORY:

```
29 Jul 2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine getagrrad(n, findex, suppdata1, suppdata2)
```

USES:

```
use LIS_coreMod, only       : LIS_rc
use LIS_suppforcingMod, only  : LIS_suppforc
use LIS_timeMgrMod, only      : LIS_tick
use agrrad_forcingMod,      only  : agrrad_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

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

The routines invoked are:

readagrrad (45.1.3)
routines that reads the AGRMET radiation and interpolates it to the LIS projection.

LIS_tick (3.31.25)
computes the AGRMET read/processing times.

45.1.3 readagrrad (Source File: readagrrad.F90)

REVISION HISTORY:

```
29Jul2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readagrrad(n,m,suppdata,yr,mo,da,hr,error)
```

USES:

```
use LIS_coreMod,        only : LIS_rc,LIS_domain
use LIS_logMod,         only : LIS_logunit, LIS_endrun
use LIS_timeMgrMod,     only : LIS_tick
use agrrad_forcingMod,  only : agrrad_struc

implicit none
```

ARGUMENTS:

```
integer,   intent(IN)    :: n
integer,   intent(IN)    :: m
real                     :: suppdata(LIS_rc%nsf(m),LIS_rc%ngrid(n))
integer,   intent(INOUT) :: error
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

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:

LIS_tick (3.31.25)
determines the AGRMET data times

agrrad_filename (45.1.4)
generates the name of the AGRMET file to be read

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

agrmet_fillgaps (**??**)
fills in the mismatches due to AGRMET and LIS masks

45.1.4 agrrad_filename (Source File: readagrrad.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

45.1.5 agrrad_fillgaps (Source File: readagrrad.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

45.1.6 readagrradcrd (Source File: readagrradcrd.F90)

REVISION HISTORY:

```
29Jul2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readagrradcrd()
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_config
use agrrad_forcingMod, only : agrrad_struc

implicit none
```

DESCRIPTION:

This routine reads the options specific to AGRMET algorithms from the LIS configuration file.

45.1.7 time_interp_agrrad (Source File: time_interp_agrrad.F90)

REVISION HISTORY:

```
08 Dec 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine time_interp_agrrad(n,findex,suppdata1,suppdata2)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,        only : LIS_rc,LIS_domain
use LIS_FORC_AttributesMod
use LIS_baseforcingMod, only : LIS_FORC_State
use LIS_logMod,         only : LIS_verify
use agrrad_forcingMod,  only : agrrad_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

Temporally interpolates the AGRMET radiation forcing data to the model timestep.

---

46 AGRMET polar stereographic radiation

This section describes the implementation of the AGRMET polar stereographic radiation product.

46.0.8 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

46.0.9 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

46.1 Fortran: Module Interface agrradps_finalize (Source File: agrradps_finalize.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine agrradps_finalize
```

USES:

```
use LIS_coreMod, only : LIS_rc
use agrradps_forcingMod, only : agrradps_struc
```

DESCRIPTION:

Routine to cleanup AGRRADPS forcing related memory allocations.

46.2 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

USES:

```
implicit none
```

PUBLIC MEMBER FUNCTIONS:

```
---------------------------------------------------------------------------
 public :: definenativeAGRRADPS    !defines the native resolution of
                                   !the input data
---------------------------------------------------------------------------
```

PUBLIC TYPES:

```
---------------------------------------------------------------------------
 public :: agrradps_struc
```

46.2.1 definenativeAGRRADPS (Source File: agrradps_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine definenativeAGRRADPS()
```

USES:

```
  use LIS_coreMod, only: LIS_rc
  use LIS_timeMgrMod, only : LIS_date2time
  use LIS_logMod, only :LIS_logunit, LIS_endrun
```

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 V
The routines invoked are:

readagrradpscrd (46.2.6)
reads the runtime options specified for AGRRADPS data

LIS_date2time (3.31.22)
converts date to the real time format

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

---

46.2.2 getagrradps (Source File: getagrradps.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 getagrradps(n, findex, suppdata1, suppdata2)
```

USES:

```
use LIS_coreMod,         only : LIS_rc
use LIS_suppforcingMod,  only : LIS_suppforc
use agrradps_forcingMod, only : agrradps_struc
use LIS_timeMgrMod

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

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 (3.31.25)
determines the data times

readagrradps (**??**)
routines that reads the AGRMET radiation (polar stereographic) and interpolates it to the LIS projection.

46.2.3 interp_agrradpsvar1 (Source File: interp_agrradpsvar1.F90)

INTERFACE:

```
subroutine interp_agrradpsvar1(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

---

46.2.4 readagrradps1 (Source File: readagrradps1.F90)

REVISION HISTORY:

```
29Jul2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readagrradps1(n,m,suppdata,yr,mo,da,hr)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,         only : LIS_rc,LIS_domain
use agrradps_forcingMod, only : agrradps_struc
use LIS_logMod,          only : LIS_logunit, LIS_verify
use LIS_FORC_AttributesMod
use LIS_baseforcingMod,  only : LIS_FORC_State

implicit none
```

ARGUMENTS:

```
integer,   intent(IN)    :: n
integer,   intent(IN)    :: m
real                     :: suppdata(LIS_rc%nsf(m),LIS_rc%ngrid(n))
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 (**??**)
generates the name of the AGRMET file to be read

bilinear_interp (5.1.1)
spatially interpolate the forcing data using bilinear interpolation

agrmet_fillgaps (**??**)
fills in the mismatches due to AGRMET and LIS masks

46.2.5 agrradps_fillgaps1 (Source File: readagrradps1.F90)

INTERFACE:

```
subroutine agrradps_fillgaps1(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
```

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

46.2.6 readagrradpscrd (Source File: readagrradpscrd.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readagrradpscrd()
```

USES:

```
use agrradps_forcingMod, only : agrradps_struc
use LIS_coreMod,         only : LIS_config,LIS_rc
use ESMF_Mod
use LIS_logMod,          only : LIS_logunit
```

DESCRIPTION:

This routine reads the options specific to AGRRADPS forcing from the LIS configuration file.

46.2.7 time_interp_agrradps (Source File: time_interp_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 time_interp_agrradps(n,findex,suppdata1,suppdata2)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,         only : LIS_rc, LIS_domain
use LIS_FORC_AttributesMod
use LIS_baseforcingMod,  only : LIS_FORC_State
use LIS_logMod,          only : LIS_verify
use agrradps_forcingMod, only : agrradps_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

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

---

47 ARMS forcing

This section describes the implementation of the ARMS data.

47.1 Fortran: Module Interface arms_forcingMod (Source File: arms_forcingMod.F90)

Contains routines and data structures that are used for the implementation of the station data from various Walnut Gulch 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 `arms_struc` that includes the variables to specify the runtime options, and the calculation of weights for spatial interpolation.

REVISION HISTORY:

```
08 Jan 2004: Mike Tischler and Matt Garcia : Initial Specification
28 Jun 2009: Sujay Kumar: Reengineered for LIS version 6

use ESMF_Mod

implicit none
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
---------------------------------------------------------------------------
public :: defineNativeARMS      !defines the native resolution of
                                !the input data
---------------------------------------------------------------------------
```

PUBLIC TYPES:

```
---------------------------------------------------------------------------
public :: arms_struc
```

47.1.1 defineNativeARMS (Source File: arms_forcingMod.F90)

REVISION HISTORY:

```
28 Jun 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defineNativeARMS(findex)
```

USES:

```
use UTM_utils,  only : UTM2geo
use LIS_coreMod,only : LIS_rc
use LIS_timeMgrMod, only : LIS_date2time, LIS_calendar
use LIS_logMod, only : LIS_logunit, LIS_endrun, LIS_verify, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber

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

DESCRIPTION:

This routines reads the runtime configurations for using the ARMS 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:

readarmscrd (47.1.7)
reads the runtime options specified for ARMS station data

utm2geo (**??**)
converts utm coordinates (easting, northing) to latitude and longitude values.

compute_stnwts (5.1.24)
computes the weights for spatial interpolation

47.1.2 armsforcing_finalize (Source File: armsforcing_finalize.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine armsforcing_finalize(findex)
```

USES:

```
use arms_forcingMod, only : arms_struc
```

DESCRIPTION:

Routine to cleanup ARMS forcing related memory allocations.

47.1.3 getarms (Source File: getarms.F90)

REVISION HISTORY:

```
08 Dec 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine getarms(n, findex, suppdata1, suppdata2)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod, only : LIS_tick
use LIS_logMod, only : LIS_logunit
use arms_forcingMod, only : arms_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

Opens, reads, and interpolates the ARMS 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 (3.31.25)
determines the ARMS data times

readarms (47.1.4)
Interpolates the appropriate ARMS station data to LIS grid

47.1.4 readarms (Source File: readarms.F90)

REVISION HISTORY:

```
28 Jun 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine readarms(n, currTime, findex,suppdata)
```

USES:

```
use ESMF_Mod
use LIS_logMod, only      : LIS_logunit, LIS_verify
use LIS_timeMgrMod, only  : LIS_calendar, LIS_time2date
use LIS_coreMod, only     : LIS_rc,LIS_domain
use LIS_logMod,  only     : LIS_getNextUnitNumber, LIS_releaseUnitNumber, &
     LIS_endrun
use LIS_constantsMod, only : LIS_CONST_TKFRZ, LIS_CONST_STEBOL
use arms_forcingMod, only : arms_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)           :: n
real*8,  intent(in)           :: currTime
integer, intent(in)           :: findex
real                          :: suppdata(LIS_rc%nsf(findex),LIS_rc%ngrid(n))
```

DESCRIPTION:

For the given time, reads parameters from the Walnut Gulch station data (ASCII), transforms into LIS forcing parameters and interpolates to the LIS domain. Please note that the routine applies the met forcing (except precip) uniformly over the entire domain.
The arguments are:

n index of the nest

currTime time at which forcing data should be read

findex index of the supplemental forcing

suppdata output forcing array that stores the spatially interpolated forcing 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.

47.1.5 doytoCalday (Source File: readarms.F90)

INTERFACE:

```
subroutine doytoCalday(yr,doy, mo, da)

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: yr
  integer, intent(in) :: doy
  integer, intent(out) :: mo
  integer, intent(out) :: da
```

DESCRIPTION:

This subroutine converts the day of the year to the calendar month and day

---

47.1.6 RHtoq (Source File: readarms.F90)

INTERFACE:

```
subroutine RHtoq(TC,ph,RHp,Es,q)
  !Revision History:
  03/06/1998  Pablo Grunmann  Initial specification for Noah LSM as 'qdatap.f'
  06/24/1998  Pablo Grunmann  Eliminated subroutine SVP
   04 Dec 04  Matthew Garcia  Edit/update for incorporation in LISv3.2
   24 May 05  Matthew Garcia  Edit/update for LISv4.1 and calling arguments
  implicit none
  !INPUT ARGUMENTS:
  In:    TC     Temperature (C)
         ph     Pressure (hPa)
         RHp    Relative humidity (%)
  real, intent(in) :: TC,ph,RHp
  !OUTPUT ARGUMENTS:
  Out:   q      Specific humidity (Kg/Kg)
         Es     Saturation vapor pressure (hPa)
  real, intent(out) :: q,Es
```

DESCRIPTION:

OBTAIN SPECIFIC HUMIDITY (q) FROM RELATIVE HUMIDITY AND GIVEN PRESSURE AND TEMPERATURE.
FORMULAS AND CONSTANTS FROM ROGERS AND YAU, 1989: 'A SHORT COURSE IN CLOUD PHYSICS', PERGAMON PRESS, 3rd ED.

47.1.7 readarmscrd (Source File: readarmscrd.F90)

REVISION HISTORY:

```
28 Jun 2009; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readarmscrd()
```

USES:

```
use ESMF_Mod
use LIS_logMod, only : LIS_logunit
use LIS_coreMod, only : LIS_rc,LIS_config
use arms_forcingMod, only : arms_struc
```

DESCRIPTION:

This routine reads the options specific to ARMS station data forcing from the LIS configuration file.

47.1.8 time_interp_arms (Source File: time_interp_arms.F90)

REVISION HISTORY:

```
28 Jun 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine time_interp_arms(n,findex,suppdata1,suppdata2)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_baseforcingMod, only : LIS_FORC_State
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use arms_forcingMod, only : arms_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

Temporally interpolates the ARMS forcing data to the model timestep. All variables except precipitation is linearly interpolated.

---

48 CEOP forcing

This section describes the implementation of the CEOP station data.

48.1 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 :: defineNativeCEOP      !defines the native resolution of
                                !the input data
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
public :: ceop_struc
```

48.1.1 defineNativeCEOP (Source File: ceop_forcingMod.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defineNativeCEOP(findex)
```

USES:

```
use LIS_coreMod,only : LIS_rc
use LIS_timeMgrMod, only : LIS_date2time
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:

readceopcrd (48.1.6)
reads the runtime options specified for CEOP station data

LIS_date2time (3.31.22)
converts date to the real time format

compute_stnwts (5.1.24)
computes the weights for spatial interpolation

48.1.2 ceopforcing_finalize (Source File: ceopforcing_finalize.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine ceopforcing_finalize(findex)
```

USES:

```
use ceop_forcingMod, only : ceop_struc
```

DESCRIPTION:

Routine to cleanup CEOP forcing related memory allocations.

48.1.3 getceop (Source File: getceop.F90)

REVISION HISTORY:

```
08 Dec 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine getceop(n, findex, suppdata1, suppdata2)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod, only : LIS_tick
use LIS_logMod, only : LIS_logunit
use ceop_forcingMod, only : ceop_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

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 (3.31.25)
determines the CEOP data times

readceop (48.1.5)
Interpolates the appropriate CEOP station data to LIS grid

48.1.4 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

48.1.5 readceop (Source File: readceop.F90)

REVISION HISTORY:

```
08 Dec 2002: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine readceop(n,findex,suppdata)
```

USES:

```
use LIS_logMod, only        : LIS_logunit
use LIS_coreMod, only     : LIS_rc,LIS_domain
use ceop_forcingMod, only : ceop_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)           :: n
integer, intent(in)           :: findex
real                          :: suppdata(LIS_rc%nsf(findex),LIS_rc%ngrid(n))
```

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 (48.1.4)
reads the Bondville, IL data

readfpk (48.1.7)
reads the Fort Peck, MT data

readsgp (48.1.8)
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.

48.1.6 readceopcrd (Source File: readceopcrd.F90)

REVISION HISTORY:

```
08 Dec 2004; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readceopcrd()
```

USES:

```
use ESMF_Mod
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.

48.1.7 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

48.1.8 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

48.1.9 time_interp_ceop (Source File: time_interp_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 time_interp_ceop(n,findex,suppdata1,suppdata2)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_baseforcingMod, only : LIS_FORC_State
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
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

Temporally interpolates the CEOP forcing data to the model timestep. All variables except precipitation is linearly interpolated.

---

49 CMAP

This section describes the implementation of the precipitation forcing from the Climate Prediction Center's (CPC) Merged Analysis of Precipitation (CMAP)

49.1 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 :: defineNativeCMAP      !defines the native resolution of
                                !the input data
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: cmap_struc
```

49.1.1 defineNativeCMAP (Source File: cmap_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defineNativeCMAP(findex)
```

USES:

```
use LIS_coreMod, only: LIS_rc
use LIS_timeMgrMod, only : LIS_date2time
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 V
The routines invoked are:

readcmapcrd (49.1.8)
reads the runtime options specified for CMAP data

LIS_date2time (3.31.22)
converts date to the real time format

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

---

49.1.2 cmapforcing_finalize (Source File: cmapforcing_finalize.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine cmapforcing_finalize(findex)
```

USES:

```
use cmap_forcingMod, only : cmap_struc
```

DESCRIPTION:

Routine to cleanup CMAP forcing related memory allocations.

49.1.3 getcmap (Source File: getcmap.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 getcmap(n,findex, suppdata1, suppdata2)
```

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
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

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 (3.31.25)
determines the CMAP data times

cmapfile (49.1.4)
Puts together appropriate file name for 6 hour intervals

glbprecip_cmap (49.1.6)
Interpolates CMAP data to LIS grid

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

---

49.1.4 cmapfile (Source File: getcmap.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

49.1.5 cmapfile_old (Source File: getcmap.F90)

This subroutine puts together CMAP file name for 6 hour file intervals

INTERFACE:

```
 subroutine cmapfile_old( name, cmapdir, yr, mo, da, hr)

!-------------------------------------------------------------------------
! Make variables for the time used to create the file
! We don't want these variables being passed out
!-------------------------------------------------------------------------
  uyr = yr
  umo = mo
  uda = da
  uhr = 6*(hr/6)  !hour needs to be a multiple of 6 hours
  umn = 0
  uss = 0
  ts1 = -24*60*60 !one day interval to roll back date.

  name=''

  write(UNIT=temp, fmt='(a40)') cmapdir
  read(UNIT=temp, fmt='(80a1)') (fbase(i), i=1,80)

  write(UNIT=temp, fmt='(a1,i4,a1)') '/', uyr, '/'
  read(UNIT=temp, fmt='(6a1)') fdir
  do i = 1, 6
     if ( fdir(i) == ' ' ) fdir(i) = '0'
  end do

  write(UNIT=temp, fmt='(a11)') '_drean_smth'
  read (UNIT=temp, fmt='(11a1)') (fsubs(i), i=1,11)

  write(UNIT=temp, fmt='(i4,i2,i2,i2)') uyr, umo, uda, uhr
  read(UNIT=temp, fmt='(10a1)') ftime
  do i = 1, 10
     if ( ftime(i) == ' ' ) ftime(i) = '0'
  end do
```

```
write(UNIT=temp, fmt='(a4)') '.grb'
read (UNIT=temp, fmt='(4a1)') (fsubs2(i), i=1,4)
c = 0
do i = 1, 80
   if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
end do

write(UNIT=temp, fmt='(80a1)') (fbase(i), i=1,c), (fdir(i), i=1,6),  &
                     (ftime(i), i=1,10),(fsubs(i), i=1,11)

read(UNIT=temp, fmt='(a80)') name

return
```

49.1.6 glbprecip_cmap (Source File: glbprecip_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
```

INTERFACE:

```
subroutine glbprecip_cmap( n, fname, findex, suppdata, ferror_cmap, filehr)
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_logMod,         only : LIS_logunit
use LIS_suppforcingMod, only : LIS_suppforc
use cmap_forcingMod,    only : cmap_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
character(len=80)   :: fname
integer, intent(in) :: findex
real                :: suppdata(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
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 (49.1.7)
spatially interpolates the CMAP data

49.1.7 interp_cmap (Source File: interp_cmap.F90)

INTERFACE:

```
subroutine interp_cmap(n, kpds,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
integer                :: kpds(200)
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

kpds grib decoding array

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 (5.1.25)
spatially interpolate the forcing data using conservative interpolation

---

49.1.8 readcmapcrd (Source File: readcmapcrd.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcmapcrd()
```

USES:

```
use ESMF_Mod
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.

49.1.9 time_interp_cmap (Source File: time_interp_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 time_interp_cmap(n,findex,suppdata1,suppdata2)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only: LIS_rc, LIS_domain
use LIS_baseforcingMod, only : LIS_FORC_State
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
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep.
The arguments are:

n index of the nest

---

50 CMORPH

This section describes the implementation of the precipitation product from CPC's MORPHing technique, known as CMORPH.

