

SPHERLS

1.0

Generated by Doxygen 1.5.6

Fri Apr 20 14:22:28 2012

Contents

1	Using and Modifying SPHERLS	1
1.1	Overview	1
1.1.1	The Basics	1
1.2	The Equations	2
1.3	Program Flow	2
1.4	Installing SPHERLS	3
1.4.1	Requirements	3
1.4.2	Optional Requirements	4
1.4.3	Installing PETSC Library	4
1.4.4	Installing FFTW Library	4
1.4.5	Installing HDF4 Library	5
1.4.6	Installing Doxygen	5
1.4.7	Installing Python	5
1.4.8	Installing SPHERLS	5
1.5	Using SPHERLS	5
1.6	Modifing or Developing SPHERLS	6
1.7	Message Passing	7
2	Todo List	9
3	Boundary Conditions	13
4	Class Index	45
4.1	Class List	45
5	File Index	47
5.1	File List	47
6	Class Documentation	49
6.1	eos Class Reference	49

6.1.1	Detailed Description	50
6.1.2	Constructor & Destructor Documentation	50
6.1.2.1	eos	50
6.1.2.2	eos	50
6.1.2.3	eos	50
6.1.2.4	~eos	50
6.1.3	Member Function Documentation	51
6.1.3.1	operator=	51
6.1.3.2	readAscii	51
6.1.3.3	readBobsAscii	51
6.1.3.4	writeAscii	51
6.1.3.5	readBin	51
6.1.3.6	writeBin	52
6.1.3.7	dGetPressure	52
6.1.3.8	dGetEnergy	52
6.1.3.9	dGetOpacity	53
6.1.3.10	dDRhoDP	53
6.1.3.11	dSoundSpeed	53
6.1.3.12	getEKappa	54
6.1.3.13	getPEKappa	54
6.1.3.14	getPEKappaGamma	54
6.1.3.15	getPEKappaGammaCp	55
6.1.3.16	getPKappaGamma	55
6.1.3.17	gamma1DelAdC_v	56
6.1.3.18	getPAndDRhoDP	56
6.1.3.19	getEAndDTDE	56
6.1.4	Member Data Documentation	56
6.1.4.1	nNumRho	57
6.1.4.2	nNumT	57
6.1.4.3	dXMassFrac	57
6.1.4.4	dYMassFrac	57
6.1.4.5	dLogRhoMin	57
6.1.4.6	dLogRhoDelta	57
6.1.4.7	dLogTMin	57
6.1.4.8	dLogTDelta	58
6.1.4.9	dLogP	58

6.1.4.10	dLogE	58
6.1.4.11	dLogKappa	58
6.2	Functions Class Reference	59
6.2.1	Detailed Description	59
6.2.2	Constructor & Destructor Documentation	59
6.2.2.1	Functions	59
6.2.3	Member Data Documentation	60
6.2.3.1	fpCalculateNewVelocities	60
6.2.3.2	fpCalculateNewGridVelocities	60
6.2.3.3	fpCalculateNewRadii	60
6.2.3.4	fpCalculateNewDensities	60
6.2.3.5	fpCalculateNewEnergies	60
6.2.3.6	fpCalculateDeltat	60
6.2.3.7	fpCalculateAveDensities	60
6.2.3.8	fpCalculateNewEOSVars	61
6.2.3.9	fpCalculateNewAV	61
6.2.3.10	fpModelWrite	61
6.2.3.11	fpWriteWatchZones	61
6.2.3.12	fpUpdateLocalBoundaryVelocitiesNewGrid	61
6.2.3.13	fpImplicitSolve	61
6.2.3.14	fpCalculateNewEddyVisc	61
6.3	Global Class Reference	62
6.3.1	Detailed Description	62
6.3.2	Constructor & Destructor Documentation	62
6.3.2.1	Global	62
6.3.3	Member Data Documentation	62
6.3.3.1	procTop	62
6.3.3.2	messPass	63
6.3.3.3	grid	63
6.3.3.4	time	63
6.3.3.5	parameters	63
6.3.3.6	output	63
6.3.3.7	performance	63
6.3.3.8	functions	63
6.3.3.9	implicit	63
6.4	Grid Class Reference	64

6.4.1	Detailed Description	65
6.4.2	Constructor & Destructor Documentation	67
6.4.2.1	Grid	67
6.4.3	Member Data Documentation	68
6.4.3.1	nM	68
6.4.3.2	nTheta	68
6.4.3.3	nPhi	68
6.4.3.4	nDM	68
6.4.3.5	nR	68
6.4.3.6	nD	69
6.4.3.7	nU	70
6.4.3.8	nU0	70
6.4.3.9	nV	71
6.4.3.10	nW	71
6.4.3.11	nT	72
6.4.3.12	nE	72
6.4.3.13	nP	72
6.4.3.14	nKappa	73
6.4.3.15	nGamma	73
6.4.3.16	nDenAve	73
6.4.3.17	nQ0	74
6.4.3.18	nQ1	74
6.4.3.19	nQ2	75
6.4.3.20	nDTheta	75
6.4.3.21	nDPhi	75
6.4.3.22	nSinThetaIJK	76
6.4.3.23	nSinThetaIJp1halfK	76
6.4.3.24	nCotThetaIJp1halfK	76
6.4.3.25	nCotThetaIJK	76
6.4.3.26	nDCosThetaIJK	77
6.4.3.27	nEddyVisc	77
6.4.3.28	nNumDims	77
6.4.3.29	nNumVars	77
6.4.3.30	nNumIntVars	78
6.4.3.31	nNum1DZones	78
6.4.3.32	nNumGhostCells	78

6.4.3.33	nGlobalGridDims	78
6.4.3.34	nVariables	78
6.4.3.35	nLocalGridDims	79
6.4.3.36	dLocalGridNew	79
6.4.3.37	dLocalGridOld	80
6.4.3.38	nStartUpdateExplicit	80
6.4.3.39	nEndUpdateExplicit	81
6.4.3.40	nStartUpdateImplicit	81
6.4.3.41	nEndUpdateImplicit	82
6.4.3.42	nStartGhostUpdateExplicit	82
6.4.3.43	nEndGhostUpdateExplicit	82
6.4.3.44	nStartGhostUpdateImplicit	83
6.4.3.45	nEndGhostUpdateImplicit	83
6.4.3.46	nCenIntOffset	83
6.4.3.47	nGlobalGridPositionLocalGrid	84
6.5	Implicit Class Reference	85
6.5.1	Detailed Description	85
6.5.2	Constructor & Destructor Documentation	85
6.5.2.1	Implicit	86
6.5.3	Member Data Documentation	86
6.5.3.1	nNumImplicitZones	86
6.5.3.2	matCoeff	86
6.5.3.3	vecTCorrections	86
6.5.3.4	vecRHS	86
6.5.3.5	vecTCorrectionsLocal	86
6.5.3.6	kspContext	86
6.5.3.7	vecscatTCorrections	87
6.5.3.8	nMaxNumIterations	87
6.5.3.9	dTolerance	87
6.5.3.10	nNumRowsALocal	87
6.5.3.11	nNumRowsALocalSB	87
6.5.3.12	nNumDerPerRow	87
6.5.3.13	nTypeDer	88
6.5.3.14	nLocDer	88
6.5.3.15	nLocFun	88
6.5.3.16	dDerivativeStepFraction	88

6.5.3.17	dCurrentRelTError	88
6.5.3.18	nCurrentNumIterations	88
6.5.3.19	nMaxNumSolverIterations	89
6.5.3.20	dMaxErrorInRHS	89
6.5.3.21	dAverageRHS	89
6.6	MessPass Class Reference	90
6.6.1	Detailed Description	90
6.6.2	Constructor & Destructor Documentation	90
6.6.2.1	MessPass	90
6.6.3	Member Data Documentation	90
6.6.3.1	typeSendNewGrid	90
6.6.3.2	typeRecvOldGrid	91
6.6.3.3	typeSendNewVar	91
6.6.3.4	typeRecvNewVar	91
6.6.3.5	requestSend	91
6.6.3.6	requestRecv	91
6.6.3.7	statusSend	91
6.6.3.8	statusRecv	91
6.7	Output Class Reference	92
6.7.1	Detailed Description	92
6.7.2	Constructor & Destructor Documentation	92
6.7.2.1	Output	92
6.7.3	Member Data Documentation	92
6.7.3.1	nDumpFrequencyStep	93
6.7.3.2	dDumpFrequencyTime	93
6.7.3.3	dTimeLastDump	93
6.7.3.4	nNumTimeStepsSinceLastPrint	93
6.7.3.5	bDump	93
6.7.3.6	bPrint	93
6.7.3.7	nPrintMode	93
6.7.3.8	sBaseOutputFileName	94
6.7.3.9	ofWatchZoneFiles	94
6.7.3.10	watchzoneList	94
6.7.3.11	nPrintFrequencyStep	94
6.7.3.12	dPrintFrequencyTime	94
6.7.3.13	dTimeLastPrint	94

6.8	Parameters Class Reference	95
6.8.1	Detailed Description	95
6.8.2	Constructor & Destructor Documentation	95
6.8.2.1	Parameters	95
6.8.3	Member Data Documentation	96
6.8.3.1	bEOSGammaLaw	96
6.8.3.2	bAdiabatic	96
6.8.3.3	nTypeTurbulenceMod	96
6.8.3.4	dPi	96
6.8.3.5	dSigma	96
6.8.3.6	dG	97
6.8.3.7	dGamma	97
6.8.3.8	sEOSFileName	97
6.8.3.9	eosTable	97
6.8.3.10	dA	97
6.8.3.11	dAVThreshold	98
6.8.3.12	dDonorFrac	98
6.8.3.13	dAlpha	98
6.8.3.14	dTolerance	98
6.8.3.15	nMaxIterations	99
6.8.3.16	dEddyViscosity	99
6.8.3.17	dMaxConvectiveVelocity	99
6.8.3.18	dMaxConvectiveVelocity_c	99
6.8.3.19	dPrt	99
6.9	Performance Class Reference	101
6.9.1	Detailed Description	101
6.9.2	Constructor & Destructor Documentation	101
6.9.2.1	Performance	101
6.9.3	Member Data Documentation	101
6.9.3.1	dStartTimer	101
6.9.3.2	dEndTimer	101
6.10	ProcTop Class Reference	103
6.10.1	Detailed Description	103
6.10.2	Constructor & Destructor Documentation	103
6.10.2.1	ProcTop	103
6.10.3	Member Data Documentation	103

6.10.3.1	nNumProcs	103
6.10.3.2	nProcDims	104
6.10.3.3	nPeriodic	104
6.10.3.4	nCoords	104
6.10.3.5	nRank	104
6.10.3.6	nNumNeighbors	104
6.10.3.7	nNeighborRanks	105
6.10.3.8	nNumRadialNeighbors	105
6.10.3.9	nRadialNeighborRanks	105
6.10.3.10	nRadialNeighborNeighborIDs	105
6.11	Time Class Reference	106
6.11.1	Detailed Description	106
6.11.2	Constructor & Destructor Documentation	106
6.11.2.1	Time	106
6.11.3	Member Data Documentation	106
6.11.3.1	dDeltat_np1half	107
6.11.3.2	dDeltat_nm1half	107
6.11.3.3	dDeltat_n	107
6.11.3.4	dt	107
6.11.3.5	dEndTime	107
6.11.3.6	nEndTimeStep	108
6.11.3.7	dTimeStepFactor	108
6.11.3.8	nTimeStepIndex	108
6.11.3.9	bVariableTimeStep	108
6.11.3.10	dConstTimeStep	108
6.11.3.11	dPerChange	108
6.11.3.12	dDelRho_t_Rho_max	108
6.11.3.13	dDelT_t_T_max	109
6.11.3.14	dDelE_t_E_max	109
6.12	WatchZone Class Reference	110
6.12.1	Detailed Description	110
7	File Documentation	111
7.1	/home/cgeroux/SPHERLS/src/eos.cpp File Reference	111
7.1.1	Detailed Description	111
7.2	/home/cgeroux/SPHERLS/src/eos.h File Reference	112
7.2.1	Detailed Description	112

7.3	dataManipulation.cpp File Reference	113
7.3.1	Detailed Description	114
7.3.2	Function Documentation	114
7.3.2.1	average3DTo1DBoundariesNew	114
7.3.2.2	average3DTo1DBoundariesOld	114
7.3.2.3	fin	115
7.3.2.4	init	115
7.3.2.5	initImplicitCalculation	116
7.3.2.6	initUpdateLocalBoundaries	117
7.3.2.7	modelRead	118
7.3.2.8	modelWrite_GL	118
7.3.2.9	modelWrite_TEOS	119
7.3.2.10	setupLocalGrid	119
7.3.2.11	updateLocalBoundaries	120
7.3.2.12	updateLocalBoundariesNewGrid	120
7.3.2.13	updateLocalBoundaryVelocitiesNewGrid_R	121
7.3.2.14	updateLocalBoundaryVelocitiesNewGrid_RT	121
7.3.2.15	updateLocalBoundaryVelocitiesNewGrid_RTP	121
7.3.2.16	updateNewGridWithOld	121
7.3.2.17	updateOldGrid	121
7.4	dataManipulation.h File Reference	123
7.4.1	Detailed Description	123
7.4.2	Function Documentation	123
7.4.2.1	average3DTo1DBoundariesNew	124
7.4.2.2	average3DTo1DBoundariesOld	124
7.4.2.3	fin	124
7.4.2.4	init	125
7.4.2.5	initImplicitCalculation	126
7.4.2.6	initUpdateLocalBoundaries	126
7.4.2.7	modelRead	127
7.4.2.8	modelWrite_GL	128
7.4.2.9	modelWrite_TEOS	128
7.4.2.10	setupLocalGrid	129
7.4.2.11	updateLocalBoundaries	129
7.4.2.12	updateLocalBoundariesNewGrid	130
7.4.2.13	updateLocalBoundaryVelocitiesNewGrid_R	130

7.4.2.14	updateLocalBoundaryVelocitiesNewGrid_RT	130
7.4.2.15	updateLocalBoundaryVelocitiesNewGrid_RTP	131
7.4.2.16	updateNewGridWithOld	131
7.4.2.17	updateOldGrid	131
7.5	dataMonitoring.cpp File Reference	132
7.5.1	Detailed Description	132
7.5.2	Function Documentation	132
7.5.2.1	bFileExists	133
7.5.2.2	finWatchZones	133
7.5.2.3	initWatchZones	133
7.5.2.4	writeWatchZones_R_GL	134
7.5.2.5	writeWatchZones_R_TEOS	134
7.5.2.6	writeWatchZones_RT_GL	134
7.5.2.7	writeWatchZones_RT_TEOS	135
7.5.2.8	writeWatchZones_RTP_GL	135
7.5.2.9	writeWatchZones_RTP_TEOS	136
7.6	dataMonitoring.h File Reference	137
7.6.1	Detailed Description	137
7.6.2	Function Documentation	137
7.6.2.1	bFileExists	137
7.6.2.2	finWatchZones	138
7.6.2.3	initWatchZones	138
7.6.2.4	writeWatchZones_R_GL	138
7.6.2.5	writeWatchZones_R_TEOS	139
7.6.2.6	writeWatchZones_RT_GL	139
7.6.2.7	writeWatchZones_RT_TEOS	139
7.6.2.8	writeWatchZones_RTP_GL	140
7.6.2.9	writeWatchZones_RTP_TEOS	140
7.7	global.cpp File Reference	141
7.7.1	Detailed Description	141
7.8	global.h File Reference	142
7.8.1	Detailed Description	142
7.8.2	Define Documentation	142
7.8.2.1	DUMP_VERSION	143
7.8.2.2	SEDOV	143
7.8.2.3	SIGNEGDEN	143

7.8.2.4	SIGNEGENG	143
7.8.2.5	SIGNEGTEMP	143
7.8.2.6	TRACKMAXSOLVERERROR	143
7.8.2.7	VISCOUS_ENERGY_EQ	143
7.9	main.cpp File Reference	144
7.9.1	Detailed Description	144
7.9.2	Function Documentation	144
7.9.2.1	main	144
7.9.2.2	signalHandler	146
7.10	main.h File Reference	147
7.10.1	Detailed Description	147
7.10.2	Function Documentation	147
7.10.2.1	main	147
7.10.2.2	signalHandler	148
7.11	physEquations.cpp File Reference	149
7.11.1	Detailed Description	152
7.11.2	Function Documentation	152
7.11.2.1	calDelt_CONST	152
7.11.2.2	calDelt_R_GL	153
7.11.2.3	calDelt_R_TEOS	153
7.11.2.4	calDelt_RT_GL	153
7.11.2.5	calDelt_RT_TEOS	154
7.11.2.6	calDelt_RTP_GL	154
7.11.2.7	calDelt_RTP_TEOS	154
7.11.2.8	calNewD_R	155
7.11.2.9	calNewD_RT	155
7.11.2.10	calNewD_RTP	155
7.11.2.11	calNewDenave_None	156
7.11.2.12	calNewDenave_R	156
7.11.2.13	calNewDenave_RT	156
7.11.2.14	calNewDenave_RTP	156
7.11.2.15	calNewE_R_AD	157
7.11.2.16	calNewE_R_NA	157
7.11.2.17	calNewE_R_NA_LES	157
7.11.2.18	calNewE_RT_AD	158
7.11.2.19	calNewE_RT_NA	158

7.11.2.20 calNewE_RT_NA_LES	159
7.11.2.21 calNewE_RTP_AD	160
7.11.2.22 calNewE_RTP_NA	160
7.11.2.23 calNewE_RTP_NA_LES	161
7.11.2.24 calNewEddyVisc_None	161
7.11.2.25 calNewEddyVisc_R_CN	162
7.11.2.26 calNewEddyVisc_R_SM	162
7.11.2.27 calNewEddyVisc_RT_CN	162
7.11.2.28 calNewEddyVisc_RT_SM	162
7.11.2.29 calNewEddyVisc_RTP_CN	162
7.11.2.30 calNewEddyVisc_RTP_SM	163
7.11.2.31 calNewP_GL	163
7.11.2.32 calNewPEKappaGamma_TEOS	163
7.11.2.33 calNewQ0_R_GL	164
7.11.2.34 calNewQ0_R_TEOS	164
7.11.2.35 calNewQ0Q1_RT_GL	164
7.11.2.36 calNewQ0Q1_RT_TEOS	164
7.11.2.37 calNewQ0Q1Q2_RTP_GL	165
7.11.2.38 calNewQ0Q1Q2_RTP_TEOS	165
7.11.2.39 calNewR	165
7.11.2.40 calNewTPKappaGamma_TEOS	166
7.11.2.41 calNewU0_R	166
7.11.2.42 calNewU0_RT	166
7.11.2.43 calNewU0_RTP	167
7.11.2.44 calNewU_R	167
7.11.2.45 calNewU_R_LES	168
7.11.2.46 calNewU_RT	169
7.11.2.47 calNewU_RT_LES	169
7.11.2.48 calNewU_RTP	170
7.11.2.49 calNewU_RTP_LES	171
7.11.2.50 calNewV_RT	172
7.11.2.51 calNewV_RT_LES	173
7.11.2.52 calNewV_RTP	173
7.11.2.53 calNewV_RTP_LES	174
7.11.2.54 calNewVelocities_R	174
7.11.2.55 calNewVelocities_R_LES	175

7.11.2.56 calNewVelocities_RT	175
7.11.2.57 calNewVelocities_RT_LES	175
7.11.2.58 calNewVelocities_RTP	176
7.11.2.59 calNewVelocities_RTP_LES	176
7.11.2.60 calNewW_RTP	176
7.11.2.61 calNewW_RTP_LES	177
7.11.2.62 calOldDenave_None	177
7.11.2.63 calOldDenave_R	178
7.11.2.64 calOldDenave_RT	178
7.11.2.65 calOldDenave_RTP	178
7.11.2.66 calOldEddyVisc_R_CN	178
7.11.2.67 calOldEddyVisc_R_SM	178
7.11.2.68 calOldEddyVisc_RT_CN	179
7.11.2.69 calOldEddyVisc_RT_SM	179
7.11.2.70 calOldEddyVisc_RTP_CN	179
7.11.2.71 calOldEddyVisc_RTP_SM	179
7.11.2.72 calOldP_GL	180
7.11.2.73 calOldPEKappaGamma_TEOS	180
7.11.2.74 calOldQ0_R_GL	180
7.11.2.75 calOldQ0_R_TEOS	181
7.11.2.76 calOldQ0Q1_RT_GL	181
7.11.2.77 calOldQ0Q1_RT_TEOS	181
7.11.2.78 calOldQ0Q1Q2_RTP_GL	182
7.11.2.79 calOldQ0Q1Q2_RTP_TEOS	182
7.11.2.80 dEOS_GL	182
7.11.2.81 dImplicitEnergyFunction_None	182
7.11.2.82 dImplicitEnergyFunction_R	183
7.11.2.83 dImplicitEnergyFunction_R_LES	183
7.11.2.84 dImplicitEnergyFunction_R_LES_SB	184
7.11.2.85 dImplicitEnergyFunction_R_SB	184
7.11.2.86 dImplicitEnergyFunction_RT	185
7.11.2.87 dImplicitEnergyFunction_RT_LES	186
7.11.2.88 dImplicitEnergyFunction_RT_LES_SB	186
7.11.2.89 dImplicitEnergyFunction_RT_SB	187
7.11.2.90 dImplicitEnergyFunction_RTP	188
7.11.2.91 dImplicitEnergyFunction_RTP_LES	188

7.11.2.92	dImplicitEnergyFunction_RTP_LES_SB	189
7.11.2.93	dImplicitEnergyFunction_RTP_SB	190
7.11.2.94	implicitSolve_None	191
7.11.2.95	implicitSolve_R	191
7.11.2.96	implicitSolve_RT	191
7.11.2.97	implicitSolve_RTP	192
7.11.2.98	initDonorFracAndMaxConVel_R_GL	192
7.11.2.99	initDonorFracAndMaxConVel_R_TEOS	192
7.11.2.100	initDonorFracAndMaxConVel_RT_GL	192
7.11.2.101	initDonorFracAndMaxConVel_RT_TEOS	193
7.11.2.102	initDonorFracAndMaxConVel_RTP_GL	193
7.11.2.103	initDonorFracAndMaxConVel_RTP_TEOS	193
7.11.2.104	initInternalVars	193
7.11.2.105	setInternalVarInf	194
7.11.2.106	setMainFunctions	194
7.12	physEquations.h File Reference	195
7.12.1	Detailed Description	198
7.12.2	Function Documentation	198
7.12.2.1	calDelt_CONST	198
7.12.2.2	calDelt_R_GL	198
7.12.2.3	calDelt_R_TEOS	199
7.12.2.4	calDelt_RT_GL	199
7.12.2.5	calDelt_RT_TEOS	200
7.12.2.6	calDelt_RTP_GL	200
7.12.2.7	calDelt_RTP_TEOS	201
7.12.2.8	calNewD_R	201
7.12.2.9	calNewD_RT	202
7.12.2.10	calNewD_RTP	202
7.12.2.11	calNewDenave_None	203
7.12.2.12	calNewDenave_R	203
7.12.2.13	calNewDenave_RT	203
7.12.2.14	calNewDenave_RTP	203
7.12.2.15	calNewE_R_AD	204
7.12.2.16	calNewE_R_NA	204
7.12.2.17	calNewE_R_NA_LES	205
7.12.2.18	calNewE_RT_AD	205

7.12.2.19 calNewE_RT_NA	206
7.12.2.20 calNewE_RT_NA_LES	207
7.12.2.21 calNewE_RTP_AD	207
7.12.2.22 calNewE_RTP_NA	208
7.12.2.23 calNewE_RTP_NA_LES	208
7.12.2.24 calNewEddyVisc_None	209
7.12.2.25 calNewEddyVisc_R_CN	209
7.12.2.26 calNewEddyVisc_R_SM	210
7.12.2.27 calNewEddyVisc_RT_CN	210
7.12.2.28 calNewEddyVisc_RT_SM	210
7.12.2.29 calNewEddyVisc_RTP_CN	211
7.12.2.30 calNewEddyVisc_RTP_SM	211
7.12.2.31 calNewP_GL	211
7.12.2.32 calNewPEKappaGamma_TEOS	212
7.12.2.33 calNewQ0_R_GL	212
7.12.2.34 calNewQ0_R_TEOS	212
7.12.2.35 calNewQ0Q1_RT_GL	213
7.12.2.36 calNewQ0Q1_RT_TEOS	213
7.12.2.37 calNewQ0Q1Q2_RTP_GL	214
7.12.2.38 calNewQ0Q1Q2_RTP_TEOS	214
7.12.2.39 calNewR	214
7.12.2.40 calNewTPKappaGamma_TEOS	215
7.12.2.41 calNewU0_R	215
7.12.2.42 calNewU0_RT	216
7.12.2.43 calNewU0_RTP	216
7.12.2.44 calNewU_R	217
7.12.2.45 calNewU_R_LES	217
7.12.2.46 calNewU_RT	218
7.12.2.47 calNewU_RT_LES	219
7.12.2.48 calNewU_RTP	221
7.12.2.49 calNewU_RTP_LES	222
7.12.2.50 calNewV_RT	223
7.12.2.51 calNewV_RT_LES	223
7.12.2.52 calNewV_RTP	224
7.12.2.53 calNewV_RTP_LES	224
7.12.2.54 calNewVelocities_R	225

7.12.2.55 calNewVelocities_R_LES	225
7.12.2.56 calNewVelocities_RT	226
7.12.2.57 calNewVelocities_RT_LES	226
7.12.2.58 calNewVelocities_RTP	227
7.12.2.59 calNewVelocities_RTP_LES	227
7.12.2.60 calNewW_RTP	227
7.12.2.61 calNewW_RTP_LES	228
7.12.2.62 calOldDenave_None	229
7.12.2.63 calOldDenave_R	229
7.12.2.64 calOldDenave_RT	229
7.12.2.65 calOldDenave_RTP	229
7.12.2.66 calOldEddyVisc_R_CN	230
7.12.2.67 calOldEddyVisc_R_SM	230
7.12.2.68 calOldEddyVisc_RT_CN	230
7.12.2.69 calOldEddyVisc_RT_SM	231
7.12.2.70 calOldEddyVisc_RTP_CN	231
7.12.2.71 calOldEddyVisc_RTP_SM	231
7.12.2.72 calOldP_GL	232
7.12.2.73 calOldPEKappaGamma_TEOS	232
7.12.2.74 calOldQ0_R_GL	233
7.12.2.75 calOldQ0_R_TEOS	233
7.12.2.76 calOldQ0Q1_RT_GL	233
7.12.2.77 calOldQ0Q1_RT_TEOS	234
7.12.2.78 calOldQ0Q1Q2_RTP_GL	234
7.12.2.79 calOldQ0Q1Q2_RTP_TEOS	234
7.12.2.80 dEOS_GL	235
7.12.2.81 dImplicitEnergyFunction_None	235
7.12.2.82 dImplicitEnergyFunction_R	235
7.12.2.83 dImplicitEnergyFunction_R_LES	236
7.12.2.84 dImplicitEnergyFunction_R_LES_SB	236
7.12.2.85 dImplicitEnergyFunction_R_SB	237
7.12.2.86 dImplicitEnergyFunction_RT	238
7.12.2.87 dImplicitEnergyFunction_RT_LES	239
7.12.2.88 dImplicitEnergyFunction_RT_LES_SB	239
7.12.2.89 dImplicitEnergyFunction_RT_SB	240
7.12.2.90 dImplicitEnergyFunction_RTP	241

7.12.2.91	dImplicitEnergyFunction_RTP_LES	242
7.12.2.92	dImplicitEnergyFunction_RTP_LES_SB	242
7.12.2.93	dImplicitEnergyFunction_RTP_SB	243
7.12.2.94	implicitSolve_None	244
7.12.2.95	implicitSolve_R	244
7.12.2.96	implicitSolve_RT	245
7.12.2.97	implicitSolve_RTP	245
7.12.2.98	initDonorFracAndMaxConVel_R_GL	246
7.12.2.99	initDonorFracAndMaxConVel_R_TEOS	246
7.12.2.100	initDonorFracAndMaxConVel_RT_GL	246
7.12.2.101	initDonorFracAndMaxConVel_RT_TEOS	247
7.12.2.102	initDonorFracAndMaxConVel_RTP_GL	247
7.12.2.103	initDonorFracAndMaxConVel_RTP_TEOS	247
7.12.2.104	nitInternalVars	248
7.12.2.105	setInternalVarInf	248
7.12.2.106	setMainFunctions	249
7.13	physEquations_HEAD.cpp File Reference	251
7.13.1	Detailed Description	254
7.13.2	Function Documentation	254
7.13.2.1	calDelt_CONST	254
7.13.2.2	calDelt_R_GL	255
7.13.2.3	calDelt_R_TEOS	255
7.13.2.4	calDelt_RT_GL	255
7.13.2.5	calDelt_RT_TEOS	256
7.13.2.6	calDelt_RTP_GL	256
7.13.2.7	calDelt_RTP_TEOS	257
7.13.2.8	calNewD_R	257
7.13.2.9	calNewD_RT	258
7.13.2.10	calNewD_RTP	258
7.13.2.11	calNewDenave_None	259
7.13.2.12	calNewDenave_R	259
7.13.2.13	calNewDenave_RT	259
7.13.2.14	calNewDenave_RTP	259
7.13.2.15	calNewE_R_AD	260
7.13.2.16	calNewE_R_NA	260
7.13.2.17	calNewE_R_NA_LES	261

7.13.2.18 calNewE_RT_AD	261
7.13.2.19 calNewE_RT_NA	262
7.13.2.20 calNewE_RT_NA_LES	262
7.13.2.21 calNewE_RTP_AD	263
7.13.2.22 calNewE_RTP_NA	264
7.13.2.23 calNewE_RTP_NA_LES	264
7.13.2.24 calNewEddyVisc_None	265
7.13.2.25 calNewEddyVisc_R_CN	265
7.13.2.26 calNewEddyVisc_R_SM	265
7.13.2.27 calNewEddyVisc_RT_CN	266
7.13.2.28 calNewEddyVisc_RT_SM	266
7.13.2.29 calNewEddyVisc_RTP_CN	266
7.13.2.30 calNewEddyVisc_RTP_SM	267
7.13.2.31 calNewP_GL	267
7.13.2.32 calNewPEKappaGamma_TEOS	267
7.13.2.33 calNewQ0_R_GL	268
7.13.2.34 calNewQ0_R_TEOS	268
7.13.2.35 calNewQ0Q1_RT_GL	268
7.13.2.36 calNewQ0Q1_RT_TEOS	269
7.13.2.37 calNewQ0Q1Q2_RTP_GL	269
7.13.2.38 calNewQ0Q1Q2_RTP_TEOS	269
7.13.2.39 calNewR	270
7.13.2.40 calNewTPKappaGamma_TEOS	270
7.13.2.41 calNewU0_R	270
7.13.2.42 calNewU0_RT	271
7.13.2.43 calNewU0_RTP	271
7.13.2.44 calNewU_R	272
7.13.2.45 calNewU_R_LES	273
7.13.2.46 calNewU_RT	273
7.13.2.47 calNewU_RT_LES	274
7.13.2.48 calNewU_RTP	276
7.13.2.49 calNewU_RTP_LES	277
7.13.2.50 calNewV_RT	278
7.13.2.51 calNewV_RT_LES	278
7.13.2.52 calNewV_RTP	279
7.13.2.53 calNewV_RTP_LES	279

7.13.2.54 calNewVelocities_R	280
7.13.2.55 calNewVelocities_R_LES	280
7.13.2.56 calNewVelocities_RT	281
7.13.2.57 calNewVelocities_RT_LES	281
7.13.2.58 calNewVelocities_RTP	281
7.13.2.59 calNewVelocities_RTP_LES	282
7.13.2.60 calNewW_RTP	282
7.13.2.61 calNewW_RTP_LES	282
7.13.2.62 calOldDenave_None	283
7.13.2.63 calOldDenave_R	283
7.13.2.64 calOldDenave_RT	283
7.13.2.65 calOldDenave_RTP	284
7.13.2.66 calOldEddyVisc_R_CN	284
7.13.2.67 calOldEddyVisc_R_SM	284
7.13.2.68 calOldEddyVisc_RT_CN	285
7.13.2.69 calOldEddyVisc_RT_SM	285
7.13.2.70 calOldEddyVisc_RTP_CN	285
7.13.2.71 calOldEddyVisc_RTP_SM	286
7.13.2.72 calOldP_GL	286
7.13.2.73 calOldPEKappaGamma_TEOS	286
7.13.2.74 calOldQ0_R_GL	287
7.13.2.75 calOldQ0_R_TEOS	287
7.13.2.76 calOldQ0Q1_RT_GL	287
7.13.2.77 calOldQ0Q1_RT_TEOS	288
7.13.2.78 calOldQ0Q1Q2_RTP_GL	288
7.13.2.79 calOldQ0Q1Q2_RTP_TEOS	288
7.13.2.80 dEOS_GL	289
7.13.2.81 dImplicitEnergyFunction_None	289
7.13.2.82 dImplicitEnergyFunction_R	289
7.13.2.83 dImplicitEnergyFunction_R_LES	290
7.13.2.84 dImplicitEnergyFunction_R_LES_SB	290
7.13.2.85 dImplicitEnergyFunction_R_SB	291
7.13.2.86 dImplicitEnergyFunction_RT	292
7.13.2.87 dImplicitEnergyFunction_RT_LES	293
7.13.2.88 dImplicitEnergyFunction_RT_LES_SB	293
7.13.2.89 dImplicitEnergyFunction_RT_SB	294

7.13.2.90	dImplicitEnergyFunction_RTP	295
7.13.2.91	dImplicitEnergyFunction_RTP_LES	295
7.13.2.92	dImplicitEnergyFunction_RTP_LES_SB	296
7.13.2.93	dImplicitEnergyFunction_RTP_SB	297
7.13.2.94	implicitSolve_None	298
7.13.2.95	implicitSolve_R	298
7.13.2.96	implicitSolve_RT	299
7.13.2.97	implicitSolve_RTP	299
7.13.2.98	initDonorFracAndMaxConVel_R_GL	299
7.13.2.99	initDonorFracAndMaxConVel_R_TEOS	300
7.13.2.100	initDonorFracAndMaxConVel_RT_GL	300
7.13.2.101	initDonorFracAndMaxConVel_RT_TEOS	300
7.13.2.102	initDonorFracAndMaxConVel_RTP_GL	300
7.13.2.103	initDonorFracAndMaxConVel_RTP_TEOS	301
7.13.2.104	nitInternalVars	301
7.13.2.105	setInternalVarInf	302
7.13.2.106	setMainFunctions	302
7.14	physEquations_new.cpp File Reference	304
7.14.1	Detailed Description	307
7.14.2	Function Documentation	307
7.14.2.1	calDelt_CONST	307
7.14.2.2	calDelt_R_GL	308
7.14.2.3	calDelt_R_TEOS	308
7.14.2.4	calDelt_RT_GL	308
7.14.2.5	calDelt_RT_TEOS	309
7.14.2.6	calDelt_RTP_GL	309
7.14.2.7	calDelt_RTP_TEOS	310
7.14.2.8	calNewD_R	310
7.14.2.9	calNewD_RT	311
7.14.2.10	calNewD_RTP	311
7.14.2.11	calNewDenave_None	312
7.14.2.12	calNewDenave_R	312
7.14.2.13	calNewDenave_RT	312
7.14.2.14	calNewDenave_RTP	312
7.14.2.15	calNewE_R_AD	313
7.14.2.16	calNewE_R_NA	313

7.14.2.17 calNewE_R_NA_LES	314
7.14.2.18 calNewE_RT_AD	314
7.14.2.19 calNewE_RT_NA	315
7.14.2.20 calNewE_RT_NA_LES	315
7.14.2.21 calNewE_RTP_AD	316
7.14.2.22 calNewE_RTP_NA	317
7.14.2.23 calNewE_RTP_NA_LES	317
7.14.2.24 calNewEddyVisc_None	318
7.14.2.25 calNewEddyVisc_R_CN	318
7.14.2.26 calNewEddyVisc_R_SM	318
7.14.2.27 calNewEddyVisc_RT_CN	319
7.14.2.28 calNewEddyVisc_RT_SM	319
7.14.2.29 calNewEddyVisc_RTP_CN	319
7.14.2.30 calNewEddyVisc_RTP_SM	320
7.14.2.31 calNewP_GL	320
7.14.2.32 calNewPEKappaGamma_TEOS	320
7.14.2.33 calNewQ0_R_GL	321
7.14.2.34 calNewQ0_R_TEOS	321
7.14.2.35 calNewQ0Q1_RT_GL	321
7.14.2.36 calNewQ0Q1_RT_TEOS	322
7.14.2.37 calNewQ0Q1Q2_RTP_GL	322
7.14.2.38 calNewQ0Q1Q2_RTP_TEOS	322
7.14.2.39 calNewR	323
7.14.2.40 calNewTPKappaGamma_TEOS	323
7.14.2.41 calNewU0_R	323
7.14.2.42 calNewU0_RT	324
7.14.2.43 calNewU0_RTP	324
7.14.2.44 calNewU_R	325
7.14.2.45 calNewU_R_LES	326
7.14.2.46 calNewU_RT	326
7.14.2.47 calNewU_RT_LES	327
7.14.2.48 calNewU_RTP	329
7.14.2.49 calNewU_RTP_LES	330
7.14.2.50 calNewV_RT	331
7.14.2.51 calNewV_RT_LES	331
7.14.2.52 calNewV_RTP	332

7.14.2.53 calNewV_RTP_LES	332
7.14.2.54 calNewVelocities_R	333
7.14.2.55 calNewVelocities_R_LES	333
7.14.2.56 calNewVelocities_RT	334
7.14.2.57 calNewVelocities_RT_LES	334
7.14.2.58 calNewVelocities_RTP	334
7.14.2.59 calNewVelocities_RTP_LES	335
7.14.2.60 calNewW_RTP	335
7.14.2.61 calNewW_RTP_LES	335
7.14.2.62 calOldDenave_None	336
7.14.2.63 calOldDenave_R	336
7.14.2.64 calOldDenave_RT	336
7.14.2.65 calOldDenave_RTP	337
7.14.2.66 calOldEddyVisc_R_CN	337
7.14.2.67 calOldEddyVisc_R_SM	337
7.14.2.68 calOldEddyVisc_RT_CN	338
7.14.2.69 calOldEddyVisc_RT_SM	338
7.14.2.70 calOldEddyVisc_RTP_CN	338
7.14.2.71 calOldEddyVisc_RTP_SM	339
7.14.2.72 calOldP_GL	339
7.14.2.73 calOldPEKappaGamma_TEOS	339
7.14.2.74 calOldQ0_R_GL	340
7.14.2.75 calOldQ0_R_TEOS	340
7.14.2.76 calOldQ0Q1_RT_GL	340
7.14.2.77 calOldQ0Q1_RT_TEOS	341
7.14.2.78 calOldQ0Q1Q2_RTP_GL	341
7.14.2.79 calOldQ0Q1Q2_RTP_TEOS	342
7.14.2.80 dEOS_GL	342
7.14.2.81 dImplicitEnergyFunction_None	342
7.14.2.82 dImplicitEnergyFunction_R	342
7.14.2.83 dImplicitEnergyFunction_R_LES	343
7.14.2.84 dImplicitEnergyFunction_R_LES_SB	343
7.14.2.85 dImplicitEnergyFunction_R_SB	344
7.14.2.86 dImplicitEnergyFunction_RT	345
7.14.2.87 dImplicitEnergyFunction_RT_LES	346
7.14.2.88 dImplicitEnergyFunction_RT_LES_SB	346

7.14.2.89 dImplicitEnergyFunction_RT_SB	347
7.14.2.90 dImplicitEnergyFunction_RTP	348
7.14.2.91 dImplicitEnergyFunction_RTP_LES	348
7.14.2.92 dImplicitEnergyFunction_RTP_LES_SB	349
7.14.2.93 dImplicitEnergyFunction_RTP_SB	350
7.14.2.94 implicitSolve_None	351
7.14.2.95 implicitSolve_R	351
7.14.2.96 implicitSolve_RT	352
7.14.2.97 implicitSolve_RTP	352
7.14.2.98 initDonorFracAndMaxConVel_R_GL	352
7.14.2.99 initDonorFracAndMaxConVel_R_TEOS	353
7.14.2.100 initDonorFracAndMaxConVel_RT_GL	353
7.14.2.101 initDonorFracAndMaxConVel_RT_TEOS	353
7.14.2.102 nitDonorFracAndMaxConVel_RTP_GL	353
7.14.2.103 nitDonorFracAndMaxConVel_RTP_TEOS	354
7.14.2.104 nitInternalVars	354
7.14.2.105 setInternalVarInf	355
7.14.2.106 setMainFunctions	355
7.15 userguide.h File Reference	357
7.15.1 Detailed Description	357
7.16 watchzone.cpp File Reference	358
7.16.1 Detailed Description	358
7.17 watchzone.h File Reference	359
7.17.1 Detailed Description	359

Chapter 1

Using and Modifying SPHERLS

This manual is divided into two main parts the current chapter, and the rest of the chapters. All chapters other than the current, contain specific reference material for the SPHERLS code while the current chapter contains a more descriptive how-to approach explaining the usage and modification of SPHERLS. The chapters following chapter 1 will serve as a usefull reference when specific details need to be found, for example a discription of a particular variable in the code. The current chapter on the other hand is the best place to go to get a quick understanding of SPHERLS that will enable you to use it.

1.1 Overview

SPHERLS stands for Stellar Pulsation with a Horizontal Eulearian Radial Lagrangian Scheme. There are three components to SPHERLS: SPHERLS itself which does the hydodynamics calculations, SPHERLSgen which creates starting models, and SPHERLSanal which is able to manipulate the output files. Both SPHERLSgen and SPHERLSanal have there own manuals which can be consulted for their specific uses and installations.