50.0.10 cmorforcing_finalize (Source File: cmorforcing_finalize.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine cmorforcing_finalize(findex)
```

USES:

```
use cmorph_forcingMod, only : cmor_struc
```

DESCRIPTION:

Routine to cleanup CMORPH forcing related memory allocations.

50.1 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 `cmor_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 :: defineNativeCMORPH      !defines the native resolution of
                                  !the input data
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: cmor_struc
```

50.1.1 defineNativeCMORPH (Source File: cmorph_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
06Jan2006: Yudong Tian; modification for LISv4.2
```

INTERFACE:

```
subroutine defineNativeCMORPH(findex)
```

USES:

```
use LIS_coreMod, only: LIS_rc
use LIS_timeMgrMod, only : LIS_date2time

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:

readcmorcrd (50.1.6)
reads the runtime options specified for CMORPH data

LIS_date2time (3.31.22)
converts date to the real time format

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

---

50.1.2 getcmor (Source File: getcmor.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 getcmor(n, findex, suppdata1, suppdata2)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_timeMgrMod, only : LIS_tick, LIS_get_nstep
use cmorph_forcingMod, only :cmor_struc
use LIS_logMod, only : LIS_logunit

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

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 (3.31.25)
determines the CMORPH data times

cmorfile (50.1.3)
Puts together appropriate file name for 6 hour intervals

glbprecip_cmor (50.1.4)
Interpolates CMORPH data to LIS grid

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

---

50.1.3 cmorfile (Source File: getcmor.F90)

INTERFACE:

```
subroutine cmorfile( name, cmordir, yr, mo, da, hr)

  implicit none
```

ARGUMENTS:

```
  character(len=*) :: name
  character(len=*) :: cmordir
  integer            :: yr, mo, da, hr
```

DESCRIPTION:

This subroutine puts together CMORPH file name for 30min file intervals
The arguments are:

cmordir Name of the CMORPH directory

yr year

mo month

da day of month

hr hour of day

name name of the timestamped CMORPH file

50.1.4 glbprecip_cmor (Source File: glbprecip_cmor.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 glbprecip_cmor (n, name_cmor, findex, suppdata, ferror_cmor, iflg )
```

USES:

```
  use LIS_coreMod, only : LIS_rc, LIS_domain, LIS_masterproc
  use LIS_logMod, only : LIS_logunit, LIS_getNextUnitNumber, &
       LIS_releaseUnitNumber
  use cmorph_forcingMod, only : cmor_struc

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: n
  integer, intent(in) :: findex
  real                :: suppdata(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

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_cmor name of the CMORPH file

ferror_cmor flag to indicate success of the call (=0 indicates success)

iflg flag indicating which 1/2 hour to read

The routines invoked are:

interp_cmor (50.1.5)
spatially interpolates the CMORPH data

50.1.5 interp_cmor (Source File: interp_cmor.F90)

INTERFACE:

```
subroutine interp_cmor(n, nx, ny, finput, lis_gds, nc, nr, varfield)
```

USES:

```
use cmorph_forcingMod, only : cmor_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 (5.1.25)
spatially interpolate the forcing data using conservative interpolation

---

50.1.6 readcmorcrd (Source File: readcmorcrd.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 readcmorcrd()
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_config, LIS_rc
use LIS_logMod, only : LIS_logunit
use cmorph_forcingMod, only : cmor_struc
```

DESCRIPTION:

This routine reads the options specific to CMORPH forcing from the LIS configuration file.

50.1.7 time_interp_cmor (Source File: time_interp_cmor.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 time_interp_cmor(n,findex, suppdata1, suppdata2)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,        only : LIS_rc,LIS_domain
use LIS_FORC_AttributesMod
use LIS_baseforcingMod, only : LIS_FORC_State
use LIS_logMod,         only : LIS_verify
use cmorph_forcingMod,  only : cmor_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep.
The arguments are:

n index of the nest

---

51 GDAS LSWG forcing

This section describes the implementation of the GDAS LSWG data.

51.1 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_Mod

implicit none

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

PUBLIC MEMBER FUNCTIONS:

```
-------------------------------------------------------------------------------
public :: defineNativeGDASLSWG      !defines the native resolution of
                                 !the input data
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
public :: gdasLSWG_struc
```

51.1.1 defineNativeGdasLSWG (Source File: gdasLSWG_forcingMod.F90)

REVISION HISTORY:

```
30 APR 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defineNativeGdasLSWG(findex)
```

USES:

```
  use LIS_coreMod,    only : LIS_rc
  use LIS_timeMgrMod, only : LIS_calendar, LIS_date2time
  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.

51.2 Fortran: Module Interface gdasLSWGforcing_finalize (Source File: gdasLSWGforcing_finalize.F90)

REVISION HISTORY:

```
17 March 2009; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine gdasLSWGforcing_finalize
```

USES:

DESCRIPTION:

Routine to cleanup allocated structures for GDAS forcing.

51.2.1 getgdasLSWG (Source File: getgdasLSWG.F90)

REVISION HISTORY:

```
20 Oct 2009; Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine getgdasLSWG(n, findex, suppdata1, suppdata2)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,        only : LIS_rc
use LIS_baseforcingMod, 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                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(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.

51.2.2 readgdasLSWGcrd (Source File: readgdasLSWGcrd.F90)

REVISION HISTORY:

```
20 Oct 2009; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readgdasLSWGcrd()
```

USES:

```
use ESMF_Mod
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.

51.2.3 retgdasLSWG (Source File: retgdasLSWG.F90)

REVISION HISTORY:

```
20 Oct 2009: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine retgdasLSWG(n, order, m, suppdata)
```

USES:

```
use ESMF_Mod
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                  :: suppdata(LIS_rc%nsf(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.

51.2.4 time_interp_gdasLSWG (Source File: time_interp_gdasLSWG.F90)

REVISION HISTORY:

```
20 Oct 2009: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine time_interp_gdasLSWG(n, findex, suppdata1, suppdata2)
```

USES:

```
  use ESMF_Mod
  use LIS_coreMod,    only : LIS_rc, LIS_domain
  use LIS_logMod,     only : LIS_verify
  use LIS_FORC_AttributesMod
  use LIS_baseforcingMod, only : LIS_FORC_State
  use gdasLSWG_forcingMod,  only : gdasLSWG_struc
#if (defined CRTM)
  USE units_conversion
  USE Type_Kinds, ONLY: fp
#endif

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: n
  integer, intent(in) :: findex
  real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
  real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

This routine performs the temporal interpolation of the GDAS data to the LIS model timestep

---

52 NLDAS

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.

52.0.5 apply_conusmask (Source File: apply_conusmask.F90)

INTERFACE:

```
subroutine apply_conusmask(n, nc,nr,varfield)
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use nldas_forcingMod,  only : nldas_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: n
integer, intent(in)   :: nc
integer, intent(in)   :: nr
real, intent(inout)   :: varfield(nc,nr)
```

DESCRIPTION:

52.0.6 getnldas (Source File: getnldas.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
```

INTERFACE:

```
subroutine getnldas(n, findex, suppdata1, suppdata2)
```

USES:

```
use LIS_coreMod, only        : LIS_rc, LIS_domain
use LIS_timeMgrMod,     only : LIS_tick
use LIS_logMod,         only : LIS_logunit, LIS_endrun
use nldas_forcingMod,   only : nldas_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

Opens, reads, and interpolates hourly, NLDAS 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 (3.31.25)
determines the NLDAS data times

nldasfile (52.0.7)
Puts together appropriate timestamped filename

retnldas (52.2.4)
Interpolates NLDAS data to LIS grid

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

---

52.0.7 nldasfile (Source File: getnldas.F90)

REVISION HISTORY:

```
04 Sep 2001: Brian Cosgrove; Use of NASA data enabled, updated
             reading of data directory structure to read new format
```

INTERFACE:

```
subroutine nldasfile(name,nldasdir,yr,mo,da,hr)

  implicit none
```

ARGUMENTS:

```
  character*80, intent(out)::  name
  character*40, intent(in) ::  nldasdir
  integer, intent(in)      :: yr,mo,da,hr
```

DESCRIPTION:

This subroutine puts together NLDAS file name for 1 hour file intervals
The arguments are:

nldasdir Name of the NLDAS directory

yr year

mo month

da day of month

hr hour of day

name name of the timestamped NLDAS file

52.0.8 interp_nldas (Source File: interp_nldas.F90)

INTERFACE:

```
subroutine interp_nldas(n, findex, kpds, nldas,f,lb,lis_gds,nc,nr, &
     varfield)
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use nldas_forcingMod,  only : nldas_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: n
integer, intent(in)   :: findex
integer, intent(in)   :: kpds(200)
integer, intent(in)   :: nldas
real, intent(in)      :: f(nldas)
logical*1, intent(in) :: lb(nldas)
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 NLDAS 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 (5.1.25)
spatially interpolate the forcing data using conservative interpolation

neighbor_interp (5.3.10)
spatially interpolate the forcing data using neighbor interpolation

52.0.9 nldas_ec_removal (Source File: nldas_ec_removal.F90)

REVISION HISTORY:

```
20 Oct 2006: Kristi Arsenault; Adapted elevation correction code
             to remove such a correction from NLDAS fields
```

INTERFACE:

```
subroutine nldas_ec_removal( nest, point, force_tmp, force_hum, &
                             force_lwd, force_prs )
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use nldas_forcingMod,   only : nldas_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 (NLDAS) 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

52.1 Fortran: Module Interface nldas_forcingMod (Source File: nldas_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.
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 `nldas_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

nldastime1 The nearest, previous hourly instance of the incoming data (as a real time).

nldastime2 The nearest, next hourly instance of the incoming data (as a real time).

nldasdir Directory containing the input data

elevfile File with the elevation definition for the input data.

mi Number of points in the input grid

orig_ediff Original NLDAS 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.

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: defineNativeNLDAS      !defines the native resolution of
                                 !the input data
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: nldas_struc
```

52.1.1 defineNativeNLDAS (Source File: nldas_forcingMod.F90)

REVISION HISTORY:

```
02Feb2004: Sujay Kumar; Initial Specification
20Oct2007: Kristi Arsenault; Changed to EDAS elev file for new diff
```

INTERFACE:

```
subroutine defineNativeNLDAS(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain
use LIS_logMod,     only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use map_utils,      only : ij_to_latlon

implicit none

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

DESCRIPTION:

Defines the native resolution of the input forcing for NLDAS 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:

readnldascrd (52.2.3)
reads the runtime options specified for NLDAS data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

neighbor_interp_input (5.3.11)
computes the neighbor, weights for nearest neighbor interpolation

read_edas_elev (52.2.2)
reads the native elevation of the EDAS data to be used for topographic adjustments to the baseforcing data

---

52.2 Fortran: Module Interface nldasforcing_finalize (Source File: nldasforcing_finalize.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine nldasforcing_finalize(findex)
```

USES:

```
use LIS_coreMod,      only : LIS_rc
use nldas_forcingMod, only : nldas_struc
```

DESCRIPTION:

Routine to cleanup nldas forcing related memory allocations.

52.2.1 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 nldas_forcingMod, only : nldas_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:

getNextUnitNumber (**??**)
obtain the next available unit number

releaseUnitNumber (**??**)
release the unit number after use.

52.2.2 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_suppforcingMod, only : LIS_suppforc
use LIS_logMod,         only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
use LIS_fileIOMod,      only : LIS_readData
use nldas_forcingMod,   only : nldas_struc
implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates NLDAS-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:

readforcingelev (**??**)
abstract method to read EDAS elevation data in the map projection used in the LIS model grid.

52.2.3 readnldascrd (Source File: readnldascrd.F90)

REVISION HISTORY:

```
02Feb2004; Sujay Kumar, Initial Code
04Mar2007; Kristi Arsenault, Implemented EDAS Elevation Correction
```

INTERFACE:

```
subroutine readnldascrd()
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod,  only :LIS_logunit, LIS_verify
use nldas_forcingMod, only : nldas_struc
```

DESCRIPTION:

This routine reads the options specific to NLDAS forcing from the LIS configuration file.

52.2.4 retnldas (Source File: retnldas.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 Code to Read EDAS Height Field
```

INTERFACE:

```
subroutine retnldas(n, findex, suppdata, name,ferror)
```

USES:

```
use LIS_coreMod, only        : LIS_rc, LIS_domain, LIS_localPet
use LIS_logMod, only         : LIS_logunit
use nldas_forcingMod, only : nldas_struc

implicit none
```

ARGUMENTS:

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

```
integer, intent(in)      :: findex
real                     :: suppdata(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
character*80, intent(in) :: name
integer, intent(out)     :: ferror
```

DESCRIPTION:

For the given time, reads parameters from NLDAS data, 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 hourly instance, order=2, read the next hourly instance)

n index of the nest

name name of the 3 hour GDAS forecast file

ferror flag to indicate success of the call (=0 indicates success)

The routines invoked are:

interp_nldas (52.0.8)
spatially interpolates a NLDAS variable

52.2.5 time_interp_nldas (Source File: time_interp_nldas.F90)

REVISION HISTORY:

```
02Feb2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine time_interp_nldas(n, findex, suppdata1, suppdata2)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_constantsMod, only  : LIS_CONST_SOLAR
use LIS_FORC_AttributesMod
use LIS_baseforcingMod, only : LIS_FORC_State
use LIS_timeMgrMod, only : LIS_tick, LIS_time2date
use LIS_logMod, only :LIS_logunit, LIS_verify
use nldas_forcingMod, only : nldas_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

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 (3.31.23)
converts the time to a date format

LIS_tick (3.31.25)
advances or retracts time by the specified amount

zterp (5.3.18)
zenith-angle based interpolation

---

53 NLDAS2

The atmospheric forcing used in the North American Land Data Assimilation System Phase II (NLDAS1) features products at an hourly 0.125° spatial resolution, over the continental United States.

53.0.6 getnldas2 (Source File: getnldas2.F90)

REVISION HISTORY:

```
02Feb 2004 : Sujay Kumar ; Initial Version in LIS
22Aug 2007 : Chuck Alonge ; Updated for NLDAS2 Forcing
```

INTERFACE:

```
subroutine getnldas2(n,findex,suppdata1,suppdata2)
```

USES:

```
use LIS_coreMod, only        : LIS_rc, LIS_domain
use LIS_timeMgrMod,     only : LIS_tick
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
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

Opens, reads, and interpolates hourly, NLDAS2 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 (3.31.25)
determines the NLDAS2 data times

nldas2filea (53.0.7)
Puts together appropriate timestamped filename - a data

nldas2fileb (53.0.7)
Puts together appropriate timestamped filename - b data

retnldas2a (**??**)
Reads and Interpolates NLDAS2 A data to LIS grid

retnldas2b (**??**)
Reads and Interpolates NLDAS2 B data to LIS grid

53.0.7 nldas2filea (Source File: getnldas2.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 NLDAS2 data
```

INTERFACE:

```
subroutine nldas2filea(name,nldas2dir,yr,mo,da,hr)

  implicit none
```

ARGUMENTS:

```
  character*80, intent(out)::  name
  character*40, intent(in) ::  nldas2dir
  integer, intent(in)      :: yr,mo,da,hr
```

DESCRIPTION:

This subroutine puts together NLDAS2 file name for 1 hour file intervals
The arguments are:

nldas2dir Name of the NLDAS2 directory

yr year

mo month

da day of month

hr hour of day

name name of the timestamped NLDAS2 file

53.0.8 nldas2fileb (Source File: getnldas2.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 NLDAS2 data
```

INTERFACE:

```
subroutine nldas2fileb(name,nldas2dir,yr,mo,da,hr)

  implicit none
```

ARGUMENTS:

```
  character*80, intent(out)::  name
  character*40, intent(in) ::  nldas2dir
  integer, intent(in)      :: yr,mo,da,hr
```

DESCRIPTION:

This subroutine puts together NLDAS2 file name for 3 hour file intervals
The arguments are:

nldas2dir Name of the NLDAS2 directory

yr year

mo month

da day of month

hr hour of day

name name of the timestamped NLDAS2 file

53.1 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/nldas2/nldas2forcing.pdf
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

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)

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.

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: defineNativeNLDAS2      !defines the native resolution of
                                 !the input data
```

PUBLIC TYPES:

```
public :: nldas2_struc
```

53.1.1 defineNativeNLDAS2 (Source File: nldas2_forcingMod.F90)

REVISION HISTORY:

```
02Feb2004: Sujay Kumar; Initial Specification
24 Aug 2007: Chuck Alonge; Modified for use with NLDAS2 data
```

INTERFACE:

```
subroutine defineNativeNLDAS2(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc

implicit none

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

DESCRIPTION:

Defines the native resolution of the input forcing for NLDAS2 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:

readnldas2crd (53.2.2)
reads the runtime options specified for NLDAS2 data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

read_nldas2_elev (53.2.1)
reads the native elevation of the NLDAS2 data to be used for topographic adjustments to the forcing

---

53.2 Fortran: Module Interface nldas2forcing_finalize (Source File: nldas2forcing_finalize.F9

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
24 Aug 2007: Chuck Alonge; Modified for use with NLDAS2 data
```

INTERFACE:

```
subroutine nldas2forcing_finalize(findex)
```

USES:

```
use LIS_coreMod,       only : LIS_rc
use nldas2_forcingMod, only : nldas2_struc
```

DESCRIPTION:

Routine to cleanup nldas2 forcing related memory allocations.

53.2.1 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)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_baseforcingMod, only : LIS_forc
use LIS_fileIOMod,      only : LIS_readData
use LIS_logMod, only : LIS_logunit
use nldas2_forcingMod, only : nldas2_struc

implicit none
```

ARGUMENTS:

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

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

53.2.2 readnldas2crd (Source File: readnldas2crd.F90)

REVISION HISTORY:

```
02Feb2004; Sujay Kumar, Initial Code
24 Aug 2007: Chuck Alonge; Modified for use with NLDAS2 data
```

INTERFACE:

```
subroutine readnldas2crd()
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod,  only : LIS_logunit
use nldas2_forcingMod, only : nldas2_struc
```

DESCRIPTION:

This routine reads the options specific to NLDAS2 forcing from the LIS configuration file.

53.2.3 retnldas2 (Source File: retnldas2a.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
24 Aug 2007: Chuck Alonge; Modified for use with NLDAS2 data
```

INTERFACE:

```
subroutine retnldas2a(n, findex, suppdata, name,ferror)
```

USES:

```
use LIS_coreMod, only        : LIS_rc, LIS_domain
use LIS_logMod, only         : LIS_logunit
use nldas2_forcingMod,only : nldas2_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: findex
integer, intent(in)      :: n
real                     :: suppdata(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
character*80, intent(in) :: name
integer, intent(out)     :: ferror
```

DESCRIPTION:

For the given time, reads parameters from NLDAS2 data, 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 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 (=0 indicates success)

The routines invoked are:

interp_nldas2 (53.2.4)
spatially interpolates a NLDAS2 variable

53.2.4 interp_nldas2 (Source File: retnldas2a.F90)

INTERFACE:

```
subroutine interp_nldas2(n,findex,kpds, nldas2,f,lb,lis_gds,nc,nr, &
     varfield)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use nldas2_forcingMod, only :nldas2_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: n
integer, intent(in)   :: findex
integer, intent(in)   :: kpds(200)
integer, intent(in)   :: nldas2
real, intent(in)     :: f(nldas2)
logical*1, intent(in) :: lb(nldas2)
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 NLDAS2 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 (5.1.25)
spatially interpolate the forcing data using conservative interpolation

neighbor_interp (5.3.10)
spatially interpolate the forcing data using nearest neighbor interpolation

---

53.2.5 retnldas2 (Source File: retnldas2b.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
24 Aug 2007: Chuck Alonge; Modified for use with NLDAS2 data
```

INTERFACE:

```
subroutine retnldas2b(n, findex, suppdata, name,ferror)
```

USES:

```
use LIS_coreMod, only        : LIS_rc, LIS_domain
use LIS_logMod, only         : LIS_logunit
use nldas2_forcingMod,only : nldas2_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: findex
integer, intent(in)      :: n
real                     :: suppdata(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
character*80, intent(in) :: name
integer, intent(out)     :: ferror
```

DESCRIPTION:

For the given time, reads parameters from NLDAS2 data, 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 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 (=0 indicates success)

The routines invoked are:

interp_nldas2 (53.2.4)
spatially interpolates a NLDAS2 variable

53.2.6 time_interp_nldas2 (Source File: time_interp_nldas2.F90)

REVISION HISTORY:

```
02Feb2004: Sujay Kumar; Initial Specification
24 Aug 2007: Chuck Alonge; Modified for use with NLDAS2 data
```

INTERFACE:

```
subroutine time_interp_nldas2(n,findex,suppdata1,suppdata2)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc,LIS_domain
use LIS_constantsMod, only  : LIS_CONST_SOLAR
use LIS_baseforcingMod, only : LIS_forc, LIS_FORC_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
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

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 (3.31.23)
converts the time to a date format

LIS_tick (3.31.25)
advances or retracts time by the specified amount

zterp (5.3.18)
zenith-angle based interpolation

---

54 SALDAS

The atmospheric forcing used in the South American Land Data Assimilation System (SALDAS).

54.0.7 getsaldas (Source File: getsaldas.F90)

REVISION HISTORY:

```
02Feb 2004 : Gustavo ; Initial Version in LIS
```

INTERFACE:

```
subroutine getsaldas(n, findex, suppdata1, suppdata2)
```

USES:

```
use LIS_coreMod,        only : LIS_rc
use LIS_timeMgrMod,     only : LIS_get_nstep, LIS_tick
use LIS_logMod,         only : LIS_logunit
use saldas_forcingMod,  only : saldas_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

Opens, reads, and interpolates hourly, saldas 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 (3.31.25)
determines the saldas data times

saldasfile (54.0.8)
Puts together appropriate timestamped filename

retsaldas (54.0.11)
Interpolates saldas data to LIS grid

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

read_saldas_elev (54.0.9)
reads the native elevation of the saldas data to be used for topographic adjustments to the forcing

---

54.0.8 saldasfile (Source File: getsaldas.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
```

INTERFACE:

```
subroutine saldasfile(name,saldasdir,yr,mo,da,hr)

  implicit none
```

ARGUMENTS:

```
  character*80, intent(out)::  name
  character*40, intent(in) ::  saldasdir
  integer, intent(in)      :: yr,mo,da,hr
```

DESCRIPTION:

This subroutine puts together saldas file name for 3 hour file intervals
The arguments are:

saldasdir Name of the saldas directory

yr year

mo month

da day of month

hr hour of day

name name of the timestamped saldas file

54.0.9 read_saldas_elev (Source File: read_saldas_elev.F90)

REVISION HISTORY:

```
17Dec2004; Sujay Kumar; Initial Specificaton
```

INTERFACE:

```
subroutine read_saldas_elev(n, phase)
```

USES:

```
use LIS_coreMod,         only : LIS_rc, LIS_domain
use LIS_baseforcingMod,  only : LIS_forc
use LIS_fileIOMod,       only : LIS_readData
use LIS_logMod,          only : LIS_logunit, LIS_endrun
use saldas_forcingMod,   only : saldas_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Opens, reads, and interpolates saldas 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

54.0.10 readsaldascrd (Source File: readsaldascrd.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readsaldascrd()
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod, only : LIS_logunit
use ESMF_Mod
use saldas_forcingMod, only: saldas_struc
```

DESCRIPTION:

This routine reads the options specific to SALDAS forcing from the LIS configuration file.

54.0.11 retsaldas (Source File: retsaldas.F90)

REVISION HISTORY:

```
1  Oct 1999: Jared Entin; Initial code
15 Oct 1999: Paul Houser; Significant F90 Revision
11 Apr 2000: Brian Cosgrove; changed code to use Forcing Mask (With inland
             water filled in).  Deleteted unused variables.
27 Apr 2000: Brian Cosgrove; changed code to use the original
             mask again since that is the
             mask which  NCEP has already applied to the forcing data
             by the time NASA gets it......not possible to use the
             expanded NASA forcing mask
1  May 2000: Brian Cosgrove; changed code so that if parameter 11 (sw)
             is not found in hourly ncep data, it will just use
             edas-based shortwave from the hourly ncep files
20 Jun 2000: Brian Cosgrove; changed code so that it uses  LDAS%UDEF and
              not a hard-wired undefined value of -999.9 and -999.0
18 Aug 2000: Brian Cosgrove; changed code so that FMASK and not MASK
             is used when ungribbing.  NCEP data already has a mask applied
             to it and so may not be able to supply forcing data to
             all LDAS land forcing points.  In areas where LDAS
             forcing mask states that land exists, but where NCEP forcing
             data is non-existant, assign undefined value to forcing data.
22 Aug 2000: Brian Cosgrove; Altered code for US/Mexico/Canada Mask
05 Sep 2001: Brian Cosgrove; Removed dirnom and infile variables, changed
             call to ungribncep to match removal.  Added code to make use
             of precip weighting mask
02 Feb 2004: Sujay Kumar; Initial Specification in LIS
```

INTERFACE:

```
subroutine retsaldas(n, findex, suppdata, name,ferror)
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain
use saldas_forcingMod, only : saldas_struc
use LIS_logMod, only :LIS_logunit

implicit none
```

ARGUMENTS:

```
integer, intent(in)      :: findex
real                     :: suppdata(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
integer, intent(in)      :: n
character*80, intent(in) :: name
integer, intent(out)     :: ferror
```

DESCRIPTION:

For the given time, reads parameters from SALDAS data, 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 hourly instance, order=2, read the next hourly instance)

n index of the nest

name name of the 3 hour GDAS forecast file

ferror flag to indicate success of the call (=0 indicates success)

The routines invoked are:

interp_saldas (54.0.12)
spatially interpolates a SALDAS variable

54.0.12 interp_saldas (Source File: retsaldas.F90)

INTERFACE:

```
subroutine interp_saldas(n, findex, kpds, saldas,f,lb,lis_gds,nc,nr, &
     varfield)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use saldas_forcingMod, only :saldas_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in)   :: n
integer, intent(in)   :: findex
integer, intent(in)   :: kpds(200)
integer, intent(in)   :: saldas
real, intent(out)     :: f(saldas)
logical*1, intent(in) :: lb(saldas)
real, intent(in)      :: lis_gds(50)
integer, intent(in)   :: nc
integer, intent(in)   :: nr
real, intent(inout)   :: varfield(nc,nr)
character :: varfield2(nc,nr)
```

DESCRIPTION:

This subroutine interpolates a given SALDAS 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 (5.1.25)
spatially interpolate the forcing data using conservative interpolation

---

54.1 Fortran: Module Interface saldas_forcingMod (Source File: saldas_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 (SALDAS; Cosgrove et al.(2003)). The variables are produced at 0.125 degree spatial resolution, and at hourly intervals.
Cosgrove, B. Real-time and retrospective forcing in the North American Land Data Assimilation System (SALDAS) project. Journal of Geophysical Research, 108 (D22), 8842, DOI:10.1029/2002JD003118
The implemenatation in LIS has the derived data type `saldas_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

saldastime1 The nearest, previous hourly instance of the incoming data (as a real time).

saldastime2 The nearest, next hourly instance of the incoming data (as a real time).

saldasdir 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)

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.

USES:

```
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: defineNativeSALDAS      !defines the native resolution of
                                 !the input data
```

PUBLIC TYPES:

```
public :: saldas_struc
```

54.1.1 defineNativeSALDAS (Source File: saldas_forcingMod.F90)

REVISION HISTORY:

```
02Feb2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defineNativeSALDAS(findex)
```

USES:

```
use LIS_coreMod,    only : LIS_rc

implicit none
integer :: findex
```

DESCRIPTION:

Defines the native resolution of the input forcing for SALDAS 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:

readsaldascrd (54.0.10)
reads the runtime options specified for SALDAS data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
computes the neighbor, weights for conservative interpolation

read_saldas_elev (54.0.9)
reads the native elevation of the SALDAS data to be used for topographic adjustments to the forcing

---

54.1.2 saldasforcing_finalize (Source File: saldasforcing_finalize.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine saldasforcing_finalize(findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use saldas_forcingMod, only : saldas_struc
```

DESCRIPTION:

Routine to cleanup allocated structures for SALDAS forcing.

54.1.3 time_interp_saldas (Source File: time_interp_saldas.F90)

REVISION HISTORY:

```
02Feb2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine time_interp_saldas(n, findex, suppdata1, suppdata2)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc,LIS_domain, LIS_localPet
use LIS_constantsMod, only  : LIS_CONST_SOLAR
use LIS_baseforcingMod, only : LIS_FORC_State
use LIS_FORC_AttributesMod
use LIS_timeMgrMod, only : LIS_tick, LIS_time2date
use LIS_logMod, only :LIS_logunit, LIS_verify
use saldas_forcingMod, only : saldas_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

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 (3.31.23)
converts the time to a date format

LIS_tick (3.31.25)
advances or retracts time by the specified amount

zterp (5.3.18)
zenith-angle based interpolation

---

55 SCAN forcing

This section describes the implementation of the SCAN station data.

55.0.4 getscan (Source File: getscan.F90)

REVISION HISTORY:

```
13 Apr 2007: Bailing Li; Initial Specification
```

INTERFACE:

```
subroutine getscan(n,findex, suppdata1, suppdata2)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod, only : LIS_tick
use LIS_logMod, only : LIS_logunit
use scan_forcingMod, only : scan_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

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 (3.31.25)
determines the SCAN data times

readscan (55.0.5)
Interpolates the appropriate SCAN station data to LIS grid

55.0.5 readscan (Source File: readscan.F90)

REVISION HISTORY:

```
13 Apr 2007: Bailing Li; Initial Specification
```

INTERFACE:

```
subroutine readscan(n,ftn,findex,suppdata)
```

USES:

```
use LIS_logMod, only         : LIS_logunit
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
real                          :: suppdata(LIS_rc%nsf(findex),LIS_rc%ngrid(n))
```

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.

55.0.6 readscancrd (Source File: readscancrd.F90)

REVISION HISTORY:

```
13 Apr 2007; Bailing Li, Initial Code
```

INTERFACE:

```
subroutine readscancrd()
```

USES:

```
use ESMF_Mod
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.

55.1 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 :: defineNativeSCAN      !defines the native resolution of
                                !the input data
-------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-------------------------------------------------------------------------------
public :: scan_struc
```

55.1.1 defineNativeSCAN (Source File: scan_forcingMod.F90)

INTERFACE:

```
subroutine defineNativeSCAN(findex)
```

USES:

```
  use LIS_coreMod,only : LIS_rc
  use LIS_timeMgrMod, only : LIS_date2time
  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:

readscancrd (55.0.6)
reads the runtime options specified for SCAN station data

LIS_date2time (3.31.22)
converts date to the real time format

compute_stnwts (5.1.24)
computes the weights for spatial interpolation

55.1.2 scanforcing_finalize (Source File: scanforcing_finalize.F90)

REVISION HISTORY:

```
08Dec2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine scanforcing_finalize(findex)
```

USES:

```
use scan_forcingMod, only : scan_struc
```

DESCRIPTION:

Routine to cleanup SCAN forcing related memory allocations.

55.1.3 time_interp_scan (Source File: time_interp_scan.F90)

REVISION HISTORY:

```
13 Apr 2007: Bailing Li; Initial Specification
```

INTERFACE:

```
subroutine time_interp_scan(n, findex, suppdata1, suppdata2)
```

USES:

```
use ESMF_Mod
use LIS_logMod,  only        : LIS_logunit
use LIS_coreMod,        only : LIS_rc,LIS_domain
use LIS_baseforcingMod, only : LIS_FORC_State
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
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

Temporally interpolates the SCAN forcing data to the model timestep. All variables except precipitation is linearly interpolated.

---

56 STAGEII

This section describes the implementation of the precipitation product from NCEP, known as STAGEII

56.0.4 getstg2 (Source File: getstg2.F90)

REVISION HISTORY:

```
25 May 2006: Kristi Arsenault;  Data and Code implementation
```

INTERFACE:

```
subroutine getstg2(n, findex, suppdata1, suppdata2)
```

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
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

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 (3.31.25)
determines the STAGE2 data times

bilinear_interp_input (5.1.2)
Computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
Computes the neighbor, weights for conservative interpolation

stg2file (56.2.2)
Puts together appropriate file name for 1-hour intervals

precip_stg2 (56.0.6)
Interpolates STAGE2 data to LIS grid

56.0.5 interp_stg2 (Source File: interp_stg2.F90)

INTERFACE:

```
subroutine interp_stg2 (n, findex, ksec1, 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
integer                :: ksec1(100)
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

ksec1 grib decoding array

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 (5.1.25)
spatially interpolate the forcing data using conservative (budget bil.) interpolation

---

56.0.6 precip_stg2 (Source File: precip_stg2.F90)

REVISION HISTORY:

```
25 May 2006: Kristi Arsenault;  Data and code implementation
```

INTERFACE:

```
subroutine precip_stg2( n, fname, findex, suppdata, ferror_stg2 )
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_logMod,         only : LIS_logunit
use stg2_forcingMod,    only : stg2_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
character(len=80)   :: fname
integer, intent(in) :: findex
real                :: suppdata(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
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 (56.0.5)
spatially interpolates the STG2 data

56.0.7 readstg2crd (Source File: readstg2crd.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
05 Jun 2006; Kristi Arsenault, Code and data implementation
```

INTERFACE:

```
subroutine readstg2crd()
```

USES:

```
use ESMF_Mod
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.

56.1 Fortran: Module Interface stg2_finalize (Source File: stg2_finalize.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
17Jul2006; K. Arsenault, Added Stage II
```

INTERFACE:

```
subroutine stg2_finalize(findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use stg2_forcingMod
```

DESCRIPTION:

Routine to cleanup Stage II forcing related memory allocations.

56.2 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 :: defineNativeSTG2      !defines the native resolution of
                                  !the input data
 ------------------------------------------------------------------------------
```

PUBLIC TYPES:

```
 ------------------------------------------------------------------------------
  public :: stg2_struc
```

56.2.1 defineNativeSTG2 (Source File: stg2_forcingMod.F90)

REVISION HISTORY:

```
  11Dec2003: Sujay Kumar; Initial Specification
  25May2006: Kristi Arsenault; Data and code implementation
```

INTERFACE:

```
  subroutine defineNativeSTG2(findex)
```

USES:

```
use LIS_coreMod, only: LIS_rc
use LIS_timeMgrMod, only : LIS_date2time

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 V
The routines invoked are:

readstg2crd (56.0.7)
reads the runtime options specified for STAGE2 data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

LIS_date2time (3.31.22)
converts date to the real time format - time of grid change

56.2.2 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

56.2.3 time_interp_stg2 (Source File: time_interp_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 time_interp_stg2(n,findex,suppdata1,suppdata2)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_baseforcingMod, only : LIS_FORC_State
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
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep.
The arguments are:

n index of the nest

---

57 STAGEIV

This section describes the implementation of the precipitation product from NCEP, known as STAGEIV

57.0.4 getstg4 (Source File: getstg4.F90)

REVISION HISTORY:

```
25 May 2006: Kristi Arsenault;  Data and Code implementation
```

INTERFACE:

```
subroutine getstg4(n,findex,suppdata1, suppdata2)
```

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
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

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 (3.31.25)
determines the STAGE4 data times

bilinear_interp_input (5.1.2)
Computes the neighbor, weights for bilinear interpolation

conserv_interp_input (5.1.26)
Computes the neighbor, weights for conservative interpolation

stg4file (57.2.2)
Puts together appropriate file name for 1-hour intervals

precip_stg4 (57.0.6)
Interpolates STAGE4 data to LIS grid

57.0.5 interp_stg4 (Source File: interp_stg4.F90)

INTERFACE:

```
subroutine interp_stg4 (n, findex, ksec1, 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
integer                :: ksec1(100)
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

ksec1 grib decoding array

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 (5.1.25)
spatially interpolate the forcing data using conservative (budget bil.) interpolation

---

57.0.6 precip_stg4 (Source File: precip_stg4.F90)

REVISION HISTORY:

```
25 May 2006: Kristi Arsenault;  Data and code implementation
```

INTERFACE:

```
subroutine precip_stg4( n, fname,findex,suppdata, ferror_stg4 )
```

USES:

```
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_logMod,         only : LIS_logunit
use LIS_suppforcingMod, only : LIS_suppforc
use stg4_forcingMod,    only : stg4_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
integer, intent(in) :: findex
real                :: suppdata(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
character(len=80)   :: fname
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 (57.0.5)
spatially interpolates the STAGE4 data

57.0.7 readstg4crd (Source File: readstg4crd.F90)

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
05 Jun 2006; Kristi Arsenault, Code and data implementation
```

INTERFACE:

```
subroutine readstg4crd()
```

USES:

```
use ESMF_Mod
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.

57.1 Fortran: Module Interface stg4_finalize (Source File: stg4_finalize.F90)

REVISION HISTORY:

```
25Oct2005; Sujay Kumar, Initial Code
17Jul2006; K. Arsenault, Added Stage IV
```

INTERFACE:

```
subroutine stg4_finalize(findex)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use stg4_forcingMod
```

DESCRIPTION:

Routine to cleanup Stage IV forcing related memory allocations.

57.2 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 :: defineNativeSTG4      !defines the native resolution of
                                !the input data
```

PUBLIC TYPES:

```
public :: stg4_struc
```

57.2.1 defineNativeSTG4 (Source File: stg4_forcingMod.F90)

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
25May2006: Kristi Arsenault; Data and code implementation
```

INTERFACE:

```
subroutine defineNativeSTG4(findex)
```

USES:

```
use LIS_coreMod, only: LIS_rc
use LIS_timeMgrMod, only : LIS_date2time

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 V
The routines invoked are:

readstg4crd (57.0.7)
reads the runtime options specified for STAGE4 data

bilinear_interp_input (5.1.2)
computes the neighbor, weights for bilinear interpolation

LIS_date2time (3.31.22)
converts date to the real time format - time of grid change

57.2.2 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

57.2.3 time_interp_stg4 (Source File: time_interp_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 time_interp_stg4(n,findex,suppdata1,suppdata2)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,        only : LIS_rc, LIS_domain
use LIS_baseforcingMod, only : LIS_FORC_State
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
real                :: suppdata1(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
real                :: suppdata2(LIS_rc%nsf(findex), LIS_rc%ngrid(n))
```

DESCRIPTION:

Temporally interpolates the forcing data to the current model timestep.
The arguments are:

n index of the nest

Part XII
Data Assimilation in LIS

58 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 XV and XIII, respectively.

---

59 Direct Insertion

59.1 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
```

59.1.1 di_init (Source File: directInsertion_Mod.F90)

INTERFACE:

```
subroutine di_init
```

DESCRIPTION:

This method performs the required initializations for the direct insertion method.