1.1.1 The Basics

SPHERLS calculates the radial pulsation motions together with the horizontal convective flow. The radial pulsation can be described by a radial grid velocity `Grid::nU0`, moving the grid inward and outward with the pulsation. The movement of the grid is defined by the motion required to maintaining the mass in a spherical shell through out the calculation. This motion is determined so that it will change the volume of the shell so the newly calculated density when multiplied with the new volume will produce the same shell mass. The total motion of the stellar material is simply the combination of the three velocity components, radial `Grid::nU`, theta `Grid::nV`, and phi velocities `Grid::nW`. The convective motion is the radial velocity minus the grid velocity, combined with the theta and phi velocities. This is because the grid velocity describes the bulk motion of the pulsation so subtracting it out leaves only the convective motions.

SPHERLS solves the normal hydodynamic equations of, mass, momentum, and energy conservation. The form of the mass equation, momentum conservation, and energy conservation are:

$$\frac{dM}{dt} + \oint_{\mathbb{S}} (\rho \vec{v}) \cdot \hat{n} d\sigma = 0$$

$$\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla) \vec{v} = -\frac{1}{\rho} \nabla P + \nabla \cdot \boldsymbol{\tau} - \nabla \phi$$

$$\frac{\partial E}{\partial t} + (\vec{v} \cdot \nabla) E + P \frac{d\mathbb{V}}{dt} = \epsilon + \frac{1}{\rho} [-\nabla \cdot F + \nabla \cdot (\boldsymbol{\tau} \cdot \vec{v}) - (\nabla \cdot \boldsymbol{\tau}) \cdot \vec{v}]$$

where $\boldsymbol{\tau}$ is the stress tensor for zero bulk viscosity, E is the specific internal energy, \mathbb{V} is the specific volume, and F is the radiative flux. In addition to these conservation equations an equation of state is needed, in this case the OPAL equation of state and opacities, and the Alexander opacities at low temperatures are used. The equation of state tables are functions of density and temperature, and produce the energy, pressure, opacity, and adiabatic index of the gas for a given temperature and density. In adiabatic calculations, it is also possible to use a γ -law gas equation of state but in that case an energy profile must also be included.

The simulation grid is broken up into two main sections, the 1D region towards the center of the star, the multi-dimensional region towards the surface. The inner part of the multi-dimensional region solves all the conservation equations explicitly, in that the new values for the conserved quantities are directly calculated from the information in the previous time step. In the outer parts of the multi-dimensional region the energy conservation equation is calculated semi-implicitly, which means that the new values are dependent on the new values averaged with the old values to correctly time center the equation. This semi-implicit energy conservation equation can be perturbed and linearized producing a set of linear equations the size of the region being solved implicitly. The solution of these linear equations provide corrections for the temperature which can be applied and then resolved in an iterative approach until the value of the new temperature converges. The equation of state is a function of temperature and not energy which is why the temperature is perturbed and not the energy. This set of equations for the temperature corrections are solved using the PETSC library.

- Different ways in which SPHERLS can be used, 1D,2D,3D, Adiabatic,Non-adiabatic, implicit, debugging options/test

1.2 The Equations

I will want to give a detailed description of the equations used (probably copied from my notes wiki) so that the reader can easily see a 1-1 correspondence between the equation and the terms in SPHERLS.

1.3 Program Flow

- Describe the grids
- The order of calculation
- When parts of the grid are updated

1.4 Installing SPHERLS

Todo

This should be updated to reflect the use of the GNU build system.

Once the correct libraries are installed, and their paths added to your `LD_LIBRARY_PATH` environment variable, it should just require typing `make` in the correct directories. SPHERLS is broken up into 3 main codes. SPHERLS it self, which is the main hydrodynamics code which integrates the initial static model, SPHERLSgen which creates the static model, and SPHERLSanal which is used for processing the output of SPHERLS and SPHERLSgen.

To Add

- example `.bashrc` entries, showing `LD_LIBRARY_PATH` additions, and other SPHERLS related configuration options
- also the make files will need to know where the paths for the libraries are, either describe how the user can do this, or automate it some how.

A few words on installation before we get into the details of the specific packages. In order for the SPHERLS configuration script (needed for installing SPHERLS) to find the required libraries and include files they have to be installed in at least one of the directories that it looks for them. The configuration script looks for the libraries and include files it requires in the following "standard" locations: `/lib`, `/include`, `/usr/lib` `/usr/include`, `/usr/local/lib`, `/usr/local/include`, `/home/$USER/lib`, and `/home/$USER/include` . If you install the required libraries in places other than these "standard" locations you will have to manually tell the SPHERLS configuration script where to find them. Running `configure -h` will list the available options to tell the script where to find these include files and libraries.

I am going to assume that the user is installing on a linux system, more over I will be assuming that the linux distribution follows a debian like directory structure (many distributions are based on debian). The install instructions below assume you do not have root access and must do an install to your home directory (a per-user install). The standard install location for per-user level binaries, libraries, include files and documentation (at least far as SPHERLS is concerned) is in `~/bin`, `~/lib`, `~/include`, and `~/share` directories respectively. Be aware that if you have these directories in your home directory already and are not using them as a standard place to install per-user packages you will likely want to rename your pre-existing directories or a bunch of additional files will be added to them from the installations of various packages below. Alternatively, you can install the libraries and binaries to any directory on your machine just by changing `-prefix=/home/$USER/`, mentioned below, to point to where you want to install it.

If you have root access and want to install for all users of the current machine you will likely want to install the libraries into `/usr/local` which can be achieved by setting `-prefix=/usr/local` instead of `-prefix=/home/$USER/` used in the below installations and SPHERLS will automatically check this location for the install libraries.

1.4.1 Requirements

- gcc/g++
- openMPI
- PETSc library, used as the core matrix solver

1.4.2 Optional Requirements

- python for analysis scripts
- - numpy
- - matplotlib
- fftw3 library for analysis
- hdf4 library for converting to hdf4 file format
- Doxygen used to create documentation from source code via "make docs"

1.4.3 Installing PETSC Library

Version `petsc-lite-3.1-p8`, has been tested to work with SPHERLS. `petsc-lite-3.2-p7` is known to be incompatible, which as of this writing is the current version of the petsc library. At some point in the future support for the newer version of the library maybe added. The below commands will install PETSc into your home directory. ASIDE: I have also had difficulties installing PETSc on Fundy, and Placentia ACENet machines.

- Download PETSc library, from the PETSc [website](#).
- Then untar and unzip it with `tar -xzf petsc-lite-3.1-p8.tar`
- To install the library change into the directory made when you extracted the archive and type the following commands:
 1. `./configure PETSC_DIR=$PWD --prefix=/home/$USER/ --with-c++-support --with-c-support --with-shared --download-f-blas-lapack=1 --with-x11=no --with-x11=no`
`$USER` is the environment variable corresponding to your username.
 2. `make all`
 3. `make install`
 4. `make PETSC_DIR=/home/$USER/lib test`
- You will then need to add the following line to you `.bashrc` file to assure that you will pick up the library

```
export PETSC_DIR=/home/$USER/lib
```

1.4.4 Installing FFTW Library

- Download the FFTW Library from the FFTS [website](#). Version `fftw-3.2.2` has been tested to work with SPHERLS.
- The downloaded FFTW file (e.g. `fftw-3.2.2.tar.gz`) will need to be unzipped to do so type `gunzip fftw-3.2.2.tar.gz`
- Then untar it with `tar -xf fftw-3.2.2.tar.gz`

- To install the library change into the directory made when you extracted the archive and type the following commands:

1. `./configure --prefix=<path-to-final-location-of-library>`
2. `make`
3. `make install`

#

1.4.5 Installing HDF4 Library

#

1.4.6 Installing Doxygen

#

1.4.7 Installing Python

#

1.4.8 Installing SPHERLS

1.5 Using SPHERLS

- Generating a starting model (see SPHERLSgen documentation for details)
- The XML configuration file
- Starting a calculation and the "makeFile"
- getting data
 - watchzones
 - model dumps
- post calculation analysis (see SPHERLSanal documentation for details)
- Adiabatic Calculations
 - 1D, 2D, and 3D
 - γ -law gas
 - Sedov Blast wave test
- Non-Adiabatic Calculations
 - 1D, 2D, and 3D
 - Tabulate EOS
 - Different versions of the energy equation
 - LES models

1.6 Modifing or Developing SPHERLS

- Basic layout/design of the code
 - model output
 - data monitoring
 - * watch zones
 - * peak KE tracking
 - internal/versus external variables
 - message passing
 - grid layout
 - ranges of grids
 - boundary regions
 - grid updating
- How to document SPHERLS
- Premade test for SPHERLS after modification
 - reference calculations
 - restart test
 - calculation test (if not modifying calcluation part of SPHERLS)
- How to modify SPHERLS
 - Common changes
 - * How to add a new internal variable
 1. **Add to the internal variable count:** Decide in what cases the variable will be needed, 1D calculations, 2D calculations, when there is a gamma law gas or a tabulated equation of state, adiabatic or non-adiabatic etc. Then once decided it can be added to the total number of internal variables `Grid::nNumInt Vars` by increasing the value by one in the function `modelRead` in the section below the comment "set number of internal variables ..." under the appropriate if block. If the specific if block for the situation you need isn't there, you can create your own, and add it there.
 2. **Create a new variable ID:** In the `grid::h` file under the `Grid` class are variable ID's. These ID's simply indicate the location of the variable in the array. One must add a new ID for the new variable as an integer. The value of the ID is set in the function `modelRead` in the same section as the number of internal variables. The value used should be the last integer after the last pre-existing variable ID. This should also be `Grid::nNumVars + Grid::nNumInt Vars -1`. The ID should also be initialized to -1, so that the code knows when it isn't being used. This is done in the grid class constructor, `Grid::Grid`. Simply add a line in the constructor setting your new ID = -1.
 3. **Set variable infos:** Decide what the dimensions of the new variable will be. It can be cell centered it can be or interface centered, it can also be only 1D, 2D, or 3D. Of course it will be only 1D if the entire calculation is 1D, or 2D if the calculation is 2D, but if the calculation is 3D it could also only be 2D, or 1D, and if 2D it could be only 1D. Also decide if the variable will change with time, dependent variables are only initialized and not updated during the calculations. This information is given to SPHERLS in

the `setInternalVarInf` function in the `physEquations.cpp` file. The variable that is set is `Grid::nVariables`. It is a 2D array, the first index corresponds to the particular variable in question, the ID you made in the previous step can be used as the first index of this array. The second index refers to the three directions (0-2) and the time (3). If the variable is centered in the grid in direction 0 (r-direction) then this array element should have a value of 0. If the variable is interface centered in the grid in direction 0, then this array element should have a value of 1. If it isn't defined in direction 0, for example the theta independent variable isn't defined in the 0 direction then it should be -1. This is the same for the other 2 directions. The last element (3) should be either 0 not updated every time step, or 1 if updated every timestep.

4. **Add functions:** Finally to do anything useful with your new internal variable functions must be added to initialize the values of the variables, and to update them with time if needed. Initialization functions are called within the `initInternalVars` function in the `physEquations.cpp` file. The details of these functions will depend on what the individual variables are intended for. **Functions** to be called every timestep must be called from the main program loop in the file `main.cpp` in the appropriate order.

- * How to add a new external variable
- * How to add a new physics functions
 - Function naming conventions
 - `Grid` variables
 - indices and their ranges
- SPHERLS debugging tips

1.7 Message Passing

- Explain message passing in SPHERLS

Chapter 2

Todo List

page Using and Modifying SPHERLS This should be updated to reflect the use of the GNU build system.

Member `initImplicitCalculation` `isFrom`, `isTo`, `matCoeff`, `vecTCorrections`, `vecTCorrections`, `vecRHS`, `vecTCorrectionsLocal`, `kspContext`, `vecscatTCorrections` all need to be destroyed before program finishes.

Member `modelRead` At some point should get it working with only 1 processor

Member `updateLocalBoundaries` Shouldn't need `MPI::COMM_WORLD.Barrier()` may want to test out removing this at some point as it might produce a bit of a speed up.

Member `updateLocalBoundariesNewGrid` May want to do some waiting on this message at some point before the end of the timestep, but it doesn't need to be done in this function. It might also be that this is built into the code by waiting at some other point. This is something that should be checked out at somepoint, perhaps once the preformance starts to be analyzed. I would think that if the send buffer was being modified before the send was completed, that there would be some errors popping up that would likely kill the program.

Member `calNewU0_R` At some point I will likely want to make this funciton compatiabile with a 3D domain decomposition instead of a purely radial domain decomposition.

Member `calNewU0_RT` At some point I will likely want to make this funciton compatiabile with a 3D domain decomposition instead of a purely radial domain decomposition.

Member `calNewU0_RTP` At some point I will likely want to make this funciton compatiabile with a 3D domain decomposition instead of a purely radial domain decomposition.

Member `calOldQ0_R_TEOS` should use new P and rho

Member `calNewU0_R` At some point I will likely want to make this funciton compatiabile with a 3D domain decomposition instead of a purely radial domain decomposition.

Member `calNewU0_RT` At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Member `calNewU0_RTP` At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Member `calNewU0_R` At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Member `calNewU0_RT` At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Member `calNewU0_RTP` At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Chapter 3

Boundary Conditions

Member `calNewD_R` doesn't allow mass flux through outter interface

Member `calNewD_RT` doesn't allow mass flux through outter interface

Member `calNewD_RTP` doesn't allow mass flux through outter interface

Member `calNewE_R_NA` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Member `calNewE_R_NA` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Member `calNewE_R_NA` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `calNewE_R_NA_LES` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Member `calNewE_R_NA_LES` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Member `calNewE_R_NA_LES` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `calNewE_RT_AD` `grid.dLocalGridOld[grid.nE][i+1][j][k]` is missing

Member `calNewE_RT_AD` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing using inner gradient for both

Member `calNewE_RT_NA` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Member `calNewE_RT_NA` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Member `calNewE_RT_NA` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member `calNewE_RT_NA_LES` Missing ΔM_r outside model using `Parameters::dAlpha` times ΔM_r in the last zone instead.

Member `calNewE_RT_NA_LES` Setting energy at surface equal to energy in last zone.

Member `calNewE_RT_NA_LES` missing eddy viscosity outside the model setting it to zero

Member `calNewE_RT_NA_LES` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member `calNewE_RT_NA_LES` missing energy outside the model, assuming it is the same as that in the last zone. That causes this term to be zero.

Member `calNewE_RTP_AD` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to zero.

Member `calNewE_RTP_AD` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient.

Member `calNewE_RTP_NA` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at i.

Member `calNewE_RTP_NA` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Member `calNewE_RTP_NA` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `calNewE_RTP_NA_LES` Missing ΔM_r outside model using `Parameters::dAlpha` times ΔM_r in the last zone instead.

Member `calNewE_RTP_NA_LES` Missing `W` at `i+1`, assuming the same as at `i`

Member `calNewE_RTP_NA_LES` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at `i`.

Member `calNewE_RTP_NA_LES` missing density outside model, setting it to zero

Member `calNewE_RTP_NA_LES` missing eddy viscosity outside the model setting it to zero

Member `calNewE_RTP_NA_LES` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Member `calNewE_RTP_NA_LES` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `calNewEddyVisc_RTP_SM` assuming that theta velocity is constant across surface

Member `calNewEddyVisc_RTP_SM` assume phi velocity is constant across surface

Member `calNewU0_RT` `grid.dLocalGridOld[grid.nD][i+1][j][k]` is missing

Member `calNewU0_RTP` `grid.dLocalGridOld[grid.nD][i+1][j][k]` is missing

Member `calNewU_R` Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2}$, setting it to 0.0

Member `calNewU_R` Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to $-1.0 * \text{grid.dLocalGridOld[grid.nP][nICen][j][k]}$.

Member `calNewU_R` Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Member `calNewU_R` Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewU_R_LES` Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2}$, setting it to 0.0

Member `calNewU_R_LES` missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` using velocity at i

Member `calNewU_R_LES` Assuming eddy viscosity outside model is zero.

Member `calNewU_R_LES` Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to $-1.0 * \text{grid.dLocalGridOld[grid.nP][nICen][j][k]}$.

Member `calNewU_R_LES` Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Member `calNewU_R_LES` Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

Member `calNewU_RT` assuming theta velocity is constant across surface

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0]` in calculation of $\langle \rho \rangle_{i+1/2}$, setting it to zero.

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to `-1.0*dP_ijk_n`.

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Member `calNewU_RT_LES` Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Member `calNewU_RT_LES` Missing density outside of surface, setting it to zero.

Member `calNewU_RT_LES` Missing density outside model, setting it to zero.

Member `calNewU_RT_LES` assuming theta and phi velocity same outside star as inside.

Member `calNewU_RT_LES` Assuming theta velocities are constant across surface.

Member `calNewU_RT_LES` assuming that V at $i+1$ is equal to v at i .

Member `calNewU_RT_LES` Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

Member `calNewU_RT_LES` assume viscosity is zero outside the star.

Member `calNewU_RT_LES` Missing mass outside model, setting it to zero.

Member `calNewU_RT_LES` Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewU_RTP` missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` in calculation of $u_{i+1,j,k}$ setting $u_{i+1,j,k} = u_{i+1/2,j,k}$.

Member `calNewU_RTP` Missing `grid.dLocalGridOld[grid.nD][i+1][j][k]` in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

Member `calNewU_RTP` assuming theta velocity is constant across the surface.

Member `calNewU_RTP` assuming phi velocity is constant across the surface.

Member `calNewU_RTP` Missing `grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0]` in calculation of $\langle \rho \rangle_{i+1/2}$ setting it to zero.

Member `calNewU_RTP` Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it equal to `Parameters::dAlpha` `grid.dLocalGridOld[grid.nDM][nICen][0][0]`.

Member `calNewU_RTP` Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewU_RTP` Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Member `calNewU_RTP_LES` Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Member `calNewU_RTP_LES` Missing density outside of surface, setting it to zero.

Member `calNewU_RTP_LES` Missing density outside model, setting it to zero.

Member `calNewU_RTP_LES` assuming theta and phi velocity same outside star as inside.

Member `calNewU_RTP_LES` Assuming theta velocities are constant across surface.

Member `calNewU_RTP_LES` assuming that V at $i+1$ is equal to v at i .

Member `calNewU_RTP_LES` Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

Member `calNewU_RTP_LES` assume viscosity is zero outside the star.

Member `calNewU_RTP_LES` Missing mass outside model, setting it to zero.

Member `calNewU_RTP_LES` Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewV_RT` `grid.dLocalGridOld[grid.nV][i+1][j+1][k]` is missing

Member `calNewV_RT` missing upwind gradient, using centred gradient instead

Member `calNewV_RT_LES` Assuming density outside star is zero

Member `calNewV_RT_LES` Assuming theta velocity is constant across surface.

Member `calNewV_RT_LES` Assuming eddy viscosity is zero at surface.

Member `calNewV_RTP` Assuming theta and phi velocities are the same at the surface of the star as just inside the star.

Member `calNewV_RTP` using centered gradient for upwind gradient outside star at surface.

Member `calNewV_RTP_LES` Assuming density outside star is zero

Member `calNewV_RTP_LES` Assuming theta velocity is constant across surface.

Member `calNewV_RTP_LES` Assuming eddy viscosity is zero at surface.

Member `calNewW_RTP` missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` assuming that the phi velocity at the outter most interface is the same as the phi velocity in the center of the zone.

Member `calNewW_RTP` missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` in outter most zone. This is needed to calculate the upwind gradient for donnor cell. The centered gradient is used instead when moving in the negative direction.

Member `calNewW_RTP_LES` assume theta and phi velocities are constant across surface

Member `calNewW_RTP_LES` assume eddy viscosity is zero at surface

Member `calNewW_RTP_LES` assume upwind gradient is the same as centered gradient across surface

Member `calOldEddyVisc_RTP_SM` assuming that theta velocity is constant across surface

Member `calOldEddyVisc_RTP_SM` assume phi velocity is constant across surface

Member `dImplicitEnergyFunction_R_LES_SB` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Member `dImplicitEnergyFunction_R_LES_SB` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Member `dImplicitEnergyFunction_R_LES_SB` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `dImplicitEnergyFunction_R_SB` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Member **dImplicitEnergyFunction_R_SB** `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Member **dImplicitEnergyFunction_R_SB** Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member **dImplicitEnergyFunction_RT_LES_SB** Missing ΔM_r outside model using `Parameters::dAlpha` times ΔM_r in the last zone instead.

Member **dImplicitEnergyFunction_RT_LES_SB** missing density outside model assuming it is zero

Member **dImplicitEnergyFunction_RT_LES_SB** missing density outside model assuming it is zero

Member **dImplicitEnergyFunction_RT_LES_SB** assuming V at `ip1half` is the same as V at `i`

Member **dImplicitEnergyFunction_RT_LES_SB** Assuming energy outside model is the same as the energy in the last zone inside the model.

Member **dImplicitEnergyFunction_RT_LES_SB** Using centered gradient for upwind gradient when motion is into the star at the surface

Member **dImplicitEnergyFunction_RT_LES_SB** Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member **dImplicitEnergyFunction_RT_SB** Using centered gradient for upwind gradient when motion is into the star at the surface

Member `dImplicitEnergyFunction_RT_SB` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member `dImplicitEnergyFunction_RTP_LES_SB` Missing ΔM_r outside model using `Parameters::dAlpha` times ΔM_r in the last zone instead.

Member `dImplicitEnergyFunction_RTP_LES_SB` missing density outside model assuming it is zero

Member `dImplicitEnergyFunction_RTP_LES_SB` missing density outside model assuming it is zero

Member `dImplicitEnergyFunction_RTP_LES_SB` assuming V at `ip1half` is the same as V at `i`

Member `dImplicitEnergyFunction_RTP_LES_SB` assuming W at `ip1half` is the same as W at `i`

Member `dImplicitEnergyFunction_RTP_LES_SB` Assuming energy outside model is the same as the energy in the last zone inside the model.

Member `dImplicitEnergyFunction_RTP_LES_SB` Using centered gradient for upwind gradient when motion is into the star at the surface

Member `dImplicitEnergyFunction_RTP_LES_SB` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member `dImplicitEnergyFunction_RTP_SB` Using $E_{\{i,j,k\}^{n+1/2}}$ for $E_{\{i+1/2,j,k\}^{n+1/2}}$

Member `dImplicitEnergyFunction_RTP_SB` Using centered gradient for upwind gradient when motion is into the star at the surface

Member `dImplicitEnergyFunction_RTP_SB` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member `calNewD_R` doesn't allow mass flux through outter interface

Member `calNewD_RT` doesn't allow mass flux through outter interface

Member `calNewD_RTP` doesn't allow mass flux through outter interface

Member `calNewE_R_NA` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Member `calNewE_R_NA` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Member `calNewE_R_NA` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `calNewE_R_NA_LES` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Member `calNewE_R_NA_LES` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Member `calNewE_R_NA_LES` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `calNewE_RT_AD` `grid.dLocalGridOld[grid.nE][i+1][j][k]` is missing

Member `calNewE_RT_AD` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing using inner gradient for both

Member `calNewE_RT_NA` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Member `calNewE_RT_NA` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Member `calNewE_RT_NA` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member `calNewE_RT_NA_LES` Setting energy at surface equal to energy in last zone.

Member `calNewE_RT_NA_LES` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member `calNewE_RTP_AD` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to zero.

Member `calNewE_RTP_AD` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in dA1. Using the centered gradient.

Member `calNewE_RTP_NA` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at i.

Member `calNewE_RTP_NA` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Member **calNewE_RTP_NA** Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member **calNewE_RTP_NA_LES** Missing `W` at `i+1`, assuming the same as at `i`

Member **calNewE_RTP_NA_LES** Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at `i`.

Member **calNewE_RTP_NA_LES** `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Member **calNewE_RTP_NA_LES** Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member **calNewEddyVisc_RTP_SM** assuming that theta velocity is constant across surface

Member **calNewEddyVisc_RTP_SM** assume phi velocity is constant across surface

Member **calNewU0_RT** `grid.dLocalGridOld[grid.nD][i+1][j][k]` is missing

Member **calNewU0_RTP** `grid.dLocalGridOld[grid.nD][i+1][j][k]` is missing

Member **calNewU_R** Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2}$, setting it to 0.0

Member **calNewU_R** Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to `-1.0*grid.dLocalGridOld[grid.nP][nICen][j][k]`.

Member **calNewU_R** Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Member `calNewU_R` Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewU_R_LES` Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2}$, setting it to 0.0

Member `calNewU_R_LES` missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` using velocity at i

Member `calNewU_R_LES` Assuming eddy viscosity outside model is zero.

Member `calNewU_R_LES` Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to $-1.0 * \text{grid.dLocalGridOld[grid.nP][nICen][j][k]}$.

Member `calNewU_R_LES` Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Member `calNewU_R_LES` Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

Member `calNewU_RT` assuming theta velocity is constant across surface

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0]` in calculation of $\langle \rho \rangle_{i+1/2}$, setting it to zero.

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to `-1.0*dP_ijk_n`.

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Member `calNewU_RT_LES` Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Member `calNewU_RT_LES` Missing density outside of surface, setting it to zero.

Member `calNewU_RT_LES` Missing density outside model, setting it to zero.

Member `calNewU_RT_LES` assuming theta and phi velocity same outside star as inside.

Member `calNewU_RT_LES` Assuming theta velocities are constant across surface.

Member `calNewU_RT_LES` assuming that V at $i+1$ is equal to v at i .

Member `calNewU_RT_LES` Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

Member `calNewU_RT_LES` assume viscosity is zero outside the star.

Member **calNewU_RT_LES** Missing mass outside model, setting it to zero.

Member **calNewU_RT_LES** Missing $\text{grid.dLocalGridOld}[\text{grid.nU}][i+1][j][k]$ and $\text{grid.dLocalGridOld}[\text{grid.nDM}][\text{nICen}+1][0][0]$ in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member **calNewU_RTP** missing $\text{grid.dLocalGridOld}[\text{grid.nU}][i+1][j][k]$ in calculation of $u_{i+1,j,k}$ setting $u_{i+1,j,k} = u_{i+1/2,j,k}$.

Member **calNewU_RTP** Missing $\text{grid.dLocalGridOld}[\text{grid.nD}][i+1][j][k]$ in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

Member **calNewU_RTP** assuming theta velocity is constant across the surface.

Member **calNewU_RTP** assuming phi velocity is constant across the surface.

Member **calNewU_RTP** Missing $\text{grid.dLocalGridOld}[\text{grid.nDenAve}][\text{nICen}+1][0][0]$ in calculation of $\langle \rho \rangle_{i+1/2}$ setting it to zero.

Member **calNewU_RTP** Missing $\text{grid.dLocalGridOld}[\text{grid.nDM}][\text{nICen}+1][0][0]$ in calculation of centered A_1 gradient, setting it equal to **Parameters::dAlpha** $\text{grid.dLocalGridOld}[\text{grid.nDM}][\text{nICen}][0][0]$.

Member **calNewU_RTP** Missing $\text{grid.dLocalGridOld}[\text{grid.nU}][i+1][j][k]$ and $\text{grid.dLocalGridOld}[\text{grid.nDM}][\text{nICen}+1][0][0]$ in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member **calNewU_RTP** Missing $\text{grid.dLocalGridOld}[\text{grid.nDM}][i+1][0][0]$ in calculation of S_1 using **Parameters::dAlpha** $\text{grid.dLocalGridOld}[\text{grid.nDM}][\text{nICen}][0][0]$ instead.

Member `calNewU_RTP_LES` Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Member `calNewU_RTP_LES` Missing density outside of surface, setting it to zero.

Member `calNewU_RTP_LES` Missing density outside model, setting it to zero.

Member `calNewU_RTP_LES` assuming theta and phi velocity same outside star as inside.

Member `calNewU_RTP_LES` Assuming theta velocities are constant across surface.

Member `calNewU_RTP_LES` assuming that V at $i+1$ is equal to v at i .

Member `calNewU_RTP_LES` Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

Member `calNewU_RTP_LES` assume viscosity is zero outside the star.

Member `calNewU_RTP_LES` Missing mass outside model, setting it to zero.

Member `calNewU_RTP_LES` Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewV_RT` `grid.dLocalGridOld[grid.nV][i+1][j+1][k]` is missing

Member `calNewV_RT` missing upwind gradient, using centred gradient instead

Member `calNewV_RT_LES` Assuming density outside star is zero

Member `calNewV_RT_LES` Assuming theta velocity is constant across surface.

Member `calNewV_RT_LES` Assuming eddy viscosity is zero at surface.

Member `calNewV_RTP` Assuming theta and phi velocities are the same at the surface of the star as just inside the star.

Member `calNewV_RTP` using centered gradient for upwind gradient outside star at surface.

Member `calNewV_RTP_LES` Assuming density outside star is zero

Member `calNewV_RTP_LES` Assuming theta velocity is constant across surface.

Member `calNewV_RTP_LES` Assuming eddy viscosity is zero at surface.

Member `calNewW_RTP` missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` assuming that the phi velocity at the outer most interface is the same as the phi velocity in the center of the zone.

Member `calNewW_RTP` missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` in outer most zone. This is needed to calculate the upwind gradient for donor cell. The centered gradient is used instead when moving in the negative direction.

Member `calNewW_RTP_LES` assume theta and phi velocities are constant across surface

Member `calNewW_RTP_LES` assume eddy viscosity is zero at surface

Member `calNewW_RTP_LES` assume upwind gradient is the same as centered gradient across surface

Member `calOldEddyVisc_RTP_SM` assuming that theta velocity is constant across surface

Member `calOldEddyVisc_RTP_SM` assume phi velocity is constant across surface

Member `dImplicitEnergyFunction_R_LES_SB` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Member `dImplicitEnergyFunction_R_LES_SB` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Member `dImplicitEnergyFunction_R_LES_SB` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `dImplicitEnergyFunction_R_SB` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Member `dImplicitEnergyFunction_R_SB` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Member `dImplicitEnergyFunction_R_SB` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `dImplicitEnergyFunction_RT_LES_SB` Using centered gradient for upwind gradient when motion is into the star at the surface

Member `dImplicitEnergyFunction_RT_LES_SB` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member `dImplicitEnergyFunction_RT_SB` Using centered gradient for upwind gradient when motion is into the star at the surface

Member `dImplicitEnergyFunction_RT_SB` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member `dImplicitEnergyFunction_RTP_LES_SB` assuming V at $ip1half$ is the same as V at i

Member `dImplicitEnergyFunction_RTP_LES_SB` assuming W at $ip1half$ is the same as W at i

Member `dImplicitEnergyFunction_RTP_LES_SB` Using $E_{\{i,j,k\}^{n+1/2}}$ for $E_{\{i+1/2,j,k\}^{n+1/2}}$

Member `dImplicitEnergyFunction_RTP_LES_SB` Using centered gradient for upwind gradient when motion is into the star at the surface

Member `dImplicitEnergyFunction_RTP_LES_SB` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member `dImplicitEnergyFunction_RTP_SB` Using $E_{\{i,j,k\}^{n+1/2}}$ for $E_{\{i+1/2,j,k\}^{n+1/2}}$

Member `dImplicitEnergyFunction_RTP_SB` Using centered gradient for upwind gradient when motion is into the star at the surface

Member `dImplicitEnergyFunction_RTP_SB` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member `calNewD_R` doesn't allow mass flux through outter interface

Member `calNewD_RT` doesn't allow mass flux through outter interface

Member `calNewD_RTP` doesn't allow mass flux through outter interface

Member `calNewE_R_NA` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Member `calNewE_R_NA` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Member `calNewE_R_NA` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `calNewE_R_NA_LES` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Member `calNewE_R_NA_LES` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Member `calNewE_R_NA_LES` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `calNewE_RT_AD` `grid.dLocalGridOld[grid.nE][i+1][j][k]` is missing

Member `calNewE_RT_AD` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing using inner gradient for both

Member `calNewE_RT_NA` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Member `calNewE_RT_NA` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Member `calNewE_RT_NA` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member `calNewE_RT_NA_LES` Setting energy at surface equal to energy in last zone.

Member `calNewE_RT_NA_LES` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member `calNewE_RTP_AD` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to zero.

Member `calNewE_RTP_AD` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient.

Member `calNewE_RTP_NA` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at `i`.

Member `calNewE_RTP_NA` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Member `calNewE_RTP_NA` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `calNewE_RTP_NA_LES` Missing `W` at `i+1`, assuming the same as at `i`

Member `calNewE_RTP_NA_LES` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at i .

Member `calNewE_RTP_NA_LES` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Member `calNewE_RTP_NA_LES` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `calNewEddyVisc_RTP_SM` assuming that theta velocity is constant across surface

Member `calNewEddyVisc_RTP_SM` assume phi velocity is constant across surface

Member `calNewU0_RT` `grid.dLocalGridOld[grid.nD][i+1][j][k]` is missing

Member `calNewU0_RTP` `grid.dLocalGridOld[grid.nD][i+1][j][k]` is missing

Member `calNewU_R` Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2}$, setting it to 0.0

Member `calNewU_R` Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to $-1.0 * \text{grid.dLocalGridOld[grid.nP][nICen][j][k]}$.

Member `calNewU_R` Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Member `calNewU_R` Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewU_R_LES` Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2}$, setting it to 0.0

Member `calNewU_R_LES` missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` using velocity at i

Member `calNewU_R_LES` Assuming eddy viscosity outside model is zero.

Member `calNewU_R_LES` Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to $-1.0 * \text{grid.dLocalGridOld[grid.nP][nICen][j][k]}$.

Member `calNewU_R_LES` Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Member `calNewU_R_LES` Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

Member `calNewU_RT` assuming theta velocity is constant across surface

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0]` in calculation of $\langle \rho \rangle_{i+1/2}$, setting it to zero.

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to $-1.0 * \text{dP_ijk_n}$.

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewU_RT` Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Member `calNewU_RT_LES` Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Member `calNewU_RT_LES` Missing density outside of surface, setting it to zero.

Member `calNewU_RT_LES` Missing density outside model, setting it to zero.

Member `calNewU_RT_LES` assuming theta and phi velocity same outside star as inside.

Member `calNewU_RT_LES` Assuming theta velocities are constant across surface.

Member `calNewU_RT_LES` assuming that V at $i+1$ is equal to v at i .

Member `calNewU_RT_LES` Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

Member `calNewU_RT_LES` assume viscosity is zero outside the star.

Member `calNewU_RT_LES` Missing mass outside model, setting it to zero.

Member `calNewU_RT_LES` Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewU_RTP` missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` in calculation of $u_{i+1,j,k}$ setting $u_{i+1,j,k} = u_{i+1/2,j,k}$.

Member `calNewU_RTP` Missing `grid.dLocalGridOld[grid.nD][i+1][j][k]` in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

Member `calNewU_RTP` assuming theta velocity is constant across the surface.

Member `calNewU_RTP` assuming phi velocity is constant across the surface.

Member `calNewU_RTP` Missing `grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0]` in calculation of $\langle \rho \rangle_{i+1/2}$ setting it to zero.

Member `calNewU_RTP` Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it equal to `Parameters::dAlpha` `grid.dLocalGridOld[grid.nDM][nICen][0][0]`.

Member `calNewU_RTP` Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewU_RTP` Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Member `calNewU_RTP_LES` Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Member `calNewU_RTP_LES` Missing density outside of surface, setting it to zero.

Member `calNewU_RTP_LES` Missing density outside model, setting it to zero.

Member `calNewU_RTP_LES` assuming theta and phi velocity same outside star as inside.

Member `calNewU_RTP_LES` Assuming theta velocities are constant across surface.

Member `calNewU_RTP_LES` assuming that V at $i+1$ is equal to v at i .

Member `calNewU_RTP_LES` Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

Member `calNewU_RTP_LES` assume viscosity is zero outside the star.

Member `calNewU_RTP_LES` Missing mass outside model, setting it to zero.

Member `calNewU_RTP_LES` Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Member `calNewV_RT` `grid.dLocalGridOld[grid.nV][i+1][j+1][k]` is missing

Member `calNewV_RT` missing upwind gradient, using centred gradient instead

Member `calNewV_RT_LES` Assuming density outside star is zero

Member `calNewV_RT_LES` Assuming theta velocity is constant across surface.

Member `calNewV_RT_LES` Assuming eddy viscosity is zero at surface.

Member `calNewV_RTP` Assuming theta and phi velocities are the same at the surface of the star as just inside the star.

Member `calNewV_RTP` using centered gradient for upwind gradient outside star at surface.

Member `calNewV_RTP_LES` Assuming density outside star is zero

Member `calNewV_RTP_LES` Assuming theta velocity is constant across surface.

Member `calNewV_RTP_LES` Assuming eddy viscosity is zero at surface.

Member `calNewW_RTP` missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` assuming that the phi velocity at the outer most interface is the same as the phi velocity in the center of the zone.

Member `calNewW_RTP` missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` in outer most zone. This is needed to calculate the upwind gradient for donor cell. The centered gradient is used instead when moving in the negative direction.

Member `calNewW_RTP_LES` assume theta and phi velocities are constant across surface

Member `calNewW_RTP_LES` assume eddy viscosity is zero at surface

Member `calNewW_RTP_LES` assume upwind gradient is the same as centered gradient across surface

Member `calOldEddyVisc_RTP_SM` assuming that theta velocity is constant across surface

Member `calOldEddyVisc_RTP_SM` assume phi velocity is constant across surface

Member `dImplicitEnergyFunction_R_LES_SB` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Member `dImplicitEnergyFunction_R_LES_SB` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Member `dImplicitEnergyFunction_R_LES_SB` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `dImplicitEnergyFunction_R_SB` Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Member `dImplicitEnergyFunction_R_SB` `grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Member `dImplicitEnergyFunction_R_SB` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Member `dImplicitEnergyFunction_RT_LES_SB` assuming V at ip1half is the same as V at i

Member `dImplicitEnergyFunction_RT_LES_SB` Assuming energy outside model is the same as the energy in the last zone inside the model.

Member `dImplicitEnergyFunction_RT_LES_SB` Using centered gradient for upwind gradient when motion is into the star at the surface

Member `dImplicitEnergyFunction_RT_LES_SB` Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member **dImplicitEnergyFunction_RT_SB** Using centered gradient for upwind gradient when motion is into the star at the surface

Member **dImplicitEnergyFunction_RT_SB** Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member **dImplicitEnergyFunction_RTP_LES_SB** assuming V at $ip1half$ is the same as V at i

Member **dImplicitEnergyFunction_RTP_LES_SB** assuming W at $ip1half$ is the same as W at i

Member **dImplicitEnergyFunction_RTP_LES_SB** Using $E_{\{i,j,k\}^{n+1/2}}$ for $E_{\{i+1/2,j,k\}^{n+1/2}}$

Member **dImplicitEnergyFunction_RTP_LES_SB** Using centered gradient for upwind gradient when motion is into the star at the surface

Member **dImplicitEnergyFunction_RTP_LES_SB** Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Member **dImplicitEnergyFunction_RTP_SB** Using $E_{\{i,j,k\}^{n+1/2}}$ for $E_{\{i+1/2,j,k\}^{n+1/2}}$

Member **dImplicitEnergyFunction_RTP_SB** Using centered gradient for upwind gradient when motion is into the star at the surface

Member **dImplicitEnergyFunction_RTP_SB** Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

eos	49
Functions	59
Global	62
Grid	64
Implicit	85
MessPass	90
Output	92
Parameters	95
Performance	101
ProcTop	103
Time	106
WatchZone	110

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

/home/cgeroux/SPHERLS/src/ eos.cpp	111
/home/cgeroux/SPHERLS/src/ eos.h	112
/home/cgeroux/SPHERLS/src/ exception2.h	??
/home/cgeroux/SPHERLS/src/ xmlFunctions.h	??
dataManipulation.cpp	113
dataManipulation.h	123
dataMonitoring.cpp	132
dataMonitoring.h	137
global.cpp	141
global.h	142
main.cpp	144
main.h	147
physEquations.cpp	149
physEquations.h	195
physEquations_HEAD.cpp	251
physEquations_new.cpp	304
userguide.h	357
watchzone.cpp	358
watchzone.h	359

Chapter 6

Class Documentation

6.1 eos Class Reference

```
#include <eos.h>
```

Public Member Functions

- `eos` ()
- `eos` (int `nNumT`, int `nNumRho`)
- `eos` (const `eos` &ref)
- `~eos` ()
- `eos` & `operator=` (const `eos` &eosRightSide)
- void `readAscii` (std::string sFileName)
- void `readBobsAscii` (std::string sFileName)
- void `writeAscii` (std::string sFileName)
- void `readBin` (std::string sFileName) throw (exception2)
- void `writeBin` (std::string sFileName)
- double `dGetPressure` (double dT, double dRho)
- double `dGetEnergy` (double dT, double dRho)
- double `dGetOpacity` (double dT, double dRho)
- double `dDRhoDP` (double dT, double dRho)
- double `dSoundSpeed` (double dT, double dRho)
- void `getEKappa` (double dT, double dRho, double &dE, double &dKappa)
- void `getPEKappa` (double dT, double dRho, double &dP, double &dE, double &dKappa)
- void `getPEKappaGamma` (double dT, double dRho, double &dP, double &dE, double &dKappa, double &dGamma)
- void `getPEKappaGammaCp` (double dT, double dRho, double &dP, double &dE, double &dKappa, double &dGamma, double &dCp)
- void `getPKappaGamma` (double dT, double dRho, double &dP, double &dKappa, double &dGamma)
- void `gamma1DelAdC_v` (double dT, double dRho, double &dGamma1, double &dDelAd, double &dC_v)
- void `getPAndDRhoDP` (double dT, double dRho, double &dP, double &dDRhoDP)
- void `getEAndDTDE` (double dT, double dRho, double &dE, double &dDTDE)

Public Attributes

- int [nNumRho](#)
- int [nNumT](#)
- double [dXMassFrac](#)
- double [dYMassFrac](#)
- double [dLogRhoMin](#)
- double [dLogRhoDelta](#)
- double [dLogTMin](#)
- double [dLogTDelta](#)
- double ** [dLogP](#)
- double ** [dLogE](#)
- double ** [dLogKappa](#)

6.1.1 Detailed Description

This class holds an equation of state as well as many functions useful for manipulating it

6.1.2 Constructor & Destructor Documentation

6.1.2.1 `eos::eos ()`

Constructor, doesn't really do anything

References [dLogE](#), [dLogKappa](#), [dLogP](#), [nNumRho](#), and [nNumT](#).