59.1.2 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:

getLSMstateVar (**??**)
obtain the specified LSM state prognostic variables

ObsTransform (3.42.22)
transform the observation state into an LSM space

mapObsToLSM (3.42.26)
map the observation state to generate the analysis increments

USES:

```
use ESMF_Mod
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
```

59.1.3 di_update (Source File: directInsertion_Mod.F90)

INTERFACE:

```
subroutine di_update(n,k)
```

USES:

```
use ESMF_Mod
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:

setLSMstateVar (**??**)
set the specified LSM state prognostic variables

59.1.4 di_diagnostics (Source File: directInsertion_Mod.F90)

INTERFACE:

```
subroutine di_diagnostics(n,k)
```

DESCRIPTION:

This subroutine generates the direct insertion related diagnostics

---

59.1.5 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.

---

60 GMAO Ensemble Kalman Filter

60.1 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
```

60.1.1 enkf_increments (Source File: enkf_general.F90)

INTERFACE:

```
subroutine enkf_increments( 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

---

60.1.2 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
```

60.1.3 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. ***

60.1.4 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.

60.2 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
```

60.3 Fortran: Module Interface gmaoenkf_Mod (Source File: gmaoenkf_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_Mod
use enkf_types

implicit none

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

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: gmaoenkf_init  ! Initialization for EnKF
public :: gmaoenkf_setup
public :: gmaoenkf_increments ! compute analysis increments
public :: gmaoenkf_update ! apply the analysis increments
public :: gmaoenkf_diagnostics ! write EnKF related diagnostics
public :: gmaoenkf_final ! Finalization for EnKF
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: gmaoenkf_struc ! data structure containing EnKF diagnostics
```

60.3.1 gmaoenkf_init (Source File: gmaoenkf_Mod.F90)

INTERFACE:

```
subroutine gmaoenkf_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.

60.3.2 gmaoenkf_setup (Source File: gmaoenkf_Mod.F90)

INTERFACE:

```
subroutine gmaoenkf_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
```

DESCRIPTION:

This method performs the required initializations for the GMAO EnKF method. The method reads the runtime settings from the LIS configuration file.

60.3.3 gmaoenkf_increments (Source File: gmaoenkf_Mod.F90)

INTERFACE:

```
subroutine gmaoenkf_increments(n,k)
```

USES:

```
use LIS_coreMod,         only : LIS_rc, LIS_domain, 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 enkf_general,        only : enkf_increments
use my_matrix_functions, only : row_variance
```

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 60.3.11
obtain the observations

getObsPred (3.42.24)
obtain model's estimate of the observations

getObsPert (60.3.10)
obtain the observation perturbations

generateObsparam (60.3.12)
generate the 'obsparam' (metadata for observations)

getLSMstateVar 3.42.18 obtain the specified LSM prognostic variables.

assemble_obs_cov (60.3.7) assembles the observation error covariance

getSelectedObsNumber(3.36.8)
obtain the number of selected observations for each modeling point.

enkf_increments (60.1.1)
apply the EnKF filter to obtain the prognostic variable increments.

row_variance (60.4.2)
computes the row variance HPH'

setLSMstateVar (3.42.20)
assigns the specified LSM state prognostic variables

scaleLSMstatVar (3.42.36)
scales the state variables for computational stability

descaleLSMstatVar (3.42.38)
descales the state variables to the original state

qcLSMState 3.42.32 QC the updated LSM state

60.3.4 gmaoenkf_update (Source File: gmaoenkf_Mod.F90)

INTERFACE:

```
subroutine gmaoenkf_update(n,k)
```

USES:

```
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
implicit none

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

DESCRIPTION:

This routine updates the model prognostics using the analysis increments computed earlier.

60.3.5 gmaoenkf_diagnostics (Source File: gmaoenkf_Mod.F90)

INTERFACE:

```
subroutine gmaoenkf_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 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 (3.12.3)
call to create output directory for DA statistics

getLSMvarnames (60.3.9)
retrieve the names of the LSM prognostic variables

pruneVarname (60.3.6)
trims the variable name, eliminating white spaces

LIS_create_stats_filename (3.12.6)
creates the filename for statistics

LIS_create_innov_filename (3.12.7)
creates the name of the innovations file

getLSMstateVar (3.42.18)
retrieve the lsm state variables

getLSMdata (**??**)
unpack the LSM state and retrive the data

LIS_writevar_txt (**??**)
call to write a variable in a text file

writeDAObsStats (**??**)
writes a formatted observation output

60.3.6 pruneVarname (Source File: gmaoenkf_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

60.3.7 assemble_obs_cov (Source File: gmaoenkf_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

60.3.8 gmaoenkf_final (Source File: gmaoenkf_Mod.F90)

INTERFACE:

```
subroutine gmaoenkf_final
```

DESCRIPTION:

This method performs the finalization for all EnKF related structures and subroutines.

60.3.9 getLSMvarnames (Source File: gmaoenkf_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.

60.3.10 getObsPert (Source File: gmaoenkf_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

60.3.11 generateObservations (Source File: gmaoenkf_Mod.F90)

INTERFACE:

```
subroutine generateObservations(n, Nobjs, N_obs_size, LIS_OBS_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(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.

60.3.12 generateObsparam (Source File: gmaoenkf_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

60.3.13 getLSMData (Source File: gmaoenkf_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

60.3.14 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)
```

60.3.15 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)
```

60.4 Fortran: Module Interface my_matrix_functions (Source File: my_matrix_functions.F90

this file contains a collection of matrix operation subroutines

60.4.1 (Source File: my_matrix_functions.F90)

reichle, 1 May 01 reichle, 18 Apr 06 - renamed subroutine sort()

60.4.2 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

60.4.3 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

$$third_moment = 1/N * sum_{i=1}^{N}(x_i - mean(x)) * *3 \tag{1}$$

60.4.4 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

60.4.5 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

---

60.4.6 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

60.4.7 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

60.4.8 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)
```

61 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.

---

62 GMAO Perturbation Algorithm

62.1 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
```

62.1.1 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

       integer, intent(in) :: index

       if(index.eq.1) then
          allocate(forcPert(LIS_rc%nnest, 1))
       elseif(index.eq.2) then
          allocate(progPert(LIS_rc%nnest, LIS_rc%nperts))
       elseif(index.eq.3) then
          allocate(obspert(LIS_rc%nnest, LIS_rc%ndas))
          allocate(nobs(LIS_rc%nnest,LIS_rc%ndas))
       endif
     end subroutine gmaoPert_init
```

```
%/////////////////////////////////////////////////////////////

\mbox{}\hrulefill\

\subsubsection{gmaopert\_setup (Source File: gmaopert\_Mod.F90)}


    \label{gmaopert_setup}

\bigskip{\sf INTERFACE:}
\begin{verbatim}     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
       use LIS_logMod,  only  : LIS_verify
       use nr_ran2_gasdev, only : NRANDSEED
```

DESCRIPTION:

Initializes memory structures for specified perturbation options.

62.1.2 gmaoperturb (Source File: gmaopert_Mod.F90)

INTERFACE:

```
     subroutine gmaoperturb(index, n, k, Base_State, Pert_State)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_logMod,  only  : LIS_verify
use landpert_routines, only : get_pert

implicit none
```

ARGUMENTS:

```
integer, intent(in)       :: index
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.

62.1.3 gmaopert_writerestart (Source File: gmaopert_Mod.F90)

INTERFACE:

```
subroutine gmaopert_writerestart(n)
```

USES:

```
use LIS_coreMod,     only : LIS_rc, LIS_domain, LIS_masterproc
use LIS_historyMod,  only : LIS_writevar_restart
use LIS_fileIOMod,   only : LIS_create_output_directory, &
     LIS_create_restart_filename
use LIS_logMod,      only : LIS_logunit, LIS_getNextUnitNumber, &
     LIS_releaseUnitNumber
```

ARGUMENTS:

```
implicit none

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

DESCRIPTION:

Write one single file that includes the restart info for forcing, states and observations

---

62.1.4 gmaopert_readrestart (Source File: gmaopert_Mod.F90)

INTERFACE:

```
subroutine gmaopert_readrestart()
```

USES:

```
use LIS_coreMod,      only : LIS_rc, LIS_domain
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

62.2 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
```

62.2.1 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.

62.3 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
```

62.3.1 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, only : LIS_rc
use LIS_logMod,  only : LIS_verify
use ESMF_Mod

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
```

62.3.2 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

62.3.3 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.

62.3.4 get_pert (Source File: landpert_routines.F90)

INTERFACE:

```
subroutine get_pert(                                &
     N_pert, N_ens, N_x, N_y,                       &
     dx, dy, dtstep,                                &
     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
```

```
! 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.

62.3.5 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

62.3.6 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)

62.3.7 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

---

62.4 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
```

62.4.1 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)

62.5 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
```

62.6 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
```

62.6.1 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

---

62.6.2 rfg2d_fft (Source File: random_fields.F90)

INTERFACE:

```
subroutine rfg2d_fft( &
     rseed, variance, N_x, N_y, dx, dy, lambda_x, lambda_y, &
     field1, field2 )

  implicit none

  ! use Compaq Extended Math Library (CXML)
  ! (free with Compaq Fortran Compiler)

#ifdef CXML_FFT_AVAILABLE
  include 'CXMLDEF.FOR'
#endif
```

ARGUMENTS:

```
integer, dimension(NRANDSEED), intent(inout) :: rseed
real, intent(in) :: variance, dx, dy, lambda_x, lambda_y
integer, intent(in) :: N_x, N_y
real, intent(out), dimension(N_x,N_y) :: field1, field2
```

DESCRIPTION:

generate a pair of 2d zero-mean random fields using FFT method (so far only Gaussian covariance implemented)
NOTE: implemented with index counters of type int, must have
N_x*N_y ¡ maximum integer on given machine
on yama/alborz (at MIT) int varies from -2147483648...2147483647 -¿ can handle up to $N_x * N_y = (32768)^2$
if larger fields are needed, rewrite with type long int etc
NOTE: The fft method for generating random fields produces fields that are periodic with the size of the domain, that is the field is (almost) the same on each boundary. This introduces unwanted correlations at lags shorter than the domain size. Therefore, only a part of the generated field is usable. As a rule of thumb, the fields should be generated on a grid that is two correlation lenghts bigger than the field on which the grid is desired. Then cut out fields of the necessary size. This procedure is included in rfg2d_fft().
NOTE: The variance specified as input is the theoretical variance of the complex field that is obtained from the inverse fft of the realization dZ. The sample variance of this *complex* field is a FIXED (non-random) number which depends on the size of the domain, the grid spacing, and the correlation length (but not on the random seed!!). (This number is non-random because the variance is the integral of the absolute value of the spectrum. In this integral the randomness disappears because we only choose a random phase angle.) For vanishing discretization and spectral truncation error, this number converges to the theoretical (input) value.
This function is set up to use the pair of two real fields, where
field1 = sqrt(2)*real(ifft(dZ)) field2 = sqrt(2)*imag(ifft(dZ)).
The factor sqrt(2) re-scales the variance of field1 and field2 such that for vanishing discretization error and spectral truncation error each field converges to the specified theoretical (input) variance. NOTE: The sum of the sample variances of the two real fields is equal to the (FIXED) sample variance of the complex field (before re-scaling with sqrt(2)). The individual sample variances within each pair vary from realization to realization.

62.6.3 get_fft_grid (Source File: random_fields.F90)

INTERFACE:

```
subroutine get_fft_grid( N_x, N_y, dx, dy, lambda_x, lambda_y, &
     N_x_fft, N_y_fft )

  implicit none
```

ARGUMENTS:

```
integer, intent(in) :: N_x, N_y
real, intent(in) :: dx, dy, lambda_x, lambda_y
integer, intent(out) :: N_x_fft, N_y_fft
```

DESCRIPTION:

get a grid that can be used for 2d fft random number generator (this fft grid must extend beyond the desired random field by about two correlations lengths, it must also have N_x_fft and N_y_fft that are powers of two)

62.6.4 generate_white_field (Source File: random_fields.F90)

INTERFACE:

```
subroutine generate_white_field(N_x, N_y, rseed, rfield )

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: N_x, N_y
  integer, dimension(NRANDSEED), intent(inout) :: rseed
  real, dimension(N_x,N_y), intent(out) :: rfield
```

DESCRIPTION:

generate standard-normal random field that is white in space
note that nr_gasdev always produces a pair of random numbers
do not store random numbers between subsequent calls to the random field generator - works best if fields are large (ie. avoid using this subroutine with N_x=N_y=1)

---

63 Static Perturbation Algorithm

63.1 Fortran: Module Interface staticPerturbMod (Source File: staticPerturbMod.F90)

This module contains routines to perturb the specified input using either random gaussian noise or random lognormal noise.

REVISION HISTORY:

```
08Jul05    Sujay Kumar; Initial Specification
```

63.1.1 staticPerturb_init (Source File: staticPerturbMod.F90)

INTERFACE:

```
subroutine staticPerturb_init (Base_State, Pert_State)
```

USES:

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

ARGUMENTS:

```
  type(ESMF_State)        :: Base_State(LIS_rc%nnest)
  type(ESMF_State)        :: Pert_State(LIS_rc%nnest)
```

DESCRIPTION:

Initialization for perturbation setup. This routine is simply a placeholder

---

63.1.2 staticPerturburb (Source File: staticPerturbMod.F90)

INTERFACE:

```
subroutine staticPerturb(n, Base_State, Pert_State)
```

USES:

```
use LIS_logMod,    only : LIS_logunit, LIS_verify
use LIS_numerRecipesMod, only : LIS_gasdev
```

ARGUMENTS:

```
integer, intent(in)       :: n
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 perturbations are created based on a random gaussian noise (additive) or a random lognormal noise (multiplicative).

Part XIII
Observation data handling for assimilation in LIS

64 Observation data handling

This section contains the description of observational data handling for assimilation in LIS.

---

65 AMSR-E Soil Moisture

65.1 Fortran: Module Interface AMSREsmobs_module (Source File: AMSREsmobs_module.F90)

This module contains interfaces and subroutines to handle amsr-e soil moisture observations

REVISION HISTORY:

```
06Nov07    Bailing Li; Initial Specification
```

```
module AMSREsmobs_module
```

USES:

```
use ESMF_Mod
implicit none
```

65.1.1 AMSREsmobs_setup (Source File: AMSREsmobs_module.F90)

INTERFACE:

```
subroutine AMSREsmobs_setup(k, OBS_State, OBS_Pert_State)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_config, LIS_vecGrid, LIS_ensOnGrid
use LIS_logmod, only : LIS_logunit, LIS_verify

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

65.1.2 getNSO_AMSREsm.F90 (Source File: getNSO_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_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

65.1.3 read_AMSREsmobs (Source File: read_AMSREsmobs.F90)

REVISION HISTORY:

```
06Nov2007: Bailing Li; Initial Specification
```

INTERFACE:

```
subroutine read_AMSREsmobs(n, k,  OBS_State)
```

USES:

```
  use ESMF_Mod
  use LIS_mpiMod
  use LIS_coreMod,    only : LIS_rc, LIS_domain, &
       LIS_masterproc, LIS_npes, LIS_masterproc, LIS_localPet
  use LIS_logMod,     only : LIS_logunit, LIS_verify
  use LIS_fileIOMod,  only : LIS_create_obs_filename
  use LIS_historyMod, only : LIS_write_innov

  implicit none
```

ARGUMENTS:

```
  integer, intent(in) :: n
  integer, intent(in) :: k
  type(ESMF_State)    :: OBS_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

---

66 NASA AMSR-E Soil Moisture

66.1 Fortran: Module Interface NASA_AMSREsm_Mod (Source File: NASA_AMSREsm_M

This module contains interfaces and subroutines to handle NASA AMSRE observations.

REVISION HISTORY:

```
    14 Nov 08    Sujay Kumar;   Initial Specification
```

```
 module NASA_AMSREsm_Mod
```

USES:

```
   use ESMF_Mod
```

66.1.1 NASA_AMSREsm_setup (Source File: NASA_AMSREsm_Mod.F90)

INTERFACE:

```
   subroutine NASA_AMSREsm_setup(k, OBS_State, OBS_Pert_State)
```

USES:

```
     use LIS_coreMod, only : LIS_rc, LIS_config, LIS_vecGrid, LIS_ensOnGrid, &
          LIS_masterproc
     use LIS_timeMgrMod, only : LIS_clock, LIS_calendar
     use LIS_historyMod, only : LIS_readvar_gridded
     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 NASA AMSRE 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_NASA_AMSREsm (Source File: getNSO_NASA_AMSREsm.F90)

Returns the number of selected observations (NSO) corresponding to the grid point specified

REVISION HISTORY:

```
14 Nov 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine getNSO_NASA_AMSREsm(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_NASA_AMSREsm (Source File: read_NASA_AMSREsm.F90)

REVISION HISTORY:

```
14 Nov 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_NASA_AMSREsm(n, OBS_State)
```

USES:

```
  use ESMF_Mod
  use LIS_pluginIndices, only : LIS_NASA_AMSREsmobsId
  use LIS_mpiMod
  use LIS_coreMod,  only : LIS_rc, LIS_domain, LIS_localPet, LIS_npes
  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

  implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: OBS_State
```

DESCRIPTION:

reads the NASA AMSRE 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 NASA_AMSREsm_filename (Source File: read_NASA_AMSREsm.F90)

INTERFACE:

```
subroutine NASA_AMSREsm_filename(name, ndir, yr, mo,da, aord)

  implicit none
```

ARGUMENTS:

```
character*80      :: name
integer           :: yr, mo, da
integer           :: aord
character (len=*) :: ndir
```

DESCRIPTION:

This subroutine creates the NASA AMSRE filename based 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

aord flag for ascending or descending (1-A,2-D)

66.1.5 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_Mod
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

---

66.1.6 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:

---

67 NESDIS AMSR-E Soil Moisture

67.1 Fortran: Module Interface NESDIS_AMSREsm_Mod (Source File: NESDIS_AMSREsm_Mod.F90)

This module contains interfaces and subroutines to handle NESDIS AMSRE observations.

REVISION HISTORY:

```
14 Nov 08    Sujay Kumar;   Initial Specification
```

```
module NESDIS_AMSREsm_Mod
```

USES:

```
use ESMF_Mod
```

67.1.1 NESDIS_AMSREsm_setup (Source File: NESDIS_AMSREsm_Mod.F90)

INTERFACE:

```
subroutine NESDIS_AMSREsm_setup(k, OBS_State, OBS_Pert_State)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_config, LIS_vecGrid, LIS_ensOnGrid, &
     LIS_masterproc
use LIS_timeMgrMod, only : LIS_clock, LIS_calendar
use LIS_historyMod, only : LIS_readvar_gridded
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 NESDIS AMSRE 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

67.1.2 getNSO_NESDIS_AMSREsm (Source File: getNSO_NESDIS_AMSREsm.F90)

Returns the number of selected observations (NSO) corresponding to the grid point specified

REVISION HISTORY:

```
14 Nov 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine getNSO_NESDIS_AMSREsm(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

67.1.3 read_NESDIS_AMSREsm (Source File: read_NESDIS_AMSREsm.F90)

REVISION HISTORY:

```
14 Nov 2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_NESDIS_AMSREsm(n, OBS_State)
```

USES:

```
use ESMF_Mod
use LIS_pluginIndices, only : LIS_NESDIS_AMSREsmobsId
use LIS_mpiMod
use LIS_coreMod,  only : LIS_rc, LIS_domain, LIS_localPet, LIS_npes
use LIS_timeMgrMod, only : LIS_calendar, LIS_clock
use LIS_logMod,     only : LIS_logunit, LIS_endrun, &
     LIS_getNextUnitNumber, LIS_releaseUnitNumber, LIS_verify
use NESDIS_AMSREsm_Mod, only : NESDIS_AMSREsm_struc

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: OBS_State
```

DESCRIPTION:

reads the NESDIS AMSRE 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

67.1.4 NESDIS_AMSREsm_filename (Source File: read_NESDIS_AMSREsm.F90)

INTERFACE:

```
subroutine NESDIS_AMSREsm_filename(name, ndir, yr, mo,da, aord)

  implicit none
```

ARGUMENTS:

```
  character*80      :: name
  integer           :: yr, mo, da
  integer           :: aord
  character (len=*) :: ndir
```

DESCRIPTION:

This subroutine creates the NESDIS AMSRE filename based the time and date
The arguments are:

name name of the NESDIS AMSRE soil moisture filename

ndir name of the NESDIS AMSRE soil moisture directory

yr current year

mo current month

da current day

aord flag for ascending or descending (1-A,2-D)

67.1.5 write_NESDIS_AMSREsmobs (Source File: write_NESDIS_AMSREsmobs.F90)

REVISION HISTORY:

```
  25Jan2008: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine write_NESDIS_AMSREsmobs(n, OBS_State)
```

USES:

```
  use ESMF_Mod
  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) NESDIS AMSRE observations to a file

---

67.1.6 NESDIS_AMSRE_smobsname (Source File: write_NESDIS_AMSREsmobs.F90)

INTERFACE:

```
subroutine NESDIS_AMSRE_smobsname(obsname)
```

USES:

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

ARGUMENTS:

```
character(len=*)      :: obsname
```

DESCRIPTION:

---

68 Synthetic Soil Moisture from Noah

68.0.7 getNSO_synsm1.F90 (Source File: getNSO_synsm1.F90)

Returns the number of selected observations (NSO) corresponding to the grid point specified

REVISION HISTORY:

```
21Jun2006: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine getNSO_synsm1(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

68.0.8 read_syntheticsm1obs (Source File: read_syntheticsm1obs.F90)

REVISION HISTORY:

```
21Jun2006: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_syntheticsm1obs(n, OBS_State)
```

USES:

```
use ESMF_Mod
use LIS_historyMod, only : LIS_readvar_gridded
use LIS_coreMod,  only : LIS_rc, LIS_domain
use LIS_logMod,     only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: OBS_State
```

DESCRIPTION:

reads the synthetic soil moisture observations produced from a LIS control run and packages it into an ESMF State with certain predefined attributes
The arguments are:

n index of the nest

OBS_State observations state

68.1 Fortran: Module Interface syntheticsm1obs_module (Source File: syntheticsm1obs_module.F90)

This module contains interfaces and subroutines to handle synthetic soil moisture observations generated from a previous LIS-Noah simulation

REVISION HISTORY:

```
27Feb05    Sujay Kumar;   Initial Specification
```

```
module syntheticsm1obs_module
```

USES:

```
use ESMF_Mod
```

68.1.1 syntheticsm1obs_setup (Source File: syntheticsm1obs_module.F90)

INTERFACE:

```
subroutine syntheticsm1obs_setup(k, OBS_State, OBS_Pert_State)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_config, LIS_vecGrid, LIS_ensOnGrid
use LIS_logMod, only : LIS_logunit, LIS_verify

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

---

69 Synthetic Soil Moisture from Catchment

69.0.2 getNSO_synsm2.F90 (Source File: getNSO_synsm2.F90)

Returns the number of selected observations (NSO) corresponding to the grid point specified

REVISION HISTORY:

```
21Jun2006: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine getNSO_synsm2(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

69.0.3 read_syntheticsm2obs (Source File: read_syntheticsm2obs.F90)

REVISION HISTORY:

```
21Jun2006: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_syntheticsm2obs(n, OBS_State)
```

USES:

```
use ESMF_Mod
use LIS_historyMod, only : LIS_readvar_gridded
use LIS_coreMod,  only : LIS_rc, LIS_domain
use LIS_logMod,     only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: OBS_State
```

DESCRIPTION:

reads the synthetic soil moisture observations produced from a LIS catchment control run.
The arguments are:

n index of the nest

OBS_State observations state

69.1 Fortran: Module Interface syntheticsm2obs_module (Source File: syntheticsm2obs_module.F90)

This module contains interfaces and subroutines to handle synthetic soil moisture observations produced from a LIS catchment simulation.

REVISION HISTORY:

```
27Feb05    Sujay Kumar;   Initial Specification
```

```
module syntheticsm2obs_module
```

USES:

```
use ESMF_Mod
```

69.1.1 syntheticsm2obs_setup (Source File: syntheticsm2obs_module.F90)

INTERFACE:

```
subroutine syntheticsm2obs_setup(k, OBS_State, OBS_Pert_State)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_config, LIS_vecGrid, LIS_ensOnGrid
use LIS_logMod, only : LIS_logunit, LIS_verify

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

---

70 Synthetic Snow Water Equivalent

70.0.2 getNSO_synswe1 (Source File: getNSO_synswe1.F90)

REVISION HISTORY:

```
21Jun2006: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine getNSO_synswe1(n, gid,st_id,en_id)

  implicit none
```

ARGUMENTS:

```
integer,   intent(IN)    :: n
integer,   intent(IN)    :: gid
integer,   intent(OUT)   :: st_id
integer,   intent(OUT)   :: en_id
```

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

70.0.3 read_syntheticswe1obs (Source File: read_syntheticswe1obs.F90)

REVISION HISTORY:

```
21Jun2006: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_syntheticswe1obs(n, OBS_State)
```

USES:

```
use ESMF_Mod
use LIS_historyMod, only : LIS_readvar_gridded
use LIS_coreMod,  only : LIS_rc, LIS_domain
use LIS_logMod,     only : LIS_logunit, LIS_verify

implicit none
```

ARGUMENTS:

```
integer, intent(in) :: n
type(ESMF_State)    :: OBS_State
```

DESCRIPTION:

reads the synthetic SWE observations produced from a LIS control run.
The arguments are:

n index of the nest

OBS_State observations state

70.1 Fortran: Module Interface syntheticswe1obs_module (Source File: syntheticswe1obs_module.F90)

This module contains interfaces and subroutines to handle synthetic soil moisture observations.

REVISION HISTORY:

```
27Feb05    Sujay Kumar;   Initial Specification
```

```
module syntheticswe1obs_module
```

USES:

```
use ESMF_Mod
```

70.1.1 syntheticswe1obs_setup (Source File: syntheticswe1obs_module.F90)

INTERFACE:

```
subroutine syntheticswe1obs_setup(k, OBS_State, OBS_Pert_State)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_config, LIS_vecGrid, LIS_ensOnGrid
use LIS_logMod, only : LIS_logunit, LIS_verify
```

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 SWE assimilation
The arguments are:

OBS_State observation state

OBS_Pert_State observation perturbations state

Part XIV
Parameter Estimation in LIS

71 Parameter Estimation in LIS

Part XV
Land Surface Models in LIS

72 Land Surface Models in LIS

73 SiB2 Land Surface Model

This section describes the interface implementations for the SiB2 land surface model.

73.0.2 readcrdsib2 (Source File: readcrdsib2.F90)

REVISION HISTORY:

```
14 Oct 2003: Sujay Kumar, Initial Code
 1 Mar 2004: Luis Gustavo G de Goncalves, SiB2 in LIS
 5 May 2004: David Mocko, made compatible with SiB-lings
21 Dec 2007: Sujay Kumar, Updated for LIS 5.0
```

INTERFACE:

```
subroutine readcrdsib2()
```

USES:

```
use ESMF_Mod
use sib2_lsmMod,   only : sib2_struc
use LIS_logMod,    only : LIS_logunit
use LIS_coreMod, only : LIS_rc, LIS_config
```

DESCRIPTION:

Routine to read SiB2 specific parameters from the card file

73.0.3 sib2_coldstart (Source File: sib2_coldstart.F90)

REVISION HISTORY:

```
 1 Mar 2004: Luis Gustavo G de Goncalves, SiB2 in LIS
18 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine sib2_coldstart(n)
```

USES:

```
use LIS_coreMod, only: LIS_rc
use LIS_timeMgrMod, only : LIS_date2time
use LIS_logMod, only : LIS_logunit
use sib2_lsmMod
```

DESCRIPTION:

This routine initializes the SiB2 state variables with some predefined values uniformly for the entire domain. These initial values will be overwritten by the values read from the supplied SiB2 model restart file.
The arguments are:

n index of the nest

73.0.4 sib2_dynsetup (Source File: sib2_dynsetup.F90)

REVISION HISTORY:

```
15 Apr 2002: Sujay Kumar, Initial Specification
 1 Mar 2004: Luis Gustavo G de Goncalves, SiB2 in LIS
 6 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine sib2_dynsetup(n)
```

USES:

```
use LIS_coreMod, only: LIS_rc
use sib2_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Updates the time dependent SiB2 variables The arguments are:

n index of the nest

The routines invoked are:

sib2_monthly (73.2.1)
reads the parameter file containing monthly varying data.

73.0.5 sib2_f2t (Source File: sib2_f2t.F90)

REVISION HISTORY:

```
 1 Mar 2004: Luis Gustavo G de Goncalves, SiB2 in LIS
 5 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine sib2_f2t(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc   ! LIS non-model-specific 1-D variables
use LIS_FORC_AttributesMod
use LIS_baseforcingMod, only : LIS_FORC_State
use LIS_logMod,         only : LIS_verify
use sib2_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This routine transfers the LIS provided forcing onto the SiB2 model tiles.
The arguments are:

n index of the nest

73.0.6 sib2_finalize (Source File: sib2_finalize.F90)

REVISION HISTORY:

```
04 Oct 2005: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine sib2_finalize()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use sib2_lsmMod
```

DESCRIPTION:

This routine cleans up the allocated memory structures in Sib2aic

---

73.1 Fortran: Module Interface sib2_lsmMod (Source File: sib2_lsmMod.F90)

Module for 1-D SiB2 land model driver variable initialization

REVISION HISTORY:

```
    Apr 2003: Sujay Kumar, Initial Code
  1 Mar 2004: Luis Gustavo G de Goncalves, SiB2 in LIS
  6 May 2004: David Mocko, made compatible with SiB-lings
 21 Dec 2007: Sujay Kumar, Updated for LIS 5.0
```

INTERFACE:

```
module sib2_lsmMod
```

USES:

```
use ESMF_Mod
use sib2_module
```

73.1.1 sib2_lsm_ini (Source File: sib2_lsmMod.F90)

INTERFACE:

```
subroutine sib2_lsm_ini()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod,   only : LIS_clock, LIS_calendar, LIS_seconds2time, &
     LIS_setMonthlyAlarm
use LIS_logMod,  only : LIS_verify
```

DESCRIPTION:

This routine creates the datatypes and allocates memory for SiB2-specific variables. It also invokes the routine to read the runtime specific options for noah from the configuration file.
The routines invoked are:

readcrdsib2 (73.0.2)
reads the runtime options for SiB2 LSM

73.1.2 sib2_main (Source File: sib2_main.F90)

REVISION HISTORY:

```
 1 Mar 2004: Luis Gustavo G de Goncalves, SiB2 in LIS
25 May 2004: David Mocko, removed duplicate code and provided ALMA outputs
```

INTERFACE:

```
subroutine sib2_main(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain
use sib2_lsmMod
use sib2alb_module, only : sib2alb
use LIS_histDataMod
```

DESCRIPTION:

This is the entry point for calling the SiB2 LSM physics. This routine calls the `sib2_routines` routine that performs the land surface computations, to solve for water and energy equations.
references:

sato, n., p. j. sellers, d. a randall, e. k. schneider, j. shukla, j. l kinter iii, y-t, hou, and albertazzi (1989) "effects of implementing the simple biosphere model in a general circulation model" j. atmos. sci., 46, 2757-2782.

sellers, p. j., d. a. randall, c. j. collatz, j. a. berry, c.b.field, d.a.dazlich, c.zhang, g.collelo,l.bounoua (1996) "a revised land-surface parameterization(sib2) for atmospheric general circulation models. part 1: model formulation. (accepted by jcl ,april 1996)

The arguments are:

n index of the nest

73.1.3 sib2_mapvegc (Source File: sib2_mapvegc.F90)

This subroutine converts the UMD classes to the SiB classes used by NOAH LSM (v 2.5). (Originally from Dag Lohmann at NCEP)

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault, Added SSiB LSM to LDAS
 1 Mar 2004: Luis Gustavo G de Goncalves, SiB2 in LIS
 5 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine sib2_mapvegc(vegt)

  implicit none
```

ARGUMENTS:

```
  integer, intent(inout) :: vegt
```

73.2 Fortran: Module Interface sib2_module (Source File: sib2_module.F90)

Module for 1-D SiB2 land model driver variable specification

REVISION HISTORY:

```
28 Apr 2002: K. Arsenault, added NOAH LSM 2.5 code to LDAS
14 Nov 2002: Sujay Kumar, Optimized version for LIS
 1 Mar 2004: Luis Gustavo G de Goncalves, SiB2 in LIS
22 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
module sib2_module
```

73.2.1 sib2_monthly (Source File: sib2_monthly.F90)

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault, Added NOAH LSM, Initial Code
 1 Mar 2004: Luis Gustavo G de Goncalves, SiB2 in LIS
18 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine sib2_monthly(n)
```

USES:

```
  use LIS_coreMod, only : LIS_rc, LIS_domain
  use LIS_timeMgrMod,   only : LIS_isMonthlyAlarmRinging
  use LIS_logMod,    only : LIS_logunit
  use sib2_lsmMod
```

DESCRIPTION:

This subroutine retrieves monthly SiB2 parameters
The arguments are:

n index of the nest

73.2.2 sib2_output (Source File: sib2_output.F90)

INTERFACE:

```
subroutine sib2_output(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_fileIOMod, only : LIS_create_output_directory, &
     LIS_create_output_filename,  &
     LIS_create_stats_filename
use LIS_histDataMod, only : LIS_histData
use LIS_historyMod,   only : LIS_writeModelOutput
use sib2_lsmMod

implicit none

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

DESCRIPTION:

This subroutines sets up methods to write model output from SiB2 LSM A timestamped filename and directory are created first. The routine supports model output in three data formats: Binary, Grib and NETCDF.
The arguments are:

n index of the nest

The routines invoked are:

LIS_create_output_directory (3.12.3)
creates a time stamped output directory

LIS_create_output_filename (3.12.4)
creates a time stamped output filename

LIS_create_stats_filename (3.12.6)
creates a stats filename

sib2_binout (**??**)
routine to write Sib2 model output in binary format

sib2_totint (**??**)
resets the time averaged variables

73.2.3 sib2_readrst (Source File: sib2_readrst.F90)

REVISION HISTORY:

```
 1 Mar 2004: Luis Gustavo G de Goncalves, SiB2 in LIS
 5 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine sib2_readrst
```

USES:

```
use LIS_coreMod, only : LIS_rc
use sib2_lsmMod
use LIS_historyMod, only : LIS_readvar_restart
use LIS_logMod, only : LIS_logunit, LIS_endrun
```

DESCRIPTION:

This program reads restart files for SiB2. This includes all relevant water/energy storages, tile information, and time information. It also rectifies changes in the tile space.

73.2.4 sib2_setup (Source File: sib2_setup.F90)

REVISION HISTORY:

```
 1 Mar 2004: Luis Gustavo G de Goncalves, SiB2 in LIS
22 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine sib2_setup
```

USES:

```
use LIS_coreMod, only : LIS_rc
use sib2_lsmMod
use sib2alb_module
```

DESCRIPTION:

This routine is the entry point to setup the parameters required for SiB2 LSM.

73.2.5 sib2_setvegparms (Source File: sib2_setvegparms.F90)

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault, Added NOAH LSM, Initial Code
13 Oct 2003: Sujay Kumar, Domain independent modifications
 1 Mar 2004: Luis Gustavo G de Goncalves, SiB2 in LIS
 5 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine sib2_setvegparms(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_logMod,    only : LIS_logunit
use sib2_lsmMod
```

DESCRIPTION:

This subroutine retrieves constant SiB2 parameters

73.2.6 sib2_writerst (Source File: sib2_writerst.F90)

REVISION HISTORY:

```
1 Mar 2004: Luis Gustavo G de Goncalves, SiB2 in LIS
6 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine sib2_writerst(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,   only : LIS_rc, LIS_masterproc
use LIS_logMod,    only : LIS_logunit
use lis_fileIOMod, only : LIS_create_output_directory, &
     LIS_create_restart_filename
use sib2_lsmMod

implicit none

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

DESCRIPTION:

This program writes restart files for SiB2. This includes all relevant water/energy storages, and tile information.
The routines invoked are:

LIS_create_output_directory (3.12.3)
creates a timestamped directory for the restart files

LIS_create_restart_filename (3.12.5)
generates a timestamped restart filename

sib2_dump_restart (73.2.7)
writes the sib2 variables into the restart file

73.2.7 sib2_dump_restart (Source File: sib2_writerst.F90)

REVISION HISTORY:

```
24 Aug 2004: James Geiger, Initial Specification
```

INTERFACE:

```
subroutine sib2_dump_restart(n, ftn)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_historyMod, only : LIS_writevar_restart
use sib2_lsmMod

implicit none

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

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:

drv_writevar_restart (3.17.3)
writes a variable to the restart file

73.3 Fortran: Module Interface sib2alb_module (Source File: sib2alb_module.F90)

In order to use the Sib2 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 Sib2'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 :: sib2alb_ini
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: sib2alb
```

73.3.1 sib2alb_ini (Source File: sib2alb_module.F90)

INTERFACE:

```
subroutine sib2alb_ini()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use sib2_lsmMod
use LIS_logMod,    only : LIS_logunit
```

DESCRIPTION:

Reads in runtime SiB2 albedo parameters to populate constants used in Sib2 albedo calculations

---

74 Catchment Land Surface Model

This section describes the interface implementations for the Catchment land surface model.

74.0.2 cat_dump_param (Source File: cat_dump_param.F90)

INTERFACE:

```
subroutine cat_dump_param(n)
```

DESCRIPTION:

This routine writes all of the catchment parameters to a file.