6.1.2.2 `eos::eos (int nNumT, int nNumRho)`

Constructor, allocates memory for the 2D arrays

Parameters:

- ← ***nNumT*** number of temperatures in the equaiton of state table
- ← ***nNumRho*** number of densities in the equaiton of state table

6.1.2.3 `eos::eos (const eos & ref)`

Copy constructor, simply constructs a new [eos](#) object from another [eos](#) object

References [dLogE](#), [dLogKappa](#), [dLogP](#), [dLogRhoDelta](#), [dLogRhoMin](#), [dLogTDelta](#), [dLogTMin](#), [dXMassFrac](#), [dYMassFrac](#), [nNumRho](#), and [nNumT](#).

6.1.2.4 `eos::~~eos ()`

Destructor, deletes dynamic arrays

References [dLogE](#), [dLogKappa](#), [dLogP](#), and [nNumRho](#).

6.1.3 Member Function Documentation

6.1.3.1 eos & eos::operator= (const eos & *eosRightSide*)

Assignment operator, assigns one eos object to another.

References dLogE, dLogKappa, dLogP, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, nNumRho, and nNumT.

6.1.3.2 void eos::readAscii (std::string *sFileName*)

This fuction reads in an ascii file and stores it in the current object.

Parameters:

← *sFileName* name of the equation of state file to read from.

References dLogE, dLogKappa, dLogP, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, dXMassFrac, dYMassFrac, nNumRho, and nNumT.

6.1.3.3 void eos::readBobsAscii (std::string *sFileName*)

This fuction reads in an ascii file and stores it in the current object. The ascii file is in Bob's format.

Parameters:

← *sFileName* name of the equation of state file to read from.

References dLogE, dLogKappa, dLogP, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, dXMassFrac, dYMassFrac, nNumRho, and nNumT.

6.1.3.4 void eos::writeAscii (std::string *sFileName*)

This fuction writes the equation of state stored in the current object to an ascii file.

Parameters:

← *sFileName* name of the file to write the equation of state to.

References dLogE, dLogKappa, dLogP, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, dXMassFrac, dYMassFrac, nNumRho, and nNumT.

6.1.3.5 void eos::readBin (std::string *sFileName*) throw (exception2)

This fuction reads in a binary file and stores it in the current object.

Parameters:

← *sFileName* name of the equation of state file to read from.

References `dLogE`, `dLogKappa`, `dLogP`, `dLogRhoDelta`, `dLogRhoMin`, `dLogTDelta`, `dLogTMin`, `dXMassFrac`, `dYMassFrac`, `nNumRho`, and `nNumT`.

Referenced by `init()`.

6.1.3.6 void eos::writeBin (std::string *sFileName*)

This function writes the equation of state stored in the current object to a binary file.

Parameters:

← *sFileName* name of the file to write the equation of state to.

References `dLogE`, `dLogKappa`, `dLogP`, `dLogRhoDelta`, `dLogRhoMin`, `dLogTDelta`, `dLogTMin`, `dXMassFrac`, `dYMassFrac`, `nNumRho`, and `nNumT`.

6.1.3.7 double eos::dGetPressure (double *dT*, double *dRho*)

This function linearly interpolates the pressure to a given temperature and density. Note that both *dT* and *dRho* are not in log space.

Parameters:

← *dT* temperature to interpolate to.

← *dRho* density to interpolate to.

Returns:

the interpolated pressure.

References `dLogP`, `dLogRhoDelta`, `dLogRhoMin`, `dLogTDelta`, `dLogTMin`, `nNumRho`, and `nNumT`.

Referenced by `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_LES()`, `dImplicitEnergyFunction_R_LES_SB()`, `dImplicitEnergyFunction_RT()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RTP()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, and `dImplicitEnergyFunction_RTP_SB()`.

6.1.3.8 double eos::dGetEnergy (double *dT*, double *dRho*)

This function linearly interpolates the energy to a given temperature and density. Note that both *dT* and *dRho* are not in log space.

Parameters:

← *dT* temperature to interpolate to.

← *dRho* density to interpolate to.

Returns:

the interpolated energy.

References dLogE, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, nNumRho, and nNumT.

Referenced by dImplicitEnergyFunction_R(), dImplicitEnergyFunction_R_LES(), dImplicitEnergyFunction_R_LES_SB(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), and dImplicitEnergyFunction_RTP_SB().

6.1.3.9 double eos::dGetOpacity (double *dT*, double *dRho*)

This function linearly interpolates the opacity to a given temperature and and density. Note that both *dT* and *dRho* are not in log space.

Parameters:

- ← *dT* temperature to interpolate to.
- ← *dRho* density to interpolate to.

Returns:

the interpolated opacity.

References dLogKappa, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, nNumRho, and nNumT.

Referenced by dImplicitEnergyFunction_R(), dImplicitEnergyFunction_R_LES(), dImplicitEnergyFunction_R_LES_SB(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), and dImplicitEnergyFunction_RTP_SB().

6.1.3.10 double eos::dDRhoDP (double *dT*, double *dRho*)

This function calculates the partial derivative of density w.r.t. pressure

Parameters:

- ← *dT* temperature at which the derivative is to be computed
- ← *dRho* density at which the derivative is to be computed

Returns:

the partial derivative of density w.r.t. pressure.

References dLogP, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, nNumRho, and nNumT.

6.1.3.11 double eos::dSoundSpeed (double *dT*, double *dRho*)

This function calculates the adiabatic sound speed

Parameters:

- ← ***dT*** temperature at which the derivative is to be computed
- ← ***dRho*** density at which the derivative is to be computed

Returns:

the sound speed.

References dLogE, dLogP, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, nNumRho, and nNumT.

6.1.3.12 void eos::getEKappa (double *dT*, double *dRho*, double & *dE*, double & *dKappa*)

This function linearly interpolates the three dependent quantities (Pressure, Energy , Opacity) to a given temperature and density. Note that both *dT* and *dRho* are not in log space.

Parameters:

- ← ***dT*** temperature to interpolate to.
- ← ***dRho*** density to interpolate to.
- ***dE*** energy at *dT* and *dRho*.
- ***dKappa*** opacity at *dT* and *dRho*.

References dLogE, dLogKappa, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, nNumRho, and nNumT.

6.1.3.13 void eos::getPEKappa (double *dT*, double *dRho*, double & *dP*, double & *dE*, double & *dKappa*)

This function linearly interpolates the three dependent quantities (Pressure, Energy , Opacity) to a given temperature and density. Note that both *dT* and *dRho* are not in log space.

Parameters:

- ← ***dT*** temperature to interpolate to.
- ← ***dRho*** density to interpolate to.
- ***dP*** pressure at *dT* and *dRho*.
- ***dE*** energy at *dT* and *dRho*.
- ***dKappa*** opacity at *dT* and *dRho*.

References dLogE, dLogKappa, dLogP, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, nNumRho, and nNumT.

6.1.3.14 void eos::getPEKappaGamma (double *dT*, double *dRho*, double & *dP*, double & *dE*, double & *dKappa*, double & *dGamma*)

This function linearly interpolates the energy and opacity to a given temperature and density. Note that both *dT* and *dRho* are not in log space.

Parameters:

- ← ***dT*** temperature to interpolate to.
- ← ***dRho*** density to interpolate to.
- ***dP*** pressure at dT and dRho.
- ***dE*** energy at dT and dRho.
- ***dKappa*** opacity at dT and dRho.
- ***dGamma*** adiabatic index at dT and dRho.

References dLogE, dLogKappa, dLogP, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, nNumRho, and nNumT.

Referenced by calNewPEKappaGamma_TEOS(), and calOldPEKappaGamma_TEOS().

6.1.3.15 void eos::getPEKappaGammaCp (double *dT*, double *dRho*, double & *dP*, double & *dE*, double & *dKappa*, double & *dGamma*, double & *dCp*)

This function linearly interpolates the energy and opacity to a given temperature and density. Note that both dT and dRho are not in log space.

Parameters:

- ← ***dT*** temperature to interpolate to.
- ← ***dRho*** density to interpolate to.
- ***dP*** pressure at dT and dRho.
- ***dE*** energy at dT and dRho.
- ***dKappa*** opacity at dT and dRho.
- ***dGamma*** adiabatic index at dT and dRho.
- ***dCp*** specific heat at constant pressure at dT and dRho.

References dLogE, dLogKappa, dLogP, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, nNumRho, and nNumT.

6.1.3.16 void eos::getPKappaGamma (double *dT*, double *dRho*, double & *dP*, double & *dKappa*, double & *dGamma*)

This function linearly interpolates the energy and opacity to a given temperature and density. Note that both dT and dRho are not in log space.

Parameters:

- ← ***dT*** temperature to interpolate to.
- ← ***dRho*** density to interpolate to.
- ***dP*** pressure at dT and dRho.
- ***dKappa*** opacity at dT and dRho.
- ***dGamma*** adiabatic index at dT and dRho.

References dLogE, dLogKappa, dLogP, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, nNumRho, and nNumT.

Referenced by calNewTPKappaGamma_TEOS().

6.1.3.17 `void eos::gamma1DelAdC_v (double dT, double dRho, double & dGamma1, double & dDelAd, double & dC_v)`

This function calculates gamma1 and the adiabatic gradient

Parameters:

- ← *dT* temperature at which the derivative is to be computed
- ← *dRho* density at which the derivative is to be computed
- *dGamma1* gamma1
- *dDelAd* adiabatic gradient
- *dC_v* specific heat at constant volume

References dLogE, dLogP, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, nNumRho, and nNumT.

6.1.3.18 `void eos::getPAndDRhoDP (double dT, double dRho, double & dP, double & dDRhoDP)`

This function calculates the partial derivative of density w.r.t. pressure and the pressure

Parameters:

- ← *dT* temperature at which the derivative is to be computed
- ← *dRho* density at which the derivative is to be computed
- *dP* pressure at dT and dRho
- *dDRhoDP* derivative of density w.r.t. pressure at constant temperature

References dLogP, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, nNumRho, and nNumT.

6.1.3.19 `void eos::getEAndDTDE (double dT, double dRho, double & dE, double & dDTDE)`

This function calculates the partial derivative of temperature w.r.t. energy and the energy

Parameters:

- ← *dT* temperature at which the derivative is to be computed
- ← *dRho* density at which the derivative is to be computed
- *dE* energy at dT and dRho
- *dDTDE* derivative of temperature w.r.t. energy at constant density

References dLogE, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, nNumRho, and nNumT.

Referenced by calNewTPKappaGamma_TEOS().

6.1.4 Member Data Documentation

6.1.4.1 int eos::nNumRho

Number of densities in the equation of state table

Referenced by dDRhoDP(), dGetEnergy(), dGetOpacity(), dGetPressure(), dSoundSpeed(), eos(), gamma1DelAdC_v(), getEAndDTDE(), getEKappa(), getPAndDRhoDP(), getPEKappa(), getPEKappaGamma(), getPEKappaGammaCp(), getPKappaGamma(), operator=(), readAscii(), readBin(), readBobsAscii(), writeAscii(), writeBin(), and ~eos().

6.1.4.2 int eos::nNumT

Number of temperatures in the equation of state table

Referenced by dDRhoDP(), dGetEnergy(), dGetOpacity(), dGetPressure(), dSoundSpeed(), eos(), gamma1DelAdC_v(), getEAndDTDE(), getEKappa(), getPAndDRhoDP(), getPEKappa(), getPEKappaGamma(), getPEKappaGammaCp(), getPKappaGamma(), operator=(), readAscii(), readBin(), readBobsAscii(), writeAscii(), and writeBin().

6.1.4.3 double eos::dXMassFrac

Hydrogen mass fraction of the composition used to generate the equation of state table.

Referenced by eos(), readAscii(), readBin(), readBobsAscii(), writeAscii(), and writeBin().

6.1.4.4 double eos::dYMassFrac

Helium mass fraction of the composition used to generate the equation of state table.

Referenced by eos(), readAscii(), readBin(), readBobsAscii(), writeAscii(), and writeBin().

6.1.4.5 double eos::dLogRhoMin

Minimum density of the table in log10.

Referenced by dDRhoDP(), dGetEnergy(), dGetOpacity(), dGetPressure(), dSoundSpeed(), eos(), gamma1DelAdC_v(), getEAndDTDE(), getEKappa(), getPAndDRhoDP(), getPEKappa(), getPEKappaGamma(), getPEKappaGammaCp(), getPKappaGamma(), operator=(), readAscii(), readBin(), readBobsAscii(), writeAscii(), and writeBin().

6.1.4.6 double eos::dLogRhoDelta

Increment of the density between table entries in log10.

Referenced by dDRhoDP(), dGetEnergy(), dGetOpacity(), dGetPressure(), dSoundSpeed(), eos(), gamma1DelAdC_v(), getEAndDTDE(), getEKappa(), getPAndDRhoDP(), getPEKappa(), getPEKappaGamma(), getPEKappaGammaCp(), getPKappaGamma(), operator=(), readAscii(), readBin(), readBobsAscii(), writeAscii(), and writeBin().

6.1.4.7 double eos::dLogTMin

Minimum temperature of the table in log10.

Referenced by `dDRhoDP()`, `dGetEnergy()`, `dGetOpacity()`, `dGetPressure()`, `dSoundSpeed()`, `eos()`, `gamma1DelAdC_v()`, `getEAndDTDE()`, `getEKappa()`, `getPAndDRhoDP()`, `getPEKappa()`, `getPEKappaGamma()`, `getPEKappaGammaCp()`, `getPKappaGamma()`, `operator=()`, `readAscii()`, `readBin()`, `readBobsAscii()`, `writeAscii()`, and `writeBin()`.

6.1.4.8 `double eos::dLogTDelta`

Increment of the temperature between table entries in \log_{10} .

Referenced by `dDRhoDP()`, `dGetEnergy()`, `dGetOpacity()`, `dGetPressure()`, `dSoundSpeed()`, `eos()`, `gamma1DelAdC_v()`, `getEAndDTDE()`, `getEKappa()`, `getPAndDRhoDP()`, `getPEKappa()`, `getPEKappaGamma()`, `getPEKappaGammaCp()`, `getPKappaGamma()`, `operator=()`, `readAscii()`, `readBin()`, `readBobsAscii()`, `writeAscii()`, and `writeBin()`.

6.1.4.9 `double** eos::dLogP`

2D array of \log_{10} pressures. `dLogP[i][j]` gives the \log_{10} pressure at \log_{10} density of `eos::dLogRhoDelta*i+eos::dLogRhoMin`, and at \log_{10} temperature of `eos::dLogTDelta*j+eos::dLogTMin`.

Referenced by `dDRhoDP()`, `dGetPressure()`, `dSoundSpeed()`, `eos()`, `gamma1DelAdC_v()`, `getPAndDRhoDP()`, `getPEKappa()`, `getPEKappaGamma()`, `getPEKappaGammaCp()`, `getPKappaGamma()`, `operator=()`, `readAscii()`, `readBin()`, `readBobsAscii()`, `writeAscii()`, `writeBin()`, and `~eos()`.

6.1.4.10 `double** eos::dLogE`

2D array of \log_{10} energies. `dLogE[i][j]` gives the \log_{10} energy at \log_{10} density of `eos::dLogRhoDelta*i+eos::dLogRhoMin`, and at \log_{10} temperature of `eos::dLogTDelta*j+eos::dLogTMin`.

Referenced by `dGetEnergy()`, `dSoundSpeed()`, `eos()`, `gamma1DelAdC_v()`, `getEAndDTDE()`, `getEKappa()`, `getPEKappa()`, `getPEKappaGamma()`, `getPEKappaGammaCp()`, `getPKappaGamma()`, `operator=()`, `readAscii()`, `readBin()`, `readBobsAscii()`, `writeAscii()`, `writeBin()`, and `~eos()`.

6.1.4.11 `double** eos::dLogKappa`

2D array of \log_{10} opacities. `dLogKappa[i][j]` gives the \log_{10} opacity at \log_{10} density of `eos::dLogRhoDelta*i+eos::dLogRhoMin`, and at \log_{10} temperature of `eos::dLogTDelta*j+eos::dLogTMin`.

Referenced by `dGetOpacity()`, `eos()`, `getEKappa()`, `getPEKappa()`, `getPEKappaGamma()`, `getPEKappaGammaCp()`, `getPKappaGamma()`, `operator=()`, `readAscii()`, `readBin()`, `readBobsAscii()`, `writeAscii()`, `writeBin()`, and `~eos()`.

The documentation for this class was generated from the following files:

- `/home/cgeroux/SPHERLS/src/eos.h`
- `/home/cgeroux/SPHERLS/src/eos.cpp`

6.2 Functions Class Reference

```
#include <global.h>
```

Public Member Functions

- [Functions](#) ()

Public Attributes

- void(* [fpCalculateNewVelocities](#))(Grid &, Parameters &, Time &, ProcTop &)
- void(* [fpCalculateNewGridVelocities](#))(Grid &, Parameters &, Time &, ProcTop &, MessPass &)
- void(* [fpCalculateNewRadii](#))(Grid &, Time &)
- void(* [fpCalculateNewDensities](#))(Grid &, Parameters &, Time &, ProcTop &)
- void(* [fpCalculateNewEnergies](#))(Grid &, Parameters &, Time &, ProcTop &)
- void(* [fpCalculateDeltat](#))(Grid &, Parameters &, Time &, ProcTop &)
- void(* [fpCalculateAveDensities](#))(Grid &)
- void(* [fpCalculateNewEOSVars](#))(Grid &, Parameters &)
- void(* [fpCalculateNewAV](#))(Grid &, Parameters &)
- void(* [fpModelWrite](#))(std::string sFileName, ProcTop &, Grid &, Time &, Parameters &)
- void(* [fpWriteWatchZones](#))(Output &, Grid &, Parameters &, Time &, ProcTop &)
- void(* [fpUpdateLocalBoundaryVelocitiesNewGrid](#))(ProcTop &, MessPass &, Grid &)
- void(* [fpImplicitSolve](#))(Grid &, Implicit &, Parameters &, Time &, ProcTop &, MessPass &, [Functions](#) &)
- void(* [fpCalculateNewEddyVisc](#))(Grid &, Parameters &)

6.2.1 Detailed Description

This class holds function pointers used to indicate the functions which should be used to calculate the various needed quantities. These functions can be different from processor to processor. For example [ProcTop::nRank](#)=0 processor will have only 1D versions of the conservation equations, while the rest of the processors will have 3D versions. These functions will also change depending on what kind of model is being calculated and the number of dimensions the calculation includes.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Functions::Functions ()

Constructor for the class [Functions](#).

References [fpCalculateAveDensities](#), [fpCalculateDeltat](#), [fpCalculateNewAV](#), [fpCalculateNewDensities](#), [fpCalculateNewEnergies](#), [fpCalculateNewEOSVars](#), [fpCalculateNewGridVelocities](#), [fpCalculateNewRadii](#), and [fpCalculateNewVelocities](#).

6.2.3 Member Data Documentation

6.2.3.1 `void(* Functions::fpCalculateNewVelocities)(Grid &, Parameters &, Time &, ProcTop &)`

Function pointer to the function used to calculate new velocities.

Referenced by `Functions()`, `main()`, and `setMainFunctions()`.

6.2.3.2 `void(* Functions::fpCalculateNewGridVelocities)(Grid &, Parameters &, Time &, ProcTop &, MessPass &)`

Function pointer to the function used to calculate new grid velocities.

Referenced by `Functions()`, `main()`, and `setMainFunctions()`.

6.2.3.3 `void(* Functions::fpCalculateNewRadii)(Grid &, Time &)`

Function pointer to the function used to calculate new radii.

Referenced by `Functions()`, `main()`, and `setMainFunctions()`.

6.2.3.4 `void(* Functions::fpCalculateNewDensities)(Grid &, Parameters &, Time &, ProcTop &)`

Function pointer to the function used to calculate the new densities.

Referenced by `Functions()`, `main()`, and `setMainFunctions()`.

6.2.3.5 `void(* Functions::fpCalculateNewEnergies)(Grid &, Parameters &, Time &, ProcTop &)`

Function pointer to the function used to calculate the new energies.

Referenced by `Functions()`, `main()`, and `setMainFunctions()`.

6.2.3.6 `void(* Functions::fpCalculateDeltat)(Grid &, Parameters &, Time &, ProcTop &)`

Function pointer to the function used to calculate the new time step.

Referenced by `Functions()`, `main()`, and `setMainFunctions()`.

6.2.3.7 `void(* Functions::fpCalculateAveDensities)(Grid &)`

Function pointer to the function used to calculate the new average density.

Referenced by `Functions()`, `main()`, and `setMainFunctions()`.

6.2.3.8 void(* Functions::fpCalculateNewEOSVars)(Grid &, Parameters &)

Function pointer to the function used to calculate the new variables depending on the equation of state.

Referenced by Functions(), main(), and setMainFunctions().

6.2.3.9 void(* Functions::fpCalculateNewAV)(Grid &, Parameters &)

Function pointer to the function used to calculate new Artificial viscosity.

Referenced by Functions(), main(), and setMainFunctions().

6.2.3.10 void(* Functions::fpModelWrite)(std::string sFileName, ProcTop &, Grid &, Time &, Parameters &)

Function pointer to the function used to write out models.

Referenced by fin(), main(), and setMainFunctions().

6.2.3.11 void(* Functions::fpWriteWatchZones)(Output &, Grid &, Parameters &, Time &, ProcTop &)

Function pointer to the function that is used to write out watch zone files

Referenced by main(), and setMainFunctions().

6.2.3.12 void(* Functions::fpUpdateLocalBoundaryVelocitiesNewGrid)(ProcTop &, MessPass &, Grid &)

Function pointer to the function that is used to update velocities across boundaries.

Referenced by main(), and setMainFunctions().

6.2.3.13 void(* Functions::fpImplicitSolve)(Grid &, Implicit &, Parameters &, Time &, ProcTop &, MessPass &, Functions &)

Function pointer to the function that is used to implicitly solve for the temperature, then uses the equation of state to solve for energy, opacity, and pressure.

Referenced by main(), and setMainFunctions().

6.2.3.14 void(* Functions::fpCalculateNewEddyVisc)(Grid &, Parameters &)

Function pointer to the function that is used to calculate the new eddy viscosity.

Referenced by main(), and setMainFunctions().

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

6.3 Global Class Reference

```
#include <global.h>
```

Public Member Functions

- [Global](#) ()

Public Attributes

- [ProcTop](#) procTop
- [MessPass](#) messPass
- [Grid](#) grid
- [Time](#) time
- [Parameters](#) parameters
- [Output](#) output
- [Performance](#) performance
- [Functions](#) functions
- [Implicit](#) implicit

6.3.1 Detailed Description

This class is simply a class that holds the other classes.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 [Global::Global](#) ()

Constructor for the class [Global](#).

6.3.3 Member Data Documentation

6.3.3.1 [ProcTop](#) [Global::procTop](#)

An instance of the [ProcTop](#) class.

Referenced by [main\(\)](#).

6.3.3.2 MessPass Global::messPass

An instance of the [MessPass](#) class.

Referenced by `main()`.

6.3.3.3 Grid Global::grid

An instance of the [Grid](#) class.

Referenced by `main()`.

6.3.3.4 Time Global::time

An instance of the [Time](#) class.

Referenced by `main()`.

6.3.3.5 Parameters Global::parameters

An instance of the [Parameters](#) class.

Referenced by `main()`.

6.3.3.6 Output Global::output

An instance of the [Output](#) class.

Referenced by `main()`.

6.3.3.7 Performance Global::performance

An instance of the [Performance](#) class.

Referenced by `main()`.

6.3.3.8 Functions Global::functions

An instance of the [Functions](#) class.

Referenced by `main()`.

6.3.3.9 Implicit Global::implicit

An instance of the [Implicit](#) class.

Referenced by `main()`.

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

6.4 Grid Class Reference

```
#include <global.h>
```

Public Member Functions

- [Grid \(\)](#)

Public Attributes

- int [nM](#)
- int [nTheta](#)
- int [nPhi](#)
- int [nDM](#)
- int [nR](#)
- int [nD](#)
- int [nU](#)
- int [nU0](#)
- int [nV](#)
- int [nW](#)
- int [nT](#)
- int [nE](#)
- int [nP](#)
- int [nKappa](#)
- int [nGamma](#)
- int [nDenAve](#)
- int [nQ0](#)
- int [nQ1](#)
- int [nQ2](#)
- int [nDTheta](#)
- int [nDPhi](#)
- int [nSinThetaIJK](#)
- int [nSinThetaIjp1halfK](#)
- int [nCotThetaIjp1halfK](#)
- int [nCotThetaIJK](#)
- int [nDCosThetaIJK](#)
- int [nEddyVisc](#)
- int [nNumDims](#)
- int [nNumVars](#)
- int [nNumIntVars](#)
- int [nNum1DZones](#)
- int [nNumGhostCells](#)
- int * [nGlobalGridDims](#)
- int ** [nVariables](#)
- int *** [nLocalGridDims](#)
- double **** [dLocalGridNew](#)

- double **** dLocalGridOld
- int ** nStartUpdateExplicit
- int ** nEndUpdateExplicit
- int ** nStartUpdateImplicit
- int ** nEndUpdateImplicit
- int *** nStartGhostUpdateExplicit
- int *** nEndGhostUpdateExplicit
- int *** nStartGhostUpdateImplicit
- int *** nEndGhostUpdateImplicit
- int * nCenIntOffset
- int nGlobalGridPositionLocalGrid [3]

6.4.1 Detailed Description

This class manages information which pertains to grid data.

External variables used with Gamma Law (GL) gas equation of state and their array indexes:

1D (nNumVars=7)		2D (nNumVars=9)		3D (nNumVars=11)	
Variable	Index	Variable	Index	Variable	Index
nM	0	nM	0	nM	0
nDM	1	nTheta	1	nTheta	1
nR	2	nDM	2	nPhi	2
nD	3	nR	3	nDM	3
nU	4	nD	4	nR	4
nU0	5	nU	5	nD	5
nE	6	nU0	6	nU	6
		nV	7	nU0	7
		nE	8	nV	8
				nW	9
				nE	10

External variables used with Tabulated Equation Of State (TEOS) and their array indexes:

1D (nNumVars=7)		2D (nNumVars=9)		3D (nNumVars=11)	
Variable	Index	Variable	Index	Variable	Index
nM	0	nM	0	nM	0
nDM	1	nTheta	1	nTheta	1
nR	2	nDM	2	nPhi	2
nD	3	nR	3	nDM	3
nU	4	nD	4	nR	4
nU0	5	nU	5	nD	5
nT	6	nU0	6	nU	6
		nV	7	nU0	7
		nT	8	nV	8
				nW	9
				nT	10

Internal variables with GL gas equation of state:

1D (nNumIntVars=2)		2D (nNumIntVars=8)	
Variable	Index	Variable	Index
nP	nNumVars+0	nP	nNumVars+0
nQ0	nNumVars+1	nQ0	nNumVars+1
		nDenAve	nNumVars+2
		nDCosThetaIJK	nNumVars+3
		nQ1	nNumVars+4
		nDTheta	nNumVars+5
		nSinThetaIJK	nNumVars+6
		nSinThetaIjp1halfK	nNumVars+7
3D (nNumIntVars=12)			
Variable	Index		
nP	nNumVars+0		
nQ0	nNumVars+1		
nDenAve	nNumVars+2		
nDPhi	nNumVars+3		
nDCosThetaIJK	nNumVars+4		
nQ1	nNumVars+5		
nDTheta	nNumVars+6		
nSinThetaIJK	nNumVars+7		
nSinThetaIjp1halfK	nNumVars+8		
nCotThetaIJK	nNumVars+9		
nCotThetaIjp1halfK	nNumVars+10		
nQ2	nNumVars+11		

Internal variables with TEOS:

1D (nNumIntVars=5)		2D (nNumIntVars=11)	
Variable	Index	Variable	Index
nP	nNumVars+0	nP	nNumVars+0
nQ0	nNumVars+1	nQ0	nNumVars+1
nE	nNumVars+2	nDenAve	nNumVars+2
nKappa	nNumVars+3	nDCosThetaIJK	nNumVars+3
nGamma	nNumVars+4	nE	nNumVars+4
		nKappa	nNumVars+5
		nGamma	nNumVars+6
		nQ1	nNumVars+7
		nDTheta	nNumVars+8
		nSinThetaIJK	nNumVars+9
		nSinThetaIjp1halfK	nNumVars+10
3D (nNumIntVars=15)			
Variable	Index		
nP	nNumVars+0		
nQ0	nNumVars+1		
nDenAve	nNumVars+2		
nDPhi	nNumVars+3		
nDCosThetaIJK	nNumVars+4		
nE	nNumVars+5		
nKappa	nNumVars+6		
nGamma	nNumVars+7		
nQ1	nNumVars+8		
nDTheta	nNumVars+9		
nSinThetaIJK	nNumVars+10		
nSinThetaIjp1halfK	nNumVars+11		
nCotThetaIJK	nNumVars+12		
nCotThetaIjp1halfK	nNumVars+13		
nQ2	nNumVars+14		

The variable indexes are set in [modelRead](#) based on the input model.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 Grid::Grid ()

Constructor for the class [Grid](#).

References [dLocalGridNew](#), [dLocalGridOld](#), [nCenIntOffset](#), [nCotThetaIJK](#), [nCotThetaIjp1halfK](#), [nD](#), [nDCosThetaIJK](#), [nDenAve](#), [nDM](#), [nDPhi](#), [nDTheta](#), [nE](#), [nEddyVisc](#), [nEndGhostUpdateExplicit](#), [nEndGhostUpdateImplicit](#), [nEndUpdateExplicit](#), [nEndUpdateImplicit](#), [nGamma](#), [nGlobalGridDims](#), [nKappa](#), [nLocalGridDims](#), [nM](#), [nP](#), [nPhi](#), [nQ0](#), [nQ1](#), [nQ2](#), [nR](#), [nSinThetaIJK](#), [nSinThetaIjp1halfK](#), [nStartGhostUpdateExplicit](#), [nStartGhostUpdateImplicit](#), [nStartUpdateExplicit](#), [nStartUpdateImplicit](#), [nT](#), [nTheta](#), [nU](#), [nU0](#), [nV](#), [nVariables](#), and [nW](#).

6.4.3 Member Data Documentation

6.4.3.1 `int Grid::nM`

Index of M_r independent variable in grid `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an external grid variable included in the count `Grid::nNumVars`. This is an independent grid variable.

Referenced by `calNewU_R()`, `calNewU_R_LES()`, `calNewU_RT()`, `calNewU_RT_LES()`, `calNewU_RTP()`, `calNewU_RTP_LES()`, `Grid()`, and `modelRead()`.

6.4.3.2 `int Grid::nTheta`

Index of θ independent variable in grid `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an external grid variable included in the count `Grid::nNumVars`. This is an independent grid variable.

Referenced by `Grid()`, `initInternalVars()`, and `modelRead()`.

6.4.3.3 `int Grid::nPhi`

Index of ϕ independent variable in grid `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an external grid variable included in the count `Grid::nNumVars`. This is an independent grid variable.

Referenced by `Grid()`, `initInternalVars()`, and `modelRead()`.

6.4.3.4 `int Grid::nDM`

Index of δM in grids, `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an external grid variable included in the count `Grid::nNumVars`

Referenced by `calNewE_R_AD()`, `calNewE_R_NA()`, `calNewE_R_NA_LES()`, `calNewE_RT_AD()`, `calNewE_RT_NA()`, `calNewE_RT_NA_LES()`, `calNewE_RTP_AD()`, `calNewE_RTP_NA()`, `calNewE_RTP_NA_LES()`, `calNewU_R()`, `calNewU_R_LES()`, `calNewU_RT()`, `calNewU_RT_LES()`, `calNewU_RTP()`, `calNewU_RTP_LES()`, `calNewV_RT()`, `calNewV_RT_LES()`, `calNewV_RTP()`, `calNewV_RTP_LES()`, `calNewW_RTP()`, `calNewW_RTP_LES()`, `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_LES()`, `dImplicitEnergyFunction_R_LES_SB()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RT_SB()`, `dImplicitEnergyFunction_RTP()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, `dImplicitEnergyFunction_RTP_SB()`, `Grid()`, `modelRead()`, `writeWatchZones_R_GL()`, `writeWatchZones_R_TEOS()`, `writeWatchZones_RT_GL()`, `writeWatchZones_RT_TEOS()`, `writeWatchZones_RTP_GL()`, and `writeWatchZones_RTP_TEOS()`.

6.4.3.5 `int Grid::nR`

Index of r in grids, `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an external grid variable included in the count `Grid::nNumVars`

Referenced by `average3DTo1DBoundariesNew()`, `average3DTo1DBoundariesOld()`, `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `calNewD_R()`, `calNewD_RT()`, `calNewD_RTP()`, `calNewDenave_RT()`, `calNewDenave_RTP()`, `calNewE_R_AD()`, `calNewE_R_NA()`, `calNewE_R_NA_LES()`, `calNewE_RT_AD()`, `calNewE_RT_NA()`, `calNewE_RT_NA_LES()`, `calNewE_RTP_AD()`, `calNewE_RTP_NA()`, `calNewE_RTP_NA_LES()`, `calNewEddyVisc_R_CN()`, `calNewEddyVisc_R_SM()`, `calNewEddyVisc_RT_CN()`, `calNewEddyVisc_RT_SM()`, `calNewEddyVisc_RTP_CN()`, `calNewEddyVisc_RTP_SM()`, `calNewQ0_R_GL()`, `calNewQ0_R_TEOS()`, `calNewQ0Q1_RT_GL()`, `calNewQ0Q1_RT_TEOS()`, `calNewQ0Q1Q2_RTP_GL()`, `calNewQ0Q1Q2_RTP_TEOS()`, `calNewR()`, `calNewU0_R()`, `calNewU0_RT()`, `calNewU0_RTP()`, `calNewU_R()`, `calNewU_R_LES()`, `calNewU_RT()`, `calNewU_RT_LES()`, `calNewU_RTP()`, `calNewU_RTP_LES()`, `calNewV_RT()`, `calNewV_RT_LES()`, `calNewV_RTP()`, `calNewV_RTP_LES()`, `calNewW_RTP()`, `calNewW_RTP_LES()`, `calOldDenave_RT()`, `calOldDenave_RTP()`, `calOldEddyVisc_R_CN()`, `calOldEddyVisc_R_SM()`, `calOldEddyVisc_RT_CN()`, `calOldEddyVisc_RT_SM()`, `calOldEddyVisc_RTP_CN()`, `calOldEddyVisc_RTP_SM()`, `calOldQ0_R_GL()`, `calOldQ0_R_TEOS()`, `calOldQ0Q1_RT_GL()`, `calOldQ0Q1_RT_TEOS()`, `calOldQ0Q1Q2_RTP_GL()`, `calOldQ0Q1Q2_RTP_TEOS()`, `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_LES()`, `dImplicitEnergyFunction_R_LES_SB()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RT_SB()`, `dImplicitEnergyFunction_RTP()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, `dImplicitEnergyFunction_RTP_SB()`, `Grid()`, `main()`, `modelRead()`, `writeWatchZones_R_GL()`, `writeWatchZones_R_TEOS()`, `writeWatchZones_RT_GL()`, `writeWatchZones_RT_TEOS()`, `writeWatchZones_RTP_GL()`, and `writeWatchZones_RTP_TEOS()`.

6.4.3.6 int Grid::nD

Index of ρ in grids, `Grid::dLocalGridOld` and `Grid::dLocalGridNew`. This is an external grid variable included in the count `Grid::nNumVars`

Referenced by `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `calNewD_R()`, `calNewD_RT()`, `calNewD_RTP()`, `calNewDenave_R()`, `calNewDenave_RT()`, `calNewDenave_RTP()`, `calNewE_R_AD()`, `calNewE_R_NA()`, `calNewE_R_NA_LES()`, `calNewE_RT_AD()`, `calNewE_RT_NA()`, `calNewE_RT_NA_LES()`, `calNewE_RTP_AD()`, `calNewE_RTP_NA()`, `calNewE_RTP_NA_LES()`, `calNewEddyVisc_R_SM()`, `calNewEddyVisc_RT_SM()`, `calNewEddyVisc_RTP_SM()`, `calNewP_GL()`, `calNewPEKappaGamma_TEOS()`, `calNewQ0_R_GL()`, `calNewQ0_R_TEOS()`, `calNewQ0Q1_RT_GL()`, `calNewQ0Q1_RT_TEOS()`, `calNewQ0Q1Q2_RTP_GL()`, `calNewQ0Q1Q2_RTP_TEOS()`, `calNewTPKappaGamma_TEOS()`, `calNewU0_R()`, `calNewU0_RT()`, `calNewU0_RTP()`, `calNewU_R()`, `calNewU_R_LES()`, `calNewU_RT()`, `calNewU_RT_LES()`, `calNewU_RTP()`, `calNewU_RTP_LES()`, `calNewV_RT()`, `calNewV_RT_LES()`, `calNewV_RTP()`, `calNewV_RTP_LES()`, `calNewW_RTP()`, `calNewW_RTP_LES()`, `calOldDenave_R()`, `calOldDenave_RT()`, `calOldDenave_RTP()`, `calOldEddyVisc_R_SM()`, `calOldEddyVisc_RT_SM()`, `calOldEddyVisc_RTP_SM()`, `calOldP_GL()`, `calOldPEKappaGamma_TEOS()`, `calOldQ0_R_GL()`, `calOldQ0_R_TEOS()`, `calOldQ0Q1_RT_GL()`, `calOldQ0Q1_RT_TEOS()`, `calOldQ0Q1Q2_RTP_GL()`, `calOldQ0Q1Q2_RTP_TEOS()`, `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_LES()`, `dImplicitEnergyFunction_R_LES_SB()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RT_SB()`, `dImplicitEnergyFunction_RTP()`, and `dImplicitEnergyFunction_RTP_LES()`.

dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB(), Grid(), initDonorFracAndMaxConVel_R_GL(), initDonorFracAndMaxConVel_R_TEOS(), initDonorFracAndMaxConVel_RT_GL(), initDonorFracAndMaxConVel_RT_TEOS(), initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), initWatchZones(), main(), modelRead(), setupLocalGrid(), writeWatchZones_R_GL(), writeWatchZones_R_TEOS(), writeWatchZones_RT_GL(), writeWatchZones_RT_TEOS(), writeWatchZones_RTP_GL(), and writeWatchZones_RTP_TEOS().

6.4.3.7 int Grid::nU

Index of u in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an external grid variable included in the count [Grid::nNumVars](#)

Referenced by calDelt_R_GL(), calDelt_R_TEOS(), calDelt_RT_GL(), calDelt_RT_TEOS(), calDelt_RTP_GL(), calDelt_RTP_TEOS(), calNewD_R(), calNewD_RT(), calNewD_RTP(), calNewE_R_AD(), calNewE_R_NA(), calNewE_R_NA_LES(), calNewE_RT_AD(), calNewE_RT_NA(), calNewE_RT_NA_LES(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewEddyVisc_R_SM(), calNewEddyVisc_RT_SM(), calNewEddyVisc_RTP_SM(), calNewQ0_R_GL(), calNewQ0_R_TEOS(), calNewQ0Q1_RT_GL(), calNewQ0Q1_RT_TEOS(), calNewQ0Q1Q2_RTP_GL(), calNewQ0Q1Q2_RTP_TEOS(), calNewU0_R(), calNewU0_RT(), calNewU0_RTP(), calNewU_R(), calNewU_R_LES(), calNewU_RT(), calNewU_RT_LES(), calNewU_RTP(), calNewU_RTP_LES(), calNewV_RT(), calNewV_RT_LES(), calNewV_RTP(), calNewV_RTP_LES(), calNewW_RTP(), calNewW_RTP_LES(), calOldEddyVisc_R_SM(), calOldEddyVisc_RT_SM(), calOldEddyVisc_RTP_SM(), calOldQ0_R_GL(), calOldQ0_R_TEOS(), calOldQ0Q1_RT_GL(), calOldQ0Q1_RT_TEOS(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), dImplicitEnergyFunction_R(), dImplicitEnergyFunction_R_LES(), dImplicitEnergyFunction_R_LES_SB(), dImplicitEnergyFunction_R_SB(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_RT_LES(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RT_SB(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB(), Grid(), initDonorFracAndMaxConVel_R_GL(), initDonorFracAndMaxConVel_R_TEOS(), initDonorFracAndMaxConVel_RT_GL(), initDonorFracAndMaxConVel_RT_TEOS(), initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), modelRead(), updateLocalBoundaryVelocitiesNewGrid_R(), updateLocalBoundaryVelocitiesNewGrid_RT(), updateLocalBoundaryVelocitiesNewGrid_RTP(), writeWatchZones_R_GL(), writeWatchZones_R_TEOS(), writeWatchZones_RT_GL(), writeWatchZones_RT_TEOS(), writeWatchZones_RTP_GL(), and writeWatchZones_RTP_TEOS().

6.4.3.8 int Grid::nU0

Index of u_0 in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an external grid variable included in the count [Grid::nNumVars](#)

Referenced by calDelt_R_GL(), calDelt_R_TEOS(), calDelt_RT_GL(), calDelt_RT_TEOS(), calDelt_RTP_GL(), calDelt_RTP_TEOS(), calNewD_R(), calNewD_RT(), calNewD_RTP(), calNewE_R_AD(), calNewE_R_NA(), calNewE_R_NA_LES(), calNewE_RT_AD(), calNewE_RT_NA(), calNewE_RT_NA_LES(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewEddyVisc_RT_SM(), calNewEddyVisc_RTP_SM(), calNewR(), calNewU0_R(), calNewU0_RT(), calNewU0_RTP(), calNewU_R(), calNewU_R_LES(), calNewU_RT(), calNewU_RT_

LES(), calNewU_RTP(), calNewU_RTP_LES(), calNewV_RT(), calNewV_RT_LES(), calNewV_RTP(), calNewV_RTP_LES(), calNewW_RTP(), calNewW_RTP_LES(), calOldEddyVisc_RT_SM(), calOldEddyVisc_RTP_SM(), dImplicitEnergyFunction_R(), dImplicitEnergyFunction_R_LES(), dImplicitEnergyFunction_R_LES_SB(), dImplicitEnergyFunction_R_SB(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_RT_LES(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RT_SB(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB(), Grid(), initDonorFracAndMaxConVel_R_GL(), initDonorFracAndMaxConVel_R_TEOS(), initDonorFracAndMaxConVel_RT_GL(), initDonorFracAndMaxConVel_RT_TEOS(), initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), main(), modelRead(), writeWatchZones_R_GL(), writeWatchZones_R_TEOS(), writeWatchZones_RT_GL(), writeWatchZones_RT_TEOS(), writeWatchZones_RTP_GL(), and writeWatchZones_RTP_TEOS().