74.0.3 cat_dynsetup.F90 (Source File: cat_dynsetup.F90)

Sets the time varying parameters for catchment

REVISION HISTORY:

```
28 Jan 2002: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine cat_dynsetup(n)
```

USES:

74.0.4 cat_f2t (Source File: cat_f2t.F90)

REVISION HISTORY:

```
16 Dec 2005: Sujay Kumar; Initial Specification
11 Oct 2006: Sujay Kumar; modified to work with ESMF_State design
```

INTERFACE:

```
subroutine cat_f2t(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod , only : LIS_rc
use LIS_FORC_AttributesMod
use LIS_baseforcingMod, only : LIS_FORC_State
use LIS_constantsMod,       only : LIS_CONST_TKFRZ
use LIS_logMod,             only : LIS_verify
use cat_lsmMod, only : cat_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

74.0.5 cat_finalize (Source File: cat_finalize.F90)

Complete the finalize routines for catchment

REVISION HISTORY:

```
11 Feb 2006: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine cat_finalize()
```

USES:

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

74.0.6 cat_lsmMod.F90 (Source File: cat_lsmMod.F90)

Module for 1-D catchment land model driver variable initialization

REVISION HISTORY:

```
15 Dec 2005; Sujay Kumar, Initial Code
```

INTERFACE:

```
module cat_lsmMod
```

USES:

```
use ESMF_Mod
use catch_types
use drv_catch_types
use modis_alb_types

PRIVATE

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

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: cat_varalloc
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: cat_struc

type, public ::  cat_type_dec
   integer :: catopen       !Keeps track of opening files
   integer :: numout       !Counts number of output times for Cat
   integer :: count
   integer :: outInterval       !CAT Output Interval (minutes)
   integer :: rstInterval       !CAT restart interval (minutes)
   type(ESMF_Alarm)           :: rstAlarm
```

```
    type(ESMF_Alarm)           :: outAlarm
    integer :: varid(39)         !For netcdf output
    real    :: zol_lookup(9,12)
    real    :: vgd_lookup(9,12)
    real    :: initSM          ! initial soil moisture
    real    :: initST          ! initial soil temperature
    character*100 :: rfile     !restart file
    character*100 :: zolfile     !roughness length table
    character*100 :: vgdfile     !veg displacement ht table
    character*100 :: paramdir     !directory for parameter files
    character*100 :: modisdir     !sub-directory for MODIS files
    character*100 :: vegclassfile !vegetation classification file
    character*100 :: soilparamfile!soil parameters file
    character*100 :: sltsfile     !surface layer timescales file
    character*100 :: topo_ar_file !topography parameters file
    character*100 :: topo_bf_file !topography parameters file
    character*100 :: topo_ts_file !topography parameters file

    logical, pointer              :: good_forcing_mask(:)
    type(cat_param_type), pointer :: cat_param(:)
    type(met_force_type), pointer :: met_force(:)
    type(cat_progn_type), pointer :: cat_progn(:)
    type(cat_diagn_type), pointer :: cat_diagn(:)
    type(cat_output_type), pointer :: cat_output(:)
    type(modis_alb_type), pointer :: modis_param(:,:)
  end type cat_type_dec

  type(cat_type_dec), pointer :: cat_struc(:)
```

74.0.7 cat_varalloc (Source File: cat_lsmMod.F90)

Reads in runtime catchment parameters, allocates memory for variables

INTERFACE:

```
subroutine cat_varalloc()
```

USES:

```
  use LIS_coreMod, only : LIS_rc
  use LIS_timeMgrMod,   only : LIS_clock, LIS_calendar, LIS_seconds2time
  use LIS_logMod,       only : LIS_verify
```

74.0.8 cat_main (Source File: cat_main.F90)

REVISION HISTORY:

```
13 Jun 2005 Rolf Reichle, Initial Specification
17 Dec 2005 Sujay Kumar, Implementation in LIS
```

INTERFACE:

```
subroutine cat_main(nid)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_gfracMod, only : LIS_gfrac
use LIS_albedoMod, only : LIS_alb
use LIS_laiMod, only : LIS_lai
use LIS_constantsMod, only : LIS_CONST_STEBOL, LIS_CONST_TKFRZ
use LIS_logMod,  only : LIS_logunit
use LIS_histDataMod
use cat_lsmMod, only : cat_struc
use catch_diagn_routines, only : calc_asnow
use catch_types
use catchment_model
use catch_constants
use esat_qsat

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

74.0.9 cat_output (Source File: cat_output.F90)

INTERFACE:

```
subroutine cat_output(n)
```

USES:

```
use ESMF_mod
use LIS_coreMod, only : LIS_rc,  LIS_masterproc, LIS_npes
use lis_fileIOMod, only : LIS_create_output_directory, &
                          LIS_create_output_filename,  &
                          LIS_create_stats_filename
use LIS_histDataMod, only : LIS_histData
use LIS_historyMod,  only : LIS_writeModelOutput
use cat_lsmMod, only : cat_struc

implicit none

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

DESCRIPTION:

This subroutines sets up methods to write model output from Catchment LSM A timestamped filename and directory are created first. The routine supports model output in three data formats: Binary, Grib and NETCDF.
The arguments are:

n index of the nest

The routines invoked are:

LIS_create_output_directory (3.12.3)
creates a time stamped output directory

LIS_create_output_filename (3.12.4)
creates a time stamped output filename

LIS_create_stats_filename (3.12.6)
creates a stats filename

cat_binout (**??**)
routine to write Cat model output in binary format

cat_totint (**??**)
resets the time averaged variables

74.0.10 cat_readrestart (Source File: cat_readrestart.F90)

REVISION HISTORY:

```
16 Dec 2005: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine cat_readrestart
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain,LIS_masterproc, LIS_localPet
use LIS_logMod,  only : LIS_logunit, LIS_endrun
use LIS_historyMod, only : LIS_readvar_restart
use catch_constants
use cat_lsmMod
```

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

74.0.11 cat_setup (Source File: cat_setup.F90)

REVISION HISTORY:

```
16 Dec 2005: Sujay kumar; Initial Code
```

INTERFACE:

```
subroutine cat_setup()
```

USES:

```
use LIS_coreMod, only : LIS_rc,LIS_domain, LIS_masterproc
use LIS_logMod,    only : LIS_logunit
use LIS_soilsMod,  only : LIS_soils
use cat_lsmMod
use clsm_para_mapping
```

DESCRIPTION:

This routine is the entry point to set up the parameters required for Noah LSM. These include the soils, greenness, albedo, bottom temperature and the initialization of state variables in Noah.
The routines invoked are:

read_land_parameter1 (74.5.2)

read_land_parameter2 (74.5.3)

read_land_parameter3 (74.5.4)

modis_scale_param (**??**)

LIS_mapSoilType (3.20.16)

read_rst (74.3.3)

read_catchment_parameters (74.3.1)

get_param (74.3.5)

read_catchment_parameter_final (**??**)

cat_totinit (**??**)

read

74.0.12 cat_writerestart (Source File: cat_writerestart.F90)

REVISION HISTORY:

```
18 Dec 2005: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine cat_writerestart(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_logMod, only :LIS_logunit
use LIS_fileIOMod, only : LIS_create_output_directory, &
                          LIS_create_restart_filename
use cat_lsmMod
```

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 (3.12.3)
creates a timestamped directory for the restart files

LIS_create_restart_filename (3.12.5)
generates a timestamped restart filename

cat_dump_restart (74.0.13)
writes the catchment variables into the restart file

74.0.13 cat_dump_restart (Source File: cat_writerestart.F90)

REVISION HISTORY:

```
24 Aug 2004: James Geiger, Initial Specification
```

INTERFACE:

```
subroutine cat_dump_restart(n, ftn)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_historyMod, only : LIS_writevar_restart
use catch_constants
use cat_lsmMod

implicit none
```

ARGUMENTS:

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

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

74.1 Fortran: Module Interface cat_diagn_routines (Source File: catch_diagn_routines.F90)

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
```

74.1.1 calc_tsurf (Source File: catch_diagn_routines.F90)

INTERFACE:

```
subroutine calc_tsurf( N_cat, cat_param, cat_progn, tsurf )
```

DESCRIPTION:

Calculate diagnostic surface temperature "tsurf" from prognostics

---

74.1.2 calc_arX (Source File: catch_diagn_routines.F90)

INTERFACE:

```
subroutine calc_arX( N_cat, cat_param, cat_progn, ar1, ar2, ar4 )
```

DESCRIPTION:

???

74.1.3 calc_asnow (Source File: catch_diagn_routines.F90)

INTERFACE:

```
subroutine calc_asnow( N_cat, cat_progn, asnow )
```

DESCRIPTION:

Calculate diagnostic snow area from prognostic SWE

74.2 Fortran: Module Interface catch_types (Source File: catch_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
```

74.3 Fortran: Module Interface clsm_para_mapping (Source File: clsm_para_mapping.F90)

REVISION HISTORY:

```
30 Nov 2006, Sarith Mahanama:  Initial Specification
```

DESCRIPTION:

??

74.3.1 read_catchment_parameters (Source File: clsm_para_mapping.F90)

INTERFACE:

```
subroutine read_catchment_parameters(pathout)

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

ARGUMENTS:

```
   character(len=*) :: pathout
```

DESCRIPTION:

??

74.3.2 read_catchment_parameters_final (Source File: clsm_para_mapping.F90)

INTERFACE:

```
subroutine read_catchment_parameters_final()

   implicit none
```

DESCRIPTION:

??

74.3.3 read_rst (Source File: clsm_para_mapping.F90)

INTERFACE:

```
SUBROUTINE read_rst (filein1,filein2,tile_id)

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

ARGUMENTS:

```
  character(len=*) :: filein1,filein2
  integer, dimension (i_raster,j_raster) :: tile_id
```

DESCRIPTION:

??

74.3.4 get_tile_index (Source File: clsm_para_mapping.F90)

INTERFACE:

```
SUBROUTINE get_tile_index (               &
     lat,lon,tile_id,                     &
     tile_index)

  implicit none
```

ARGUMENTS:

```
  real , intent(in) :: lat,lon
  integer,  intent(in), dimension (i_raster,j_raster) :: tile_id
  integer, intent(out) :: tile_index
```

DESCRIPTION:

??

74.3.5 get_param (Source File: clsm_para_mapping.F90)

INTERFACE:

```
SUBROUTINE get_param (                    &
     tile_index,                          &
     soil_type_in,veg_type_in,            &
     veg_type_out,                        &
     BEE, PSIS,POROS,COND,WPWET,          &
     atau, btau, soil_depth,              &
     ARS1,ARS2,ARS3,ARA1,ARA2,ARA3,ARA4,  &
     ARW1,ARW2,ARW3,ARW4, bf1, bf2, bf3,  &
     tsa1, tsa2,tsb1, tsb2)

  implicit none
```

DESCRIPTION:

??

74.4 Fortran: Module Interface drv_catch_types (Source File: drv_catch_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
```

INTERFACE:

74.5 Fortran: Module Interface esat_qsat (Source File: esat_qsat.F90)

cleaned up qsat.f and function esat() from process_new.f

REVISION HISTORY:

```
reichle, 13 Jun 2005
```

INTERFACE:

```
module esat_qsat
```

74.5.1 pmonth (Source File: pmonth_stage4.F90)

REVISION HISTORY:

INTERFACE:

```
SUBROUTINE PMONTH (&
     NCH,    ITYP,   JDAY,   ALAT,&
     GREEN, ZLT, Z0, D, BUG,&
     SQSCAT, Z2, DZM,&
     RSOIL1, RSOIL2, SATCAP, RDC,  U2FAC &
     &                  )
```

74.5.2 read_land_parameters1 (Source File: read_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
```

INTERFACE:

```
subroutine read_land_parameters1(nest)
```

USES:

```
     use catch_constants, only : N_gndtmp
     use cat_lsmMod
     use LIS_coreMod, only : LIS_rc
     use LIS_logMod,    only : LIS_logunit, LIS_endrun, &
          LIS_getNextUnitNumber, LIS_releaseUnitNumber

     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

74.5.3 read_land_parameters2 (Source File: read_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
```

INTERFACE:

```
subroutine read_land_parameters2(nest)
```

USES:

```
!use tile_coord_types
use catch_constants, only : N_gndtmp
use cat_lsmMod
use LIS_coreMod, only : LIS_rc
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

74.5.4 read_land_parameters3 (Source File: read_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
```

INTERFACE:

```
subroutine read_land_parameters3(nest)
```

USES:

```
!use tile_coord_types
use catch_constants, only : N_gndtmp
use cat_lsmMod
use LIS_coreMod, only : LIS_rc
use LIS_logMod,    only : LIS_logunit

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

74.5.5 d2g (Source File: read_land_parameters.F90)

REVISION HISTORY:

```
10 Jul 2006 James Geiger, Implementation in LIS
```

INTERFACE:

```
function d2g(n,k)
```

USES:

```
use LIS_coreMod, only : LIS_domain
implicit none
```

ARGUMENTS:

```
integer :: d2g
integer, intent(in) :: n
integer, intent(in) :: k
```

DESCRIPTION:

This routine returns the global tile number (from the .til file) corresponding to the the given local tile number.
The arguments are:

n index of the nest

k local index of the tile

74.5.6 read_modis_scale_param (Source File: read_modis_scale_param.F90)

REVISION HISTORY:

```
              Rolf Reichle, Initial Specification
10 Jul 2006 James Geiger, Implementation in LIS
```

INTERFACE:

```
subroutine read_modis_scale_param(nest)
```

USES:

```
!use tile_coord_types
use cat_lsmMod
use LIS_coreMod, only : LIS_rc
use LIS_logMod,    only : LIS_logunit

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

Reads in modis scale parameters.
The arguments are:

nest index of the nest

74.5.7 readcatchcrd (Source File: readcatchcrd.F90)

Routine to read Catchment specific parameters from the card file.

REVISION HISTORY:

```
12 Feb 2006; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcatchcrd()
```

USES:

```
use cat_lsmMod, only : cat_struc
use LIS_logMod, only :LIS_logunit
use LIS_coreMod, only : LIS_config,LIS_rc
use ESMF_Mod
```

74.5.8 turb (Source File: turb.F90)

REVISION HISTORY:

INTERFACE:

```
subroutine turb(&
     nch,em,tm,um,tc,ea,psur,z0,&
     ra,eturb,dedea,dedtc,hsturb,dhsdea,dhsdtc)
```

---

75 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.

!ROUTINE:canhtset

REVISION HISTORY:

```
15 Nov 2002: Jon Gottschalck; Initial code
```

INTERFACE:

```
subroutine canhtset (n)
```

USES:

```
use clm_lsmMod          ! CLM tile variables
use clm_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.

75.0.9 clm_dynsetup (Source File: clm_dynsetup.F90)

REVISION HISTORY:

```
20 Jan 2003  Sujay Kumar Initial Specification
```

INTERFACE:

```
subroutine clm_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

---

75.0.10 clm_finalize (Source File: clm_finalize.F90)

REVISION HISTORY:

```
26 Oct 2005: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine clm_finalize()
```

USES:

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

DESCRIPTION:

This routine cleans up the allocated memory structures in CLM

75.1 Fortran: Module Interface clm_lsmMod (Source File: clm_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 ESMF_Mod
use clmtype

implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: clm_lsm_ini
```

PUBLIC TYPES:

```
public :: clm_struc
```

75.1.1 clm_lsm_ini (Source File: clm_lsmMod.F90)

INTERFACE:

```
subroutine clm_lsm_ini()
```

USES:

```
use clm_varpar,   only : nlevsoi, nlevsno, numpft, numrad
use pft_varcon
use shr_orb_mod,  only : shr_orb_params
use LIS_timeMgrMod,   only : LIS_clock, LIS_calendar, LIS_seconds2time
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:

readclmcrd (75.3.1) reads the runtime options for CLM

clm_varder_init (75.1.2) set initial values to CLM variables

iniTimeConst (75.2.3) initialize time invariant parameters

75.1.2 clm_lsm_init (Source File: clm_lsmMod.F90)

INTERFACE:

```
subroutine clm_lsm_init(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_precisionMod
use clm_varpar,   only : nlevsoi, nlevsno, numrad
```

DESCRIPTION:

Initializes clm variables
The arguments are:

n index of the nest

75.1.3 clm_main (Source File: clm_main.F90)

INTERFACE:

```
subroutine clm_main (n)
```

USES:

```
  use LIS_precisionMod
  use clm_lsmMod
  use clm_varpar, only : nlevsoi
  use clm_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 shr_sys_mod   , only : shr_sys_flush
  use LIS_histDataMod

  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

75.1.4 clm_output (Source File: clm_output.F90)

INTERFACE:

```
subroutine clm_output(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_masterproc, LIS_npes
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
use clm_lsmMod, only : clm_struc

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutines sets up methods to write model output from CLM A timestamped filename and directory are created first. The routine supports model output in three data formats: Binary, Grib and NETCDF. The arguments are:

n index of the nest

The routines invoked are:

LIS_create_output_directory (3.12.3) creates a time stamped output directory

LIS_create_output_filename (3.12.4) creates a time stamped output filename

LIS_create_stats_filename (3.12.6) creates a stats filename

clm_binout (**??**) routine to write CLM model output in binary format

clm_gribout (**??**) routine to write CLM model output in grib format

clm_totint (**??**) resets the time averaged variables

75.1.5 clm_readrestart (Source File: clm_readrestart.F90)

REVISION HISTORY:

```
20 Jan 2003; Sujay Kumar Initial Specification
```

INTERFACE:

```
subroutine clm_readrestart()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_logMod, only : LIS_logunit, LIS_endrun
use LIS_historyMod, only : LIS_readvar_restart
use clm_varpar,     only : nlevsno, nlevsoi
use clm_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

---

75.1.6 clm_setup (Source File: clm_setup.F90)

REVISION HISTORY:

```
20 Jan 2003  Sujay Kumar Initial Specification
```

INTERFACE:

```
subroutine clm_setup()
```

USES:

```
use LIS_coreMod, only: LIS_rc
use LIS_timeMgrMod, only : LIS_get_nstep
use LIS_logMod, only :LIS_logunit
use clm_lsmMod
use clm_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 (75) reads the canopy height information

iniTimeVar (75.2.4) initialize time variant parameters

75.1.7 clm_writerestart (Source File: clm_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 clm_writerestart(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_logMod, only : LIS_logunit
use LIS_fileIOMod, only : LIS_create_output_directory, &
                           LIS_create_restart_filename
use clm_lsmMod, only : clm_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 (3.12.3) creates a timestamped directory for the restart files

LIS_create_restart_filename (3.12.5) generates a timestamped restart filename

clm_dump_restart (75.1.8) writes the CLM variables into the restart file reads a variable from the restart file

---

75.1.8 clm_dump_restart (Source File: clm_writerestart.F90)

INTERFACE:

```
subroutine clm_dump_restart(n, ftn)
```

USES:

```
use clm_lsmMod
use clm_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

---

75.1.9 clmlairead.F90: (Source File: clmlairead.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 clmlairead (n)
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_laiMod, only : LIS_lai
use LIS_saiMod, only : LIS_sai
use clm_lsmMod
```

75.2 Fortran: Module Interface clmtype (Source File: clmtype.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
```

75.2.1 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_Mod
use LIS_precisionMod
use LIS_coreMod ,      only : LIS_rc
use LIS_baseforcingMod, only : LIS_FORC_State
use LIS_FORC_AttributesMod
use LIS_logMod,         only : LIS_verify
use clm_lsmMod
use clm_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

75.2.2 clm_setwrfexport.F90 (Source File: clm_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 clm_setwrfexport(n, expState)
```

USES:

```
use ESMF_Mod
use LIS_precisionMod
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 clm_lsmMod
use clm_varcon, only : denh2o, denice, hvap, hsub, hfus, istwet
use clm_varpar, only : nlevsoi
```

75.2.3 iniTimeConst (Source File: iniTimeConst.F90)

INTERFACE:

```
subroutine iniTimeConst (n)
```

USES:

```
use LIS_precisionMod
use clm_lsmMod
use clm_varpar   , only : nlevsoi, nlevlak, numrad
use clm_varcon   , only : istsoil, istice, istdlak, istslak, istwet, spval
use pft_varcon   , only : ncorn, nwheat, roota_par, rootb_par,  &
                          z0mr, displar, dleaf, rhol, rhos, taul, taus, xl, &
                          qe25, vcmx25, mp, c3psn
use clm_varsur   , only : zlak, dzlak, zsoi, dzsoi, zisoi
use shr_const_mod, only : SHR_CONST_PI
use LIS_coreMod,   only : LIS_rc, 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