6.4.3.9 int Grid::nV

Index of v in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an external grid variable included in the count [Grid::nNumVars](#)

Referenced by calDelt_RT_GL(), calDelt_RT_TEOS(), calDelt_RTP_GL(), calDelt_RTP_TEOS(), calNewD_RT(), calNewD_RTP(), calNewE_RT_AD(), calNewE_RT_NA(), calNewE_RT_NA_LES(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewEddyVisc_RT_SM(), calNewEddyVisc_RTP_SM(), calNewQ0Q1_RT_GL(), calNewQ0Q1_RT_TEOS(), calNewQ0Q1Q2_RTP_GL(), calNewQ0Q1Q2_RTP_TEOS(), calNewU0_RT(), calNewU0_RTP(), calNewU_RT(), calNewU_RT_LES(), calNewU_RTP(), calNewU_RTP_LES(), calNewV_RT(), calNewV_RT_LES(), calNewV_RTP(), calNewV_RTP_LES(), calNewW_RTP(), calNewW_RTP_LES(), calOldEddyVisc_RT_SM(), calOldEddyVisc_RTP_SM(), calOldQ0Q1_RT_GL(), calOldQ0Q1_RT_TEOS(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_RT_LES(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RT_SB(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB(), Grid(), initDonorFracAndMaxConVel_RT_GL(), initDonorFracAndMaxConVel_RT_TEOS(), initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), modelRead(), updateLocalBoundaryVelocitiesNewGrid_RT(), updateLocalBoundaryVelocitiesNewGrid_RTP(), writeWatchZones_RT_GL(), writeWatchZones_RT_TEOS(), writeWatchZones_RTP_GL(), and writeWatchZones_RTP_TEOS().

6.4.3.10 int Grid::nW

Index of w in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an external grid variable included in the count [Grid::nNumVars](#)

Referenced by calDelt_RTP_GL(), calDelt_RTP_TEOS(), calNewD_RTP(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewEddyVisc_RTP_SM(), calNewQ0Q1Q2_RTP_GL(), calNewQ0Q1Q2_RTP_TEOS(), calNewU0_RTP(), calNewU_RTP(), calNewU_RTP_LES(), calNewV_RTP(), calNewV_RTP_LES(), calNewW_RTP(), calNewW_RTP_LES(), calOldEddyVisc_RTP_SM(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB()

SB(), Grid(), initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), modelRead(), updateLocalBoundaryVelocitiesNewGrid_RTP(), writeWatchZones_RTP_GL(), and writeWatchZones_RTP_TEOS().

6.4.3.11 int Grid::nT

Index of T in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an external grid variable included in the count [Grid::nNumVars](#). This variable is defined at cell centers.

Referenced by calDelt_R_TEOS(), calDelt_RT_TEOS(), calDelt_RTP_TEOS(), calNewE_R_NA(), calNewE_R_NA_LES(), calNewE_RT_NA(), calNewE_RT_NA_LES(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewPEKappaGamma_TEOS(), calNewTPKappaGamma_TEOS(), calOldPEKappaGamma_TEOS(), dImplicitEnergyFunction_R(), dImplicitEnergyFunction_R_LES(), dImplicitEnergyFunction_R_LES_SB(), dImplicitEnergyFunction_R_SB(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_RT_LES(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RT_SB(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB(), Grid(), implicitSolve_R(), implicitSolve_RT(), implicitSolve_RTP(), initImplicitCalculation(), initUpdateLocalBoundaries(), main(), modelRead(), writeWatchZones_R_TEOS(), writeWatchZones_RT_TEOS(), and writeWatchZones_RTP_TEOS().

6.4.3.12 int Grid::nE

Index of E in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable included in the count [Grid::nNumIntVars](#), unless the calculation is adiabatic in which case it is an external grid variable. This variable is defined at cell centers.

Referenced by calDelt_R_GL(), calDelt_RT_GL(), calDelt_RTP_GL(), calNewE_R_AD(), calNewE_R_NA(), calNewE_R_NA_LES(), calNewE_RT_AD(), calNewE_RT_NA(), calNewE_RT_NA_LES(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewP_GL(), calNewPEKappaGamma_TEOS(), calNewTPKappaGamma_TEOS(), calOldP_GL(), calOldPEKappaGamma_TEOS(), dImplicitEnergyFunction_R(), dImplicitEnergyFunction_R_LES(), dImplicitEnergyFunction_R_LES_SB(), dImplicitEnergyFunction_R_SB(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_RT_LES(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RT_SB(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB(), Grid(), initUpdateLocalBoundaries(), modelRead(), setInternalVarInf(), writeWatchZones_R_GL(), writeWatchZones_R_TEOS(), writeWatchZones_RT_GL(), writeWatchZones_RT_TEOS(), writeWatchZones_RTP_GL(), and writeWatchZones_RTP_TEOS().

6.4.3.13 int Grid::nP

Index of Pressure in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#). This variable is defined at cell centers.

Referenced by calDelt_R_GL(), calDelt_R_TEOS(), calDelt_RT_GL(), calDelt_RT_TEOS(), calDelt_RTP_GL(), calDelt_RTP_TEOS(), calNewE_R_AD(), calNewE_R_NA(), calNewE_R_NA_LES(), calNewE_RT_AD(), calNewE_RT_NA(), calNewE_RT_NA_LES(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(),

calNewP_GL(), calNewPEKappaGamma_TEOS(), calNewQ0_R_GL(), calNewQ0_R_TEOS(), calNewQ0Q1_RT_GL(), calNewQ0Q1_RT_TEOS(), calNewQ0Q1Q2_RTP_GL(), calNewQ0Q1Q2_RTP_TEOS(), calNewTPKappaGamma_TEOS(), calNewU_R(), calNewU_R_LES(), calNewU_RT(), calNewU_RT_LES(), calNewU_RTP(), calNewU_RTP_LES(), calNewV_RT(), calNewV_RT_LES(), calNewV_RTP(), calNewV_RTP_LES(), calNewW_RTP(), calNewW_RTP_LES(), calOldP_GL(), calOldPEKappaGamma_TEOS(), calOldQ0_R_GL(), calOldQ0_R_TEOS(), calOldQ0Q1_RT_GL(), calOldQ0Q1_RT_TEOS(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), Grid(), initDonorFracAndMaxConVel_R_GL(), initDonorFracAndMaxConVel_R_TEOS(), initDonorFracAndMaxConVel_RT_GL(), initDonorFracAndMaxConVel_RT_TEOS(), initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), initUpdateLocalBoundaries(), main(), modelRead(), setInternalVarInf(), writeWatchZones_R_GL(), writeWatchZones_R_TEOS(), writeWatchZones_RT_GL(), writeWatchZones_RT_TEOS(), writeWatchZones_RTP_GL(), and writeWatchZones_RTP_TEOS().

6.4.3.14 int Grid::nKappa

Index of Opacity in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#). This variable is defined at cell centers.

Referenced by calNewE_R_NA(), calNewE_R_NA_LES(), calNewE_RT_NA(), calNewE_RT_NA_LES(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewPEKappaGamma_TEOS(), calNewTPKappaGamma_TEOS(), calOldPEKappaGamma_TEOS(), Grid(), initUpdateLocalBoundaries(), modelRead(), and setInternalVarInf().

6.4.3.15 int Grid::nGamma

Index of adiabatic index in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#). This variable is defined at cell centers.

Referenced by calDelt_R_TEOS(), calDelt_RT_TEOS(), calDelt_RTP_TEOS(), calNewPEKappaGamma_TEOS(), calNewQ0_R_TEOS(), calNewQ0Q1_RT_TEOS(), calNewQ0Q1Q2_RTP_TEOS(), calNewTPKappaGamma_TEOS(), calOldPEKappaGamma_TEOS(), calOldQ0_R_TEOS(), calOldQ0Q1_RT_TEOS(), calOldQ0Q1Q2_RTP_TEOS(), Grid(), initDonorFracAndMaxConVel_R_TEOS(), initDonorFracAndMaxConVel_RT_TEOS(), initDonorFracAndMaxConVel_RTP_TEOS(), main(), modelRead(), and setInternalVarInf().

6.4.3.16 int Grid::nDenAve

Index of $\langle \rho \rangle$ in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#). This variable is defined at cell centers only in the radial direction.

Referenced by calNewDenave_R(), calNewDenave_RT(), calNewDenave_RTP(), calNewE_RT_AD(), calNewE_RT_NA(), calNewE_RT_NA_LES(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewU_RT(), calNewU_RT_LES(), calNewU_RTP(), calNewU_RTP_LES(), calNewV_RT(), calNewV_RT_LES(), calNewV_RTP(), calNewV_RTP_LES(), calNewW_RTP(), calNewW_RTP_LES(), calOldDenave_R(), calOldDenave_RT(), calOldDenave_RTP(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_RT_LES(), dImplicitEnergyFunction_RT_

LES_SB(), dImplicitEnergyFunction_RT_SB(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB(), Grid(), modelRead(), setInternalVarInf(), writeWatchZones_RT_GL(), writeWatchZones_RT_TEOS(), writeWatchZones_RTP_GL(), and writeWatchZones_RTP_TEOS().

6.4.3.17 int Grid::nQ0

Index of the radial artificial viscosity in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#). This variable is defined at cell centers.

Referenced by calDelt_R_GL(), calDelt_R_TEOS(), calDelt_RT_GL(), calDelt_RT_TEOS(), calDelt_RTP_GL(), calDelt_RTP_TEOS(), calNewE_R_AD(), calNewE_R_NA(), calNewE_R_NA_LES(), calNewE_RT_AD(), calNewE_RT_NA(), calNewE_RT_NA_LES(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewQ0_R_GL(), calNewQ0_R_TEOS(), calNewQ0Q1_RT_GL(), calNewQ0Q1_RT_TEOS(), calNewQ0Q1Q2_RTP_GL(), calNewQ0Q1Q2_RTP_TEOS(), calNewU_R(), calNewU_R_LES(), calNewU_RT(), calNewU_RT_LES(), calNewU_RTP(), calNewU_RTP_LES(), calNewV_RT(), calNewV_RT_LES(), calNewV_RTP(), calNewV_RTP_LES(), calNewW_RTP(), calNewW_RTP_LES(), calOldQ0_R_GL(), calOldQ0_R_TEOS(), calOldQ0Q1_RT_GL(), calOldQ0Q1_RT_TEOS(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), dImplicitEnergyFunction_R(), dImplicitEnergyFunction_R_LES(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_RT_LES(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RT_SB(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB(), Grid(), initDonorFracAndMaxConVel_R_GL(), initDonorFracAndMaxConVel_R_TEOS(), initDonorFracAndMaxConVel_RT_GL(), initDonorFracAndMaxConVel_RT_TEOS(), initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), modelRead(), setInternalVarInf(), writeWatchZones_R_GL(), writeWatchZones_R_TEOS(), writeWatchZones_RT_GL(), writeWatchZones_RT_TEOS(), writeWatchZones_RTP_GL(), and writeWatchZones_RTP_TEOS().

6.4.3.18 int Grid::nQ1

Index of the theta artificial viscosity in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#). This variable is defined at cell centers.

Referenced by calDelt_RT_GL(), calDelt_RT_TEOS(), calDelt_RTP_GL(), calDelt_RTP_TEOS(), calNewE_RT_AD(), calNewE_RT_NA(), calNewE_RT_NA_LES(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewQ0Q1_RT_GL(), calNewQ0Q1_RT_TEOS(), calNewQ0Q1Q2_RTP_GL(), calNewQ0Q1Q2_RTP_TEOS(), calNewU_RT_LES(), calNewU_RTP_LES(), calNewV_RT(), calNewV_RT_LES(), calNewV_RTP(), calNewV_RTP_LES(), calNewW_RTP(), calNewW_RTP_LES(), calOldQ0Q1_RT_GL(), calOldQ0Q1_RT_TEOS(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_RT_LES(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RT_SB(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB(), Grid(), initDonorFracAndMaxConVel_RT_GL(), and initDonorFracAndMaxConVel_RT_TEOS(),

initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), modelRead(), setInternalVarInf(), writeWatchZones_RT_GL(), writeWatchZones_RT_TEOS(), writeWatchZones_RTP_GL(), and writeWatchZones_RTP_TEOS().

6.4.3.19 int Grid::nQ2

Index of the phi artificial viscosity in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#). This variable is defined at cell centers.

Referenced by calDelt_RTP_GL(), calDelt_RTP_TEOS(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewQ0Q1Q2_RTP_GL(), calNewQ0Q1Q2_RTP_TEOS(), calNewU_RTP_LES(), calNewV_RTP(), calNewV_RTP_LES(), calNewW_RTP(), calNewW_RTP_LES(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB(), Grid(), initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), modelRead(), setInternalVarInf(), writeWatchZones_RTP_GL(), and writeWatchZones_RTP_TEOS().

6.4.3.20 int Grid::nDTheta

Index of $\Delta\theta$ in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#). This variable is defined at cell centers.

Referenced by calDelt_RT_GL(), calDelt_RT_TEOS(), calDelt_RTP_GL(), calDelt_RTP_TEOS(), calNewD_RTP(), calNewE_RT_AD(), calNewE_RT_NA(), calNewE_RT_NA_LES(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewEddyVisc_RT_CN(), calNewEddyVisc_RT_SM(), calNewEddyVisc_RTP_CN(), calNewEddyVisc_RTP_SM(), calNewU0_RTP(), calNewU_RT(), calNewU_RT_LES(), calNewU_RTP(), calNewU_RTP_LES(), calNewV_RT(), calNewV_RT_LES(), calNewV_RTP(), calNewV_RTP_LES(), calNewW_RTP(), calNewW_RTP_LES(), calOldEddyVisc_RT_CN(), calOldEddyVisc_RT_SM(), calOldEddyVisc_RTP_CN(), calOldEddyVisc_RTP_SM(), calOldQ0Q1_RT_GL(), calOldQ0Q1Q2_RTP_GL(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_RT_LES(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RT_SB(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB(), Grid(), initInternalVars(), modelRead(), and setInternalVarInf().

6.4.3.21 int Grid::nDPhi

Index of $\Delta\phi$ in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#). This variable is defined at cell centers.

Referenced by average3DTo1DBoundariesNew(), average3DTo1DBoundariesOld(), calDelt_RTP_GL(), calDelt_RTP_TEOS(), calNewD_RTP(), calNewDenave_RTP(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewEddyVisc_RTP_CN(), calNewEddyVisc_RTP_SM(), calNewU0_RTP(), calNewU_RTP(), calNewU_RTP_LES(), calNewV_RTP(), calNewV_RTP_LES(), calNewW_RTP(), calNewW_RTP_LES(), calOldDenave_RTP(), calOldEddyVisc_RTP_CN(), calOldEddyVisc_RTP_SM(),

calOldQ0Q1Q2_RTP_GL(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB(), Grid(), initInternalVars(), modelRead(), and setInternalVarInf().

6.4.3.22 int Grid::nSinThetaIJK

Index of $\sin \theta$ defined at zone center in grids, [Grid::dLocalGridOld](#) and [Grid::dLocalGridNew](#). This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#).

Referenced by calDelt_RTP_GL(), calDelt_RTP_TEOS(), calNewE_RT_AD(), calNewE_RT_NA(), calNewE_RT_NA_LES(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewEddyVisc_RTP_CN(), calNewEddyVisc_RTP_SM(), calNewQ0Q1_RT_GL(), calNewQ0Q1_RT_TEOS(), calNewQ0Q1Q2_RTP_GL(), calNewQ0Q1Q2_RTP_TEOS(), calNewU_RT_LES(), calNewU_RTP(), calNewU_RTP_LES(), calNewV_RT_LES(), calNewV_RTP_LES(), calNewW_RTP(), calNewW_RTP_LES(), calOldEddyVisc_RTP_CN(), calOldEddyVisc_RTP_SM(), calOldQ0Q1_RT_GL(), calOldQ0Q1_RT_TEOS(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_RT_LES(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RT_SB(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB(), Grid(), initInternalVars(), modelRead(), and setInternalVarInf().

6.4.3.23 int Grid::nSinThetaIjp1halfK

Index of $\sin \theta$ at θ interfaces in grids. This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#).

Referenced by calNewD_RT(), calNewD_RTP(), calNewE_RT_AD(), calNewE_RT_NA(), calNewE_RT_NA_LES(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewQ0Q1_RT_GL(), calNewQ0Q1_RT_TEOS(), calNewQ0Q1Q2_RTP_GL(), calNewQ0Q1Q2_RTP_TEOS(), calNewU0_RT(), calNewU0_RTP(), calNewU_RT_LES(), calNewU_RTP_LES(), calNewV_RT_LES(), calNewV_RTP(), calNewV_RTP_LES(), calNewW_RTP_LES(), calOldQ0Q1_RT_GL(), calOldQ0Q1_RT_TEOS(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_RT_LES(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RT_SB(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB(), Grid(), initInternalVars(), modelRead(), and setInternalVarInf().

6.4.3.24 int Grid::nCotThetaIjp1halfK

Index of $\cot \theta$ at θ interfaces in grids. This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#).

Referenced by calNewV_RT_LES(), calNewV_RTP(), calNewV_RTP_LES(), Grid(), initInternalVars(), modelRead(), and setInternalVarInf().

6.4.3.25 int Grid::nCotThetaIJK

Index of $\cot \theta$ at cell centers of grids. This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#).

Referenced by `calNewEddyVisc_RTP_SM()`, `calNewU_RT_LES()`, `calNewU_RTP_LES()`, `calNewW_RTP()`, `calNewW_RTP_LES()`, `calOldEddyVisc_RTP_SM()`, `Grid()`, `initInternalVars()`, `modelRead()`, and `setInternalVarInf()`.

6.4.3.26 `int Grid::nDCosThetaIJK`

Index of $\Delta \cos \theta$ defined at zone center in grids. This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#).

Referenced by `average3DTo1DBoundariesNew()`, `average3DTo1DBoundariesOld()`, `calNewD_RT()`, `calNewD_RTP()`, `calNewDenave_RT()`, `calNewDenave_RTP()`, `calNewU0_RT()`, `calNewU0_RTP()`, `calOldDenave_RT()`, `calOldDenave_RTP()`, `Grid()`, `initInternalVars()`, `modelRead()`, and `setInternalVarInf()`.

6.4.3.27 `int Grid::nEddyVisc`

Index of the eddy viscosity in the grid, it is defined at zone centers in the grids. This is an internal grid variable and is included in the count of [Grid::nNumIntVars](#).

Referenced by `calNewE_R_NA_LES()`, `calNewE_RT_NA_LES()`, `calNewE_RTP_NA_LES()`, `calNewEddyVisc_R_CN()`, `calNewEddyVisc_R_SM()`, `calNewEddyVisc_RT_CN()`, `calNewEddyVisc_RT_SM()`, `calNewEddyVisc_RTP_CN()`, `calNewEddyVisc_RTP_SM()`, `calNewU_R_LES()`, `calNewU_RT_LES()`, `calNewU_RTP_LES()`, `calNewV_RT_LES()`, `calNewV_RTP_LES()`, `calNewW_RTP_LES()`, `calOldEddyVisc_R_CN()`, `calOldEddyVisc_R_SM()`, `calOldEddyVisc_RT_CN()`, `calOldEddyVisc_RT_SM()`, `calOldEddyVisc_RTP_CN()`, `calOldEddyVisc_RTP_SM()`, `dImplicitEnergyFunction_R_LES()`, `dImplicitEnergyFunction_R_LES_SB()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, `Grid()`, `modelRead()`, `setInternalVarInf()`, `writeWatchZones_RT_GL()`, `writeWatchZones_RT_TEOS()`, `writeWatchZones_RTP_GL()`, and `writeWatchZones_RTP_TEOS()`.

6.4.3.28 `int Grid::nNumDims`

Number of dimensions of the grid. It is used to chose the appropriate conservation equations. The value of this variable is independent of processor [ProcTop::nRank](#).

Referenced by `fin()`, `initImplicitCalculation()`, `initInternalVars()`, `initWatchZones()`, `main()`, `modelRead()`, `setInternalVarInf()`, `setMainFunctions()`, `setupLocalGrid()`, and `updateNewGridWithOld()`.

6.4.3.29 `int Grid::nNumVars`

Number of grid variables. This is set when reading in the model input file in the function [modelRead](#). It is the number of variables that are printed and read from a file. The total number of variables also includes [Grid::nNumIntVars](#). The value of this variable is independent of processor [ProcTop::nRank](#).

Referenced by `average3DTo1DBoundariesOld()`, `initUpdateLocalBoundaries()`, `modelRead()`, `modelWrite_GL()`, `modelWrite_TEOS()`, `setInternalVarInf()`, `setupLocalGrid()`, `updateNewGridWithOld()`, and `updateOldGrid()`.

6.4.3.30 `int Grid::nNumIntVars`

Number of internal variables. Internal variables are variables which are not reported in model dumps, and are not required to fully specify a starting model. They are used to save important information required during computation, an example is $\sin\theta$. The value of this variable is independent of processor `ProcTop::nRank`. This variable is set depending on the model read in (adiabatic/non-adiabatic/number of dimensions) in the function `modelRead` located in the file `dataManipulation.cpp`.

Referenced by `average3DTo1DBoundariesOld()`, `initUpdateLocalBoundaries()`, `modelRead()`, `setInternalVarInf()`, `setupLocalGrid()`, `updateNewGridWithOld()`, and `updateOldGrid()`.

6.4.3.31 `int Grid::nNum1DZones`

Number of zones in 1D region of grid. The number of zones in 3D region is (`Grid::nGlobalGridDims[0]`- `Grid::nNum1DZones`). This is set when reading in the model input file in the function `modelRead`. The value of this variable is independent of processor `ProcTop::nRank`.

Referenced by `initUpdateLocalBoundaries()`, `initWatchZones()`, `modelRead()`, `modelWrite_GL()`, `modelWrite_TEOS()`, and `setupLocalGrid()`.

6.4.3.32 `int Grid::nNumGhostCells`

Number of cells which are not included in local grid updating. This number is used in all dimensions to add to each local grid. When variables are not defined in a given direction ghost cells are not included in that direction. This is set when reading in the model input file in the function `modelRead`. The value of this variable is independent of processor `ProcTop::nRank`.

Referenced by `calNewU0_RT()`, `calNewU0_RTP()`, `calOldEddyVisc_R_CN()`, `calOldEddyVisc_R_SM()`, `calOldEddyVisc_RT_CN()`, `calOldEddyVisc_RT_SM()`, `calOldEddyVisc_RTP_CN()`, `calOldEddyVisc_RTP_SM()`, `calOldPEKappaGamma_TEOS()`, `initImplicitCalculation()`, `initInternalVars()`, `initUpdateLocalBoundaries()`, `initWatchZones()`, `modelRead()`, `modelWrite_GL()`, `modelWrite_TEOS()`, `setupLocalGrid()`, and `updateNewGridWithOld()`.

6.4.3.33 `int* Grid::nGlobalGridDims`

Size of the entire global grid. It is an array of size 3 to hold size of each dimension of global grid. This size does not include `Grid::nNumGhostCells` or the extra size required for interface centered quantities. The values of this variable are independent of processor `ProcTop::nRank`. In the case of 1D or 2D calculations the θ and ϕ dimensions are set to 1 or just the ϕ dimensions is set to 1 depending on the number of dimensions. The r , θ and ϕ dimensions are in the 0, 1 and 2 indices of the array respectively.

Referenced by `Grid()`, `init()`, `initImplicitCalculation()`, `initUpdateLocalBoundaries()`, `initWatchZones()`, `modelRead()`, `modelWrite_GL()`, `modelWrite_TEOS()`, and `setupLocalGrid()`.

6.4.3.34 `int** Grid::nVariables`

Provides information on grid variables. A 2D array of size `Grid::nNumVars`+`Grid::nNumIntVars` by 3+1. `nVariables[n][1]` has values:

- -1: indicating that variable `n` is not defined

- 0: indicating that variable `n` is zone centered quantity
- 1: indicating that variable `n` is an interface centered quantity

in directions `l=0,1,2` which corresponding to \hat{r} , $\hat{\theta}$, and $\hat{\phi}$ respectively. `nVariables[n][l]` with `l=3` is used to indicate if a variable is dependent on time (1) or not(0). The values of this variable are independent of processor `ProcTop::nRank`.

Referenced by `average3DTo1DBoundariesNew()`, `average3DTo1DBoundariesOld()`, `Grid()`, `initUpdateLocalBoundaries()`, `modelRead()`, `modelWrite_GL()`, `modelWrite_TEOS()`, `setInternalVarInf()`, `setupLocalGrid()`, and `updateNewGridWithOld()`.

6.4.3.35 int*** Grid::nLocalGridDims

Local grid dimensions. It is An array of size `ProcTop::nNumProcs` by `Grid::nNumVars+Grid::nNumIntVars` by 3. `nLocalGridDims[p][n][l]` gives the dimension of the local grid on processor `p` for variable `n` in direction `l`. This variable does not include `Grid::nNumGhostCells`. The values of this variable are independent of processor `ProcTop::nRank`.

Referenced by `calNewU0_RT()`, `calNewU0_RTP()`, `Grid()`, `initImplicitCalculation()`, `initInternalVars()`, `initUpdateLocalBoundaries()`, `initWatchZones()`, `modelRead()`, `modelWrite_GL()`, `modelWrite_TEOS()`, `setupLocalGrid()`, and `updateNewGridWithOld()`.

6.4.3.36 double**** Grid::dLocalGridNew

Updated local grid values. An array of size `Grid::nNumVars+Grid::nNumIntVars` by `Grid::nLocalGridDims[0]+2*Grid::nNumGhostCells` by `Grid::nLocalGridDims[1]+2*Grid::nNumGhostCells` by `Grid::nLocalGridDims[2]+2*Grid::nNumGhostCells` provided that the variable is defined in all 3 directions. Variables that are not defined in all 3 directions will have the additional two ghost cells left out in that direction and will also have a dimension of size 1 in that direction. This array contains the current grid state as it is being updated through calculations. This is a processor dependent variable and contains only the local grid for the current processor plus ghost cells.

Referenced by `average3DTo1DBoundariesNew()`, `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `calNewD_R()`, `calNewD_RT()`, `calNewD_RTP()`, `calNewDenave_R()`, `calNewDenave_RT()`, `calNewDenave_RTP()`, `calNewE_R_AD()`, `calNewE_R_NA()`, `calNewE_R_NA_LES()`, `calNewE_RT_AD()`, `calNewE_RT_NA()`, `calNewE_RT_NA_LES()`, `calNewE_RTP_AD()`, `calNewE_RTP_NA()`, `calNewE_RTP_NA_LES()`, `calNewEddyVisc_R_CN()`, `calNewEddyVisc_R_SM()`, `calNewEddyVisc_RT_CN()`, `calNewEddyVisc_RT_SM()`, `calNewEddyVisc_RTP_CN()`, `calNewEddyVisc_RTP_SM()`, `calNewP_GL()`, `calNewPEKappaGamma_TEOS()`, `calNewQ0_R_GL()`, `calNewQ0_R_TEOS()`, `calNewQ0Q1_RT_GL()`, `calNewQ0Q1_RT_TEOS()`, `calNewQ0Q1Q2_RTP_GL()`, `calNewQ0Q1Q2_RTP_TEOS()`, `calNewR()`, `calNewTPKappaGamma_TEOS()`, `calNewU0_R()`, `calNewU0_RT()`, `calNewU0_RTP()`, `calNewU_R()`, `calNewU_R_LES()`, `calNewU_RT()`, `calNewU_RT_LES()`, `calNewU_RTP()`, `calNewU_RTP_LES()`, `calNewV_RT()`, `calNewV_RT_LES()`, `calNewV_RTP()`, `calNewV_RTP_LES()`, `calNewW_RTP()`, `calNewW_RTP_LES()`, `calOldEddyVisc_R_CN()`, `calOldEddyVisc_RT_CN()`, `calOldEddyVisc_RTP_CN()`, `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_LES()`, `dImplicitEnergyFunction_R_LES_SB()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RT_SB()`, `dImplicitEnergyFunction_RTP()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, `dImplicitEnergyFunction_RTP_SB()`, `Grid()`,

`implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, `initUpdateLocalBoundaries()`, `setupLocalGrid()`, `updateLocalBoundaries()`, `updateLocalBoundariesNewGrid()`, `updateNewGridWithOld()`, and `updateOldGrid()`.

6.4.3.37 double**** `Grid::dLocalGridOld`

`Grid` values from previous time step. An array the same size as `Grid::dLocalGridNew` but instead of containing the current grid state, it contains the last complete grid state. This is a processor dependent variable and contains only the local grid for the current processor plus ghost cells.

Referenced by `average3DTo1DBoundariesNew()`, `average3DTo1DBoundariesOld()`, `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `calNewD_R()`, `calNewD_RT()`, `calNewD_RTP()`, `calNewDenave_RT()`, `calNewDenave_RTP()`, `calNewE_R_AD()`, `calNewE_R_NA()`, `calNewE_R_NA_LES()`, `calNewE_RT_AD()`, `calNewE_RT_NA()`, `calNewE_RT_NA_LES()`, `calNewE_RTP_AD()`, `calNewE_RTP_NA()`, `calNewE_RTP_NA_LES()`, `calNewEddyVisc_R_SM()`, `calNewEddyVisc_RT_CN()`, `calNewEddyVisc_RT_SM()`, `calNewEddyVisc_RTP_CN()`, `calNewEddyVisc_RTP_SM()`, `calNewQ0_R_GL()`, `calNewQ0_R_TEOS()`, `calNewQ0Q1_RT_GL()`, `calNewQ0Q1_RT_TEOS()`, `calNewQ0Q1Q2_RTP_GL()`, `calNewQ0Q1Q2_RTP_TEOS()`, `calNewR()`, `calNewTPKappaGamma_TEOS()`, `calNewU0_R()`, `calNewU0_RT()`, `calNewU0_RTP()`, `calNewU_R()`, `calNewU_R_LES()`, `calNewU_RT()`, `calNewU_RT_LES()`, `calNewU_RTP()`, `calNewU_RTP_LES()`, `calNewV_RT()`, `calNewV_RT_LES()`, `calNewV_RTP()`, `calNewV_RTP_LES()`, `calNewW_RTP()`, `calNewW_RTP_LES()`, `calOldDenave_R()`, `calOldDenave_RT()`, `calOldDenave_RTP()`, `calOldEddyVisc_R_CN()`, `calOldEddyVisc_R_SM()`, `calOldEddyVisc_RT_CN()`, `calOldEddyVisc_RT_SM()`, `calOldEddyVisc_RTP_CN()`, `calOldEddyVisc_RTP_SM()`, `calOldP_GL()`, `calOldPEKappaGamma_TEOS()`, `calOldQ0_R_GL()`, `calOldQ0_R_TEOS()`, `calOldQ0Q1_RT_GL()`, `calOldQ0Q1_RT_TEOS()`, `calOldQ0Q1Q2_RTP_GL()`, `calOldQ0Q1Q2_RTP_TEOS()`, `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_LES()`, `dImplicitEnergyFunction_R_LES_SB()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RT_SB()`, `dImplicitEnergyFunction_RTP()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, `dImplicitEnergyFunction_RTP_SB()`, `Grid()`, `initDonorFracAndMaxConVel_R_GL()`, `initDonorFracAndMaxConVel_R_TEOS()`, `initDonorFracAndMaxConVel_RT_GL()`, `initDonorFracAndMaxConVel_RT_TEOS()`, `initDonorFracAndMaxConVel_RTP_GL()`, `initDonorFracAndMaxConVel_RTP_TEOS()`, `initInternalVars()`, `initUpdateLocalBoundaries()`, `modelRead()`, `modelWrite_GL()`, `modelWrite_TEOS()`, `setupLocalGrid()`, `updateLocalBoundaries()`, `updateNewGridWithOld()`, `updateOldGrid()`, `writeWatchZones_R_GL()`, `writeWatchZones_R_TEOS()`, `writeWatchZones_RT_GL()`, `writeWatchZones_RT_TEOS()`, `writeWatchZones_RTP_GL()`, and `writeWatchZones_RTP_TEOS()`.

6.4.3.38 int** `Grid::nStartUpdateExplicit`

Positions to begin updating grid with explicit calculations. It is an array of size `nNumVars+nNumIntVars` by 3. The start positions are defined in `initUpdateLocalBoundaries()`. These start values are dependent on processor `ProcTop::nRank`.

Referenced by `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `calNewD_R()`, `calNewD_RT()`, `calNewD_RTP()`, `calNewDenave_R()`, `calNewDenave_RT()`, `calNewDenave_RTP()`, `calNewE_R_AD()`, `calNewE_R_NA()`, `calNewE_R_NA_LES()`, `calNewE_RT_AD()`, `calNewE_RT_NA()`, `calNewE_RT_NA_LES()`, `calNewE_RTP_AD()`, `calNewE_RTP_NA()`, `calNewE_RTP_`

NA_LES(), calNewEddyVisc_R_CN(), calNewEddyVisc_R_SM(), calNewEddyVisc_RT_CN(), calNewEddyVisc_RT_SM(), calNewEddyVisc_RTP_CN(), calNewEddyVisc_RTP_SM(), calNewP_GL(), calNewQ0_R_GL(), calNewQ0_R_TEOS(), calNewQ0Q1_RT_GL(), calNewQ0Q1_RT_TEOS(), calNewQ0Q1Q2_RTP_GL(), calNewQ0Q1Q2_RTP_TEOS(), calNewR(), calNewTPKappaGamma_TEOS(), calNewU0_R(), calNewU0_RT(), calNewU0_RTP(), calNewU_R(), calNewU_R_LES(), calNewU_RT(), calNewU_RT_LES(), calNewU_RTP(), calNewU_RTP_LES(), calNewV_RT(), calNewV_RT_LES(), calNewV_RTP(), calNewV_RTP_LES(), calNewW_RTP(), calNewW_RTP_LES(), calOldDenave_R(), calOldDenave_RT(), calOldDenave_RTP(), calOldEddyVisc_R_CN(), calOldEddyVisc_R_SM(), calOldEddyVisc_RT_CN(), calOldEddyVisc_RT_SM(), calOldEddyVisc_RTP_CN(), calOldEddyVisc_RTP_SM(), calOldP_GL(), calOldPEKappaGamma_TEOS(), calOldQ0_R_GL(), calOldQ0_R_TEOS(), calOldQ0Q1_RT_GL(), calOldQ0Q1_RT_TEOS(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), Grid(), initDonorFracAndMaxConVel_R_GL(), initDonorFracAndMaxConVel_R_TEOS(), initDonorFracAndMaxConVel_RT_GL(), initDonorFracAndMaxConVel_RT_TEOS(), initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), initUpdateLocalBoundaries(), and updateOldGrid().

6.4.3.39 int** Grid::nEndUpdateExplicit

Positions to stop updating grid with explicit calculations. It is an array of size `nNumVars+nNumIntVars` by 3. The end positions are defined in `initUpdateLocalBoundaries()`. These start values are dependent on processor `ProcTop::nRank`.

Referenced by `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `calNewD_R()`, `calNewD_RT()`, `calNewD_RTP()`, `calNewDenave_R()`, `calNewDenave_RT()`, `calNewDenave_RTP()`, `calNewE_R_AD()`, `calNewE_R_NA()`, `calNewE_R_NA_LES()`, `calNewE_RT_AD()`, `calNewE_RT_NA()`, `calNewE_RT_NA_LES()`, `calNewE_RTP_AD()`, `calNewE_RTP_NA()`, `calNewE_RTP_NA_LES()`, `calNewEddyVisc_R_CN()`, `calNewEddyVisc_R_SM()`, `calNewEddyVisc_RT_CN()`, `calNewEddyVisc_RT_SM()`, `calNewEddyVisc_RTP_CN()`, `calNewEddyVisc_RTP_SM()`, `calNewP_GL()`, `calNewQ0_R_GL()`, `calNewQ0_R_TEOS()`, `calNewQ0Q1_RT_GL()`, `calNewQ0Q1_RT_TEOS()`, `calNewQ0Q1Q2_RTP_GL()`, `calNewQ0Q1Q2_RTP_TEOS()`, `calNewR()`, `calNewTPKappaGamma_TEOS()`, `calNewU0_R()`, `calNewU0_RT()`, `calNewU0_RTP()`, `calNewU_R()`, `calNewU_R_LES()`, `calNewU_RT()`, `calNewU_RT_LES()`, `calNewU_RTP()`, `calNewU_RTP_LES()`, `calNewV_RT()`, `calNewV_RT_LES()`, `calNewV_RTP()`, `calNewV_RTP_LES()`, `calNewW_RTP()`, `calNewW_RTP_LES()`, `calOldDenave_R()`, `calOldDenave_RT()`, `calOldDenave_RTP()`, `calOldEddyVisc_R_CN()`, `calOldEddyVisc_R_SM()`, `calOldEddyVisc_RT_CN()`, `calOldEddyVisc_RT_SM()`, `calOldEddyVisc_RTP_CN()`, `calOldEddyVisc_RTP_SM()`, `calOldP_GL()`, `calOldPEKappaGamma_TEOS()`, `calOldQ0_R_GL()`, `calOldQ0_R_TEOS()`, `calOldQ0Q1_RT_GL()`, `calOldQ0Q1_RT_TEOS()`, `calOldQ0Q1Q2_RTP_GL()`, `calOldQ0Q1Q2_RTP_TEOS()`, `Grid()`, `initDonorFracAndMaxConVel_R_GL()`, `initDonorFracAndMaxConVel_R_TEOS()`, `initDonorFracAndMaxConVel_RT_GL()`, `initDonorFracAndMaxConVel_RT_TEOS()`, `initDonorFracAndMaxConVel_RTP_GL()`, `initDonorFracAndMaxConVel_RTP_TEOS()`, `initUpdateLocalBoundaries()`, and `updateOldGrid()`.

6.4.3.40 int** Grid::nStartUpdateImplicit

Positions to begin updating grid with implicit calculations. It is an array of size `nNumVars+nNumIntVars` by 3. The start positions are defined in `initUpdateLocalBoundaries()`. These start values are dependent on processor `ProcTop::nRank`.

Referenced by `calNewPEKappaGamma_TEOS()`, `calOldDenave_R()`, `calOldDenave_RT()`, `calOldDenave_RTP()`, `calOldPEKappaGamma_TEOS()`, `Grid()`, `initUpdateLocalBoundaries()`, and `updateOldGrid()`.

6.4.3.41 `int** Grid::nEndUpdateImplicit`

Positions to stop updating grid with implicit calculations. It is an array of size `nNumVars+nNumIntVars` by 3. The end positions are defined in `initUpdateLocalBoundaries()`. These start values are dependent on processor `ProcTop::nRank`.

Referenced by `calNewPEKappaGamma_TEOS()`, `calOldDenave_R()`, `calOldDenave_RT()`, `calOldDenave_RTP()`, `calOldPEKappaGamma_TEOS()`, `Grid()`, `initUpdateLocalBoundaries()`, and `updateOldGrid()`.

6.4.3.42 `int*** Grid::nStartGhostUpdateExplicit`

Positions to begin updating ghost cells with explicit calculations. It is an array of size `Grid::nNumVars+Grid::nNumIntVars` by 2*3 by 3. The second dimension indicates a particular ghost region. There are 2*3 since each direction can have two ghost regions. The ghost region 0, is the outer ghost region in direction 0, 1 is the inner ghost region in direction 0, etc.

Referenced by `average3DTo1DBoundariesNew()`, `average3DTo1DBoundariesOld()`, `calNewD_R()`, `calNewD_RT()`, `calNewD_RTP()`, `calNewDenave_R()`, `calNewDenave_RT()`, `calNewDenave_RTP()`, `calNewE_R_AD()`, `calNewE_R_NA()`, `calNewE_R_NA_LES()`, `calNewE_RT_AD()`, `calNewE_RT_NA()`, `calNewE_RT_NA_LES()`, `calNewE_RTP_AD()`, `calNewE_RTP_NA()`, `calNewE_RTP_NA_LES()`, `calNewEddyVisc_R_CN()`, `calNewEddyVisc_R_SM()`, `calNewEddyVisc_RT_CN()`, `calNewEddyVisc_RT_SM()`, `calNewEddyVisc_RTP_CN()`, `calNewEddyVisc_RTP_SM()`, `calNewP_GL()`, `calNewQ0_R_GL()`, `calNewQ0_R_TEOS()`, `calNewQ0Q1_RT_GL()`, `calNewQ0Q1_RT_TEOS()`, `calNewQ0Q1Q2_RTP_GL()`, `calNewQ0Q1Q2_RTP_TEOS()`, `calNewR()`, `calNewTPKappaGamma_TEOS()`, `calNewU0_R()`, `calNewU0_RT()`, `calNewU0_RTP()`, `calNewU_R()`, `calNewU_R_LES()`, `calNewU_RT()`, `calNewU_RT_LES()`, `calNewU_RTP()`, `calNewU_RTP_LES()`, `calNewV_RT()`, `calNewV_RT_LES()`, `calNewV_RTP()`, `calNewV_RTP_LES()`, `calNewW_RTP()`, `calNewW_RTP_LES()`, `calOldDenave_R()`, `calOldDenave_RT()`, `calOldDenave_RTP()`, `calOldEddyVisc_R_CN()`, `calOldEddyVisc_R_SM()`, `calOldEddyVisc_RT_CN()`, `calOldEddyVisc_RT_SM()`, `calOldEddyVisc_RTP_CN()`, `calOldEddyVisc_RTP_SM()`, `calOldP_GL()`, `calOldPEKappaGamma_TEOS()`, `calOldQ0_R_GL()`, `calOldQ0_R_TEOS()`, `calOldQ0Q1_RT_GL()`, `calOldQ0Q1_RT_TEOS()`, `calOldQ0Q1Q2_RTP_GL()`, `calOldQ0Q1Q2_RTP_TEOS()`, `Grid()`, `initUpdateLocalBoundaries()`, and `updateOldGrid()`.

6.4.3.43 `int*** Grid::nEndGhostUpdateExplicit`

Positions to end updating ghost cells with explicit calculations. Is an array of size `Grid::nNumVars+Grid::nNumIntVars` by 2*3 by 3. The second dimension corresponds to which ghost region, since each dimension can have two ghost regions. The ghost region 0, is the outer ghost region in direction 0, 1 is the inner ghost region in direction 0, etc.

Referenced by `average3DTo1DBoundariesNew()`, `average3DTo1DBoundariesOld()`, `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `calNewD_R()`, `calNewD_RT()`, `calNewD_RTP()`, `calNewDenave_R()`, `calNewDenave_RT()`, `calNewDenave_RTP()`, `calNewE_R_AD()`, `calNewE_R_NA()`, `calNewE_R_NA_LES()`, `calNewE_RT_AD()`, `calNewE_RT_NA()`, `calNewE_`

RT_NA_LES(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewEddyVisc_R_CN(), calNewEddyVisc_R_SM(), calNewEddyVisc_RT_CN(), calNewEddyVisc_RT_SM(), calNewEddyVisc_RTP_CN(), calNewEddyVisc_RTP_SM(), calNewP_GL(), calNewQ0_R_GL(), calNewQ0_R_TEOS(), calNewQ0Q1_RT_GL(), calNewQ0Q1_RT_TEOS(), calNewQ0Q1Q2_RTP_GL(), calNewQ0Q1Q2_RTP_TEOS(), calNewR(), calNewTPKappaGamma_TEOS(), calNewU0_R(), calNewU0_RT(), calNewU0_RTP(), calNewU_R(), calNewU_R_LES(), calNewU_RT(), calNewU_RT_LES(), calNewU_RTP(), calNewU_RTP_LES(), calNewV_RT(), calNewV_RT_LES(), calNewV_RTP(), calNewV_RTP_LES(), calNewW_RTP(), calNewW_RTP_LES(), calOldDenave_R(), calOldDenave_RT(), calOldDenave_RTP(), calOldEddyVisc_R_CN(), calOldEddyVisc_R_SM(), calOldEddyVisc_RT_CN(), calOldEddyVisc_RT_SM(), calOldEddyVisc_RTP_CN(), calOldEddyVisc_RTP_SM(), calOldP_GL(), calOldPEKappaGamma_TEOS(), calOldQ0_R_GL(), calOldQ0_R_TEOS(), calOldQ0Q1_RT_GL(), calOldQ0Q1_RT_TEOS(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), Grid(), initDonorFracAndMaxConVel_R_GL(), initDonorFracAndMaxConVel_R_TEOS(), initDonorFracAndMaxConVel_RT_GL(), initDonorFracAndMaxConVel_RT_TEOS(), initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), initUpdateLocalBoundaries(), and updateOldGrid().

6.4.3.44 int*** Grid::nStartGhostUpdateImplicit

Positions to begin updating ghost cells with implicit calculations. It is an array of size [Grid::nNumVars](#)+[Grid::nNumIntVars](#) by 2*3 by 3. The second dimension indicates a particular ghost region. There are 2*3 since each direction can have two ghost regions. The ghost region 0, is the outer ghost region in direction 0, 1 is the inner ghost region in direction 0, etc.

Referenced by calNewPEKappaGamma_TEOS(), calOldDenave_R(), calOldDenave_RT(), calOldDenave_RTP(), calOldPEKappaGamma_TEOS(), Grid(), initUpdateLocalBoundaries(), and updateOldGrid().

6.4.3.45 int*** Grid::nEndGhostUpdateImplicit

Positions to end updating ghost cells with implicit calculations. Is an array of size [Grid::nNumVars](#)+[Grid::nNumIntVars](#) by 2*3 by 3. The second dimension corresponds to which ghost region, since each dimension can have two ghost regions. The ghost region 0, is the outer ghost region in direction 0, 1 is the inner ghost region in direction 0, etc.

Referenced by calNewPEKappaGamma_TEOS(), calOldDenave_R(), calOldDenave_RT(), calOldDenave_RTP(), calOldPEKappaGamma_TEOS(), Grid(), initUpdateLocalBoundaries(), and updateOldGrid().

6.4.3.46 int* Grid::nCenIntOffset

Indicates the offset between interface and center quantities. If [nCenIntOffset](#)[1]=0 then the outer interface quantities have the same index as zone centered quantities in direction 1. If [nCenIntOffset](#)[1]=1 then the outer interface quantities are given by the index for the zone centered quantities +1, in direction 1. The values are dependent on [ProcTop::nRank](#) and [ProcTop::nPeriodic](#).

Referenced by average3DTo1DBoundariesNew(), average3DTo1DBoundariesOld(), calDelt_R_GL(), calDelt_R_TEOS(), calDelt_RT_GL(), calDelt_RT_TEOS(), calDelt_RTP_GL(), calDelt_RTP_TEOS(), calNewD_R(), calNewD_RT(), calNewD_RTP(), calNewDenave_RT(), calNewDenave_RTP(), calNewE_R_AD(), calNewE_R_NA(),

calNewE_R_NA_LES(), calNewE_RT_AD(), calNewE_RT_NA(), calNewE_RT_NA_LES(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewEddyVisc_R_CN(), calNewEddyVisc_R_SM(), calNewEddyVisc_RT_CN(), calNewEddyVisc_RT_SM(), calNewEddyVisc_RTP_CN(), calNewEddyVisc_RTP_SM(), calNewQ0_R_GL(), calNewQ0_R_TEOS(), calNewQ0Q1_RT_GL(), calNewQ0Q1_RT_TEOS(), calNewQ0Q1Q2_RTP_GL(), calNewQ0Q1Q2_RTP_TEOS(), calNewU0_R(), calNewU0_RT(), calNewU0_RTP(), calNewU_R(), calNewU_R_LES(), calNewU_RT(), calNewU_RT_LES(), calNewU_RTP(), calNewU_RTP_LES(), calNewV_RT(), calNewV_RT_LES(), calNewV_RTP(), calNewV_RTP_LES(), calNewW_RTP(), calNewW_RTP_LES(), calOldDenave_RT(), calOldDenave_RTP(), calOldEddyVisc_R_CN(), calOldEddyVisc_R_SM(), calOldEddyVisc_RT_CN(), calOldEddyVisc_RT_SM(), calOldEddyVisc_RTP_CN(), calOldEddyVisc_RTP_SM(), calOldQ0_R_GL(), calOldQ0_R_TEOS(), calOldQ0Q1_RT_GL(), calOldQ0Q1_RT_TEOS(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), dImplicitEnergyFunction_R(), dImplicitEnergyFunction_R_LES(), dImplicitEnergyFunction_R_LES_SB(), dImplicitEnergyFunction_R_SB(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_RT_LES(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RT_SB(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB(), Grid(), initDonorFracAndMaxConVel_R_GL(), initDonorFracAndMaxConVel_R_TEOS(), initDonorFracAndMaxConVel_RT_GL(), initDonorFracAndMaxConVel_RT_TEOS(), initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), initInternalVars(), initUpdateLocalBoundaries(), setupLocalGrid(), writeWatchZones_R_GL(), writeWatchZones_R_TEOS(), writeWatchZones_RT_GL(), writeWatchZones_RT_TEOS(), writeWatchZones_RTP_GL(), and writeWatchZones_RTP_TEOS().

6.4.3.47 int Grid::nGlobalGridPositionLocalGrid[3]

The location at which the local grid starts in the global grid. This starts at 0, for the inner most cell, including ghost zones.

Referenced by setupLocalGrid().

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

6.5 Implicit Class Reference

```
#include <global.h>
```

Public Member Functions

- [Implicit](#) ()

Public Attributes

- int [nNumImplicitZones](#)
- Mat [matCoeff](#)
- Vec [vecTCorrections](#)
- Vec [vecRHS](#)
- Vec [vecTCorrectionsLocal](#)
- KSP [kspContext](#)
- VecScatter [vecscatTCorrections](#)
- int [nMaxNumIterations](#)
- double [dTolerance](#)
- int [nNumRowsALocal](#)
- int [nNumRowsALocalSB](#)
- int * [nNumDerPerRow](#)
- int ** [nTypeDer](#)
- int *** [nLocDer](#)
- int ** [nLocFun](#)
- double [dDerivativeStepFraction](#)
- double [dCurrentRelTError](#)
- int [nCurrentNumIterations](#)
- int [nMaxNumSolverIterations](#)
- double [dMaxErrorInRHS](#)
- double [dAverageRHS](#)

6.5.1 Detailed Description

This class holds data required for the implicit calculation.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 Implicit::Implicit ()

constructor the the class [Implicit](#).

References `dCurrentRelTError`, `dDerivativeStepFraction`, `dMaxErrorInRHS`, `dTolerance`, `nCurrentNumIterations`, `nLocDer`, `nLocFun`, `nMaxNumIterations`, `nMaxNumSolverIterations`, `nNumDerPerRow`, `nNumImplicitZones`, `nNumRowsALocal`, `nNumRowsALocalSB`, and `nTypeDer`.

6.5.3 Member Data Documentation

6.5.3.1 int Implicit::nNumImplicitZones

The number of zones in the region near the surface which should used the implicit calculation of the energy equation. If zero no zones will use the implicit calculation of energy.

Referenced by `fin()`, `Implicit()`, `init()`, `initImplicitCalculation()`, `initUpdateLocalBoundaries()`, `main()`, and `setMainFunctions()`.

6.5.3.2 Mat Implicit::matCoeff

Parallel coefficient matrix (spread across all processors)

Referenced by `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `initImplicitCalculation()`.

6.5.3.3 Vec Implicit::vecTCorrections

Temperature corrections solution vector (spread across all processors)

Referenced by `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `initImplicitCalculation()`.

6.5.3.4 Vec Implicit::vecRHS

RHS vector (spread across all processors)

Referenced by `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `initImplicitCalculation()`.

6.5.3.5 Vec Implicit::vecTCorrectionsLocal

Corrections to local temperatures only (on local processor only).

Referenced by `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `initImplicitCalculation()`.

6.5.3.6 KSP Implicit::kspContext

PETSc solver context.

Referenced by `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `initImplicitCalculation()`.

6.5.3.7 VecScatter Implicit::vecscatTCorrections

Scatter context, used to hold information about retrieving the distributed temperature corrections from `vecTCorrections` and placing them into the local vector `vecTCorrectionsLocal`.

Referenced by `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `initImplicitCalculation()`.

6.5.3.8 int Implicit::nMaxNumIterations

The maximum number of iterations to try to get the largest value of `vecTCorrections` relative to the temperature below `dTolerance`. After which the calculation continues.

Referenced by `Implicit()`, `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, `init()`, and `initImplicitCalculation()`.

6.5.3.9 double Implicit::dTolerance

The amount of relative error that is allowed in the calculation of the temperature with the implicit calculation.

Referenced by `Implicit()`, `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, `init()`, and `initImplicitCalculation()`.

6.5.3.10 int Implicit::nNumRowsALocal

The number of rows of the coefficient matrix which is on the local processor.

Referenced by `Implicit()`, `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `initImplicitCalculation()`.

6.5.3.11 int Implicit::nNumRowsALocalSB

The number of rows of the coefficient matrix which is on the local processor, and that are in the surface boundary region.

Referenced by `Implicit()`, `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `initImplicitCalculation()`.

6.5.3.12 int* Implicit::nNumDerPerRow

An array of size `nNumRowsALocal` which contains the number of non-zero derivatives for a given row of A.

Referenced by `Implicit()`, `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `initImplicitCalculation()`.

6.5.3.13 `int** Implicit::nTypeDer`

An array of size `nNumRowsALocal` by `nNumDerPerRow [q]` , where `q` is a row index. Thus each row of the array can have a different length. This gives the type of derivative of row `q` for each derivative in that row. The value of this variable is set in the function `initImplicitCalculation` .

Referenced by `Implicit()`, `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `initImplicitCalculation()`.

6.5.3.14 `int*** Implicit::nLocDer`

An array of size `nNumRowsALocal` by 2 by `nNumDerPerRow [q]` , where `q` is a row index. This array holds the global position of the current row `q` for the current derivative e.g. the `p` th derivative in the `q` th row would be in row and column (`nLocDer[q][0][p]` , `nLocDer[q][1][p]`). The value of this variable is set in the function `initImplicitCalculation` .

Referenced by `Implicit()`, `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `initImplicitCalculation()`.

6.5.3.15 `int** Implicit::nLocFun`

An array of size `nNumRowsALocal` by 3 [`q`] , where `q` is a row index. This array holds the local grid position of the current row `q` e.g. the (`i,j,k`) location of the the current row in the local grid. The value of this variable is set in the function `initImplicitCalculation` .

Referenced by `Implicit()`, `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `initImplicitCalculation()`.

6.5.3.16 `double Implicit::dDerivativeStepFraction`

Dicates the size of the step that should be used to evaluate the numerical derivitves of the energy equation, for solving for the temperature implicitly. This value multiplies the temperature to produce the step size. A good value is around 5e-7.

Referenced by `Implicit()`, `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `init()`.

6.5.3.17 `double Implicit::dCurrentRelTError`

keeps track of the largest relative error in the calculation of the temperature

Referenced by `fin()`, `Implicit()`, `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `main()`.

6.5.3.18 `int Implicit::nCurrentNumIterations`

keeps track of the number of iterations needed to converge to a solution

Referenced by `fin()`, `Implicit()`, `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `main()`.

6.5.3.19 int Implicit::nMaxNumSolverIterations

If [TRACKMAXSOLVERERROR](#) set to 1, then this will be the current maximum number of iterations required for the linear equaiton solver to solve for the temperature correction over all iterations and time steps since the last model dump.

Referenced by `fin()`, `Implicit()`, `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `main()`.

6.5.3.20 double Implicit::dMaxErrorInRHS

If [TRACKMAXSOLVERERROR](#) set to 1, then this will be the current maximum absolute error between the RHS as calculated from the solution and the coeffecient matrix, and the actual RHS. This value is the maximum from all values at each iteration of the solution, from each time step since the last model dump.

Referenced by `fin()`, `Implicit()`, `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `main()`.

6.5.3.21 double Implicit::dAverageRHS

Holds the average value of the right hand side for the timestep where the error in the RHS is the largest [dMaxErrorInRHS](#). Only set if [TRACKMAXSOLVERERROR](#) is set to 1.

Referenced by `fin()`, `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, and `main()`.

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

6.6 MessPass Class Reference

```
#include <global.h>
```

Public Member Functions

- [MessPass](#) ()

Public Attributes

- MPI::Datatype * [typeSendNewGrid](#)
- MPI::Datatype * [typeRecvOldGrid](#)
- MPI::Datatype ** [typeSendNewVar](#)
- MPI::Datatype ** [typeRecvNewVar](#)
- MPI::Request * [requestSend](#)
- MPI::Request * [requestRecv](#)
- MPI::Status * [statusSend](#)
- MPI::Status * [statusRecv](#)

6.6.1 Detailed Description

This class manages information which pertains to message passing between processors.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 MessPass::MessPass ()

Constructor for class [MessPass](#).

References [requestRecv](#), [requestSend](#), [statusRecv](#), [statusSend](#), [typeRecvNewVar](#), [typeRecvOldGrid](#), [typeSendNewGrid](#), and [typeSendNewVar](#).

6.6.3 Member Data Documentation

6.6.3.1 MPI::Datatype* MessPass::typeSendNewGrid

Send data types for entire grid. It is of size [ProcTop::nNumNeighbors](#).

Referenced by [initUpdateLocalBoundaries\(\)](#), [MessPass\(\)](#), and [updateLocalBoundaries\(\)](#).

6.6.3.2 MPI::Datatype* MessPass::typeRecvOldGrid

Recv data types for entire grid. It is of size [ProcTop::nNumNeighbors](#).

Referenced by [initUpdateLocalBoundaries\(\)](#), [MessPass\(\)](#), and [updateLocalBoundaries\(\)](#).

6.6.3.3 MPI::Datatype** MessPass::typeSendNewVar

Send data types for variables. It is of size [ProcTop::nNumNeighbors](#) by [Grid::nNumVars](#).

Referenced by [calNewU0_R\(\)](#), [calNewU0_RT\(\)](#), [calNewU0_RTP\(\)](#), [initUpdateLocalBoundaries\(\)](#), [MessPass\(\)](#), and [updateLocalBoundariesNewGrid\(\)](#).

6.6.3.4 MPI::Datatype** MessPass::typeRecvNewVar

Recieve data types for variables. It is of size [ProcTop::nNumNeighbors](#) by [Grid::nNumVars](#).

Referenced by [calNewU0_R\(\)](#), [calNewU0_RT\(\)](#), [calNewU0_RTP\(\)](#), [initUpdateLocalBoundaries\(\)](#), [MessPass\(\)](#), and [updateLocalBoundariesNewGrid\(\)](#).

6.6.3.5 MPI::Request* MessPass::requestSend

Message handles.

Referenced by [initUpdateLocalBoundaries\(\)](#), [MessPass\(\)](#), and [updateLocalBoundaries\(\)](#).

6.6.3.6 MPI::Request* MessPass::requestRecv

Message handles.

Referenced by [initUpdateLocalBoundaries\(\)](#), [MessPass\(\)](#), [updateLocalBoundaries\(\)](#), and [updateLocalBoundariesNewGrid\(\)](#).

6.6.3.7 MPI::Status* MessPass::statusSend

Message status.

Referenced by [initUpdateLocalBoundaries\(\)](#), [MessPass\(\)](#), and [updateLocalBoundaries\(\)](#).

6.6.3.8 MPI::Status* MessPass::statusRecv

Message status.

Referenced by [initUpdateLocalBoundaries\(\)](#), [MessPass\(\)](#), [updateLocalBoundaries\(\)](#), and [updateLocalBoundariesNewGrid\(\)](#).

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

6.7 Output Class Reference

```
#include <global.h>
```

Public Member Functions

- [Output \(\)](#)

Public Attributes

- int [nDumpFrequencyStep](#)
- double [dDumpFrequencyTime](#)
- double [dTimeLastDump](#)
- int [nNumTimeStepsSinceLastPrint](#)
- bool [bDump](#)
- bool [bPrint](#)
- int [nPrintMode](#)
- std::string [sBaseOutputFileName](#)
- std::ofstream * [ofWatchZoneFiles](#)
- std::vector< [WatchZone](#) > [watchzoneList](#)
- int [nPrintFrequencyStep](#)
- double [dPrintFrequencyTime](#)
- double [dTimeLastPrint](#)

6.7.1 Detailed Description

This class manages information pertaining to the output of data to files.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 [Output::Output \(\)](#)

Constructor for this class.

References [bDump](#), [nDumpFrequencyStep](#), [nNumTimeStepsSinceLastPrint](#), [ofWatchZoneFiles](#), and [sBaseOutputFileName](#).

6.7.3 Member Data Documentation

6.7.3.1 int Output::nDumpFrequencyStep

How often a the grid state should be written to a file according to time step index. If it is 1 the will state will be written every time step, if it equals 2 it will be written every other time step etc. If it is 0 no dumps will be made according to the time step index.

Referenced by `init()`, `main()`, and `Output()`.

6.7.3.2 double Output::dDumpFrequencyTime

How often a the grid state should be written to a file according to simulation time in seconds. If it is 0 no dumps will be made according to simulation time.

Referenced by `init()`, and `main()`.

6.7.3.3 double Output::dTimeLastDump

The simulation time at which the last dump was made using the [Output::dDumpFrequencyTime](#) criterion.

Referenced by `init()`, and `main()`.

6.7.3.4 int Output::nNumTimeStepsSinceLastPrint

The number of time steps since the last model dump.

Referenced by `fin()`, `main()`, and `Output()`.

6.7.3.5 bool Output::bDump

The number of time steps since the last print. Should the grid state be written to a file at a frequency of [Output::nDumpFrequencyStep](#) timesteps, and/or every [Output::dDumpFrequencyTime](#) seconds of simulation time. This is set to true by putting a "<dump>" node into the "SPHERLS.xml" configuration file.

Referenced by `init()`, `main()`, and `Output()`.

6.7.3.6 bool Output::bPrint

Should status updates be printed to the screen.

Referenced by `init()`, and `main()`.

6.7.3.7 int Output::nPrintMode

Sets the way in which information should be printed to the standard output during the run. If it is 0, it will print the standard information reporting on the progress of the code. If it is 1 it will print out information to diagnose timestepping problems.

Referenced by `fin()`, `init()`, and `main()`.

6.7.3.8 `std::string Output::sBaseOutputFileName`

Base filename used for output, default is "out". All model dumps, and output information will contain this file name and extend it to indicate their specific information. The value of this variable is independent of processor [ProcTop::nRank](#).

Referenced by `fin()`, `init()`, `initWatchZones()`, `main()`, and `Output()`.

6.7.3.9 `std::ofstream* Output::ofWatchZoneFiles`

An array of output streams of size [Output::watchzoneList.size\(\)](#) which are used to write out the information of the watched zones.

Referenced by `finWatchZones()`, `initWatchZones()`, `Output()`, `writeWatchZones_R_GL()`, `writeWatchZones_R_TEOS()`, `writeWatchZones_RT_GL()`, `writeWatchZones_RT_TEOS()`, `writeWatchZones_RTP_GL()`, and `writeWatchZones_RTP_TEOS()`.

6.7.3.10 `std::vector<WatchZone> Output::watchzoneList`

A vector used to keep information used to specify the zones to be watched.

Referenced by `finWatchZones()`, `initWatchZones()`, `writeWatchZones_R_GL()`, `writeWatchZones_R_TEOS()`, `writeWatchZones_RT_GL()`, `writeWatchZones_RT_TEOS()`, `writeWatchZones_RTP_GL()`, and `writeWatchZones_RTP_TEOS()`.

6.7.3.11 `int Output::nPrintFrequencyStep`

How often the status is printed to the screen in time steps.

Referenced by `init()`, and `main()`.

6.7.3.12 `double Output::dPrintFrequencyTime`

How often the status is printed to the screen in simulation time.

Referenced by `init()`, and `main()`.

6.7.3.13 `double Output::dTimeLastPrint`

Simulation time when last status was printed.

Referenced by `init()`, and `main()`.

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

6.8 Parameters Class Reference

```
#include <global.h>
```

Public Member Functions

- [Parameters](#) ()

Public Attributes

- bool [bEOSGammaLaw](#)
- bool [bAdiabatic](#)
- int [nTypeTurbulenceMod](#)
- double [dPi](#)
- double [dSigma](#)
- double [dG](#)
- double [dGamma](#)
- std::string [sEOSFileName](#)
- eos [eosTable](#)
- double [dA](#)
- double [dAVThreshold](#)
- double [dDonorFrac](#)
- double [dAlpha](#)
- double [dTolerance](#)
- int [nMaxIterations](#)
- double [dEddyViscosity](#)
- double [dMaxConvectiveVelocity](#)
- double [dMaxConvectiveVelocity_c](#)
- double [dPrt](#)

6.8.1 Detailed Description

This class holds parameters and constants used for calculation.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Parameters::Parameters ()

Constructor for the class [Parameters](#)

References [dA](#), [dAlpha](#), [dAVThreshold](#), [dEddyViscosity](#), [dG](#), [dMaxConvectiveVelocity](#), [dMaxConvectiveVelocity_c](#), [dPi](#), [dPrt](#), and [dSigma](#).

6.8.3 Member Data Documentation

6.8.3.1 bool Parameters::bEOSGammaLaw

If true SPHERLS will use a gamma law gas instead of a tabulated equation of state. This is set in the starting model.

Referenced by `init()`, `initInternalVars()`, `initWatchZones()`, `modelRead()`, `setInternalVarInf()`, and `setMainFunctions()`.

6.8.3.2 bool Parameters::bAdiabatic

If true SPHERLS will use adiabatic functions to calculate the energy. This can be used for both gamma law gas and tabulated equations of state (see [Parameters::bEOSGammaLaw](#)).

Referenced by `init()`, and `setMainFunctions()`.

6.8.3.3 int Parameters::nTypeTurbulenceMod

This variable indicates the type of turbulence model to be used. If 0, no turbulence model will be used, if 1 it will use a constant times the zoning size, and if 2 it will use the Smagorinsky turbulence model which increases the value of the eddy viscosity parameter when there are large amounts of shear, and decrease it when there isn't.

Referenced by `init()`, `initInternalVars()`, `modelRead()`, `setInternalVarInf()`, `setMainFunctions()`, `writeWatchZones_RT_GL()`, `writeWatchZones_RT_TEOS()`, `writeWatchZones_RTP_GL()`, and `writeWatchZones_RTP_TEOS()`.

6.8.3.4 double Parameters::dPi

The value of π .

Referenced by `calNewE_R_AD()`, `calNewE_R_NA()`, `calNewE_R_NA_LES()`, `calNewE_RT_AD()`, `calNewE_RT_NA()`, `calNewE_RT_NA_LES()`, `calNewE_RTP_AD()`, `calNewE_RTP_NA()`, `calNewE_RTP_NA_LES()`, `calNewU_R()`, `calNewU_R_LES()`, `calNewU_RT()`, `calNewU_RT_LES()`, `calNewU_RTP()`, `calNewU_RTP_LES()`, `calNewV_RT()`, `calNewV_RT_LES()`, `calNewV_RTP()`, `calNewV_RTP_LES()`, `calNewW_RTP()`, `calNewW_RTP_LES()`, `calOldQ0_R_GL()`, `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_LES()`, `dImplicitEnergyFunction_R_LES_SB()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RT_SB()`, `dImplicitEnergyFunction_RTP()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, `dImplicitEnergyFunction_RTP_SB()`, `Parameters()`, `writeWatchZones_R_GL()`, `writeWatchZones_R_TEOS()`, `writeWatchZones_RT_GL()`, `writeWatchZones_RT_TEOS()`, `writeWatchZones_RTP_GL()`, and `writeWatchZones_RTP_TEOS()`.

6.8.3.5 double Parameters::dSigma

The value of σ , the Stefan-Boltzmann constant.

Referenced by `calNewE_R_NA()`, `calNewE_R_NA_LES()`, `calNewE_RT_NA()`, `calNewE_RT_NA_LES()`, `calNewE_RTP_NA()`, `calNewE_RTP_NA_LES()`, `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_LES()`, `dImplicitEnergyFunction_R_LES_SB()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RT_SB()`, `dImplicitEnergyFunction_RTP()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, `dImplicitEnergyFunction_RTP_SB()`, and `Parameters()`.

6.8.3.6 double Parameters::dG

The Gravitational constant G .

Referenced by `calNewU_R()`, `calNewU_R_LES()`, `calNewU_RT()`, `calNewU_RT_LES()`, `calNewU_RTP()`, `calNewU_RTP_LES()`, and `Parameters()`.

6.8.3.7 double Parameters::dGamma

The adiabatic γ , used in calculating the equation of state. If using a gamma law gas.

Referenced by `calDelt_R_GL()`, `calDelt_RT_GL()`, `calDelt_RTP_GL()`, `calNewQ0_R_GL()`, `calNewQ0Q1_RT_GL()`, `calNewQ0Q1Q2_RTP_GL()`, `calOldQ0_R_GL()`, `calOldQ0Q1_RT_GL()`, `calOldQ0Q1Q2_RTP_GL()`, `dEOS_GL()`, `initDonorFracAndMaxConVel_R_GL()`, `initDonorFracAndMaxConVel_RT_GL()`, `initDonorFracAndMaxConVel_RTP_GL()`, `initWatchZones()`, `modelRead()`, and `modelWrite_GL()`.

6.8.3.8 std::string Parameters::sEOSFileName

File name of equation of state table. This value is set either by the configuration file, SPHERLS.xml or in the model file read in. If it is specified in SPHERLS.xml it will override the file name set in the model.

Referenced by `init()`, `modelRead()`, and `modelWrite_TEOS()`.

6.8.3.9 eos Parameters::eosTable

Holds the equation of state table. If using a tabulated equation of state.

Referenced by `calNewPEKappaGamma_TEOS()`, `calNewTPKappaGamma_TEOS()`, `calOldPEKappaGamma_TEOS()`, `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_LES()`, `dImplicitEnergyFunction_R_LES_SB()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RT_SB()`, `dImplicitEnergyFunction_RTP()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, `dImplicitEnergyFunction_RTP_SB()`, and `init()`.

6.8.3.10 double Parameters::dA

Artificial viscosity parameter, reasonable values range from 0 to ~ 3 .

Referenced by `calNewQ0_R_GL()`, `calNewQ0_R_TEOS()`, `calNewQ0Q1_RT_GL()`, `calNewQ0Q1_RT_TEOS()`, `calNewQ0Q1Q2_RTP_GL()`, `calNewQ0Q1Q2_RTP_TEOS()`,

calOldQ0_R_GL(), calOldQ0_R_TEOS(), calOldQ0Q1_RT_GL(), calOldQ0Q1_RT_TEOS(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), init(), modelRead(), modelWrite_GL(), modelWrite_TEOS(), and Parameters().

6.8.3.11 double Parameters::dAVThreshold

The amount of compression before AV is turned on. It is in terms of a velocity difference between zone sides and is in fractions of the local sound speed.

Referenced by calNewQ0_R_GL(), calNewQ0_R_TEOS(), calNewQ0Q1_RT_GL(), calNewQ0Q1_RT_TEOS(), calNewQ0Q1Q2_RTP_GL(), calNewQ0Q1Q2_RTP_TEOS(), calOldQ0_R_GL(), calOldQ0_R_TEOS(), calOldQ0Q1_RT_GL(), calOldQ0Q1_RT_TEOS(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), init(), modelRead(), modelWrite_GL(), modelWrite_TEOS(), and Parameters().

6.8.3.12 double Parameters::dDonorFrac

Fraction of the upwind gradient to contribute to the advection term

Referenced by calDelt_R_GL(), calDelt_R_TEOS(), calDelt_RT_GL(), calDelt_RT_TEOS(), calDelt_RTP_GL(), calDelt_RTP_TEOS(), calNewE_R_NA(), calNewE_R_NA_LES(), calNewE_RT_AD(), calNewE_RT_NA(), calNewE_RT_NA_LES(), calNewE_RTP_AD(), calNewE_RTP_NA(), calNewE_RTP_NA_LES(), calNewU_R(), calNewU_R_LES(), calNewU_RT(), calNewU_RT_LES(), calNewU_RTP(), calNewU_RTP_LES(), calNewV_RT(), calNewV_RT_LES(), calNewV_RTP(), calNewV_RTP_LES(), calNewW_RTP(), calNewW_RTP_LES(), dImplicitEnergyFunction_R(), dImplicitEnergyFunction_R_LES(), dImplicitEnergyFunction_R_LES_SB(), dImplicitEnergyFunction_R_SB(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_RT_LES(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RT_SB(), dImplicitEnergyFunction_RTP(), dImplicitEnergyFunction_RTP_LES(), dImplicitEnergyFunction_RTP_LES_SB(), dImplicitEnergyFunction_RTP_SB(), initDonorFracAndMaxConVel_R_GL(), initDonorFracAndMaxConVel_R_TEOS(), initDonorFracAndMaxConVel_RT_GL(), initDonorFracAndMaxConVel_RT_TEOS(), initDonorFracAndMaxConVel_RTP_GL(), and initDonorFracAndMaxConVel_RTP_TEOS().

6.8.3.13 double Parameters::dAlpha

This parameter controls the amount of extra mass above the outter interface. it is read in from the starting model, so that it will be consistent with the value used in calculating the starting model.

Referenced by calNewE_RT_NA_LES(), calNewE_RTP_NA_LES(), calNewU_R(), calNewU_R_LES(), calNewU_RT(), calNewU_RT_LES(), calNewU_RTP(), calNewU_RTP_LES(), dImplicitEnergyFunction_RT_LES_SB(), dImplicitEnergyFunction_RTP_LES_SB(), modelRead(), modelWrite_GL(), modelWrite_TEOS(), and Parameters().

6.8.3.14 double Parameters::dTolerance

Amount of error to tolerate when calculating temperature from the equation of state.

Referenced by calNewTPKappaGamma_TEOS(), and init().

6.8.3.15 int Parameters::nMaxIterations

The maximum number of iterations to try to get the the relative error in the temperture below parameters::dTolerance.

Referenced by calNewTPKappaGamma_TEOS(), and init().

6.8.3.16 double Parameters::dEddyViscosity

Used in calculating the eddy viscosity, larger values will produce a larger value of the eddy viscosity, causing the rethermalization to happen at larger scales. This value should be kept small, a good value is 0.17, which seems to correspond with experiments.

Referenced by calNewEddyVisc_R_CN(), calNewEddyVisc_R_SM(), calNewEddyVisc_RT_CN(), calNewEddyVisc_RT_SM(), calNewEddyVisc_RTP_CN(), calNewEddyVisc_RTP_SM(), calOldEddyVisc_R_CN(), calOldEddyVisc_R_SM(), calOldEddyVisc_RT_CN(), calOldEddyVisc_RT_SM(), calOldEddyVisc_RTP_CN(), calOldEddyVisc_RTP_SM(), init(), and Parameters().

6.8.3.17 double Parameters::dMaxConvectiveVelocity

Holds the maximum convective velocity, it is set in the functions which calculate the timestep (see [calDelt_R_GL](#), [calDelt_R_TEOS](#), [calDelt_RT_GL](#), [calDelt_RT_TEOS](#), [calDelt_RTP_GL](#), [calDelt_RTP_TEOS](#), [calDelt_CONST](#)).

Referenced by calDelt_R_GL(), calDelt_R_TEOS(), calDelt_RT_GL(), calDelt_RT_TEOS(), calDelt_RTP_GL(), calDelt_RTP_TEOS(), calNewEddyVisc_R_CN(), calNewEddyVisc_RT_CN(), calNewEddyVisc_RTP_CN(), calOldEddyVisc_R_CN(), calOldEddyVisc_RT_CN(), calOldEddyVisc_RTP_CN(), fin(), initDonorFracAndMaxConVel_R_GL(), initDonorFracAndMaxConVel_R_TEOS(), initDonorFracAndMaxConVel_RT_GL(), initDonorFracAndMaxConVel_RT_TEOS(), initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), main(), and Parameters().

6.8.3.18 double Parameters::dMaxConvectiveVelocity_c

Holds the maximum of convective velocity divided by the sound speed. It is set in the functions which calculate the timestep (see [calDelt_R_GL](#), [calDelt_R_TEOS](#), [calDelt_RT_GL](#), [calDelt_RT_TEOS](#), [calDelt_RTP_GL](#), [calDelt_RTP_TEOS](#), [calDelt_CONST](#)).

Referenced by calDelt_R_GL(), calDelt_R_TEOS(), calDelt_RT_GL(), calDelt_RT_TEOS(), calDelt_RTP_GL(), calDelt_RTP_TEOS(), initDonorFracAndMaxConVel_R_GL(), initDonorFracAndMaxConVel_R_TEOS(), initDonorFracAndMaxConVel_RT_GL(), initDonorFracAndMaxConVel_RT_TEOS(), initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), and Parameters().

6.8.3.19 double Parameters::dPrt

This is the value of the Prandtl number, a value of 0.7 is what is suggested by Lawrence D. Cloutman in "The LUVd11 Large Eddy Simulation Model" April 15, 1991 a Lawrence Livermore National Labratory report.

Referenced by calNewE_RT_NA_LES(), calNewE_RTP_NA_LES(), dImplicitEnergyFunction_RT_LES(), dImplicitEnergyFunction_RT_LES_SB(),

`dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, and `Parameters()`.

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

6.9 Performance Class Reference

```
#include <global.h>
```

Public Member Functions

- [Performance](#) ()

Public Attributes

- double [dStartTimer](#)
- double [dEndTimer](#)

6.9.1 Detailed Description

This class manages information pertaining to performance analysis of the code.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Performance::Performance ()

Constructor for the class [Performance](#).

References [dEndTimer](#), and [dStartTimer](#).

6.9.3 Member Data Documentation

6.9.3.1 double Performance::dStartTimer

The time that the code timer was started.

Referenced by [fin\(\)](#), [init\(\)](#), and [Performance\(\)](#).

6.9.3.2 double Performance::dEndTimer

The time that the code timer was ended. The difference between [Performance::dStartTimer](#) and [dEndTimer](#) gives the total run time

Referenced by [fin\(\)](#), and [Performance\(\)](#).

The documentation for this class was generated from the following files:

- [global.h](#)

- [global.cpp](#)

6.10 ProcTop Class Reference

```
#include <global.h>
```

Public Member Functions

- [ProcTop](#) ()

Public Attributes

- int [nNumProcs](#)
- int * [nProcDims](#)
- int * [nPeriodic](#)
- int ** [nCoords](#)
- int [nRank](#)
- int [nNumNeighbors](#)
- int * [nNeighborRanks](#)
- int [nNumRadialNeighbors](#)
- int * [nRadialNeighborRanks](#)
- int * [nRadialNeighborNeighborIDs](#)

6.10.1 Detailed Description

This class manages information which pertains to the processor topology.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 ProcTop::ProcTop ()

Constructor for class [ProcTop](#).

References [nCoords](#), [nNeighborRanks](#), [nNumNeighbors](#), [nNumRadialNeighbors](#), [nPeriodic](#), [nProcDims](#), [nRadialNeighborNeighborIDs](#), and [nRadialNeighborRanks](#).

6.10.3 Member Data Documentation

6.10.3.1 int ProcTop::nNumProcs

Number of processors in global communicator MPI::COMM_WORLD. The value of this variable is independent of processor [ProcTop::nRank](#).

Referenced by [init\(\)](#), [initImplicitCalculation\(\)](#), [initUpdateLocalBoundaries\(\)](#), [initWatchZones\(\)](#), [modelRead\(\)](#), and [setupLocalGrid\(\)](#).

6.10.3.2 `int* ProcTop::nProcDims`

Dimensions of the processor topology. It is an array of size 3 to hold the size of the processor grid in each dimension. The value of this variable is set in the configuration file "config.xml" which is parsed by the function [init](#). The values of this variable are independent of processor [ProcTop::nRank](#).

Referenced by [init\(\)](#), [initImplicitCalculation\(\)](#), [initUpdateLocalBoundaries\(\)](#), [initWatchZones\(\)](#), [modelRead\(\)](#), [modelWrite_GL\(\)](#), [modelWrite_TEOS\(\)](#), [ProcTop\(\)](#), and [setupLocalGrid\(\)](#).

6.10.3.3 `int* ProcTop::nPeriodic`

Periodic boundary conditions. It is an array of size 3 to tell if a dimension is periodic (wraps) or not. It contains an interger value of 0 or 1. 0, the boundary condition is not periodic, 1 the boundary condition is periodic. The value of this variable is set in the configuration file "config.xml" which is parsed by the function [init](#). The values of this variable are independent of processor [ProcTop::nRank](#).

Referenced by [initUpdateLocalBoundaries\(\)](#), [modelRead\(\)](#), [modelWrite_GL\(\)](#), [modelWrite_TEOS\(\)](#), [ProcTop\(\)](#), and [setupLocalGrid\(\)](#).

6.10.3.4 `int** ProcTop::nCoords`

Coordinates of the processors. It is of size [ProcTop::nNumProcs](#) by 3. The values of this variable are independent of processor [ProcTop::nRank](#).

Referenced by [calNewU0_R\(\)](#), [calNewU0_RT\(\)](#), [calNewU0_RTP\(\)](#), [initImplicitCalculation\(\)](#), [initUpdateLocalBoundaries\(\)](#), [initWatchZones\(\)](#), [modelRead\(\)](#), [modelWrite_GL\(\)](#), [modelWrite_TEOS\(\)](#), [ProcTop\(\)](#), and [setupLocalGrid\(\)](#).

6.10.3.5 `int ProcTop::nRank`

Is a unique integer which identifies the processor. The values of [ProcTop::nRank](#) range from 0 to [ProcTop::nNumProcs](#)-1 depending on the processor.

Referenced by [calDelt_R_GL\(\)](#), [calDelt_R_TEOS\(\)](#), [calDelt_RT_GL\(\)](#), [calDelt_RT_TEOS\(\)](#), [calDelt_RTP_GL\(\)](#), [calDelt_RTP_TEOS\(\)](#), [calNewD_R\(\)](#), [calNewD_RT\(\)](#), [calNewD_RTP\(\)](#), [calNewE_R_AD\(\)](#), [calNewE_R_NA\(\)](#), [calNewE_R_NA_LES\(\)](#), [calNewE_RT_AD\(\)](#), [calNewE_RT_NA\(\)](#), [calNewE_RT_NA_LES\(\)](#), [calNewE_RTP_AD\(\)](#), [calNewE_RTP_NA\(\)](#), [calNewE_RTP_NA_LES\(\)](#), [calNewU0_R\(\)](#), [calNewU0_RT\(\)](#), [calNewU0_RTP\(\)](#), [fin\(\)](#), [implicitSolve_R\(\)](#), [implicitSolve_RT\(\)](#), [implicitSolve_RTP\(\)](#), [init\(\)](#), [initImplicitCalculation\(\)](#), [initInternalVars\(\)](#), [initUpdateLocalBoundaries\(\)](#), [initWatchZones\(\)](#), [main\(\)](#), [modelRead\(\)](#), [modelWrite_GL\(\)](#), [modelWrite_TEOS\(\)](#), [setMainFunctions\(\)](#), [setupLocalGrid\(\)](#), [updateLocalBoundaries\(\)](#), [updateLocalBoundariesNewGrid\(\)](#), and [updateNewGridWithOld\(\)](#).

6.10.3.6 `int ProcTop::nNumNeighbors`

The number of neighbors surrounding the current processor. The maximum number of neighbors possible is 27, 3x3x3 don't forget the current processor itself can be its own neighbor because of periodic boundary conditions. The value of this variable is dependent on processor [ProcTop::nRank](#).

Referenced by [initUpdateLocalBoundaries\(\)](#), [ProcTop\(\)](#), [updateLocalBoundaries\(\)](#), and [updateLocalBoundariesNewGrid\(\)](#).

6.10.3.7 int* ProcTop::nNeighborRanks

[ProcTop::nRank](#) s of the neighboring processors. An array of size [nNumNeighbors](#) to hold ranks of neighbouring processors.