75.2.4 iniTimeVar (Source File: iniTimeVar.F90)

INTERFACE:

```
subroutine iniTimeVar (n, eccen, obliqr, lambm0 , mvelpp)
```

USES:

```
use LIS_precisionMod
use LIS_coreMod, only : LIS_rc
use clm_lsmMod
use LIS_timeMgrMod, only : LIS_get_curr_calday
use clm_varpar,   only : nlevsno, nlevsoi
use clm_varcon  , only : bdsno, istice, istwet, &
     istsoil, denice, denh2o, tfrz, spval, doalb
use shr_sys_mod , only : 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)

75.3 Fortran: Module Interface atmdrvMod (Source File: atmdrvMod.F90)

This module defines the variables and routines to enable the interaction with an atmospheric component

USES:

75.3.1 atmdrv (Source File: atmdrvMod.F90)

INTERFACE:

```
subroutine atmdrv(n)
```

USES:

```
use ESMF_Mod
use clm_lsmMod
use clm_varcon  , only : rair, cpair, po2, pco2, tcrit, tfrz
use LIS_precisionMod
use LIS_coreMod, only : LIS_rc
use LIS_FORC_AttributesMod
use LIS_baseforcingMod, 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 : readclmcrd

REVISION HISTORY:

```
14 Oct 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readclmcrd()
```

USES:

```
use LIS_logMod, only : LIS_logunit
use LIS_coreMod, only : LIS_rc, LIS_config
use ESMF_Mod
use clm_lsmMod, only : clm_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
```

75.3.2 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
```

75.3.3 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
```

75.4 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:

75.4.1 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
```

75.4.2 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
```

75.4.3 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
```

75.5 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
```

75.5.1 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
```

75.5.2 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
```

75.5.3 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
```

75.5.4 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
```

75.5.5 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
```

75.5.6 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
```

75.5.7 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
```

75.5.8 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
```

75.5.9 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
```

75.5.10 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
!
```

75.5.11 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
!
```

75.5.12 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
```

75.6 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
```

75.6.1 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
```

75.6.2 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
```

75.6.3 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
```

75.6.4 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
```

75.6.5 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
```

75.6.6 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
```

75.6.7 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
```

75.7 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
```

75.7.1 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
```

75.7.2 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
```

75.7.3 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
```

75.7.4 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
```

75.7.5 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
```

75.7.6 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
```

75.7.7 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
```

75.7.8 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
```

75.7.9 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
```

75.7.10 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
```

75.7.11 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
```

75.7.12 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
```

75.7.13 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
```

75.7.14 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
```

75.7.15 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
```

75.7.16 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
```

75.7.17 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
```

75.7.18 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
```

75.7.19 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
```

75.7.20 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
```

75.7.21 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
```

75.7.22 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
```

75.7.23 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
```

75.7.24 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
```

75.7.25 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
```

75.7.26 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
```

75.7.27 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
```

75.7.28 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
```

75.7.29 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
```

75.7.30 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
```

75.8 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
```

75.8.1 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
```

75.8.2 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
```

75.8.3 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
```

75.8.4 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
```

75.8.5 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 Redistribution-Type 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
```

76 HySSIB Land Surface Model

This section describes the interface implementations for the HySSIB land surface model.

76.0.6 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.

76.0.7 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

---

76.0.8 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
```

INTERFACE:

```
subroutine hyssib_f2t(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod       , only : LIS_rc
use LIS_FORC_AttributesMod
use LIS_baseforcingMod, 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

76.0.9 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

---

76.0.10 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, LIS_domain
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

---

76.1 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

initsm initial soil moisture for a cold start run

initTemp initial soil temperature for a cold start run

outInterval output writing interval

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
```

USES:

```
use ESMF_Mod
use hyssib_module

 implicit none

 PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: hyssib_lsm_ini
```

PUBLIC TYPES:

```
public :: hyssib_struc
```

76.1.1 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_seconds2time
use LIS_logMod,       only : LIS_verify
```

DESCRIPTION:

Reads in runtime HY-SSiB parameters, allocates memory for variables
The routines invoked are:

hyssib_readcrd (76.2.2)
reads the runtime options for Hyssib LSM

76.1.2 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 November: Chuck Alonge, Updates for LIS 5.0
```

INTERFACE:

```
subroutine hyssib_main(n)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain
use LIS_albedoMod, only : LIS_alb
use LIS_logMod,    only : LIS_logunit
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

76.1.3 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

---

76.2 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

albsf1 month 1 albedo value

albsf2 month 2 albedo value

albsf interpolated albedo value

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

soilmoist1 soil moisture layer 1 (kg/m2)

soilmoist2 soil moisture layer 2 (kg/m2)

soilmoist3 soil moisture layer 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
```

76.2.1 hyssib_output (Source File: hyssib_output.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_output(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use lis_fileIOMod, only : LIS_create_output_directory, &
     LIS_create_output_filename,  &
     LIS_create_stats_filename
use LIS_histDataMod, only : LIS_histData
use LIS_historyMod, only : LIS_writeModelOutput
```

```
use LIS_logMod,  only   : LIS_logunit, LIS_endrun
use hyssib_lsmMod         ! HY-SSiB-specific variables

implicit none

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

DESCRIPTION:

This subroutines sets up methods to write model output from Hyssib LSM A timestamped filename and directory are created first. The routine supports model output in three data formats: Binary, and Grib. The arguments are:

n index of the nest

The routines invoked are:

LIS_create_output_directory (3.12.3)
creates a time stamped output directory

LIS_create_output_filename (3.12.4)
creates a time stamped output filename

LIS_create_stats_filename (3.12.6)
creates a stats filename

hyssib_binout (**??**)
routine to write Hyssib model output in binary format

hyssib_totint (76.2.8)
resets the time averaged variables

76.2.2 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
```

INTERFACE:

```
subroutine hyssib_readcrd()
```

USES:

```
use ESMF_Mod
use LIS_coreMod,   only : LIS_rc, LIS_config
use LIS_logMod,    only : LIS_logunit
use hyssib_lsmMod, only : hyssib_struc
```

DESCRIPTION:

Routine to read HY-SSiB specific parameters from the LIS configuration file

---

76.2.3 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, only : LIS_logunit, LIS_endrun
use hyssib_lsmMod
```

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,nch    - 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

76.2.4 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_domain
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:

readtbot (3.48.2)
retrieves the bottom temperature data

76.2.5 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_domain
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.

76.2.6 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
```

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 (76.4.1)
initializes the Hyssib vegetation parameter module

hyssibalb_vegini (76.3.1)
initializes the Hyssib albedo parameter module

hyssib_setvegparms (76.2.7)
initializes the vegetation-related parameters in Hyssib

hyssib_settbot (76.2.4)
initializes the bottom temperature fields

hyssib_settopostd (76.2.5)
initializes the std. dev. of topography in hyssib

hyssib_gfrac (76.0.10)
initializes monthly vegetation parameters in hyssib

hyssib_coldstart (76.0.6)
initializes the hyssib state variables

LIS_setMonthlyAlarm (3.31.14)
sets a monthly alarm if a climatology of bottom temperature is being read

---

76.2.7 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_domain
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
```

76.2.8 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
```

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

76.2.9 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_Mod
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_logMod, only :LIS_logunit
use lis_fileIOMod, only : LIS_create_output_directory, &
                          LIS_create_restart_filename
use hyssib_lsmMod
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 (3.12.3)
creates a timestamped directory for the restart files

LIS_create_restart_filename (3.12.5)
generates a timestamped restart filename

hyssib_dump_restart (76.2.10)
writes the hyssib variables into the restart file

76.2.10 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)
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_historyMod, only : LIS_writevar_restart
use hyssib_lsmMod

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 following is the list of variables written in the Hyssib restart file:

```
nc,nr,nch    - 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_writevar_restart (3.17.3)
writes a variable to the restart file

76.3 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
```

76.3.1 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

---

76.4 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
```

76.4.1 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

---

77 Mosaic Land Surface Model

This section describes the interface implementations for the Mosaic land surface model.

77.0.2 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.**
```

77.0.3 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.

77.0.4 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_Mod
use LIS_coreMod ,     only : LIS_rc
use LIS_FORC_AttributesMod
use LIS_baseforcingMod, 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

77.0.5 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

---

77.0.6 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 ESMF_Mod
use mos_module
implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
---------------------------------------------------------------------------
 public :: mos_lsm_ini
---------------------------------------------------------------------------
```

PUBLIC TYPES:

```
---------------------------------------------------------------------------
 public :: mos_struc
```

77.0.7 mos_lsm_ini (Source File: mos_lsmMod.F90)

INTERFACE:

```
subroutine mos_lsm_ini()
```

USES:

```
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod,   only : LIS_clock, LIS_calendar, LIS_seconds2time
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 (77.1.8)
reads the runtime options for Mosaic LSM

77.0.8 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 MOS25 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 mosto 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
  use LIS_logMod, only : LIS_logunit
  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 {\tt MOSTILE} routine that performs the land surface
   computations, to solve for water and energy equations. For documentation
   of the {\tt 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:
   \begin{description}
   \item[n]
    index of the nest
   \end{description}
```

77.1 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
```

77.1.1 mos_output (Source File: mos_output.F90)

REVISION HISTORY:

```
25 Sep 2007: Sujay Kumar; Upgraded for LIS5.0
```

INTERFACE:

```
subroutine mos_output(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use LIS_histDataMod, only : LIS_histData
use LIS_historyMod,  only : LIS_writeModelOutput
use mos_lsmMod
use lis_fileIOMod, only : LIS_create_output_directory, &
     LIS_create_output_filename,  &
     LIS_create_stats_filename
use LIS_logMod,   only : LIS_logunit, LIS_endrun

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutines sets up methods to write model output from Mosaic LSM A timestamped filename and directory are created first. The routine supports model output binary format.
The arguments are:

n index of the nest

The routines invoked are:

LIS_create_output_directory (3.12.3)
creates a time stamped output directory

LIS_create_output_filename (3.12.4)
creates a time stamped output filename

LIS_create_stats_filename (3.12.6)
creates a stats filename

mos_binout (**??**)
routine to write Mosaic model output in binary format

mos_gribout (**??**)
routine to write Mosaic model output in grib format

mos_totint (**??**)
resets the time averaged variables

77.1.2 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

INTERFACE:

```
subroutine mos_readrestart
```

USES:

```
use LIS_coreMod, only : LIS_rc
use mos_lsmMod
use LIS_historyMod, only : LIS_readvar_restart
use LIS_logMod,     only : LIS_logunit, LIS_endrun
```

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,nch    - 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:

drv_readvar_restart (3.17.6)
reads a variable from the restart file

77.1.3 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 (77.1.9)
initializes the static soil and topography fields

mapsib2umd (77.0.2)
maps the SIB vegetation types to UMD classification

mos_coldstart (77.0.3)
initializes the noah state variables

77.1.4 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 INTERFACE:

```
subroutine mos_writerst(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_masterproc
use mos_lsmMod, only    : mos_struc
use LIS_logMod, only    : LIS_logunit
use lis_fileIOMod, only : LIS_create_output_directory, &
     LIS_create_restart_filename

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 (3.12.3)
creates a timestamped directory for the restart files

LIS_create_restart_filename (3.12.5)
generates a timestamped restart filename

mos_dump_restart (77.1.5)
writes the Mosaic variables into the restart file

77.1.5 mos_dump_restart (Source File: mos_writerst.F90)

REVISION HISTORY:

```
25 Sep 2007: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine mos_dump_restart(ftn,n)
```

USES:

```
use LIS_coreMod, only  : LIS_rc, LIS_masterproc
use mos_lsmMod
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 following is the list of variables written in the Mosaic restart file:

```
nc,nr,nch    - 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_writevar_restart (3.17.3)
writes a variable to the restart file

77.1.6 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
use LIS_timeMgrMod, only : LIS_date2time
use LIS_laiMod,     only : LIS_lai
use LIS_saiMod,     only : LIS_sai
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

---

77.1.7 paramb (Source File: paramb.F90)

REVISION HISTORY:

```
11 Mar 2001 : Matt Rodell, Initial Specification
```

INTERFACE:

```
subroutine paramb (nch,tex,b)

implicit none
```

ARGUMENTS:

```
integer :: nch
integer :: tex(nch)
real    :: b(nch)
```

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:

nch number of tiles

tex texture class

b b parameter

77.1.8 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_Mod
use LIS_logMod, only : LIS_logunit
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.

77.1.9 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_domain
use LIS_soilsMod, only : LIS_soils
use LIS_logMod,   only : LIS_logunit, LIS_endrun
use mos_lsmMod

 implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This subroutine sets the static mosaic parameters related to vegetation, soils and topography.

77.2 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
```

77.2.1 texture (Source File: texture.F90)

REVISION HISTORY:

```
7 Mar 2001 : Matt Rodell, Initial Specification
```

INTERFACE:

```
subroutine texture (n, nc, nr, nch, sand,silt,clay,tex)
```

USES:

```
use LIS_coreMod, only : LIS_domain, LIS_rc

implicit none
```

ARGUMENTS:

```
integer, intent(in)    :: n
integer, intent(in)    :: nc,nr,nch
real,    intent(in)    :: sand(nc,nr),silt(nc,nr),clay(nc,nr)
integer, intent(inout) :: tex(nch)
```

DESCRIPTION:

Finds the USDA texture class based on percentages of sand, silt, and clay.

---

78 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.

78.0.2 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_Mod
use LIS_coreMod,        only : LIS_rc, LIS_localPet
use LIS_FORC_AttributesMod
use LIS_baseforcingMod, 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

78.0.3 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_Mod
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%nch(n))
real :: t2v, rho
real :: temp1(LIS_rc%lnc(n), LIS_rc%lnr(n))
```

78.0.4 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_Mod
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

integer          :: c,r, t,gid
```

```
type(ESMF_Field) :: qleField, qhField, qgField, avgsurftField, albedoField, &
     emissField, z0Field, xiceField, qsfcField, cqs2Field, chs2Field, &
     t2Field, th2Field,q2Field, psfcField,qfxField,chField
real, pointer    :: qle(:,:), qh(:,:), qg(:,:), avgsurft(:,:), albedo(:,:),&
     emiss(:,:), z0(:,:), xice(:,:), qsfc(:,:), cqs2(:,:), chs2(:,:), &
     t2(:,:),q2(:,:),th2(:,:),psfc(:,:),qfx(:,:),ch(:,:)
integer          :: rc

integer :: nobjs
```

78.0.5 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_Mod
use LIS_coreMod ,      only : LIS_rc
use LIS_baseforcingMod, 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

78.0.6 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_Mod
  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
  integer, intent(in) :: n
  type(ESMF_State)      :: expState
  integer               :: i,j,k
  real                  :: temp(LIS_rc%nch(n))

  call LIS_tile2grid(n,LISWRF_export(n)%avgsurft,noah271_struc(n)%noah%t1)
  call LIS_tile2grid(n,LISWRF_export(n)%qh,noah271_struc(n)%noah%qh)
  call LIS_tile2grid(n,LISWRF_export(n)%eta_kinematic,noah271_struc(n)%noah%eta_kinematic)
  call LIS_tile2grid(n,LISWRF_export(n)%qle,noah271_struc(n)%noah%qle)
  call LIS_tile2grid(n,LISWRF_export(n)%qg,noah271_struc(n)%noah%qg)
  call LIS_tile2grid(n,LISWRF_export(n)%rootmoist,noah271_struc(n)%noah%rootmoist) !units?
  call LIS_tile2grid(n,LISWRF_export(n)%soilm,noah271_struc(n)%noah%soilm)
  call LIS_tile2grid(n,LISWRF_export(n)%qs,noah271_struc(n)%noah%qs)
  call LIS_tile2grid(n,LISWRF_export(n)%qsb,noah271_struc(n)%noah%qsb)
  call LIS_tile2grid(n,LISWRF_export(n)%albedo,noah271_struc(n)%noah%albedo)
  call LIS_tile2grid(n,LISWRF_export(n)%znt,noah271_struc(n)%noah%z0)
  call LIS_tile2grid(n,LISWRF_export(n)%snocvr,noah271_struc(n)%noah%sca)
  call LIS_tile2grid(n,LISWRF_export(n)%q1,noah271_struc(n)%noah%q1)
  call LIS_tile2grid(n,LISWRF_export(n)%snow,noah271_struc(n)%noah%sneqv*1000.0)
  call LIS_tile2grid(n,LISWRF_export(n)%cmc,noah271_struc(n)%noah%cmc)
  call LIS_tile2grid(n,LISWRF_export(n)%qsm,noah271_struc(n)%noah%qsm)
  call LIS_tile2grid(n,LISWRF_export(n)%chs2,noah271_struc(n)%noah%chs2)
  call LIS_tile2grid(n,LISWRF_export(n)%cqs2,noah271_struc(n)%noah%cqs2)
  call LIS_tile2grid(n,LISWRF_export(n)%emiss,noah271_struc(n)%noah%emiss)
  call LIS_tile2grid(n,LISWRF_export(n)%xice,noah271_struc(n)%noah%xice)

  do i=1,LIS_rc%nch(n)
     temp(i) = noah271_struc(n)%noah(i)%smc(1)
   print*, 'LISsideout:', noah271_struc(n)%noah(i)%chs2, noah271_struc(n)%noah(i)%cqs2
  enddo
  call LIS_tile2grid(n,LISWRF_export(n)%smc1,temp)
  do i=1,LIS_rc%nch(n)
     temp(i) = noah271_struc(n)%noah(i)%smc(2)
  enddo
  call LIS_tile2grid(n,LISWRF_export(n)%smc2,temp)
  do i=1,LIS_rc%nch(n)
     temp(i) = noah271_struc(n)%noah(i)%smc(3)
  enddo
  call LIS_tile2grid(n,LISWRF_export(n)%smc3,temp)
  do i=1,LIS_rc%nch(n)
     temp(i) = noah271_struc(n)%noah(i)%smc(4)
  enddo
  call LIS_tile2grid(n,LISWRF_export(n)%smc4,temp)
  do i=1,LIS_rc%nch(n)
     temp(i) = noah271_struc(n)%noah(i)%stc(1)
```

```
   enddo
   call LIS_tile2grid(n,LISWRF_export(n)%stc1,temp)
   do i=1,LIS_rc%nch(n)
      temp(i) = noah271_struc(n)%noah(i)%stc(2)
   enddo
   call LIS_tile2grid(n,LISWRF_export(n)%stc2,temp)
   do i=1,LIS_rc%nch(n)
      temp(i) = noah271_struc(n)%noah(i)%stc(3)
   enddo
   call LIS_tile2grid(n,LISWRF_export(n)%stc3,temp)
   do i=1,LIS_rc%nch(n)
      temp(i) = noah271_struc(n)%noah(i)%stc(4)
   enddo
   call LIS_tile2grid(n,LISWRF_export(n)%stc4,temp)
   do i=1,LIS_rc%nch(n)
      temp(i) = noah271_struc(n)%noah(i)%sh2o(1)
   enddo
   call LIS_tile2grid(n,LISWRF_export(n)%sh2o1,temp)
   do i=1,LIS_rc%nch(n)
      temp(i) = noah271_struc(n)%noah(i)%sh2o(2)
   enddo
   call LIS_tile2grid(n,LISWRF_export(n)%sh2o2,temp)
   do i=1,LIS_rc%nch(n)
      temp(i) = noah271_struc(n)%noah(i)%sh2o(3)
   enddo
   call LIS_tile2grid(n,LISWRF_export(n)%sh2o3,temp)
   do i=1,LIS_rc%nch(n)
      temp(i) = noah271_struc(n)%noah(i)%sh2o(4)
   enddo
   call LIS_tile2grid(n,LISWRF_export(n)%sh2o4,temp)

   do j=1,LIS_rc%lnr(n)
      do i=1,LIS_rc%lnc(n)
         k = LIS_domain(n)%gindex(i,j)
         if(LIS_domain(n)%gindex(i,j).ne.-1) then
            LISWRF_export(n)%xland(i,j) = noah271_struc(n)%noah(k)%vegt
         endif
      enddo
   enddo
   print*, 'EXITING noah271_setwrfexport '
 end subroutine noah271_setwrfexport