Referenced by [initUpdateLocalBoundaries\(\)](#), [ProcTop\(\)](#), [updateLocalBoundaries\(\)](#), and [updateLocalBoundariesNewGrid\(\)](#).

6.10.3.8 int ProcTop::nNumRadialNeighbors

The number of neighbors in the radial direction. Can range from 1 to 2 depending on whether there is a processor beneath or above the current processor.

Referenced by [calNewU0_R\(\)](#), [calNewU0_RT\(\)](#), [calNewU0_RTP\(\)](#), [initUpdateLocalBoundaries\(\)](#), and [ProcTop\(\)](#).

6.10.3.9 int* ProcTop::nRadialNeighborRanks

[ProcTop::nRank](#) s of the neighboring radial processors. It is an array of size [ProcTop::nNumRadialNeighbors](#).

Referenced by [calNewU0_R\(\)](#), [calNewU0_RT\(\)](#), [calNewU0_RTP\(\)](#), [initUpdateLocalBoundaries\(\)](#), and [ProcTop\(\)](#).

6.10.3.10 int* ProcTop::nRadialNeighborNeighborIDs

Holds the ID of a radial neighbor, to be used to obtain their [ProcTop::nRank](#) from [ProcTop::nNeighborRanks](#)

Referenced by [calNewU0_R\(\)](#), [calNewU0_RT\(\)](#), [calNewU0_RTP\(\)](#), [initUpdateLocalBoundaries\(\)](#), and [ProcTop\(\)](#).

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

6.11 Time Class Reference

```
#include <global.h>
```

Public Member Functions

- [Time](#) ()

Public Attributes

- double [dDeltat_np1half](#)
- double [dDeltat_nm1half](#)
- double [dDeltat_n](#)
- double [dt](#)
- double [dEndTime](#)
- int [nEndTimeStep](#)
- double [dTimeStepFactor](#)
- int [nTimeStepIndex](#)
- bool [bVariableTimeStep](#)
- double [dConstTimeStep](#)
- double [dPerChange](#)
- double [dDelRho_t_Rho_max](#)
- double [dDelT_t_T_max](#)
- double [dDelE_t_E_max](#)

6.11.1 Detailed Description

This class manages information which pertains to time variables.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 [Time::Time](#) ()

Constructor for the class [Time](#).

References [dDelE_t_E_max](#), [dDelRho_t_Rho_max](#), [dDelT_t_T_max](#), [dDeltat_n](#), [dDeltat_np1half](#), [dEndTime](#), [dPerChange](#), [dt](#), [dTimeStepFactor](#), [nEndTimeStep](#), and [nTimeStepIndex](#).

6.11.3 Member Data Documentation

6.11.3.1 double Time::dDeltat_np1half

The time step centered at $n + 1/2$ in seconds. It is used for calculating new variables defined at time step n , e.g. the density [Grid::nD](#).

Referenced by `calDelt_CONST()`, `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `calNewD_R()`, `calNewD_RT()`, `calNewD_RTP()`, `calNewR()`, `fin()`, `main()`, `modelRead()`, `modelWrite_TEOS()`, and `Time()`.

6.11.3.2 double Time::dDeltat_nm1half

The previously used timestep centered at $n - 1/2$ in seconds. It is used for calculating [dDeltat_n](#) the n centered time step.

Referenced by `calDelt_CONST()`, `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `modelRead()`, `modelWrite_GL()`, and `modelWrite_TEOS()`.

6.11.3.3 double Time::dDeltat_n

The time step centered at n in seconds. It is used for calculating new variables defined at time step $n + 1/2$, e.g. the radial velocity [Grid::nU](#). This value is determined by averaging the current [Time::dDeltat_np1half](#), and the last [Time::dDeltat_np1half](#).

Referenced by `calDelt_CONST()`, `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `calNewE_R_AD()`, `calNewE_R_NA()`, `calNewE_R_NA_LES()`, `calNewE_RT_AD()`, `calNewE_RT_NA()`, `calNewE_RT_NA_LES()`, `calNewE_RTP_AD()`, `calNewE_RTP_NA()`, `calNewE_RTP_NA_LES()`, `calNewU_R()`, `calNewU_R_LES()`, `calNewU_RT()`, `calNewU_RT_LES()`, `calNewU_RTP()`, `calNewU_RTP_LES()`, `calNewV_RT()`, `calNewV_RT_LES()`, `calNewV_RTP()`, `calNewV_RTP_LES()`, `calNewW_RTP()`, `calNewW_RTP_LES()`, `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_LES()`, `dImplicitEnergyFunction_R_LES_SB()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RT_SB()`, `dImplicitEnergyFunction_RTP()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, `dImplicitEnergyFunction_RTP_SB()`, `modelRead()`, and `Time()`.

6.11.3.4 double Time::dt

The current time of the simulation in seconds.

Referenced by `calDelt_CONST()`, `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `fin()`, `init()`, `main()`, `modelRead()`, `modelWrite_GL()`, `modelWrite_TEOS()`, `Time()`, `writeWatchZones_R_GL()`, `writeWatchZones_R_TEOS()`, `writeWatchZones_RT_GL()`, `writeWatchZones_RT_TEOS()`, `writeWatchZones_RTP_GL()`, and `writeWatchZones_RTP_TEOS()`.

6.11.3.5 double Time::dEndTime

The end time of the current calculation in seconds.

Referenced by `init()`, `main()`, and `Time()`.

6.11.3.6 `int Time::nEndTimeStep`

The last time step to calculate, will stop if the current time step is larger than this. The default value is the largest integer of the system.

Referenced by `init()`, `main()`, and `Time()`.

6.11.3.7 `double Time::dTimeStepFactor`

Used for determining the time step. It is the factor which the courant time step is multiplied by in order to determine `Time::dDeltat_nplhalf`.

Referenced by `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `init()`, and `Time()`.

6.11.3.8 `int Time::nTimeStepIndex`

An index indicating the current time step. An index of zero corresponds to a `Time::dt=0`.

Referenced by `calDelt_CONST()`, `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `fin()`, `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, `initWatchZones()`, `main()`, `modelRead()`, `modelWrite_GL()`, `modelWrite_TEOS()`, `Time()`, `writeWatchZones_R_GL()`, `writeWatchZones_R_TEOS()`, `writeWatchZones_RT_GL()`, `writeWatchZones_RT_TEOS()`, `writeWatchZones_RTP_GL()`, and `writeWatchZones_RTP_TEOS()`.

6.11.3.9 `bool Time::bVariableTimeStep`

If true a variable time step is used as specified by the Courant condition, times the `dTimeStepFactor`.

Referenced by `init()`, and `setMainFunctions()`.

6.11.3.10 `double Time::dConstTimeStep`

If set to a value other than 0, will use that constant time step in place of the courant time step.

Referenced by `calDelt_CONST()`, and `init()`.

6.11.3.11 `double Time::dPerChange`

A percentage amount to allow the maximum horizontal temperature variation and radial, theta and phi convective velocities to change by from one time step to the next. The time step is reduced accordingly to keep this percent change intact.

Referenced by `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `init()`, and `Time()`.

6.11.3.12 `double Time::dDelRho_t_Rho_max`

Keeps track of the maximum relative change in density from one time step to the next.

Referenced by `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `fin()`, `main()`, and `Time()`.

6.11.3.13 `double Time::dDelT_t_T_max`

Keeps track of the maximum relative change in temperature from one time step to the next. This quantity is only tracked if the calculation is non-adiabatic, else the energy is tracked instead, see

[Time::dDelE_t_E_max](#)

Referenced by `calDelt_R_TEOS()`, `calDelt_RT_TEOS()`, `calDelt_RTP_TEOS()`, `fin()`, `main()`, and `Time()`.

6.11.3.14 `double Time::dDelE_t_E_max`

Keeps track of the maximum relative change in energy from one time step to the next. This quantity is only tracked if the calculation is adiabatic, else the temperature is tracked instead, see

[Time::dDelT_t_T_max](#)

Referenced by `calDelt_R_GL()`, `calDelt_RT_GL()`, `calDelt_RTP_GL()`, and `Time()`.

The documentation for this class was generated from the following files:

- [global.h](#)
- [global.cpp](#)

6.12 WatchZone Class Reference

```
#include <watchzone.h>
```

6.12.1 Detailed Description

This class contains information used to monitor a particular zone of the grid.

The documentation for this class was generated from the following files:

- [watchzone.h](#)
- [watchzone.cpp](#)

Chapter 7

File Documentation

7.1 `/home/cgeroux/SPHERLS/src/eos.cpp` File Reference

```
#include <string>
#include <fstream>
#include <sstream>
#include <iostream>
#include <cmath>
#include "eos.h"
#include "exception2.h"
```

7.1.1 Detailed Description

Implements the [eos](#) (equation of state) class defined in [eos.h](#)

7.2 /home/cgeroux/SPHERLS/src/eos.h File Reference

```
#include <string>
#include "exception2.h"
```

Classes

- class [eos](#)

7.2.1 Detailed Description

Header file for [eos.cpp](#)

7.3 dataManipulation.cpp File Reference

```
#include <cmath>
#include <sstream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <fenv.h>
#include "dataManipulation.h"
#include "global.h"
#include "xmlFunctions.h"
#include "exception2.h"
#include "dataMonitoring.h"
#include "physEquations.h"
#include <string>
```

Functions

- void [init](#) ([ProcTop](#) &procTop, [Grid](#) &grid, [Output](#) &output, [Time](#) &time, [Parameters](#) ¶meters, [MessPass](#) &messPass, [Performance](#) &performance, [Implicit](#) &implicit, int nNumArgs, char *cArgs[])
- void [setupLocalGrid](#) ([ProcTop](#) &procTop, [Grid](#) &grid)
- void [fin](#) (bool bWriteCurrentStateToFile, [Time](#) &time, [Output](#) &output, [ProcTop](#) &procTop, [Grid](#) &grid, [Parameters](#) ¶meters, [Functions](#) &functions, [Performance](#) &performance, [Implicit](#) &implicit)
- void [modelWrite_GL](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) ¶meters)
- void [modelWrite_TEOS](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) ¶meters)
- void [modelRead](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) ¶meters)
- void [initUpdateLocalBoundaries](#) ([ProcTop](#) &procTop, [Grid](#) &grid, [MessPass](#) &messPass, [Implicit](#) &implicit)
- void [updateLocalBoundaries](#) ([ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)
- void [updateLocalBoundariesNewGrid](#) (int nVar, [ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)
- void [updateOldGrid](#) ([ProcTop](#) &procTop, [Grid](#) &grid)
- void [updateNewGridWithOld](#) ([Grid](#) &grid, [ProcTop](#) &procTop)
- void [average3DTo1DBoundariesOld](#) ([Grid](#) &grid)
- void [average3DTo1DBoundariesNew](#) ([Grid](#) &grid, int nVar)
- void [updateLocalBoundaryVelocitiesNewGrid_R](#) ([ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)

- void [updateLocalBoundaryVelocitiesNewGrid_RT](#) ([ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)
- void [updateLocalBoundaryVelocitiesNewGrid_RTP](#) ([ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)
- void [initImplicitCalculation](#) ([Implicit](#) &implicit, [Grid](#) &grid, [ProcTop](#) &procTop, int nNumArgs, char *cArgs[])

7.3.1 Detailed Description

This file holds functions for manipulating data. This includes initializing the program, parsing the configuration file "config.xml", allocating memory for the model to be read in, reading in the input model, etc.

7.3.2 Function Documentation

7.3.2.1 void [average3DTo1DBoundariesNew](#) ([Grid](#) & *grid*, int *nVar*)

This function averages the 3D boundary recieved by the 1D processor ([ProcTop::nRank](#) ==0) into 1D. This average is volume weighted. This function only needs to be called by the 1D processor, and if called by other processors may have unexpected results. This function calculates the average from the new grid, and places the average into new old grid. It does so for only the specified variable. This function is used every time the grid boundaries are updated with [updateLocalBoundariesNewGrid](#).

Parameters:

- ↔ *grid* supplies the information for calculating the averages and recieves the averages.
- ← *nVar* index of the variable to be averaged with in the grid.

References [Grid::dLocalGridNew](#), [Grid::dLocalGridOld](#), [Grid::nCenIntOffset](#),
[Grid::nDCosThetaIJK](#), [Grid::nDPhi](#), [Grid::nEndGhostUpdateExplicit](#), [Grid::nR](#),
[Grid::nStartGhostUpdateExplicit](#), and [Grid::nVariables](#).

Referenced by [updateLocalBoundariesNewGrid\(\)](#).

7.3.2.2 void [average3DTo1DBoundariesOld](#) ([Grid](#) & *grid*)

This function averages the 3D boundary recieved by the 1D processor ([ProcTop::nRank](#) ==0) into 1D. This average is volume weighted. This function only needs to be called by the 1D processor, and if called by other processors may have unexpected results. This function calculates the average from the old grid, and places the average into the old grid. It does so for all variables external and internal. This function is used every time the grid boundaries are updated with [updateLocalBoundaries](#).

Parameters:

- ↔ *grid* supplies the information for calculating the averages and recieves the averages.

References `Grid::dLocalGridOld`, `Grid::nCenIntOffset`, `Grid::nDCosThetaIJK`, `Grid::nDPhi`, `Grid::nEndGhostUpdateExplicit`, `Grid::nNumIntVars`, `Grid::nNumVars`, `Grid::nR`, `Grid::nStartGhostUpdateExplicit`, and `Grid::nVariables`.

Referenced by `updateLocalBoundaries()`.

7.3.2.3 `void fin (bool bWriteCurrentStateToFile, Time & time, Output & output, ProcTop & procTop, Grid & grid, Parameters & parameters, Functions & functions, Performance & performance, Implicit & implicit)`

Finishes program execution by writing out last grid state, closing output files, and writing out run time.

Parameters:

- ← ***bWriteCurrentStateToFile*** is a bool value which indicates wheather or not to write out current model state.
- ← ***time***
- ← ***output***
- ← ***procTop***
- ← ***grid***
- ← ***parameters***
- ← ***functions***
- ← ***performance***
- ← ***implicit***

References `Implicit::dAverageRHS`, `Implicit::dCurrentRelTError`, `Time::dDelRho_t_Rho_max`, `Time::dDelT_t_T_max`, `Time::dDeltat_np1half`, `Time::dDelUmU0_t_UmU0_max`, `Time::dDelV_t_V_max`, `Time::dDelW_t_W_max`, `Performance::dEndTimer`, `Parameters::dMaxConvectiveVelocity`, `Implicit::dMaxErrorInRHS`, `Performance::dStartTimer`, `Time::dt`, `finWatchZones()`, `Functions::fpModelWrite`, `Implicit::nCurrentNumIterations`, `Implicit::nMaxNumSolverIterations`, `Grid::nNumDims`, `Implicit::nNumImplicitZones`, `Output::nNumTimeStepsSinceLastPrint`, `Output::nPrintMode`, `ProcTop::nRank`, `Time::nTimeStepIndex`, and `Output::sBaseOutputFileName`.

Referenced by `main()`.

7.3.2.4 `void init (ProcTop & procTop, Grid & grid, Output & output, Time & time, Parameters & parameters, MessPass & messPass, Performance & performance, Implicit & implicit, int argc, char * argv[])`

Initializes the program. It does this by reading a number of configuration options from the config file "SPHERLS.xml". It also reads in the starting model, as specified in the "SPHERLS.xml" file, using the function `modelRead`. During the reading of the initial model the `modelRead` function also calls `setupLocalGrid` to determine the sizes of the local grids and allocate memory for them.

Other things of note that are done in this function are:

- the calculation timer is started, `Performance::dStartTimer`
- It also reads in the equation of state table if using a tabulated equation of state (`Parameters::bEOSGammaLaw = false`) by calling `eos::readBin`

- Initializes the watchZones, i.e. figure out which processors have which watch zones, opens the files and prints headers.

Parameters:

- **procTop** all parts of this stucture are set, and do not change throughout the rest of the calculation.
- **grid** through the function `modelRead` the function `setupLocalGrid` is called to allocate memory for the grid, and set sizes of it.
- **output**
- **time**
- **parameters**
- **messPass**
- **performance**
- **implicit**
- ← **argc**
- ← **argv**

References Parameters::bAdiabatic, Output::bDump, Parameters::bEOSGammaLaw, Output::bPrint, Time::bVariableTimeStep, Parameters::dA, Parameters::dAlphaExtra, Parameters::dAVThreshold, Time::dConstTimeStep, Implicit::dDerivativeStepFraction, Output::dDumpFrequencyTime, Parameters::dEddyViscosity, Time::dEndTime, Time::dPerChange, Output::dPrintFrequencyTime, Performance::dStartTimer, Time::dt, Output::dTimeLastDump, Output::dTimeLastPrint, Time::dTimeStepFactor, Implicit::dTolerance, Parameters::dTolerance, Parameters::eosTable, initImplicitCalculation(), initInternalVars(), initUpdateLocalBoundaries(), initWatchZones(), modelRead(), Output::nDumpFrequencyStep, Time::nEndTimeStep, Grid::nGlobalGridDims, Parameters::nMaxIterations, Implicit::nMaxNumIterations, Implicit::nNumImplicitZones, ProcTop::nNumProcs, Output::nPrintFrequencyStep, Output::nPrintMode, ProcTop::nProcDims, ProcTop::nRank, Parameters::nTypeTurbulenceMod, eos::readBin(), Output::sBaseOutputFileName, and Parameters::sEOSFileName.

Referenced by main().

7.3.2.5 void initImplicitCalculation (Implicit & *implicit*, Grid & *grid*, ProcTop & *procTop*, int *nNumArgs*, char * *cArgs*[])

This function initializes data structures and defines indexes of non-zero elements in the coefficient matrix. It also sets up pathways for collection of the temperature corrections back to the processors which need them for their local grids.

Parameters:

- ↔ **implicit**
- ← **grid** size information of the grid is used
- ← **procTop**
- ← **nNumArgs** number of command line arguments, PETSc wants them
- ← **cArgs** a list of command line arguments, PETSc wants them

Todo

isFrom, isTo, matCoeff, vecTCorrections, vecTCorrections, vecRHS, vecTCorrectionsLocal, kspContext, vecscatTCorrections all need to be destroyed before program finishes.

References `Implicit::dTolerance`, `Implicit::kspContext`, `Implicit::matCoeff`, `ProcTop::nCoords`, `Grid::nGlobalGridDims`, `Grid::nLocalGridDims`, `Implicit::nLocDer`, `Implicit::nLocFun`, `Implicit::nMaxNumIterations`, `Implicit::nNumDerPerRow`, `Grid::nNumDims`, `Grid::nNumGhostCells`, `Implicit::nNumImplicitZones`, `ProcTop::nNumProcs`, `Implicit::nNumRowsALocal`, `Implicit::nNumRowsALocalSB`, `ProcTop::nProcDims`, `ProcTop::nRank`, `Grid::nT`, `Implicit::nTypeDer`, `Implicit::vecRHS`, `Implicit::vecscatTCorrections`, `Implicit::vecTCorrections`, and `Implicit::vecTCorrectionsLocal`.

Referenced by `init()`.

7.3.2.6 void initUpdateLocalBoundaries (ProcTop & *procTop*, Grid & *grid*, MessPass & *messPass*, Implicit & *implicit*)

Sets up MPI derived data types used for updating the local grid boundaries between processors. It sets where the local grids should start/stop updating the local grids (`Grid::nStartUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nStartUpdateImplicit`, `Grid::nEndUpdateImplicit`, `Grid::nStartGhostUpdateExplicit`, `Grid::nEndGhostUpdateExplicit`, `Grid::nStartGhostUpdateImplicit`, `Grid::nEndGhostUpdateImplicit`). It sets the radial processor neighbors (`ProcTop::nNumRadialNeighbors`).

It also allocates memory for:

- `MessPass::requestSend`
- `MessPass::requestRecv`
- `statusSend`
- `statusRecv`

Parameters:

↔ *procTop*

↔ *grid*

↔ *messPass*

↔ *implicit*

References `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Grid::nCenIntOffset`, `ProcTop::nCoords`, `Grid::nE`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndGhostUpdateImplicit`, `Grid::nEndUpdateExplicit`, `Grid::nEndUpdateImplicit`, `Grid::nGlobalGridDims`, `Grid::nKappa`, `Grid::nLocalGridDims`, `ProcTop::nNeighborRanks`, `Grid::nNum1DZones`, `Grid::nNumGhostCells`, `Implicit::nNumImplicitZones`, `Grid::nNumIntVars`, `ProcTop::nNumNeighbors`, `ProcTop::nNumProcs`, `ProcTop::nNumRadialNeighbors`, `Grid::nNumVars`, `Grid::nP`, `ProcTop::nPeriodic`, `ProcTop::nProcDims`, `ProcTop::nRadialNeighborNeighborIDs`, `ProcTop::nRadialNeighborRanks`, `ProcTop::nRank`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartGhostUpdateImplicit`, `Grid::nStartUpdateExplicit`, `Grid::nStartUpdateImplicit`, `Grid::nT`, `Grid::nVariables`, `MessPass::requestRecv`, `MessPass::requestSend`, `MessPass::statusRecv`, `MessPass::statusSend`, `MessPass::typeRecvNewVar`, `MessPass::typeRecvOldGrid`, `MessPass::typeSendNewGrid`, and `MessPass::typeSendNewVar`.

Referenced by `init()`.

7.3.2.7 void modelRead (std::string *sFileName*, ProcTop & *procTop*, Grid & *grid*, Time & *time*, Parameters & *parameters*)

Reads in a collected binary file into the local grid and calls [setupLocalGrid](#) to allocate memory and set various parameters of the model. Works for both gamma-law gas, and tabulated equation of state models.

Parameters:

- ← *sFileName* name of the file containing the model to be read in
- *procTop*
- *grid*
- *time*
- *parameters*

Todo

At some point should get it working with only 1 processor

References Parameters::bEOSGammaLaw, Parameters::dA, Parameters::dAlpha, Parameters::dAVThreshold, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Parameters::dGamma, Grid::dLocalGridOld, Time::dt, DUMP_VERSION, ProcTop::nCoords, Grid::nCotThetaIJK, Grid::nCotThetaIjp1halfK, Grid::nD, Grid::nDCosThetaIJK, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nGamma, Grid::nGlobalGridDims, Grid::nKappa, Grid::nLocalGridDims, Grid::nM, Grid::nNum1DZones, Grid::nNumDims, Grid::nNumGhostCells, Grid::nNumIntVars, ProcTop::nNumProcs, Grid::nNumVars, Grid::nP, ProcTop::nPeriodic, Grid::nPhi, ProcTop::nProcDims, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nT, Grid::nTheta, Time::nTimeStepIndex, Parameters::nTypeTurbulenceMod, Grid::nU, Grid::nU0, Grid::nV, Grid::nVariables, Grid::nW, Parameters::sEOSFileName, setInternalVarInf(), and setupLocalGrid().

Referenced by init().

7.3.2.8 void modelWrite_GL (std::string *sFileName*, ProcTop & *procTop*, Grid & *grid*, Time & *time*, Parameters & *parameters*)

Writes out a model in distributed model format, meaning that each processor writes its own local grid to a file in binary format. They can be combined, and or converted to ascii format using SPHERLSanal. This is for a gamma-law gas model.

Parameters:

- ← *sFileName* base name of the output files
- ← *procTop*
- ← *grid*
- ← *time*
- ← *parameters*

References Parameters::dA, Parameters::dAlpha, Parameters::dAVThreshold, Time::dDeltat_nm1half, Parameters::dGamma, Grid::dLocalGridOld, Time::dt, DUMP_VERSION, ProcTop::nCoords, Grid::nGlobalGridDims, Grid::nLocalGridDims, Grid::nNum1DZones,

Grid::nNumGhostCells, Grid::nNumVars, ProcTop::nPeriodic, ProcTop::nProcDims, ProcTop::nRank, Time::nTimeStepIndex, and Grid::nVariables.

Referenced by setMainFunctions().

7.3.2.9 void modelWrite_TEOS (std::string *sFileName*, ProcTop & *procTop*, Grid & *grid*, Time & *time*, Parameters & *parameters*)

Writes out a model in distributed model format, meaning that each processor writes its own local grid to a file in binary format. They can be combined, and or converted to ascii format using SPHERLSanal. This is for a tabulated equation of state model.

Parameters:

- ← *sFileName* base name of the output files
- ← *procTop*
- ← *grid*
- ← *time*
- ← *parameters*

References Parameters::dA, Parameters::dAlpha, Parameters::dAVThreshold, Time::dDeltat_nm1half, Time::dDeltat_np1half, Grid::dLocalGridOld, Time::dt, DUMP_VERSION, ProcTop::nCoords, Grid::nGlobalGridDims, Grid::nLocalGridDims, Grid::nNum1DZones, Grid::nNumGhostCells, Grid::nNumVars, ProcTop::nPeriodic, ProcTop::nProcDims, ProcTop::nRank, Time::nTimeStepIndex, Grid::nVariables, and Parameters::sEOSFileName.

Referenced by setMainFunctions().

7.3.2.10 void setupLocalGrid (ProcTop & *procTop*, Grid & *grid*)

Determines size of local grids (Grid::nLocalGridDims) based on processor topology, and allocates memory for the local grids (Grid::dLocalGridNew, Grid::dLocalGridOld). It sets various other quantities as well such as,

- the coordinates of all processors (ProcTop::nCoords)
- the offset for interface centered quantities (Grid::nCenIntOffset, which depends on zoning and boundary conditions)
- the position the local grid is in relative to the global grid (Grid::nGlobalGridPositionLocalGrid).

Parameters:

- ↔ *procTop* contains information about the processor topology
- ↔ *grid* contains information about grid

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, ProcTop::nCoords, Grid::nD, Grid::nGlobalGridDims, Grid::nGlobalGridPositionLocalGrid, Grid::nLocalGridDims, Grid::nNum1DZones, Grid::nNumDims, Grid::nNumGhostCells, Grid::nNumIntVars, ProcTop::nNumProcs, Grid::nNumVars, ProcTop::nPeriodic, ProcTop::nProcDims, ProcTop::nRank, and Grid::nVariables.

Referenced by modelRead().

7.3.2.11 void updateLocalBoundaries (ProcTop & *procTop*, MessPass & *messPass*, Grid & *grid*)

Updates the boundaries of the local grids from the data in the local grids of other processors. It does this for all variables and updates to the old grid. It also has processor `ProcTop::nRank=0` call `average3DTo1DBoundariesOld` which averages the 3D information into the 1D boundaries.

Parameters:

← *procTop*
 ← *messPass*
 ↔ *grid*

Todo

Shouldn't need `MPI::COMM_WORLD.Barrier()` may want to test out removing this at some point as it might produce a bit of a speed up.

References `average3DTo1DBoundariesOld()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `ProcTop::nNeighborRanks`, `ProcTop::nNumNeighbors`, `ProcTop::nRank`, `MessPass::requestRecv`, `MessPass::requestSend`, `MessPass::statusRecv`, `MessPass::statusSend`, `MessPass::typeRecvOldGrid`, `MessPass::typeSendNewGrid`, and `updateOldGrid()`.

Referenced by `main()`.

7.3.2.12 void updateLocalBoundariesNewGrid (int *nVar*, ProcTop & *procTop*, MessPass & *messPass*, Grid & *grid*)

Updates the boundaries of the local grids from the data in the local grids of other processors. It does this for a specific variable specified by `nVar` and updates to the new grid. It also has processor `ProcTop::nRank=0` call `average3DTo1DBoundariesNew` which averages the 3D information into the 1D boundaries for that specific variable.

Parameters:

← *procTop*
 ← *messPass*
 ↔ *grid*

Todo

May want to do some waiting on this message at some point before the end of the timestep, but it doesn't need to be done in this function. It might also be that this is built into the code by waiting at some other point. This is something that should be checked out at somepoint, perhaps once the preformance starts to be analyzed. I would think that if the send buffer was being modified before the send was completed, that there would be some errors popping up that would likely kill the program.

References `average3DTo1DBoundariesNew()`, `Grid::dLocalGridNew`, `ProcTop::nNeighborRanks`, `ProcTop::nNumNeighbors`, `ProcTop::nRank`, `MessPass::requestRecv`, `MessPass::statusRecv`, `MessPass::typeRecvNewVar`, and `MessPass::typeSendNewVar`.

Referenced by `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, `main()`, `updateLocalBoundaryVelocitiesNewGrid_R()`, `updateLocalBoundaryVelocitiesNewGrid_RT()`, and `updateLocalBoundaryVelocitiesNewGrid_RTP()`.

7.3.2.13 void updateLocalBoundaryVelocitiesNewGrid_R (ProcTop & *procTop*, MessPass & *messPass*, Grid & *grid*)

Updates velocity boundaries of the new grid in a 1D calculations after the velocities have been newly calculated.

References Grid::nU, and updateLocalBoundariesNewGrid().

Referenced by setMainFunctions().

7.3.2.14 void updateLocalBoundaryVelocitiesNewGrid_RT (ProcTop & *procTop*, MessPass & *messPass*, Grid & *grid*)

Updates velocity boundaries of the new grid in a 2D calculations after the velocities have been newly calculated.

References Grid::nU, Grid::nV, and updateLocalBoundariesNewGrid().

Referenced by setMainFunctions().

7.3.2.15 void updateLocalBoundaryVelocitiesNewGrid_RTP (ProcTop & *procTop*, MessPass & *messPass*, Grid & *grid*)

Updates velocity boundaries of the new grid in a 3D calculations after the velocities have been newly calculated.

References Grid::nU, Grid::nV, Grid::nW, and updateLocalBoundariesNewGrid().

Referenced by setMainFunctions().

7.3.2.16 void updateNewGridWithOld (Grid & *grid*, ProcTop & *procTop*)

Copies the contents of the old grid to the new grid including ghost cells.

Parameters:

↔ *grid*

← *procTop*

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nLocalGridDims, Grid::nNumDims, Grid::nNumGhostCells, Grid::nNumIntVars, Grid::nNumVars, ProcTop::nRank, and Grid::nVariables.

Referenced by main().

7.3.2.17 void updateOldGrid (ProcTop & *procTop*, Grid & *grid*)

Updates the old grid with the new grid, not including boundaries.

Parameters:

← *procTop*

↔ *grid*

References `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Grid::nEndGhostUpdateExplicit`,
`Grid::nEndGhostUpdateImplicit`, `Grid::nEndUpdateExplicit`, `Grid::nEndUpdateImplicit`,
`Grid::nNumIntVars`, `Grid::nNumVars`, `Grid::nStartGhostUpdateExplicit`,
`Grid::nStartGhostUpdateImplicit`, `Grid::nStartUpdateExplicit`, and `Grid::nStartUpdateImplicit`.

Referenced by `updateLocalBoundaries()`.

7.4 dataManipulation.h File Reference

```
#include <mpi.h>
#include "global.h"
```

Functions

- void [init](#) ([ProcTop](#) &procTop, [Grid](#) &grid, [Output](#) &output, [Time](#) &time, [Parameters](#) ¶meters, [MessPass](#) &messPass, [Performance](#) &performance, [Implicit](#) &implicit, int argc, char *argv[])
- void [setupLocalGrid](#) ([ProcTop](#) &procTop, [Grid](#) &grid)
- void [fin](#) (bool bWriteCurrentStateToFile, [Time](#) &time, [Output](#) &output, [ProcTop](#) &procTop, [Grid](#) &grid, [Parameters](#) ¶meters, [Functions](#) &functions, [Performance](#) &performance, [Implicit](#) &implicit)
- void [modelWrite_GL](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) ¶meters)
- void [modelWrite_TEOS](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) ¶meters)
- void [modelRead](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) ¶meters)
- void [initUpdateLocalBoundaries](#) ([ProcTop](#) &procTop, [Grid](#) &grid, [MessPass](#) &messPass, [Implicit](#) &implicit)
- void [updateLocalBoundaries](#) ([ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)
- void [updateLocalBoundariesNewGrid](#) (int nVar, [ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)
- void [updateOldGrid](#) ([ProcTop](#) &procTop, [Grid](#) &grid)
- void [updateNewGridWithOld](#) ([Grid](#) &grid, [ProcTop](#) &procTop)
- void [average3DTo1DBoundariesOld](#) ([Grid](#) &grid)
- void [average3DTo1DBoundariesNew](#) ([Grid](#) &grid, int nVar)
- void [updateLocalBoundaryVelocitiesNewGrid_R](#) ([ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)
- void [updateLocalBoundaryVelocitiesNewGrid_RT](#) ([ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)
- void [updateLocalBoundaryVelocitiesNewGrid_RTP](#) ([ProcTop](#) &procTop, [MessPass](#) &messPass, [Grid](#) &grid)
- void [initImplicitCalculation](#) ([Implicit](#) &implicit, [Grid](#) &grid, [ProcTop](#) &procTop, int nNumArgs, char *cArgs[])

7.4.1 Detailed Description

Header file for [dataManipulation.cpp](#)

7.4.2 Function Documentation

7.4.2.1 void average3DTo1DBoundariesNew (Grid & *grid*, int *nVar*)

This function averages the 3D boundary recieved by the 1D processor (`ProcTop::nRank == 0`) into 1D. This average is volume weighted. This function only needs to be called by the 1D processor, and if called by other processors may have unexpected results. This function calculates the average from the new grid, and places the average into new old grid. It does so for only the specified variable. This function is used every time the grid boundaries are updated with `updateLocalBoundariesNewGrid`.

Parameters:

- ↔ ***grid*** supplies the information for calculating the averages and recieves the averages.
- ← ***nVar*** index of the variable to be averaged with in the grid.

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset,
Grid::nDCosThetaIJK, Grid::nDPhi, Grid::nEndGhostUpdateExplicit, Grid::nR,
Grid::nStartGhostUpdateExplicit, and Grid::nVariables.

Referenced by `updateLocalBoundariesNewGrid()`.

7.4.2.2 void average3DTo1DBoundariesOld (Grid & *grid*)

This function averages the 3D boundary recieved by the 1D processor (`ProcTop::nRank == 0`) into 1D. This average is volume weighted. This function only needs to be called by the 1D processor, and if called by other processors may have unexpected results. This function calculates the average from the old grid, and places the average into the old grid. It does so for all variables external and internal. This function is used every time the grid boundaries are updated with `updateLocalBoundaries`.

Parameters:

- ↔ ***grid*** supplies the information for calculating the averages and recieves the averages.

References Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nDCosThetaIJK, Grid::nDPhi,
Grid::nEndGhostUpdateExplicit, Grid::nNumIntVars, Grid::nNumVars, Grid::nR,
Grid::nStartGhostUpdateExplicit, and Grid::nVariables.

Referenced by `updateLocalBoundaries()`.

7.4.2.3 void fin (bool *bWriteCurrentStateToFile*, Time & *time*, Output & *output*, ProcTop & *procTop*, Grid & *grid*, Parameters & *parameters*, Functions & *functions*, Performance & *performance*, Implicit & *implicit*)

Finishes program execution by writing out last grid state, closing output files, and writting out run time.

Parameters:

- ← ***bWriteCurrentStateToFile*** is a bool value which indicates wheather or not to write out current model state.
- ← ***time***
- ← ***output***
- ← ***procTop***

← *grid*
 ← *parameters*
 ← *functions*
 ← *performance*
 ← *implicit*

References Implicit::dAverageRHS, Implicit::dCurrentRelTError, Time::dDelRho_t_Rho_max, Time::dDelT_t_T_max, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Performance::dEndTimer, Parameters::dMaxConvectiveVelocity, Implicit::dMaxErrorInRHS, Performance::dStartTimer, Time::dt, finWatchZones(), Functions::fpModelWrite, Implicit::nCurrentNumIterations, Implicit::nMaxNumSolverIterations, Grid::nNumDims, Implicit::nNumImplicitZones, Output::nNumTimeStepsSinceLastPrint, Output::nPrintMode, ProcTop::nRank, Time::nTimeStepIndex, and Output::sBaseOutputFileName.

Referenced by main().

7.4.2.4 void init (ProcTop & *procTop*, Grid & *grid*, Output & *output*, Time & *time*, Parameters & *parameters*, MessPass & *messPass*, Performance & *performance*, Implicit & *implicit*, int *argc*, char * *argv*[])

Initializes the program. It does this by reading a number of configuration options from the config file "SPHERLS.xml". It also reads in the starting model, as specified in the "SPHERLS.xml" file, using the function [modelRead](#). During the reading of the initial model the [modelRead](#) function also calls [setupLocalGrid](#) to determine the sizes of the local grids and allocate memory for them.

Other things of note that are done in this function are:

- the calculation timer is started, [Performance::dStartTimer](#)
- It also reads in the equation of state table if using a tabulated equation of state ([Parameters::bEOSGammaLaw](#) = false) by calling [eos::readBin](#)
- Initializes the watchZones, i.e. figure out which processors have which watch zones, opens the files and prints headers.

Parameters:

→ *procTop* all parts of this structure are set, and do not change throughout the rest of the calculation.
 → *grid* through the function [modelRead](#) the function [setupLocalGrid](#) is called to allocate memory for the grid, and set sizes of it.
 → *output*
 → *time*
 → *parameters*
 → *messPass*
 → *performance*
 → *implicit*
 ← *argc*
 ← *argv*

References Parameters::bAdiabatic, Output::bDump, Parameters::bEOSGammaLaw, Output::bPrint, Time::bVariableTimeStep, Parameters::dA, Parameters::dAlphaExtra, Parameters::dAVThreshold, Time::dConstTimeStep, Implicit::dDerivativeStepFraction, Output::dDumpFrequencyTime, Parameters::dEddyViscosity, Time::dEndTime, Time::dPerChange, Output::dPrintFrequencyTime, Performance::dStartTimer, Time::dt, Output::dTimeLastDump, Output::dTimeLastPrint, Time::dTimeStepFactor, Implicit::dTolerance, Parameters::dTolerance, Parameters::eosTable, initImplicitCalculation(), initInternalVars(), initUpdateLocalBoundaries(), initWatchZones(), modelRead(), Output::nDumpFrequencyStep, Time::nEndTimeStep, Grid::nGlobalGridDims, Parameters::nMaxIterations, Implicit::nMaxNumIterations, Implicit::nNumImplicitZones, ProcTop::nNumProcs, Output::nPrintFrequencyStep, Output::nPrintMode, ProcTop::nProcDims, ProcTop::nRank, Parameters::nTypeTurbulenceMod, eos::readBin(), Output::sBaseOutputFileName, and Parameters::sEOSFileName.

Referenced by main().

7.4.2.5 void initImplicitCalculation (Implicit & *implicit*, Grid & *grid*, ProcTop & *procTop*, int *nNumArgs*, char * *cArgs*[])

This function initializes data structures and defines indexes of non-zero elements in the coefficient matrix. It also sets up pathways for collection of the temperature corrections back to the processors which need them for their local grids.

Parameters:

- ↔ *implicit*
- ← *grid* size information of the grid is used
- ← *procTop*
- ← *nNumArgs* number of command line arguments, PETSc wants them
- ← *cArgs* a list of command line arguments, PETSc wants them

Todo

isFrom, isTo, matCoeff, vecTCorrections, vecTCorrections, vecRHS, vecTCorrectionsLocal, kspContext, vecscatTCorrections all need to be destroyed before program finishes.

References Implicit::dTolerance, Implicit::kspContext, Implicit::matCoeff, ProcTop::nCoords, Grid::nGlobalGridDims, Grid::nLocalGridDims, Implicit::nLocDer, Implicit::nLocFun, Implicit::nMaxNumIterations, Implicit::nNumDerPerRow, Grid::nNumDims, Grid::nNumGhostCells, Implicit::nNumImplicitZones, ProcTop::nNumProcs, Implicit::nNumRowsALocal, Implicit::nNumRowsALocalSB, ProcTop::nProcDims, ProcTop::nRank, Grid::nT, Implicit::nTypeDer, Implicit::vecRHS, Implicit::vecscatTCorrections, Implicit::vecTCorrections, and Implicit::vecTCorrectionsLocal.

Referenced by init().

7.4.2.6 void initUpdateLocalBoundaries (ProcTop & *procTop*, Grid & *grid*, MessPass & *messPass*, Implicit & *implicit*)

Sets up MPI derived data types used for updating the local grid boundaries between processors. It sets where the local grids should start/stop updating the local grids (Grid::nStartUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nStartUpdateImplicit, Grid::nEndUpdateImplicit, Grid::nStartGhostUpdateExplicit, Grid::nEndGhostUpdateExplicit,

[Grid::nStartGhostUpdateImplicit](#), [Grid::nEndGhostUpdateImplicit](#)). It sets the radial processor neighbors ([ProcTop::nNumRadialNeighbors](#)).

It also allocates memeory for:

- [MessPass::requestSend](#)
- [MessPass::requestRecv](#)
- [statusSend](#)
- [statusRecv](#)

Parameters:

- ↔ *procTop*
- ↔ *grid*
- ↔ *messPass*
- ↔ *implicit*

References [Grid::dLocalGridNew](#), [Grid::dLocalGridOld](#), [Grid::nCenIntOffset](#), [ProcTop::nCoords](#), [Grid::nE](#), [Grid::nEndGhostUpdateExplicit](#), [Grid::nEndGhostUpdateImplicit](#), [Grid::nEndUpdateExplicit](#), [Grid::nEndUpdateImplicit](#), [Grid::nGlobalGridDims](#), [Grid::nKappa](#), [Grid::nLocalGridDims](#), [ProcTop::nNeighborRanks](#), [Grid::nNum1DZones](#), [Grid::nNumGhostCells](#), [Implicit::nNumImplicitZones](#), [Grid::nNumIntVars](#), [ProcTop::nNumNeighbors](#), [ProcTop::nNumProcs](#), [ProcTop::nNumRadialNeighbors](#), [Grid::nNumVars](#), [Grid::nP](#), [ProcTop::nPeriodic](#), [ProcTop::nProcDims](#), [ProcTop::nRadialNeighborNeighborIDs](#), [ProcTop::nRadialNeighborRanks](#), [ProcTop::nRank](#), [Grid::nStartGhostUpdateExplicit](#), [Grid::nStartGhostUpdateImplicit](#), [Grid::nStartUpdateExplicit](#), [Grid::nStartUpdateImplicit](#), [Grid::nT](#), [Grid::nVariables](#), [MessPass::requestRecv](#), [MessPass::requestSend](#), [MessPass::statusRecv](#), [MessPass::statusSend](#), [MessPass::typeRecvNewVar](#), [MessPass::typeRecvOldGrid](#), [MessPass::typeSendNewGrid](#), and [MessPass::typeSendNewVar](#).

Referenced by [init\(\)](#).

7.4.2.7 void modelRead (std::string *sFileName*, ProcTop & *procTop*, Grid & *grid*, Time & *time*, Parameters & *parameters*)

Reads in a collected binary file into the local grid and calls [setupLocalGrid](#) to allocate memory and set various parameters of the model. Works for both gamma-law gas, and tabulated equation of state models.

Parameters:

- ← *sFileName* name of the file containing the model to be read in
- *procTop*
- *grid*
- *time*
- *parameters*

Todo

At some point should get it working with only 1 processor

References Parameters::bEOSGammaLaw, Parameters::dA, Parameters::dAlpha, Parameters::dAVThreshold, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Parameters::dGamma, Grid::dLocalGridOld, Time::dt, DUMP_VERSION, ProcTop::nCoords, Grid::nCotThetaIJK, Grid::nCotThetaIjp1halfK, Grid::nD, Grid::nDCosThetaIJK, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nGamma, Grid::nGlobalGridDims, Grid::nKappa, Grid::nLocalGridDims, Grid::nM, Grid::nNum1DZones, Grid::nNumDims, Grid::nNumGhostCells, Grid::nNumIntVars, ProcTop::nNumProcs, Grid::nNumVars, Grid::nP, ProcTop::nPeriodic, Grid::nPhi, ProcTop::nProcDims, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nT, Grid::nTheta, Time::nTimeStepIndex, Parameters::nTypeTurbulenceMod, Grid::nU, Grid::nU0, Grid::nV, Grid::nVariables, Grid::nW, Parameters::sEOSFileName, setInternalVarInf(), and setupLocalGrid().

Referenced by init().

7.4.2.8 void modelWrite_GL (std::string *sFileName*, ProcTop & *procTop*, Grid & *grid*, Time & *time*, Parameters & *parameters*)

Writes out a model in distributed model format, meaning that each processor writes its own local grid to a file in binary format. They can be combined, and or converted to ascii format using SPHERLSanal. This is for a gamma-law gas model.

Parameters:

- ← *sFileName* base name of the output files
- ← *procTop*
- ← *grid*
- ← *time*
- ← *parameters*

References Parameters::dA, Parameters::dAlpha, Parameters::dAVThreshold, Time::dDeltat_nm1half, Parameters::dGamma, Grid::dLocalGridOld, Time::dt, DUMP_VERSION, ProcTop::nCoords, Grid::nGlobalGridDims, Grid::nLocalGridDims, Grid::nNum1DZones, Grid::nNumGhostCells, Grid::nNumVars, ProcTop::nPeriodic, ProcTop::nProcDims, ProcTop::nRank, Time::nTimeStepIndex, and Grid::nVariables.

Referenced by setMainFunctions().

7.4.2.9 void modelWrite_TEOS (std::string *sFileName*, ProcTop & *procTop*, Grid & *grid*, Time & *time*, Parameters & *parameters*)

Writes out a model in distributed model format, meaning that each processor writes its own local grid to a file in binary format. They can be combined, and or converted to ascii format using SPHERLSanal. This is for a tabulated equation of state model.

Parameters:

- ← *sFileName* base name of the output files
- ← *procTop*
- ← *grid*
- ← *time*
- ← *parameters*

References Parameters::dA, Parameters::dAlpha, Parameters::dAVThreshold, Time::dDeltat_nmlhalf, Time::dDeltat_nplhalf, Grid::dLocalGridOld, Time::dt, DUMP_VERSION, ProcTop::nCoords, Grid::nGlobalGridDims, Grid::nLocalGridDims, Grid::nNum1DZones, Grid::nNumGhostCells, Grid::nNumVars, ProcTop::nPeriodic, ProcTop::nProcDims, ProcTop::nRank, Time::nTimeStepIndex, Grid::nVariables, and Parameters::sEOSFileName.

Referenced by setMainFunctions().

7.4.2.10 void setupLocalGrid (ProcTop & *procTop*, Grid & *grid*)

Determines size of local grids (Grid::nLocalGridDims) based on processor topology, and allocates memory for the local grids (Grid::dLocalGridNew, Grid::dLocalGridOld). It sets various other quantities aswell such as,

- the coordinates of all processors (ProcTop::nCoords)
- the offset for interface centered quantities (Grid::nCenIntOffset, which depends on zoning and boundary conditions)
- the position the local grid is in relative to the global grid (Grid::nGlobalGridPositionLocalGrid).

Parameters:

↔ *procTop* contains information about the processor topology

↔ *grid* contains information about grid

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, ProcTop::nCoords, Grid::nD, Grid::nGlobalGridDims, Grid::nGlobalGridPositionLocalGrid, Grid::nLocalGridDims, Grid::nNum1DZones, Grid::nNumDims, Grid::nNumGhostCells, Grid::nNumIntVars, ProcTop::nNumProcs, Grid::nNumVars, ProcTop::nPeriodic, ProcTop::nProcDims, ProcTop::nRank, and Grid::nVariables.

Referenced by modelRead().

7.4.2.11 void updateLocalBoundaries (ProcTop & *procTop*, MessPass & *messPass*, Grid & *grid*)

Updates the boundaries of the local grids from the data in the local grids of other processors. It does this for all variables and updates to the old grid. It also has processor ProcTop::nRank=0 call [average3DTo1DBoundariesOld](#) which averages the 3D information into the 1D boundaries.

Parameters:

← *procTop*

← *messPass*

↔ *grid*

Todo

Shouldn't need MPI::COMM_WORLD.Barrier() may want to test out removing this at some point as it might produce a bit of a speed up.

References `average3DTo1DBoundariesOld()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `ProcTop::nNeighborRanks`, `ProcTop::nNumNeighbors`, `ProcTop::nRank`, `MessPass::requestRecv`, `MessPass::requestSend`, `MessPass::statusRecv`, `MessPass::statusSend`, `MessPass::typeRecvOldGrid`, `MessPass::typeSendNewGrid`, and `updateOldGrid()`.

Referenced by `main()`.

7.4.2.12 `void updateLocalBoundariesNewGrid (int nVar, ProcTop & procTop, MessPass & messPass, Grid & grid)`

Updates the boundaries of the local grids from the data in the local grids of other processors. It does this for a specific variable specified by `nVar` and updates to the new grid. It also has processor `ProcTop::nRank=0` call `average3DTo1DBoundariesNew` which averages the 3D information into the 1D boundaries for that specific variable.

Parameters:

← *procTop*
 ← *messPass*
 ↔ *grid*

Todo

May want to do some waiting on this message at some point before the end of the timestep, but it doesn't need to be done in this function. It might also be that this is built into the code by waiting at some other point. This is something that should be checked out at somepoint, perhaps once the preformance starts to be analyzed. I would think that if the send buffer was being modified before the send was completed, that there would be some errors popping up that would likely kill the program.

References `average3DTo1DBoundariesNew()`, `Grid::dLocalGridNew`, `ProcTop::nNeighborRanks`, `ProcTop::nNumNeighbors`, `ProcTop::nRank`, `MessPass::requestRecv`, `MessPass::statusRecv`, `MessPass::typeRecvNewVar`, and `MessPass::typeSendNewVar`.

Referenced by `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, `main()`, `updateLocalBoundaryVelocitiesNewGrid_R()`, `updateLocalBoundaryVelocitiesNewGrid_RT()`, and `updateLocalBoundaryVelocitiesNewGrid_RTP()`.

7.4.2.13 `void updateLocalBoundaryVelocitiesNewGrid_R (ProcTop & procTop, MessPass & messPass, Grid & grid)`

Updates velocity boundaries of the new grid in a 1D calculations after the velocities have been newly calculated.

References `Grid::nU`, and `updateLocalBoundariesNewGrid()`.

Referenced by `setMainFunctions()`.

7.4.2.14 `void updateLocalBoundaryVelocitiesNewGrid_RT (ProcTop & procTop, MessPass & messPass, Grid & grid)`

Updates velocity boundaries of the new grid in a 2D calculations after the velocities have been newly calculated.

References `Grid::nU`, `Grid::nV`, and `updateLocalBoundariesNewGrid()`.

Referenced by setMainFunctions().

7.4.2.15 void updateLocalBoundaryVelocitiesNewGrid_RTP (ProcTop & *procTop*, MessPass & *messPass*, Grid & *grid*)

Updates velocity boundaries of the new grid in a 3D calculations after the velocities have been newly calculated.

References Grid::nU, Grid::nV, Grid::nW, and updateLocalBoundariesNewGrid().

Referenced by setMainFunctions().

7.4.2.16 void updateNewGridWithOld (Grid & *grid*, ProcTop & *procTop*)

Copies the contents of the old grid to the new grid including ghost cells.

Parameters:

↔ *grid*

← *procTop*

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nLocalGridDims, Grid::nNumDims, Grid::nNumGhostCells, Grid::nNumIntVars, Grid::nNumVars, ProcTop::nRank, and Grid::nVariables.

Referenced by main().

7.4.2.17 void updateOldGrid (ProcTop & *procTop*, Grid & *grid*)

Updates the old grid with the new grid, not including boundaries.

Parameters:

← *procTop*

↔ *grid*

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nEndGhostUpdateExplicit, Grid::nEndGhostUpdateImplicit, Grid::nEndUpdateExplicit, Grid::nEndUpdateImplicit, Grid::nNumIntVars, Grid::nNumVars, Grid::nStartGhostUpdateExplicit, Grid::nStartGhostUpdateImplicit, Grid::nStartUpdateExplicit, and Grid::nStartUpdateImplicit.

Referenced by updateLocalBoundaries().

7.5 dataMonitoring.cpp File Reference

```
#include <mpi.h>
#include <sstream>
#include <fstream>
#include <iostream>
#include <cmath>
#include <iomanip>
#include <string>
#include "watchzone.h"
#include "exception2.h"
#include "xmlFunctions.h"
#include "dataMonitoring.h"
#include "global.h"
```

Functions

- void [initWatchZones](#) (XMLNode xParent, [ProcTop](#) &procTop, [Grid](#) &grid, [Output](#) &output, [Parameters](#) ¶meters, [Time](#) &time)
- void [writeWatchZones_R_GL](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones_R_TEOS](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones_RT_GL](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones_RT_TEOS](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones_RTP_GL](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones_RTP_TEOS](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [finWatchZones](#) ([Output](#) &output)
- bool [bFileExists](#) (std::string sFilename)

7.5.1 Detailed Description

This file holds functions used for examining the grid data during execution. This includes initializing structures, handling watching zones during the execution of the program, opening files to write out the peak kinetic energy, etc.

7.5.2 Function Documentation

7.5.2.1 bool bFileExists (std::string *sFilename*)

Tests if the file exists by attempting to open the file for reading, if it fails it returns false, if it succeeds it returns true. This does not take into consideration permissions but that is ok for this project.

Parameters:

← *sFilename* file name of the file to check if it exists or not

Returns:

returns true or false depending on whether the file exists

Referenced by initWatchZones().

7.5.2.2 void finWatchZones (Output & *output*)

Closes the files opened for writing out the watchzones

Parameters:

← *output*

References Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by fin().

7.5.2.3 void initWatchZones (XMLNode *xParent*, ProcTop & *procTop*, Grid & *grid*, Output & *output*, Parameters & *parameters*, Time & *time*)

Reads in watchzones set in configuration file "SPHERLS.xml". A list is created on each processor containing the watchzones on that processor's local grid. It also opens file streams for each watchzone and writes out a header.

Parameters:

← *xParent*

← *procTop*

← *grid*

↔ *output*

← *parameters*

← *time*

References Parameters::bEOSGammaLaw, bFileExists(), Parameters::dGamma, ProcTop::nCoords, Grid::nD, Grid::nGlobalGridDims, Grid::nLocalGridDims, Grid::nNum1DZones, Grid::nNumDims, Grid::nNumGhostCells, ProcTop::nNumProcs, ProcTop::nProcDims, ProcTop::nRank, Time::nTimeStepIndex, Output::ofWatchZoneFiles, Output::sBaseOutputFileName, and Output::watchzoneList.

Referenced by init().

7.5.2.4 void writeWatchZones_R_GL (Output & *output*, Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 1D gamma-law gas.

Parameters:

- ↔ *output*
- ← *grid*
- ← *parameters*
- ← *time*
- ← *procTop*

References Grid::dLocalGridOld, Parameters::dPi, Time::dt, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nP, Grid::nQ0, Grid::nR, Time::nTimeStepIndex, Grid::nU, Grid::nU0, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

7.5.2.5 void writeWatchZones_R_TEOS (Output & *output*, Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 1D tabulated equation of state.

Parameters:

- ↔ *output*
- ← *grid*
- ← *parameters*
- ← *time*
- ← *procTop*

References Grid::dLocalGridOld, Parameters::dPi, Time::dt, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nP, Grid::nQ0, Grid::nR, Grid::nT, Time::nTimeStepIndex, Grid::nU, Grid::nU0, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

7.5.2.6 void writeWatchZones_RT_GL (Output & *output*, Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 2D gamma-law gas.

Parameters:

- ↔ *output*
- ← *grid*
- ← *parameters*

← *time*
 ← *procTop*

References Grid::dLocalGridOld, Parameters::dPi, Time::dt, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, Time::nTimeStepIndex, Parameters::nTypeTurbulenceMod, Grid::nU, Grid::nU0, Grid::nV, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

7.5.2.7 void writeWatchZones_RT_TEOS (Output & *output*, Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 2D tabulated equation of state.

Parameters:

↔ *output*
 ← *grid*
 ← *parameters*
 ← *time*
 ← *procTop*

References Grid::dLocalGridOld, Parameters::dPi, Time::dt, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nT, Time::nTimeStepIndex, Parameters::nTypeTurbulenceMod, Grid::nU, Grid::nU0, Grid::nV, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

7.5.2.8 void writeWatchZones_RTP_GL (Output & *output*, Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 3D gamma-law gas.

Parameters:

↔ *output*
 ← *grid*
 ← *parameters*
 ← *time*
 ← *procTop*

References Grid::dLocalGridOld, Parameters::dPi, Time::dt, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Time::nTimeStepIndex, Parameters::nTypeTurbulenceMod, Grid::nU, Grid::nU0, Grid::nV, Grid::nW, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

7.5.2.9 void writeWatchZones_RTP_TEOS (Output & *output*, Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 3D tabulated equation of state.

Parameters:

- ↔ *output*
- ← *grid*
- ← *parameters*
- ← *time*
- ← *procTop*

References Grid::dLocalGridOld, Parameters::dPi, Time::dt, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nT, Time::nTimeStepIndex, Parameters::nTypeTurbulenceMod, Grid::nU, Grid::nU0, Grid::nV, Grid::nW, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

7.6 dataMonitoring.h File Reference

```
#include <string>
#include "xmlParser.h"
#include "global.h"
```

Functions

- void `initWatchZones` (XMLNode xParent, `ProcTop` &procTop, `Grid` &grid, `Output` &output, `Parameters` ¶meters, `Time` &time)
- void `writeWatchZones_R_GL` (`Output` &output, `Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `writeWatchZones_R_TEOS` (`Output` &output, `Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `writeWatchZones_RT_GL` (`Output` &output, `Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `writeWatchZones_RT_TEOS` (`Output` &output, `Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `writeWatchZones_RTP_GL` (`Output` &output, `Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `writeWatchZones_RTP_TEOS` (`Output` &output, `Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `finWatchZones` (`Output` &output)
- bool `bFileExists` (std::string sFilename)

7.6.1 Detailed Description

Header file for `dataMonitoring.cpp`

7.6.2 Function Documentation

7.6.2.1 bool bFileExists (std::string *sFilename*)

Tests if the file exists by attempting to open the file for reading, if it fails it returns false, if it succeeds it returns true. This does not take into consideration permissions but that is ok for this project.

Parameters:

← *sFilename* file name of the file to check if it exists or not

Returns:

returns true or false depending on whether the file exists

Referenced by `initWatchZones()`.

7.6.2.2 void finWatchZones (Output & *output*)

Closes the files opened for writing out the watchzones

Parameters:

← *output*

References Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by fin().

7.6.2.3 void initWatchZones (XMLNode *xParent*, ProcTop & *procTop*, Grid & *grid*, Output & *output*, Parameters & *parameters*, Time & *time*)

Reads in watchzones set in configuration file "SPHERLS.xml". A list is created on each processor containing the watchzones on that processor's local grid. It also opens file streams for each watchzone and writes out a header.

Parameters:

← *xParent*

← *procTop*

← *grid*

↔ *output*

← *parameters*

← *time*

References Parameters::bEOSGammaLaw, bFileExists(), Parameters::dGamma, ProcTop::nCoords, Grid::nD, Grid::nGlobalGridDims, Grid::nLocalGridDims, Grid::nNum1DZones, Grid::nNumDims, Grid::nNumGhostCells, ProcTop::nNumProcs, ProcTop::nProcDims, ProcTop::nRank, Time::nTimeStepIndex, Output::ofWatchZoneFiles, Output::sBaseOutputFileName, and Output::watchzoneList.

Referenced by init().

7.6.2.4 void writeWatchZones_R_GL (Output & *output*, Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 1D gamma-law gas.

Parameters:

↔ *output*

← *grid*

← *parameters*

← *time*

← *procTop*

References Grid::dLocalGridOld, Parameters::dPi, Time::dt, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nP, Grid::nQ0, Grid::nR, Time::nTimeStepIndex, Grid::nU, Grid::nU0, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

7.6.2.5 void writeWatchZones_R_TEOS (Output & *output*, Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 1D tabulated equation of state.

Parameters:

- ↔ *output*
- ← *grid*
- ← *parameters*
- ← *time*
- ← *procTop*

References Grid::dLocalGridOld, Parameters::dPi, Time::dt, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nP, Grid::nQ0, Grid::nR, Grid::nT, Time::nTimeStepIndex, Grid::nU, Grid::nU0, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

7.6.2.6 void writeWatchZones_RT_GL (Output & *output*, Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 2D gamma-law gas.

Parameters:

- ↔ *output*
- ← *grid*
- ← *parameters*
- ← *time*
- ← *procTop*

References Grid::dLocalGridOld, Parameters::dPi, Time::dt, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, Time::nTimeStepIndex, Parameters::nTypeTurbulenceMod, Grid::nU, Grid::nU0, Grid::nV, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

7.6.2.7 void writeWatchZones_RT_TEOS (Output & *output*, Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 2D tabulated equation of state.

Parameters:

- ↔ *output*
- ← *grid*

← *parameters*
 ← *time*
 ← *procTop*

References Grid::dLocalGridOld, Parameters::dPi, Time::dt, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nT, Time::nTimeStepIndex, Parameters::nTypeTurbulenceMod, Grid::nU, Grid::nU0, Grid::nV, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

7.6.2.8 void writeWatchZones_RTP_GL (Output & *output*, Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 3D gamma-law gas.

Parameters:

↔ *output*
 ← *grid*
 ← *parameters*
 ← *time*
 ← *procTop*

References Grid::dLocalGridOld, Parameters::dPi, Time::dt, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Time::nTimeStepIndex, Parameters::nTypeTurbulenceMod, Grid::nU, Grid::nU0, Grid::nV, Grid::nW, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

7.6.2.9 void writeWatchZones_RTP_TEOS (Output & *output*, Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

Writes out the information for each watchzone specified in "SPHERLS.xml" in the case of a 3D tabulated equation of state.

Parameters:

↔ *output*
 ← *grid*
 ← *parameters*
 ← *time*
 ← *procTop*

References Grid::dLocalGridOld, Parameters::dPi, Time::dt, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nT, Time::nTimeStepIndex, Parameters::nTypeTurbulenceMod, Grid::nU, Grid::nU0, Grid::nV, Grid::nW, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

7.7 global.cpp File Reference

```
#include "global.h"
```

7.7.1 Detailed Description

Declares global variables used across files and functions. This file contains the constructors used to initialize the classes defined in [global.h](#), and does little more than initilize the default values of various parameters.

7.8 global.h File Reference

```
#include <vector>
#include <mpi.h>
#include "watchzone.h"
#include "eos.h"
#include "petscksp.h"
#include <csignal>
#include <limits>
```

Classes

- class [ProcTop](#)
- class [MessPass](#)
- class [Grid](#)
- class [Time](#)
- class [Parameters](#)
- class [Output](#)
- class [Performance](#)
- class [Implicit](#)
- class [Functions](#)
- class [Global](#)

Defines

- `#define` [SIGNEGDEN](#) 0
- `#define` [SIGNEGENG](#) 0
- `#define` [SIGNEGTEMP](#) 0
- `#define` [TRACKMAXSOLVERERROR](#) 0
- `#define` [SEDOV](#) 0
- `#define` [VISCOUS_ENERGY_EQ](#) 1
- `#define` [DUMP_VERSION](#) 1

7.8.1 Detailed Description

Header file for [global.cpp](#).

This file contains definitions which are required throughout the program. The classes defined herein are used through out the program.

7.8.2 Define Documentation

7.8.2.1 `#define DUMP_VERSION 1`

Sets the version of the dump file. Should be incremented if changes are made to the information that is printed out in a dump.

Referenced by `modelRead()`, `modelWrite_GL()`, and `modelWrite_TEOS()`.

7.8.2.2 `#define SEDOV 0`

If 1 we are performing the sedov test, which sets special boundary conditions, if 0 we use normal boundary conditions. It also handles artificial viscosity, and timestep slightly differently.

7.8.2.3 `#define SIGNEG DEN 0`

Raise signal on calculation of negative density if set to 1. Useful when debugging, it will stop the debugger at the location of the calculation of the negative density. If not 1, it will speed up calculation slightly and generate more useful output upon detection of negative densities. If 1 and not being run in the debugger, it likely won't generate any usefull output upon negative density, and wil simply abort the program.

7.8.2.4 `#define SIGNEG ENG 0`

Raise signal on calculation of negative energy if set to 1, else don't rais a signal. Otherwise it will be handled through the normal exception method. This is useful when debugging, it will stop the debugger at the location of the calculation of the negative energy. If not 1, it will speed up calculation slightly and generate more useful output upon detection of negative energy. If 1 and not being run in the debugger, it likely won't generate any usefull output upon negative energies, and wil simply abort the program.

7.8.2.5 `#define SIGNEG TEMP 0`

Raise signal on calculation of negative temperature if set to 1, else don't rais a signal. Otherwise it will be handled through the normal exception method. This is useful when debugging, it will stop the debugger at the location of the calculation of the negative energy. If not 1, it will speed up calculation slightly and generate more useful output upon detection of negative energy. If 1 and not being run in the debugger, it likely won't generate any usefull output upon negative energies, and wil simply abort the program.

7.8.2.6 `#define TRACKMAXSOLVERERROR 0`

Report the error of the linear equation solver if set to 1, else don't. Not tracking the error reduces the calculations per iteration and will speed up running, however if there is question of weather the solver is working accurately this is very handy to turn on.

7.8.2.7 `#define VISCOUS_ENERGY_EQ 1`

If 1 will include viscosity in the energy equation. If 0 it won't. This normally should be set to 1

7.9 main.cpp File Reference

```
#include <mpi.h>
#include <sstream>
#include <string>
#include <fstream>
#include <cmath>
#include <vector>
#include <algorithm>
#include <iomanip>
#include <csignal>
#include <fenv.h>
#include "main.h"
#include "global.h"
#include "watchzone.h"
#include "exception2.h"
#include "xmlParser.h"
#include "xmlFunctions.h"
#include "dataManipulation.h"
#include "dataMonitoring.h"
#include "physEquations.h"
```

Functions

- int [main](#) (int argc, char *argv[])
- void [signalHandler](#) (int nSig)

7.9.1 Detailed Description

This file contains the main function which is the driver for SPHERLS.

7.9.2 Function Documentation

7.9.2.1 int main (int argc, char * argv[])

Main driving function of SPHERLS.

Parameters:

- ← *argc* number of arguments passed from the command line
- ← *argv* array of character strings of size *argc* containing the arguments from the command line.

The flow of this function is as follows:

- Initialize program by calling `init()`
- Set function pointers by calling `setMainFunctions()`
- Update new grid with old grid by calling `updateNewGridWithOld()`
- Update boundaries of local grids
- Calculate the first time step by calling `Functions::fpCalculateDeltat()`
- Enter while loop until end time (`Time::dEndTime`) is reached, and for each iteration of the loop:
 - Test to see if a model dump is needed (by checking `Output::bDump` and `Output::nDumpFrequency`), if so dump one by calling `modelWrite()`
 - Write out information for any watchzones present by calling `writeWatchZones()`
 - Write out information for peak kinetic energy per period by calling `writePeakKE()`
 - calculate time step by calling function pointed to by `Functions::fpCalculateDeltat`
- Calculate new velocities by calling the function pointed to by `Functions::fpCalculateNewVelocities()`
- Update velocities on new grid boundaries between processors by calling `updateLocalBoundariesNewGrid()` three times indicating the *r*-velocity (U), *θ*-velocity (V) and the *φ*-velocity (W).
- Calculate new grid velocities with `Functions::fpCalculateNewGridVelocities()`.
- Calculate new radii with `Functions::fpCalculateNewRadii()`.
- Update radii on new grid boundaries between processors by calling `updateLocalBoundariesNewGrid()` indicating radius is to be updated (R).
- Calculate new densities with `Functions::fpCalculateNewDensities()`
- Calculate new energies with `Functions::fpCalculateNewEnergies()`
- Update the old grid boundaries and centers by calling `updateLocalBoundaries()`
- Calculating the next time step with `Functions::fpCalculateDeltat()`

Finish by dumping the last model computed

References `Output::bDump`, `Output::bPrint`, `Implicit::dAverageRHS`, `Implicit::dCurrentRelTErr`, `Time::dDelRho_t_Rho_max`, `Time::dDelT_t_T_max`,
`Time::dDeltat_np1half`, `Time::dDelUmU0_t_UmU0_max`, `Time::dDelV_t_V_max`,
`Time::dDelW_t_W_max`, `Output::dDumpFrequencyTime`, `Time::dEndTime`, `Parameters::dMaxConvectiveVelocity`,
`Implicit::dMaxErrorInRHS`, `Output::dPrintFrequencyTime`,
`Time::dt`, `Output::dTimeLastDump`, `Output::dTimeLastPrint`, `fin()`, `Functions::fpCalculateAveDensities`,
`Functions::fpCalculateDeltat`, `Functions::fpCalculateNewAV`,

Functions::fpCalculateNewDensities, Functions::fpCalculateNewEddyVisc, Functions::fpCalculateNewEOSVars, Functions::fpCalculateNewGridVelocities, Functions::fpCalculateNewRadii, Functions::fpCalculateNewVelocities, Functions::fpImplicitSolve, Functions::fpModelWrite, Functions::fpUpdateLocalBoundaryVelocitiesNewGrid, Functions::fpWriteWatchZones, Global::functions, Global::grid, Global::implicit, init(), Global::messPass, Implicit::nCurrentNumIterations, Grid::nD, Output::nDumpFrequencyStep, Time::nEndTimeStep, Grid::nGamma, Implicit::nMaxNumSolverIterations, Grid::nNumDims, Implicit::nNumImplicitZones, Output::nNumTimeStepsSinceLastDump, Output::nNumTimeStepsSinceLastPrint, Grid::nP, Output::nPrintFrequencyStep, Output::nPrintMode, Grid::nR, ProcTop::nRank, Grid::nT, Time::nTimeStepIndex, Grid::nU0, Global::output, Global::parameters, Global::performance, Global::procTop, Output::sBaseOutputFileName, setMainFunctions(), signalHandler(), Global::time, updateLocalBoundaries(), updateLocalBoundariesNewGrid(), and updateNewGridWithOld().

7.9.2.2 void signalHandler (int *nSig*)

Used for catching signals.

Referenced by main().

7.10 main.h File Reference

Functions

- void [signalHandler](#) (int nSig)
- int [main](#) (int argc, char *argv[])

7.10.1 Detailed Description

Header file for [main.cpp](#)

7.10.2 Function Documentation

7.10.2.1 int main (int *argc*, char * *argv*[])

Main driving function of SPHERLS.

Parameters:

- ← *argc* number of arguments passed from the command line
- ← *argv* array of character strings of size argc containing the arguments from the command line.

The flow of this function is as follows:

- Initilize program by calling [init\(\)](#)
- Set function pointers by calling [setMainFunctions\(\)](#)
- Update new grid with old grid by calling [updateNewGridWithOld\(\)](#)
- Update boundaries of local grids
- Calculate the first time step by calling [Functions::fpCalculateDeltat\(\)](#)
- Enter while loop until end time ([Time::dEndTime](#)) is reached, and for each iteration of the loop:
 - Test to see if a model dump is needed (by checking [Output::bDump](#) and [Output::nDumpFrequency](#)), if so dump one by calling [modelWrite\(\)](#)
 - Write out information for any watchzones present by calling [writeWatchZones\(\)](#)
 - Write out information for peak kinetic energy per period by calling [writePeakKE\(\)](#)
 - calculate time step by calling function pointed to by [Functions::fpCalculateDeltat](#)
- Calculate new velocities by calling the function pointed to by [Functions::fpCalculateNewVelocities\(\)](#)

- Update velocities on new grid boundaries between processors by calling `updateLocalBoundariesNewGrid()` three times indicating the r -velocity (U), θ -velocity (V) and the ϕ -velocity (W).
- Calculate new grid velocities with `Functions::fpCalculateNewGridVelocities()`.
- Calculate new radii with `Functions::fpCalculateNewRadii()`.
- Update radii on new grid boundaries between processors by calling `updateLocalBoundariesNewGrid()` indicating radius is to be updated (R).
- Calculate new densities with `Functions::fpCalculateNewDensities()`
- Calculate new energies with `Functions::fpCalculateNewEnergies()`
- Update the old grid boundaries and centers by calling `updateLocalBoundaries()`
- Calculating the next time step with `Functions::fpCalculateDeltat()`

Finish by dumping the last model computed

References Output::bDump, Output::bPrint, Implicit::dAverageRHS, Implicit::dCurrentRelTError, Time::dDelRho_t_Rho_max, Time::dDelT_t_T_max, Time::dDeltat_nplhalf, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Output::dDumpFrequencyTime, Time::dEndTime, Parameters::dMaxConvectiveVelocity, Implicit::dMaxErrorInRHS, Output::dPrintFrequencyTime, Time::dt, Output::dTimeLastDump, Output::dTimeLastPrint, `fin()`, Functions::fpCalculateAveDensities, Functions::fpCalculateDeltat, Functions::fpCalculateNewAV, Functions::fpCalculateNewDensities, Functions::fpCalculateNewEddyVisc, Functions::fpCalculateNewEnergies, Functions::fpCalculateNewEOSVars, Functions::fpCalculateNewGridVelocities, Functions::fpCalculateNewRadii, Functions::fpCalculateNewVelocities, Functions::fpImplicitSolve, Functions::fpModelWrite, Functions::fpUpdateLocalBoundaryVelocitiesNewGrid, Functions::fpWriteWatchZones, Global::functions, Global::grid, Global::implicit, `init()`, Global::messPass, Implicit::nCurrentNumIterations, Grid::nD, Output::nDumpFrequencyStep, Time::nEndTimeStep, Grid::nGamma, Implicit::nMaxNumSolverIterations, Grid::nNumDims, Implicit::nNumImplicitZones, Output::nNumTimeStepsSinceLastDump, Output::nNumTimeStepsSinceLastPrint, Grid::nP, Output::nPrintFrequencyStep, Output::nPrintMode, Grid::nR, ProcTop::nRank, Grid::nT, Time::nTimeStepIndex, Grid::nU0, Global::output, Global::parameters, Global::performance, Global::procTop, Output::sBaseOutputFileName, `setMainFunctions()`, `signalHandler()`, Global::time, `updateLocalBoundaries()`, `updateLocalBoundariesNewGrid()`, and `updateNewGridWithOld()`.

7.10.2.2 void signalHandler (int *nSig*)

Used for catching signals.

Referenced by `main()`.

7.11 physEquations.cpp File Reference

```
#include <cmath>
#include <sstream>
#include <signal.h>
#include "exception2.h"
#include "physEquations.h"
#include "dataManipulation.h"
#include "dataMonitoring.h"
#include "global.h"
#include <limits>
```

Functions

- void [setMainFunctions](#) ([Functions](#) &functions, [ProcTop](#) &procTop, [Parameters](#) ¶meters, [Grid](#) &grid, [Time](#) &time, [Implicit](#) &implicit)
- void [setInternalVarInf](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [initInternalVars](#) ([Grid](#) &grid, [ProcTop](#) &procTop, [Parameters](#) ¶meters)
- void [calNewVelocities_R](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_R_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RT_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RTP_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_R](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_R_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RT_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RTP_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RT_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)

- void `calNewV_RTP_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewW_RTP` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewW_RTP_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewU0_R` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewU0_RT` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewU0_RTP` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewR` (`Grid` &grid, `Time` &time)
- void `calNewD_R` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewD_RT` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewD_RTP` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_R_AD` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_R_NA` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_R_NA_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RT_AD` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RT_NA` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RT_NA_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RTP_AD` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RTP_NA` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RTP_NA_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewDenave_None` (`Grid` &grid)
- void `calNewDenave_R` (`Grid` &grid)
- void `calNewDenave_RT` (`Grid` &grid)
- void `calNewDenave_RTP` (`Grid` &grid)
- void `calNewP_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewTPKappaGamma_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewPEKappaGamma_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0_R_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0_R_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0Q1_RT_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0Q1_RT_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0Q1Q2_RTP_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0Q1Q2_RTP_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewEddyVisc_None` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewEddyVisc_R_CN` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewEddyVisc_RT_CN` (`Grid` &grid, `Parameters` ¶meters)

- void `calNewEddyVisc_RTP_CN` (Grid &grid, Parameters ¶meters)
- void `calNewEddyVisc_R_SM` (Grid &grid, Parameters ¶meters)
- void `calNewEddyVisc_RT_SM` (Grid &grid, Parameters ¶meters)
- void `calNewEddyVisc_RTP_SM` (Grid &grid, Parameters ¶meters)
- void `calOldDenave_None` (Grid &grid)
- void `calOldDenave_R` (Grid &grid)
- void `calOldDenave_RT` (Grid &grid)
- void `calOldDenave_RTP` (Grid &grid)
- void `calOldP_GL` (Grid &grid, Parameters ¶meters)
- void `calOldPEKappaGamma_TEOS` (Grid &grid, Parameters ¶meters)
- void `calOldQ0_R_GL` (Grid &grid, Parameters ¶meters)
- void `calOldQ0_R_TEOS` (Grid &grid, Parameters ¶meters)
- void `calOldQ0Q1_RT_GL` (Grid &grid, Parameters ¶meters)
- void `calOldQ0Q1_RT_TEOS` (Grid &grid, Parameters ¶meters)
- void `calOldQ0Q1Q2_RTP_GL` (Grid &grid, Parameters ¶meters)
- void `calOldQ0Q1Q2_RTP_TEOS` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_R_CN` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RT_CN` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RTP_CN` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_R_SM` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RT_SM` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RTP_SM` (Grid &grid, Parameters ¶meters)
- void `calDelt_R_GL` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_R_TEOS` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RT_GL` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RT_TEOS` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RTP_GL` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RTP_TEOS` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_CONST` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `implicitSolve_None` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- void `implicitSolve_R` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- void `implicitSolve_RT` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- void `implicitSolve_RTP` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- double `dImplicitEnergyFunction_None` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R_SB` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)

- double `dImplicitEnergyFunction_RT` (`Grid` &`grid`, `Parameters` &`parameters`, `Time` &`time`, double `dTemps`[], int `i`, int `j`, int `k`)
- double `dImplicitEnergyFunction_RT_SB` (`Grid` &`grid`, `Parameters` &`parameters`, `Time` &`time`, double `dTemps`[], int `i`, int `j`, int `k`)
- double `dImplicitEnergyFunction_RTP` (`Grid` &`grid`, `Parameters` &`parameters`, `Time` &`time`, double `dTemps`[], int `i`, int `j`, int `k`)
- double `dImplicitEnergyFunction_RTP_SB` (`Grid` &`grid`, `Parameters` &`parameters`, `Time` &`time`, double `dTemps`[], int `i`, int `j`, int `k`)
- double `dImplicitEnergyFunction_R_LES` (`Grid` &`grid`, `Parameters` &`parameters`, `Time` &`time`, double `dTemps`[], int `i`, int `j`, int `k`)
- double `dImplicitEnergyFunction_R_LES_SB` (`Grid` &`grid`, `Parameters` &`parameters`, `Time` &`time`, double `dTemps`[], int `i`, int `j`, int `k`)
- double `dImplicitEnergyFunction_RT_LES` (`Grid` &`grid`, `Parameters` &`parameters`, `Time` &`time`, double `dTemps`[], int `i`, int `j`, int `k`)
- double `dImplicitEnergyFunction_RT_LES_SB` (`Grid` &`grid`, `Parameters` &`parameters`, `Time` &`time`, double `dTemps`[], int `i`, int `j`, int `k`)
- double `dImplicitEnergyFunction_RTP_LES` (`Grid` &`grid`, `Parameters` &`parameters`, `Time` &`time`, double `dTemps`[], int `i`, int `j`, int `k`)
- double `dImplicitEnergyFunction_RTP_LES_SB` (`Grid` &`grid`, `Parameters` &`parameters`, `Time` &`time`, double `dTemps`[], int `i`, int `j`, int `k`)
- double `dEOS_GL` (double `dRho`, double `dE`, `Parameters` `parameters`)
- void `initDonorFracAndMaxConVel_R_GL` (`Grid` &`grid`, `Parameters` &`parameters`)
- void `initDonorFracAndMaxConVel_R_TEOS` (`Grid` &`grid`, `Parameters` &`parameters`)
- void `initDonorFracAndMaxConVel_RT_GL` (`Grid` &`grid`, `Parameters` &`parameters`)
- void `initDonorFracAndMaxConVel_RT_TEOS` (`Grid` &`grid`, `Parameters` &`parameters`)
- void `initDonorFracAndMaxConVel_RTP_GL` (`Grid` &`grid`, `Parameters` &`parameters`)
- void `initDonorFracAndMaxConVel_RTP_TEOS` (`Grid` &`grid`, `Parameters` &`parameters`)

7.11.1 Detailed Description

This file is used to specify the functions which contain physics. This includes conservation equations, equation of state, etc.. It also sets function pointers for these functions, so that `main()` will know which functions to call. This implementation also allows the functions called to calculate, for example new densities, to be different depending on the processor. This allows one processor to handle the 1D region and other processors to handle a 3D region.

7.11.2 Function Documentation

7.11.2.1 void `calDelt_CONST` (`Grid` & *grid*, `Parameters` & *parameters*, `Time` & *time*, `ProcTop` & *procTop*)

This function is used when a constant tie step is desired.

Referenced by `setMainFunctions()`.

7.11.2.2 void calDelt_R_GL (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial direction only and is compatible with a gamma law gas EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

Referenced by setMainFunctions().

7.11.2.3 void calDelt_R_TEOS (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial direction only and is compatible with a tabulated EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

Referenced by setMainFunctions().

7.11.2.4 void calDelt_RT_GL (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial and theta directions only and is compatible with a gamma law gas EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

Referenced by setMainFunctions().

7.11.2.5 void calDelt_RT_TEOS (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial and theta directions and is compatible with a tabulated EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

Referenced by setMainFunctions().

7.11.2.6 void calDelt_RTP_GL (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial, theta and phi directions only and is compatible with a gamma law gas EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

Referenced by setMainFunctions().

7.11.2.7 void calDelt_RTP_TEOS (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial, theta and phi directions and is compatible with a tabulated EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

Referenced by setMainFunctions().

7.11.2.8 void calNewD_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new densities using terms in the radial direction only

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated densities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology, uses [ProcTop::nRank](#) when reporting negative densities

Boundary Conditions

doesn't allow mass flux through outer interface

Referenced by setMainFunctions().

7.11.2.9 void calNewD_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new densities using terms in the radial and theta directions

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated densities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology, uses [ProcTop::nRank](#) when reporting negative densities

Boundary Conditions

doesn't allow mass flux through outer interface

Referenced by setMainFunctions().

7.11.2.10 void calNewD_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new densities using terms in the radial, theta, and phi directions

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated densities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology, uses [ProcTop::nRank](#) when reporting negative densities

Boundary Conditions

doesn't allow mass flux through outer interface

Referenced by `setMainFunctions()`.

7.11.2.11 void calNewDenave__None (Grid & *grid*)

This function is a dummy function, and doesn't do anything. In the case of a 1D calculation the average density is undefined, and only the density is used. This is different from the case where the 1D region exists on the rank 0 processor, but the grid as a whole is really 2D or 3D. In which case `calNewDenave__R` should be used instead.

Parameters:

↔ *grid*

Referenced by `setMainFunctions()`.

7.11.2.12 void calNewDenave__R (Grid & *grid*)

This function calculates the horizontal average density in a 3D region. This really just copies the density from the particular radial zone into the averaged density variable. This way it can be used exactly the same way in the 1D region as it is in the 3D region. This is done using the density in the new grid, and places the result into the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

Referenced by `setMainFunctions()`.

7.11.2.13 void calNewDenave__RT (Grid & *grid*)

This function calculates the horizontal average density in a 2D region from the new grid density and stores the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

Referenced by `setMainFunctions()`.

7.11.2.14 void calNewDenave__RTP (Grid & *grid*)

This function calculates the horizontal average density in a 3D region from the new grid density and stores the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

Referenced by setMainFunctions().

7.11.2.15 void calNewE_R_AD (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new adiabatic energies using terms in the radial direction.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Referenced by setMainFunctions().

7.11.2.16 void calNewE_R_NA (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial direction and includes radiation diffusion terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Boundary Conditions

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Referenced by setMainFunctions().

7.11.2.17 void calNewE_R_NA_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial direction and includes radiation diffusion terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ **grid** contains the local grid, and will hold the newly updated densities
- ← **parameters** various parameters needed for the calculation
- ← **time** contains time information, e.g. time step, current time etc.
- ← **procTop**

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Boundary Conditions

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

7.11.2.18 void calNewE_RT_AD (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new adiabatic energies using terms in the radial and theta directions.