\markboth{Left}{Source File: noah271\_wrf\_f2t.F90,  Date: Wed Jun  9 09:46:35 EDT 2010
}
```


```
  -----------------------BEGIN NOTICE -- DO NOT EDIT-----------------------
   NASA Goddard Space Flight Center Land Information System (LIS)
   See RELEASE_NOTES.txt for more information.
   The LIS source code and documentation are not in the public domain
   and may not be freely distributed.  Only qualified entities may receive
   the source code and documentation.
   Qualified entities must be covered by a Software Usage Agreement.
   The Software Usage Agreement contains all the terms and conditions
   regarding the release of the LIS software.
```

```
   NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE
   SOFTWARE FOR ANY PURPOSE.  IT IS PROVIDED AS IS WITHOUT EXPRESS OR
   IMPLIED WARRANTY.  NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE
   LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE.
   See the Software Usage Agreement for the full disclaimer of warranty.
  -------------------------END NOTICE -- DO NOT EDIT-----------------------
%//////////////////////////////////////////////////////////////////////////

\mbox{}\hrulefill\

\subsubsection{noah271\_wrf\_f2t (Source File: noah271\_wrf\_f2t.F90)}


   \label{noah271_wrf_f2t}

\bigskip{\sf REVISION HISTORY:}
\begin{verbatim}    15 Oct 2002: Sujay Kumar; Initial Code
```

INTERFACE:

```
 subroutine noah271_wrf_f2t(n)
```

USES:

```
   use ESMF_Mod
   use LIS_coreMod ,      only : LIS_rc
   use LIS_baseforcingMod, 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

78.0.7 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_Mod
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

78.0.8 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_Mod
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

78.0.9 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_Mod
use LIS_coreMod, only : LIS_rc,LIS_domain
use noah271_lsmMod
```

78.0.10 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_Mod
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

78.0.11 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_Mod
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

78.0.12 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_Mod
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

78.0.13 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_Mod
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

78.0.14 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_Mod
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.

78.0.15 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_Mod
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

78.0.16 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_Mod
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.

78.0.17 noah271_getsoilm (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_Mod
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

!ROUTINE : noah_physics.F90
DESCRIPTION: SUB-DRIVER FOR "NOAH/OSU LSM" FAMILY OF PHYSICS SUBROUTINES FOR A SOIL/VEG/SNOWPACK LAND-SURFACE MODEL TO UPDATE SOIL MOISTURE, SOIL ICE, SOIL TEMPERATURE, SKIN TEMPERATURE, SNOWPACK WATER CONTENT, SNOWDEPTH, AND ALL TERMS OF THE SURFACE ENERGY BALANCE AND SURFACE WATER BALANCE (EXCLUDING INPUT ATMOSPHERIC FORCINGS OF DOWNWARD RADIATION AND PRECIP)

REVISION HISTORY:

```
    28 Apr 2002: Kristi Arsenault; Added NOAH LSM 2.5 code to LDAS
    15 May 2002: Urszula Jambor; Changed LOGICAL to LOGICAL*1 to match new
                  GRIB libraries
    28 May 2002: Kristi Arsenault; Updated NOAH code to 2.5.1, and
                 corrected problem with FX in DEVAP function.
    12 Jun 2002: Kristi Arsenault; Updated NOAH code to 2.5.2
    04 Nov 2002: Kristi Arsenault; Incorporated new TBOT fields
    24 Jun 2003: Kristi Arsenault; Updated Noah LSM to 2.6 version
    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)
```

78.0.18 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_gfracMod,     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%nch(n))
 real, intent(out)       :: t1(LIS_rc%nch(n))
real                    :: t1(LIS_rc%nch(n))
```

DESCRIPTION:

Diagnostic routine to update noah variables based on an updated T1 value

---

78.0.19 noah271_coldstart (Source File: noah271_coldstart.F90)

INTERFACE:

```
subroutine noah271_coldstart()
```

USES:

```
use LIS_coreMod,      only : LIS_rc, LIS_domain
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 noah state variables with some predefined values uniformly for the entire domain. These initial values will be overwritten by the values read from the supplied Noah model restart file.

78.0.20 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 Noah. Currently this routine is a placeholder

---

78.0.21 noah271_f2t (Source File: noah271_f2t.F90)

REVISION HISTORY:

```
15 Oct 2002: Sujay Kumar; Initial Code
```

INTERFACE:

```
subroutine noah271_f2t(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,        only : LIS_rc
use LIS_FORC_AttributesMod
use LIS_baseforcingMod, 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

78.0.22 noah271_finalize (Source File: noah271_finalize.F90)

REVISION HISTORY:

```
28 Apr 2002: Sujay Kumar; Initial code
```

INTERFACE:

```
subroutine noah271_finalize()
```

USES:

```
use LIS_coreMod,  only : LIS_rc
use noah271_lsmMod
```

DESCRIPTION:

This routine cleans up the allocated memory structures in Noah

---

78.1 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 Noah LSM. It also provides the entry method for the initialization of Noah-specific variables. The derived data type `noah_struc` includes the variables that specify the runtime options and other control variables as described below:

rfile name of the noah 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

noahopen 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

npr number of prognostic state variables used in data assimilation

varid NETCDF ids for variables (used for netcdf output)

initsm initial soil moisture for a cold start run

initTemp initial soil temperature for a cold start run

outInterval output writing interval

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 noah LSM specific variables

REVISION HISTORY:

```
Apr 2003; Sujay Kumar, Initial Code
```

USES:

```
use ESMF_Mod
use noah271_module

implicit none

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

PUBLIC MEMBER FUNCTIONS:

```
-----------------------------------------------------------------------------
public :: noah271_lsm_ini
-----------------------------------------------------------------------------
```

PUBLIC TYPES:

```
-----------------------------------------------------------------------------
public :: noah271_struc
```

78.1.1 noah271_lsm_ini (Source File: noah271_lsmMod.F90)

INTERFACE:

```
subroutine noah271_lsm_ini()
```

USES:

```
use LIS_coreMod,      only : LIS_rc
use LIS_timeMgrMod,   only : LIS_clock, LIS_calendar, LIS_seconds2time
use LIS_logMod,       only : LIS_verify
```

DESCRIPTION:

This routine creates the datatypes and allocates memory for noah-specific variables. It also invokes the routine to read the runtime specific options for noah from the configuration file.
The routines invoked are:

readnoahcrd (**??**)
reads the runtime options for Noah LSM

78.1.2 noah271_main (Source File: noah271_main.F90)

INTERFACE:

```
subroutine noah271_main(n)
```

USES:

```
use LIS_coreMod,       only : LIS_rc, LIS_domain
use LIS_gfracMod,      only : LIS_gfrac
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
use LIS_histDataMod
use module_sf_noah271lsm, only : SFLX, SFCDIF, SNFRAC
use noah271_lsmMod


implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This is the entry point for calling the Noah 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 noah routines, please see the "NOAH LSM USERS GUIDE" maintained at the public server at NOAA NCEP.
The arguments are:

n index of the nest

78.1.3 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

! ----------------------------------------------------------------------
! COLD START:  determine liquid soil water content (SH2O)
! NSOIL number of soil layers
! ----------------------------------------------------------------------
!  SH2O <= SMC for T < 273.149K (-0.001C)
  IF (STC .LT. 273.149) THEN
! ----------------------------------------------------------------------
! first guess following explicit solution for Flerchinger Eqn from Koren
! et al, JGR, 1999, Eqn 17 (KCOUNT=0 in FUNCTION FRH2O).
! ----------------------------------------------------------------------
     BX = BETA
     IF ( BETA .GT. BLIM )  BX = BLIM
     FK=(((HLICE/(GRAV*(-PSIS)))* &
          ((STC-T0)/STC))**(-1/BX))*SMCMAX
     IF (FK .LT. 0.02) FK = 0.02
     SH2O = MIN ( FK, SMC )
! ----------------------------------------------------------------------
! now use iterative solution for liquid soil water content using
! FUNCTION FRH2O with the initial guess for SH2O from above explicit
! first guess.
! ----------------------------------------------------------------------
     SH2O = FRH2O(STC,SMC,SH2O,SMCMAX,BETA,PSIS)

  ELSE
! ----------------------------------------------------------------------
!  SH2O = SMC for T => 273.149K (-0.001C)
     SH2O=SMC
! ----------------------------------------------------------------------
  ENDIF

  RETURN
```

78.2 Fortran: Module Interface noah271_module (Source File: noah271_module.F90)

The code in this file provides a description of the data structure containing the noah 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

vegmp1 month 1 greenness fraction value

vegmp2 month 2 greenness fraction value

albsf1 date 1 snow free albedo value

albsf2 date 2 snow free albedo value

albsf interpolated snow free albedo value

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

mxsnoalb maximum albedo expected over deep snow

tempbot bottom boundary temperature

tempbot1 month 1 bottom boundary temperature

tempbot2 month 2 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

soilm_prev soil moisture from the previous model output

swe_prev snow water equivalent from the previous model output

tair_agl_min minimum 2 meter AGL temperature

rhmin minimum 2 meter relative humidity

REVISION HISTORY:

```
28 Apr 2002: K. Arsenault added NOAH LSM 2.5 code to LDAS.
14 Nov 2002: Sujay Kumar Optimized version for LIS
```

78.2.1 noah271_output (Source File: noah271_output.F90)

INTERFACE:

```
subroutine noah271_output(n)
```

USES:

```
#if ( defined USE_NETCDF )
  use netcdf
#endif
  use ESMF_Mod
  use LIS_coreMod,    only : LIS_rc, LIS_masterproc
  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
  use LIS_logMod,     only : LIS_logunit
  use noah271_lsmMod

  implicit none

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

DESCRIPTION:

This subroutines sets up methods to write model output from Noah LSM A timestamped filename and directory are created first. The routine supports model output in three data formats: Binary, Grib and NETCDF.
The arguments are:

n index of the nest

The routines invoked are:

LIS_create_output_directory (3.12.3)
creates a time stamped output directory

LIS_create_output_filename (3.12.4)
creates a time stamped output filename

LIS_create_stats_filename (3.12.6)
creates a stats filename

noah_binout (**??**)
routine to write Noah model output in binary format

noah_gribout (**??**)
routine to write Noah model output in grib format

noah_netcdfout (**??**)
routine to write Noah model output in netcdf format

noah_totint (**??**)
resets the time averaged variables

78.2.2 noah271_setsoils (Source File: noah271_setsoils.F90)

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault;  Added NOAH 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
```

INTERFACE:

```
subroutine noah271_setsoils
```

USES:

```
use LIS_coreMod,   only : LIS_rc, LIS_domain
use LIS_soilsMod,  only : LIS_soils
use LIS_logMod,    only : LIS_logunit
use noah271_lsmMod
```

DESCRIPTION:

This subroutine sets the soil parameters in Noah. Noah uses a lookup table based on the input soil texture to derive these parameters.
The routines invoked are:

LIS_mapSoilType (3.20.16)
method to derive the soil texture type from the sand, silt, and clay fractions.

78.2.3 noah271_settbot (Source File: noah271_settbot.F90)

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault;  Added NOAH LSM, Initial Code
13 Oct 2003: Sujay Kumar; Domain independent modifications
```

INTERFACE:

```
subroutine noah271_settbot
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_domain
use noah271_lsmMod
```

DESCRIPTION:

This subroutine retrieves bottom boundary temperature for Noah 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:

readtbot (3.48.2)
retrieves the bottom temperature data

78.2.4 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 Noah LSM. These include the soils, greenness, albedo, bottom temperature and the initialization of state variables in Noah.
The routines invoked are:

noah_setvegparms (78.2.5)
initializes the vegetation-related parameters in Noah

noah_settbot (78.2.3)
initializes the bottom temperature fields

noah_setsoils (78.2.2)
initializes the soil parameters

setMonthlyAlarm (**??**)
sets a monthly alarm if a climatology of bottom temperature is being read

---

78.2.5 noah271_setvegparms (Source File: noah271_setvegparms.F90)

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault;  Added NOAH 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
```

USES:

```
use LIS_coreMod, only : LIS_rc, LIS_domain
use noah271_lsmMod
```

DESCRIPTION:

This subroutine retrieves NOAH vegetation parameters. The current implementation uses a table-based lookup based on SIB 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
```

78.2.6 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 NOAH LSM into LDAS
28 May 2002: Kristi Arsenault; For STARTCODE=4, corrected SNEQV values
              and put SMC, SH2O, STC limit for GDAS and GEOS forcing.
14 Jun 2003: Sujay Kumar , Separated the write restart from the original
              code
```

INTERFACE:

```
subroutine noah271_writerst(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,   only : LIS_rc, LIS_masterproc
use LIS_logMod,    only : LIS_logunit
use LIS_fileIOMod, only : LIS_create_output_directory, &
                          LIS_create_restart_filename
use noah271_lsmMod

implicit none
```

ARGUMENTS:

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

DESCRIPTION:

This program writes restart files for Noah. This includes all relevant water/energy storage and tile information
The routines invoked are:

LIS_create_output_directory (3.12.3)
creates a timestamped directory for the restart files

LIS_create_restart_filename (3.12.5)
generates a timestamped restart filename

noah_dump_restart (78.2.7)
writes the noah variables into the restart file

78.2.7 noah271_dump_restart (Source File: noah271_writerst.F90)

REVISION HISTORY:

```
24 Aug 2004: James Geiger, Initial Specification
```

INTERFACE:

```
subroutine noah271_dump_restart(n, ftn)
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_historyMod, only : LIS_writevar_restart
use noah271_lsmMod

implicit none

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

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 Noah restart file:

```
nc,nr,nch    - grid and tile space dimensions
t1           - noah skin temperature
cmc          - noah canopy moisture storage
snowh        - noah snow depth
sneqv        - noah snow water equivalent
stc          - noah soil temperature (for each layer)
smc          - noah soil moisture (for each layer)
sh2o         - noah liquid only soil moisture (for each layer)
ch           - noah heat/moisture exchange coefficient
cm           - noah momentum exchange coefficient
```

The routines invoked are:

LIS_writevar_restart (3.17.3)
writes a variable to the restart file

78.2.8 noah271rst (Source File: noah271rst.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 NOAH LSM into LDAS
28 May 2002: Kristi Arsenault; For STARTCODE=4, corrected SNEQV values
              and put SMC, SH2O, STC limit for GDAS and GEOS forcing.
```

INTERFACE:

```
subroutine noah271rst
```

USES:

```
use LIS_coreMod,    only : LIS_rc, LIS_masterproc
use LIS_historyMod, only : LIS_readvar_restart
use LIS_logMod,     only : LIS_logunit, LIS_endrun
use noah271_lsmMod
```

DESCRIPTION:

This program reads restart files for Noah. This includes all relevant water/energy storages and tile information. The following is the list of variables specified in the Noah restart file:

```
nc,nr,nch    - grid and tile space dimensions
t1           - noah skin temperature
cmc          - noah canopy moisture storage
```

```
snowh        - noah snow depth
sneqv        - noah snow water equivalent
stc          - noah soil temperature (for each layer)
smc          - noah soil moisture (for each layer)
sh2o         - noah liquid only soil moisture (for each layer)
ch           - noah heat/moisture exchange coefficient
cm           - noah momentum exchange coefficient
```

The routines invoked are:

LIS_readvar_restart (3.17.6)
reads a variable from the restart file

noah_coldstart (78.0.19)
initializes the noah state variables

78.2.9 readnoah271crd (Source File: readnoah271crd.F90)

REVISION HISTORY:

```
14 Oct 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readnoah271crd()
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc, LIS_config
use LIS_logMod,  only : LIS_logunit, LIS_verify
use noah271_lsmMod, only : noah271_struc
```

DESCRIPTION:

This routine reads the options specific to Noah LSM from the LIS configuration file.

---

79 Template

This is a dummy placeholder implementation of a LSM, to be used as a template for new implementations.

79.0.10 readtemplatecrd (Source File: readtemplatecrd.F90)

REVISION HISTORY:

```
21 Jul 2004; Sujay Kumar, Initial Code
23 Oct 2007; Kristi Arsenault, Updated for use in LISv5.0
```

INTERFACE:

```
subroutine readtemplatecrd()
```

USES:

```
use ESMF_Mod
use LIS_coreMod,     only : LIS_rc, LIS_config
use LIS_logMod,      only : LIS_logunit
use template_lsmMod, only : template_struc
```

DESCRIPTION:

This routine reads the options specific to template LSM option from the LIS configuration file.

79.0.11 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

79.0.12 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_Mod
use LIS_coreMod,        only : LIS_rc
use LIS_baseforcingMod, 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

79.0.13 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

79.1 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 ESMF_Mod
use template_module

implicit none

PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
public :: template_lsm_ini
```

PUBLIC TYPES:

```
public :: template_struc
```

79.1.1 template_lsm_ini (Source File: template_lsmMod.F90)

INTERFACE:

```
subroutine template_lsm_ini()
```

USES:

```
use ESMF_Mod
use LIS_coreMod, only : LIS_rc
use LIS_timeMgrMod,   only : LIS_clock, LIS_calendar, LIS_seconds2time
use LIS_logMod,       only : LIS_verify
```

DESCRIPTION:

79.1.2 template_main (Source File: template_main.F90)

79.1.3 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)

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

79.2 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
```

79.2.1 template_output (Source File: template_output.F90)

INTERFACE:

```
subroutine template_output(n)
```

USES:

```
use ESMF_Mod
use LIS_coreMod,   only : LIS_rc, LIS_masterproc
use LIS_histDataMod, only : LIS_histData
use LIS_historyMod,  only : LIS_writeModelOutput
use LIS_logMod,    only : LIS_logunit, LIS_endrun
use LIS_fileIOMod, only : LIS_create_output_directory, &
     LIS_create_output_filename,  &
     LIS_create_stats_filename
use template_lsmMod,only : template_struc

implicit none

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

DESCRIPTION:

Forcing-only (Template) option for calling the output routines.
The arguments are:

n index of the nest

The routines invoked are:

LIS_create_output_directory (3.12.3)
creates a time stamped output directory

LIS_create_output_filename (3.12.4)
creates a time stamped output filename

LIS_create_stats_filename (3.12.6)
creates a stats filename

template_binout (**??**)
routine to write Template model output in binary format

79.2.2 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)

79.2.3 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

79.2.4 templaterst (Source File: templaterst.F90)

INTERFACE:

```
subroutine templaterst()
```

USES:

```
implicit none
```

ARGUMENTS:

DESCRIPTION:

Calls the restart reading routines for the forcing-only option (template)
The arguments are:

n index of the nest

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.