Parameters:

- ↔ **grid** contains the local grid, and will hold the newly updated densities
- ← **parameters** various parameters needed for the calculation
- ← **time** contains time information, e.g. time step, current time etc.
- ← **procTop**

Boundary Conditions

`grid.dLocalGridOld[grid.nE][i+1][j][k]` is missing

Boundary Conditions

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing using inner gradient for both

Referenced by `setMainFunctions()`.

7.11.2.19 void calNewE_RT_NA (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial and theta directions and includes radiation diffusion terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at *i*.

Boundary Conditions

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Referenced by `setMainFunctions()`.

7.11.2.20 void calNewE_RT_NA_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial and theta directions and includes radiation diffusion terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing ΔM_r outside model using `Parameters::dAlpha` times ΔM_r in the last zone instead.

Boundary Conditions

Setting energy at surface equal to energy in last zone.

Boundary Conditions

missing eddy viscosity outside the model setting it to zero

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Boundary Conditions

missing energy outside the model, assuming it is the same as that in the last zone. That causes this term to be zero.

Referenced by `setMainFunctions()`.

7.11.2.21 void calNewE_RTP_AD (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new adiabatic energies using terms in the radial, theta, and phi directions.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to zero.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient.

Referenced by setMainFunctions().

7.11.2.22 void calNewE_RTP_NA (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial, theta, and phi directions and includes radiation diffusion terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at i.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

Referenced by setMainFunctions().

7.11.2.23 void calNewE_RTP_NA_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial, theta, and phi directions and includes radiation diffusion terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing ΔM_r outside model using [Parameters::dAlpha](#) times ΔM_r in the last zone instead.

Boundary Conditions

Missing W at i+1, assuming the same as at i

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at i.

Boundary Conditions

missing density outside model, setting it to zero

Boundary Conditions

missing eddy viscosity outside the model setting it to zero

Boundary Conditions

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

Referenced by `setMainFunctions()`.

7.11.2.24 void calNewEddyVisc_None (Grid & *grid*, Parameters & *parameters*)

This function is a empty function used as a place holder when no eddy viscosity model is being used.

Parameters:

- ↔ *grid*
- ← *parameters*

Referenced by `setMainFunctions()`.

7.11.2.25 void calNewEddyVisc_R_CN (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity using a constant times the zoning with only the radial terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

7.11.2.26 void calNewEddyVisc_R_SM (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity with only the radial terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

7.11.2.27 void calNewEddyVisc_RT_CN (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity using a constant times the zoning with only the radial and theta terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

Referenced by setMainFunctions().

7.11.2.28 void calNewEddyVisc_RT_SM (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity with only the radial and theta terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

Referenced by setMainFunctions().

7.11.2.29 void calNewEddyVisc_RTP_CN (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity using a constant times the zoning with only the radial, theta, and phi terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

Referenced by `setMainFunctions()`.

7.11.2.30 void calNewEddyVisc_RTP_SM (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity with only the radial, theta, and phi terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

Boundary Conditions

assuming that theta velocity is constant across surface

Boundary Conditions

assume phi velocity is constant across surface

Referenced by `setMainFunctions()`.

7.11.2.31 void calNewP_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the pressure. It is calculated using the new values of quantities and places the result in the new grid. It uses a gamma law gas give in [dEOS_GL](#) to calculate the pressure.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculations.
- ← *parameters* contains parameters used in calculating the pressure, namely the adiabatic gamma that is used.

Referenced by `setMainFunctions()`.

7.11.2.32 void calNewPEKappaGamma_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the Energy, pressure and opacity of a cell. It calculates it using the new vaules of quantities and places the result in the new grid.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← *parameters* contains parameters used in calculating the pressure.

Referenced by `implicitSolve_R()`, `implicitSolve_RT()`, and `implicitSolve_RTP()`.

7.11.2.33 void calNewQ0_R_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial component of the viscosity only. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation.
- ← *parameters* contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

Referenced by setMainFunctions().

7.11.2.34 void calNewQ0_R_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for the radial component of the viscosity only. It uses a sound speed derived from a tabulated equation of state for the calculation.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

Referenced by setMainFunctions().

7.11.2.35 void calNewQ0Q1_RT_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial and theta components of the viscosity. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculations.
- ← *parameters* contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

Referenced by setMainFunctions().

7.11.2.36 void calNewQ0Q1_RT_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for two components of the viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

Referenced by setMainFunctions().

7.11.2.37 void calNewQ0Q1Q2_RTP_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial, theta, and phi components of the viscosity. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculations.
- ← *parameters* contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

Referenced by setMainFunctions().

7.11.2.38 void calNewQ0Q1Q2_RTP_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for the three component of the viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

Referenced by setMainFunctions().

7.11.2.39 void calNewR (Grid & *grid*, Time & *time*)

This function calculates the radii, from the new radial grid velocities

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities
- ← *time* contains time information, e.g. time step, current time etc.

Referenced by setMainFunctions().

7.11.2.40 void calNewTPKappaGamma_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the Temperature, pressure and opacity of a cell. It calculates it using the new vaules of quantities and places the result in the new grid.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← *parameters* contains parameters used in calculating the pressure.

Referenced by setMainFunctions().

7.11.2.41 void calNewU0_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*)

This function calculates the radial grid velocity, it does so by considering only the radial terms

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial grid velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology
- ← *messPass*

Todo

At some point I will likely want to make this funciton compatiabile with a 3D domain decomposition instead of a purely radial domain decomposition.

Referenced by setMainFunctions().

7.11.2.42 void calNewU0_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*)

This function calculates the radial grid velocity, and does it by only considering the radial and theta terms

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial grid velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology
- ↔ *messPass* handles data needed for message passing

Todo

At some point I will likely want to make this funciton compatiabile with a 3D domain decomposition instead of a purely radial domain decomposition.

Boundary Conditions

grid.dLocalGridOld[grid.nD][i+1][j][k] is missing

Referenced by setMainFunctions().

7.11.2.43 void calNewU0_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*)

This function calculates the radial grid velocity, and does it by considering all radial, theta and phi terms

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated radial grid velocities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology
- ↔ ***messPass*** handles data needed for message passing

Todo

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Boundary Conditions

grid.dLocalGridOld[grid.nD][i+1][j][k] is missing

Referenced by setMainFunctions().

7.11.2.44 void calNewU_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by only considering the radial terms.

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated radial velocities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology

Boundary Conditions

Missing grid.dLocalGridOld[grid.nD][nICen+1][j][k] in calculation of $\rho_{i+1/2}$, setting it to 0.0

Boundary Conditions

Missing grid.dLocalGridOld[grid.nP][nICen+1][j][k] in calculation of S_1 , setting it to $-1.0 * \text{grid.dLocalGridOld[grid.nP][nICen][j][k]}$.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Referenced by `calNewVelocities_R()`.

7.11.2.45 void calNewU_R_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by only considering the radial terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated radial velocities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2}$, setting it to 0.0

Boundary Conditions

missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` using velocity at i

Boundary Conditions

Assuming eddy viscosity outside model is zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to $-1.0 * \text{grid.dLocalGridOld[grid.nP][nICen][j][k]}$.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Referenced by `calNewVelocities_R_LES()`.

7.11.2.46 void calNewU_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by only considering the radial and theta terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

Boundary Conditions

assuming theta velocity is constant across surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0]` in calculation of $\langle \rho \rangle_{i+1/2}$, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to $-1.0 * dP_ijk_n$.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Referenced by `calNewVelocities_RT()`.

7.11.2.47 void calNewU_RT_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by only considering the radial and theta terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using
Parameters: `dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Boundary Conditions

Missing density outside of surface, setting it to zero.

Boundary Conditions

Missing density outside model, setting it to zero.

Boundary Conditions

assuming theta and phi velocity same outside star as inside.

Boundary Conditions

Assuming theta velocities are constant across surface.

Boundary Conditions

assuming that V at $i+1$ is equal to v at i .

Boundary Conditions

Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

Boundary Conditions

assume viscosity is zero outside the star.

Boundary Conditions

Missing mass outside model, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]`
in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Referenced by `calNewVelocities_RT_LES()`.

7.11.2.48 `void calNewU_RTP (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)`

This function calculates the radial velocity, and does it by including all radial, theta and phi terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` in calculation of $u_{i+1,j,k}$ setting $u_{i+1,j,k} = u_{i+1/2,j,k}$.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nD][i+1][j][k]` in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

Boundary Conditions

assuming theta velocity is constant across the surface.

Boundary Conditions

assuming phi velocity is constant across the surface.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0]` in calculation of $\langle \rho \rangle_{i+1/2}$ setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it equal to `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]`.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Referenced by `calNewVelocities_RTP()`.

7.11.2.49 void calNewU_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by including all radial, theta and phi terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities

- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Boundary Conditions

Missing density outside of surface, setting it to zero.

Boundary Conditions

Missing density outside model, setting it to zero.

Boundary Conditions

assuming theta and phi velocity same outside star as inside.

Boundary Conditions

Assuming theta velocities are constant across surface.

Boundary Conditions

assuming that V at $i+1$ is equal to v at i .

Boundary Conditions

Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

Boundary Conditions

assume viscosity is zero outside the star.

Boundary Conditions

Missing mass outside model, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Referenced by `calNewVelocities_RTP_LES()`.

7.11.2.50 void calNewV_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the theta velocity, and does it by only considering the radial and theta terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

grid.dLocalGridOld[grid.nV][i+1][j+1][k] is missing

Boundary Conditions

missing upwind gradient, using centred gradient instead

Referenced by calNewVelocities_RT().

7.11.2.51 void calNewV_RT_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the theta velocity, and does it by only considering the radial and theta terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Assuming density outside star is zero

Boundary Conditions

Assuming theta velocity is constant across surface.

Boundary Conditions

Assuming eddy viscosity is zero at surface.

Referenced by calNewVelocities_RT_LES().

7.11.2.52 void calNewV_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the theta velocity, and does it by considering the radial, theta, and phi terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities

- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Assuming theta and phi velocities are the same at the surface of the star as just inside the star.

Boundary Conditions

Assuming centered gradient for upwind gradient outside star at surface.

Referenced by `calNewVelocities_RTP()`.

7.11.2.53 void `calNewV_RTP_LES` (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the theta velocity, and does it by considering the radial, theta, and phi terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Assuming density outside star is zero

Boundary Conditions

Assuming theta velocity is constant across surface.

Boundary Conditions

Assuming eddy viscosity is zero at surface.

Referenced by `calNewVelocities_RTP_LES()`.

7.11.2.54 void `calNewVelocities_R` (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls a function that calculate the radial velocity. Calls the function `calNewU_R` to calculate radial velocity, including only radial terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.

- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

Referenced by setMainFunctions().

7.11.2.55 void calNewVelocities_R_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls a function that calculate the radial velocity. Calls the function [calNewU_R](#) to calculate radial velocity, including only radial terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

7.11.2.56 void calNewVelocities_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls two other functions that calculate the radial and theta velocities. Calls the two functions [calNewU_RT](#) and [calNewV_RT](#) to calculate radial and theta velocities, including both radial and theta terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

Referenced by setMainFunctions().

7.11.2.57 void calNewVelocities_RT_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls two other functions that calculate the radial and theta velocities. Calls the two functions [calNewU_RT](#) and [calNewV_RT](#) to calculate radial and theta velocities, including both radial and theta terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.

- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

Referenced by setMainFunctions().

7.11.2.58 void calNewVelocities_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls three other functions that calculate the radial, theta and phi velocities. Calls the two functions [calNewU_RTP](#), [calNewV_RTP](#) and [calNewW_RTP](#) to calculate radial, theta, and phi velocities, including radial, theta, and phi terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

Referenced by setMainFunctions().

7.11.2.59 void calNewVelocities_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls three other functions that calculate the radial, theta and phi velocities. Calls the two functions [calNewU_RTP](#), [calNewV_RTP](#) and [calNewW_RTP](#) to calculate radial, theta, and phi velocities, including radial, theta, and phi terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

Referenced by setMainFunctions().

7.11.2.60 void calNewW_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the phi velocity, and does it by only considering the radial, theta, and phi terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities

- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` assuming that the phi velocity at the outter most interface is the same as the phi velocity in the center of the zone.

Boundary Conditions

missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` in outter most zone. This is needed to calculate the upwind gradient for donnor cell. The centered gradient is used instead when moving in the negative direction.

Referenced by `calNewVelocities_RTP()`.

7.11.2.61 void calNewW_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the phi velocity, and does it by only considering the radial, theta, and phi terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

assume theta and phi velocities are constant across surface

Boundary Conditions

assume eddy viscosity is zero at surface

Boundary Conditions

assume upwind gradient is the same as centered gradient across surface

Referenced by `calNewVelocities_RTP_LES()`.

7.11.2.62 void calOldDenave_None (Grid & *grid*)

This function is a dumpy functon, and doesn't do anything. In the case of a 1D calculation the average density is undefined, and only the density is used. This is different from the case where the 1D region exists on the rank 0 processor, but the grid as a whole is really 2D or 3D. In which case `calOldDenave_R` should be used instead.

7.11.2.63 void calOldDenave_R (Grid & *grid*)

This function does nothing as the averaged density is not needed in 1D calculations.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

Referenced by `initInternalVars()`.

7.11.2.64 void calOldDenave_RT (Grid & *grid*)

This function calculates the horizontal average density in a 2D region. This function differs from `calNewDenave_RT` in that it calculates the average density from the old grid density and stores the result in the old grid. While `calNewDenave_RT` calculates the average density from the new grid density and places the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

Referenced by `initInternalVars()`.

7.11.2.65 void calOldDenave_RTP (Grid & *grid*)

This function calculates the horizontal average density in a 3D region. This function differs from `calNewDenave_RTP` in that it calculates the average density from the old grid density and stores the result in the old grid. While `calNewDenave_RTP` calculates the average density from the new grid density and places the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

Referenced by `initInternalVars()`.

7.11.2.66 void calOldEddyVisc_R_CN (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity using a constant times the zoning including only the radial terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

Referenced by `initInternalVars()`.

7.11.2.67 void calOldEddyVisc_R_SM (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity including only the radial terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution. It uses the Smagorinsky model for calculating the eddy viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

Referenced by `initInternalVars()`.

7.11.2.68 void calOldEddyVisc_RT_CN (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity using a constant times the zoning including only the radial and theta terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

Referenced by `initInternalVars()`.

7.11.2.69 void calOldEddyVisc_RT_SM (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity including only the radial and theta terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution. It uses the Smagorinsky model for calculating the eddy viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

Referenced by `initInternalVars()`.

7.11.2.70 void calOldEddyVisc_RTP_CN (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity using a constant times the zoning including the radial, theta, and phi terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

Referenced by `initInternalVars()`.

7.11.2.71 void calOldEddyVisc_RTP_SM (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity including the radial, theta, and phi terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution. It uses the Smagorinsky model for calculating the eddy viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

Boundary Conditions

assuming that theta velocity is constant across surface

Boundary Conditions

assume phi velocity is constant across surface

Referenced by `initInternalVars()`.

7.11.2.72 void calOldP_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the pressure using a gamma law gas, calculate by `dEOS_GL`.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the results of the pressure calculations
- ← *parameters* contains parameters used in calculating the pressure, namely the value of the adiabatic gamma

Referenced by `initInternalVars()`.

7.11.2.73 void calOldPEKappaGamma_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the pressure, energy, opacity, and adiabatic index of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. This function is used to initialize the internal variables pressure, energy and kappa, and is suitable for both 1D and 3D calculations.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← *parameters* contains parameters used in calculating the pressure.

Referenced by `initInternalVars()`.

7.11.2.74 void calOldQ0_R_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the radial component of the viscosity only. This function is used when using a gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation

← *parameters* contains parameters used in calculating the artificial viscosity.

Referenced by `initInternalVars()`.

7.11.2.75 void `calOldQ0_R_TEOS` (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for 1D viscosity only.

Parameters:

↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation

← *parameters* contains parameters used in calculating the artificial viscosity.

Todo

should use new P and rho

Referenced by `initInternalVars()`.

7.11.2.76 void `calOldQ0Q1_RT_GL` (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the two components of the viscosity. This function is used when using a gamma law gas equation of state.

Parameters:

↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation

← *parameters* contains parameters used in calculating the artificial viscosity.

Referenced by `initInternalVars()`.

7.11.2.77 void `calOldQ0Q1_RT_TEOS` (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for two components of the viscosity. This function is used when using a tabulated equation of state.

Parameters:

↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation

← *parameters* contains parameters used in calculating the artificial viscosity.

Referenced by `initInternalVars()`.

7.11.2.78 void calOldQ0Q1Q2_RTP_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for the three components of the viscosity. This function is used when using a gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

Referenced by `initInternalVars()`.

7.11.2.79 void calOldQ0Q1Q2_RTP_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for the three components of the viscosity. This function is used when using a tabulated equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

Referenced by `initInternalVars()`.

7.11.2.80 double dEOS_GL (double *dRho*, double *dE*, Parameters *parameters*)

Calculates the pressure from the energy and density using a γ -law gas.

Parameters:

- ← *dRho* the density of a cell
- ← *dE* the energy of a cell
- ← *parameters* contains various parameters, including γ needed to calculate the pressure.

Returns:

the pressure

This version of `dEOS_GL` uses the same value of γ through out the model. The equation of state is given by $\rho(\gamma - 1)E$.

Referenced by `calNewP_GL()`, and `calOldP_GL()`.

7.11.2.81 double dImplicitEnergyFunction_None (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This is an empty function, that isn't even called when no implicit solution is needed. This safe guards against future addition which may need to call an empty function when no implicit solve is being done.

Referenced by `setMainFunctions()`.

7.11.2.82 double dImplicitEnergyFunction_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n+1$, with each other in the non-adiabatic energy equation. The *_R* version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* *dTemps*[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n+1$, *dTemps*[1]=dT_ip1jk_np1 is the temperature at radial position $(i+1, j, k)$ and time $n+1$, *dTemps*[2]=dT_im1jk_np1 is the temperature at radial position $(i-1, j, k)$ and time $n+1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

Referenced by setMainFunctions().

7.11.2.83 double dImplicitEnergyFunction_R_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n+1$, with each other in the non-adiabatic energy equation. The *_R* version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* *dTemps*[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n+1$, *dTemps*[1]=dT_ip1jk_np1 is the temperature at radial position $(i+1, j, k)$ and time $n+1$, *dTemps*[2]=dT_im1jk_np1 is the temperature at radial position $(i-1, j, k)$ and time $n+1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

7.11.2.84 double dImplicitEnergyFunction_R_LES_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_R` version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the `"_SB"` suffix ([dImplicitEnergyFunction_R](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* `dTemps[0]=dT_ijk_np1` is the temperature at radial position (i, j, k) and time $n + 1$ and time $n + 1$, `dTemps[1]=dT_im1jk_np1` is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at *i*.

Boundary Conditions

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

7.11.2.85 double dImplicitEnergyFunction_R_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_R` version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the `"_SB"` suffix ([dImplicitEnergyFunction_R](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*

- ← *time*
- ← **dTemps** dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$ and time $n + 1$, dTemps[1]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

Referenced by setMainFunctions().

7.11.2.86 double dImplicitEnergyFunction_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The _RT version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← **dTemps** dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$, dTemps[1]=dT_ip1jk_np1 is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, dTemps[2]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, dTemps[3]=dT_ijp1k_np1 is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, dTemps[4]=dT_ijm1k_np1 is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

Referenced by setMainFunctions().

7.11.2.87 `double dImplicitEnergyFunction_RT_LES (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)`

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RT` version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* `dTemps[0]=dT_ijk_np1` is the temperature at radial position (i, j, k) and time $n + 1$, `dTemps[1]=dT_ip1jk_np1` is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, `dTemps[2]=dT_im1jk_np1` is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, `dTemps[3]=dT_ijp1k_np1` is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, `dTemps[4]=dT_ijm1k_np1` is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

Referenced by `setMainFunctions()`.

7.11.2.88 `double dImplicitEnergyFunction_RT_LES_SB (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)`

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RT` version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the `"_SB"` suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* `dTemps[0]=dT_ijk_np1` is the temperature at radial position (i, j, k) and time $n + 1$, `dTemps[1]=dT_ip1jk_np1` is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, `dTemps[2]=dT_im1jk_np1` is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, `dTemps[3]=dT_ijp1k_np1` is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, `dTemps[4]=dT_ijm1k_np1` is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

Boundary Conditions

Missing ΔM_r outside model using [Parameters::dAlpha](#) times ΔM_r in the last zone instead.

Boundary Conditions

missing density outside model assuming it is zero

Boundary Conditions

missing density outside model assuming it is zero

Boundary Conditions

assuming V at $ip1/2$ is the same as V at i

Boundary Conditions

Assuming energy outside model is the same as the energy in the last zone inside the model.

Boundary Conditions

Assuming energy outside model is the same as the energy in the last zone inside the model.

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Referenced by `setMainFunctions()`.

7.11.2.89 `double dImplicitEnergyFunction_RT_SB (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)`

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RT` version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the `"_SB"` suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

← *grid*

← *parameters*

← *time*

← *dTemps* `dTemps[0]`=`dT_ijk_np1` is the temperature at radial position (i, j, k) and time $n + 1$, `dTemps[1]`=`dT_ip1jk_np1` is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, `dTemps[2]`=`dT_im1jk_np1` is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, `dTemps[3]`=`dT_ijp1k_np1` is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, `dTemps[4]`=`dT_ijm1k_np1` is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.

- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Referenced by `setMainFunctions()`.

7.11.2.90 `double dImplicitEnergyFunction_RTP (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)`

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RTP` version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "`_SB`" suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps*, *dTemps*[0]=*dT_ijk_np1* is the temperature at radial position (*i*, *j*, *k*) and time $n + 1$, *dTemps*[1]=*dT_ip1jk_np1* is the temperature at radial position ($i + 1$, *j*, *k*) and time $n + 1$, *dTemps*[2]=*dT_im1jk_np1* is the temperature at radial position ($i - 1$, *j*, *k*) and time $n + 1$, *dTemps*[3]=*dT_ijp1k_np1* is the temperature at radial position (*i*, $j + 1$, *k*) and time $n + 1$, *dTemps*[4]=*dT_ijm1k_np1* is the temperature at radial position (*i*, $j - 1$, *k*) and time $n + 1$, *dTemps*[5]=*dT_ijkp1_np1* is the temperature at radial position (*i*, *j*, $k + 1$) and time $n + 1$, *dTemps*[6]=*dT_ijkm1_np1* is the temperature at radial position (*i*, *j*, $k - 1$) and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

Referenced by `setMainFunctions()`.

7.11.2.91 `double dImplicitEnergyFunction_RTP_LES (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)`

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RTP` version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "`_SB`" suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps*, *dTemps[0]=dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps[1]=dT_ip1jk_np1* is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, *dTemps[2]=dT_im1jk_np1* is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, *dTemps[3]=dT_ijp1k_np1* is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, *dTemps[4]=dT_ijm1k_np1* is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, *dTemps[5]=dT_ijkp1_np1* is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, *dTemps[6]=dT_ijkm1_np1* is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

Referenced by setMainFunctions().

7.11.2.92 double dImplicitEnergyFunction_RTP_LES_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The _RTP version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "_SB" suffix ([dImplicitEnergyFunction_RT](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* *dTemps[0]=dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps[1]=dT_ip1jk_np1* is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, *dTemps[2]=dT_im1jk_np1* is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, *dTemps[3]=dT_ijp1k_np1* is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, *dTemps[4]=dT_ijm1k_np1* is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, *dTemps[5]=dT_ijkp1_np1* is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, *dTemps[6]=dT_ijkm1_np1* is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

Boundary Conditions

Missing ΔM_r outside model using [Parameters::dAlpha](#) times ΔM_r in the last zone instead.

Boundary Conditions

missing density outside model assuming it is zero

Boundary Conditions

missing density outside model assuming it is zero

Boundary Conditions

assuming V at $ip1half$ is the same as V at i

Boundary Conditions

assuming W at $ip1half$ is the same as W at i

Boundary Conditions

Assuming energy outside model is the same as the energy in the last zone inside the model.

Boundary Conditions

Assuming energy outside model is the same as the energy in the last zone inside the model.

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Referenced by `setMainFunctions()`.

7.11.2.93 **double dImplicitEnergyFunction_RTP_SB** (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RTP` version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the `"_SB"` suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

← *grid*

← *parameters*

← *time*

← *dTemps* `dTemps[0]=dT_ijk_np1` is the temperature at radial position (i, j, k) and time $n + 1$, `dTemps[1]=dT_ip1jk_np1` is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, `dTemps[2]=dT_im1jk_np1` is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, `dTemps[3]=dT_ijp1k_np1` is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, `dTemps[4]=dT_ijm1k_np1` is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, `dTemps[5]=dT_ijkp1_np1` is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, `dTemps[6]=dT_ijkm1_np1` is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.

- ← i is the radial index to evaluate the function at.
- ← j is the theta index to evaluate the function at.
- ← k is the phi index to evaluate the function at.

Boundary Conditions

Using $E_{\{i,j,k\}^{n+1/2}}$ for $E_{\{i+1/2,j,k\}^{n+1/2}}$

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

Referenced by `setMainFunctions()`.

7.11.2.94 `void implicitSolve_None (Grid & grid, Implicit & implicit, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass, Functions & functions)`

This is an empty function, to be called when no implicit solution is needed. This allows the same code in the main program to be executed wheather or not an implicit solution is being preformed by setting the functon pointer to this functon if there is no implicit solution required.

Referenced by `setMainFunctions()`.

7.11.2.95 `void implicitSolve_R (Grid & grid, Implicit & implicit, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass, Functions & functions)`

This function solves for temperature corrections based on derivatives of the radial non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coeffecient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equaitons. Solving this system of equaitons provides the corrections needed for the new temperature. This processes is then repeated until the corrections are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equaiton of state.

Referenced by `setMainFunctions()`.

7.11.2.96 `void implicitSolve_RT (Grid & grid, Implicit & implicit, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass, Functions & functions)`

This function solves for temperature corrections based on derivatives of the radial-theta non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coeffecient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equaitons. Solving this system of equaitons provides the corrections needed for the new temperature. This processes is then repeated until the corrections are small. At this point the new

temperature is used to update the energy, pressure, and opacity in the new grid via the equaiton of state.

Referenced by setMainFunctions().

7.11.2.97 void implicitSolve_RTP (Grid & *grid*, Implicit & *implicit*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*, Functions & *functions*)

This function solves for temperature corrections based on derivatives of the radial-theta-phi non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coeffecient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equaitons. Solving this system of equaitons provides the corrections needed for the new temperature. This processes is then repeated until the corrections are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equaiton of state.

Referenced by setMainFunctions().

7.11.2.98 void initDonorFracAndMaxConVel_R_GL (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the fuction is for 1D, gamma law calculations.

Referenced by initInternalVars().

7.11.2.99 void initDonorFracAndMaxConVel_R_TEOS (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the fuction is for 1D, tabulated equation of state calculations.

Referenced by initInternalVars().

7.11.2.100 void initDonorFracAndMaxConVel_RT_GL (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the fuction is for 2D, gamma law calculations.

Referenced by initInternalVars().

7.11.2.101 void initDonorFracAndMaxConVel_RT_TEOS (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 2D, tabulated equation of state calculations.

Referenced by initInternalVars().

7.11.2.102 void initDonorFracAndMaxConVel_RTP_GL (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 3D, gamma law calculations.

Referenced by initInternalVars().

7.11.2.103 void initDonorFracAndMaxConVel_RTP_TEOS (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 3D, tabulated equation of state calculations.

Referenced by initInternalVars().

7.11.2.104 void initInternalVars (Grid & *grid*, ProcTop & *procTop*, Parameters & *parameters*)

This function is used to set the initial values of the internal variables. While external variables are initialized from the starting model, internal variables are calculated at startup.

Parameters:

- ↔ *grid* supplies information needed for initializing internal variables as well as storing the initialized internal variables
- ← *procTop* contains information about processor topology
- ← *parameters* contains parameters used in initializing the internal variables.

Warning:

$\Delta\theta$, $\Delta\phi$, $\sin\theta_{i,j,k}$, $\Delta\cos\theta_{i,j,k}$, all don't have the first zone calculated. At the moment this is a ghost cell that doesn't matter, but it may become a problem if calculations require this quantity. This is an issue for quantities that aren't updated in time, as those that are will have boundary cells updated with periodic boundary conditions.

Referenced by init().

7.11.2.105 void setInternalVarInf (Grid & *grid*, Parameters & *parameters*)

This function sets the information for internal variables. While external verabile information is derived from the starting model, internal variables infos are set in this function. In other words this function sets the values of [Grid::nVariables](#).

Parameters:

- ↔ ***grid*** supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.
- ← ***parameters*** is used when setting variable infos, since one needs to know if the code is calculating using a gamma law gas, or a tabulated equation of state.

Referenced by `modelRead()`.

7.11.2.106 void setMainFunctions (Functions & *functions*, ProcTop & *procTop*, Parameters & *parameters*, Grid & *grid*, Time & *time*, Implicit & *implicit*)

Used to set the functions that `main()` uses to evolve the input model.

Parameters:

- ***functions*** is of class [Functions](#) and is used to specify the functions called to calculate the evolution of the input model.
- ← ***procTop*** is of type [ProcTop](#). [ProcTop::nRank](#) is used to set different functions based on processor rank. For instance processor rank 1 requires 1D versions of the equations.
- ← ***parameters*** is of class [Parameters](#). It holds various constants and runtime parameters.
- ← ***grid*** of type [Grid](#). This function requires the number of dimensions, specified by [Grid::nNumDims](#).
- ← ***time*** of type [Time](#). This function requires knowledge of the type of time setp being used, specified by [Time::bVariableTimeStep](#).
- ← ***implicit*** of type [Implicit](#). This function needs to know if there is an implicit region, specified when [Implicit::nNumImplicitZones](#)>0.

The functions are picked based on model geometry, and the physics requested or required by the input model, and the configuration file. The specific functions pointers that are set are described in the [Functions](#) class.

Referenced by `main()`.

7.12 physEquations.h File Reference

```
#include "global.h"
```

Functions

- void `setMainFunctions` (`Functions` &functions, `ProcTop` &procTop, `Parameters` ¶meters, `Grid` &grid, `Time` &time, `Implicit` &implicit)
- void `setInternalVarInf` (`Grid` &grid, `Parameters` ¶meters)
- void `initInternalVars` (`Grid` &grid, `ProcTop` &procTop, `Parameters` ¶meters)
- void `calNewVelocities_R` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewVelocities_R_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewVelocities_RT` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewVelocities_RT_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewVelocities_RTP` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewVelocities_RTP_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewU_R` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewU_R_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewU_RT` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewU_RT_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewU_RTP` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewU_RTP_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewV_RT` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewV_RT_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewV_RTP` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewV_RTP_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewW_RTP` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewW_RTP_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewU0_R` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewU0_RT` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)

- void `calNewU0_RTP` (`Grid &grid`, `Parameters ¶meters`, `Time &time`, `ProcTop &procTop`, `MessPass &messPass`)
- void `calNewR` (`Grid &grid`, `Time &time`)
- void `calNewD_R` (`Grid &grid`, `Parameters ¶meters`, `Time &time`, `ProcTop &procTop`)
- void `calNewD_RT` (`Grid &grid`, `Parameters ¶meters`, `Time &time`, `ProcTop &procTop`)
- void `calNewD_RTP` (`Grid &grid`, `Parameters ¶meters`, `Time &time`, `ProcTop &procTop`)
- void `calNewE_R_AD` (`Grid &grid`, `Parameters ¶meters`, `Time &time`, `ProcTop &procTop`)
- void `calNewE_R_NA` (`Grid &grid`, `Parameters ¶meters`, `Time &time`, `ProcTop &procTop`)
- void `calNewE_R_NA_LES` (`Grid &grid`, `Parameters ¶meters`, `Time &time`, `ProcTop &procTop`)
- void `calNewE_RT_AD` (`Grid &grid`, `Parameters ¶meters`, `Time &time`, `ProcTop &procTop`)
- void `calNewE_RT_NA` (`Grid &grid`, `Parameters ¶meters`, `Time &time`, `ProcTop &procTop`)
- void `calNewE_RT_NA_LES` (`Grid &grid`, `Parameters ¶meters`, `Time &time`, `ProcTop &procTop`)
- void `calNewE_RTP_AD` (`Grid &grid`, `Parameters ¶meters`, `Time &time`, `ProcTop &procTop`)
- void `calNewE_RTP_NA` (`Grid &grid`, `Parameters ¶meters`, `Time &time`, `ProcTop &procTop`)
- void `calNewE_RTP_NA_LES` (`Grid &grid`, `Parameters ¶meters`, `Time &time`, `ProcTop &procTop`)
- void `calNewDenave_None` (`Grid &grid`)
- void `calNewDenave_R` (`Grid &grid`)
- void `calNewDenave_RT` (`Grid &grid`)
- void `calNewDenave_RTP` (`Grid &grid`)
- void `calNewP_GL` (`Grid &grid`, `Parameters ¶meters`)
- void `calNewTPKappaGamma_TEOS` (`Grid &grid`, `Parameters ¶meters`)
- void `calNewPEKappaGamma_TEOS` (`Grid &grid`, `Parameters ¶meters`)
- void `calNewQ0_R_GL` (`Grid &grid`, `Parameters ¶meters`)
- void `calNewQ0_R_TEOS` (`Grid &grid`, `Parameters ¶meters`)
- void `calNewQ0Q1_RT_GL` (`Grid &grid`, `Parameters ¶meters`)
- void `calNewQ0Q1_RT_TEOS` (`Grid &grid`, `Parameters ¶meters`)
- void `calNewQ0Q1Q2_RTP_GL` (`Grid &grid`, `Parameters ¶meters`)
- void `calNewQ0Q1Q2_RTP_TEOS` (`Grid &grid`, `Parameters ¶meters`)
- void `calNewEddyVisc_None` (`Grid &grid`, `Parameters ¶meters`)
- void `calNewEddyVisc_R_CN` (`Grid &grid`, `Parameters ¶meters`)
- void `calNewEddyVisc_RT_CN` (`Grid &grid`, `Parameters ¶meters`)
- void `calNewEddyVisc_RTP_CN` (`Grid &grid`, `Parameters ¶meters`)
- void `calNewEddyVisc_R_SM` (`Grid &grid`, `Parameters ¶meters`)
- void `calNewEddyVisc_RT_SM` (`Grid &grid`, `Parameters ¶meters`)
- void `calNewEddyVisc_RTP_SM` (`Grid &grid`, `Parameters ¶meters`)
- void `calOldDenave_None` (`Grid &grid`)
- void `calOldDenave_R` (`Grid &grid`)
- void `calOldDenave_RT` (`Grid &grid`)
- void `calOldDenave_RTP` (`Grid &grid`)
- void `calOldP_GL` (`Grid &grid`, `Parameters ¶meters`)
- void `calOldPEKappaGamma_TEOS` (`Grid &grid`, `Parameters ¶meters`)

- void `calOldQ0_R_GL` (Grid &grid, Parameters ¶meters)
- void `calOldQ0_R_TEOS` (Grid &grid, Parameters ¶meters)
- void `calOldQ0Q1_RT_GL` (Grid &grid, Parameters ¶meters)
- void `calOldQ0Q1_RT_TEOS` (Grid &grid, Parameters ¶meters)
- void `calOldQ0Q1Q2_RTP_GL` (Grid &grid, Parameters ¶meters)
- void `calOldQ0Q1Q2_RTP_TEOS` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_R_CN` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RT_CN` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RTP_CN` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_R_SM` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RT_SM` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RTP_SM` (Grid &grid, Parameters ¶meters)
- void `calDelt_R_GL` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_R_TEOS` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RT_GL` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RT_TEOS` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RTP_GL` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RTP_TEOS` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_CONST` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `implicitSolve_None` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- void `implicitSolve_R` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- void `implicitSolve_RT` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- void `implicitSolve_RTP` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- double `dImplicitEnergyFunction_None` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R_SB` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RT` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RT_SB` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RTP` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RTP_SB` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R_LES` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R_LES_SB` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)

- double [dImplicitEnergyFunction_RT_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RT_LES_SB](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RTP_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RTP_LES_SB](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dEOS_GL](#) (double dRho, double dE, [Parameters](#) parameters)
- void [initDonorFracAndMaxConVel_R_GL](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [initDonorFracAndMaxConVel_R_TEOS](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [initDonorFracAndMaxConVel_RT_GL](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [initDonorFracAndMaxConVel_RT_TEOS](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [initDonorFracAndMaxConVel_RTP_GL](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [initDonorFracAndMaxConVel_RTP_TEOS](#) ([Grid](#) &grid, [Parameters](#) ¶meters)

7.12.1 Detailed Description

Header file for [physEquations.cpp](#)

7.12.2 Function Documentation

7.12.2.1 void calDelt_CONST ([Grid](#) & *grid*, [Parameters](#) & *parameters*, [Time](#) & *time*, [ProcTop](#) & *procTop*)

This function is used when a constant tie step is desired.

References [Time::dConstTimeStep](#), [Time::dDeltat_n](#), [Time::dDeltat_nm1half](#), [Time::dDeltat_np1half](#), [Time::dt](#), and [Time::nTimeStepIndex](#).

Referenced by [setMainFunctions\(\)](#).

7.12.2.2 void calDelt_R_GL ([Grid](#) & *grid*, [Parameters](#) & *parameters*, [Time](#) & *time*, [ProcTop](#) & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial direction only and is compatible with a gamma law gas EOS.

Parameters:

- ← ***grid*** contains the local grid, and will hold the newly updated densities
- ← ***parameters*** various parameters needed for the calculation
- ↔ ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References [Time::dDelE_t_E_max](#), [Time::dDelRho_t_Rho_max](#), [Time::dDeltat_n](#), [Time::dDeltat_nm1half](#), [Time::dDeltat_np1half](#), [Time::dDelUmU0_t_UmU0_max](#), [Time::dDelV_t_V_max](#), [Time::dDelW_t_W_max](#), [Parameters::dDonorFrac](#),

Parameters::dGamma, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartUpdateExplicit, Time::nTimeStepIndex, Grid::nU, and Grid::nU0.

Referenced by setMainFunctions().

7.12.2.3 void calDelt_R_TEOS (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial direction only and is compatible with a tabulated EOS.

Parameters:

- ← ***grid*** contains the local grid, and will hold the newly updated densities
- ← ***parameters*** various parameters needed for the calculation
- ↔ ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelRho_t_Rho_max, Time::dDelT_t_T_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartUpdateExplicit, Grid::nT, Time::nTimeStepIndex, Grid::nU, and Grid::nU0.

Referenced by setMainFunctions().

7.12.2.4 void calDelt_RT_GL (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial and theta directions only and is compatible with a gamma law gas EOS.

Parameters:

- ← ***grid*** contains the local grid, and will hold the newly updated densities
- ← ***parameters*** various parameters needed for the calculation
- ↔ ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelE_t_E_max, Time::dDelRho_t_Rho_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Parameters::dGamma, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange,

Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, ProcTop::nRank, Grid::nStartUpdateExplicit, Time::nTimeStepIndex, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().

7.12.2.5 void calDelt_RT_TEOS (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial and theta directions and is compatible with a tabulated EOS.

Parameters:

- ← ***grid*** contains the local grid, and will hold the newly updated densities
- ← ***parameters*** various parameters needed for the calculation
- ↔ ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelRho_t_Rho_max, Time::dDelT_t_T_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, ProcTop::nRank, Grid::nStartUpdateExplicit, Grid::nT, Time::nTimeStepIndex, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().

7.12.2.6 void calDelt_RTP_GL (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial, theta and phi directions only and is compatible with a gamma law gas EOS.

Parameters:

- ← ***grid*** contains the local grid, and will hold the newly updated densities
- ← ***parameters*** various parameters needed for the calculation
- ↔ ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelE_t_E_max, Time::dDelRho_t_Rho_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Parameters::dGamma, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange,

Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nStartUpdateExplicit, Time::nTimeStepIndex, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

7.12.2.7 void calDelt_RTP_TEOS (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial, theta and phi directions and is compatible with a tabulated EOS.

Parameters:

- ← ***grid*** contains the local grid, and will hold the newly updated densities
- ← ***parameters*** various parameters needed for the calculation
- ↔ ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelRho_t_Rho_max, Time::dDelT_t_T_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nStartUpdateExplicit, Grid::nT, Time::nTimeStepIndex, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

7.12.2.8 void calNewD_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new densities using terms in the radial direction only

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated densities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology, uses [ProcTop::nRank](#) when reporting negative densities

Boundary Conditions

doesn't allow mass flux through outer interface

References Time::dDeltat_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit,

Grid::nR, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

Referenced by setMainFunctions().

7.12.2.9 void calNewD_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new densities using terms in the radial and theta directions

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated densities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology, uses [ProcTop::nRank](#) when reporting negative densities

Boundary Conditions

doesn't allow mass flux through outter interface

References Time::dDeltat_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDCosThetaIJK, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, ProcTop::nRank, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().

7.12.2.10 void calNewD_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new densities using terms in the radial, theta, and phi directions

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated densities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology, uses [ProcTop::nRank](#) when reporting negative densities

Boundary Conditions

doesn't allow mass flux through outter interface

References Time::dDeltat_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDCosThetaIJK, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, ProcTop::nRank, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

7.12.2.11 void calNewDenave__None (Grid & *grid*)

This function is a dummy function, and doesn't do anything. In the case of a 1D calculation the average density is undefined, and only the density is used. This is different from the case where the 1D region exists on the rank 0 processor, but the grid as a whole is really 2D or 3D. In which case [calNewDenave__R](#) should be used instead.

Parameters:

↔ *grid*

Referenced by `setMainFunctions()`.

7.12.2.12 void calNewDenave__R (Grid & *grid*)

This function calculates the horizontal average density in a 3D region. This really just copies the density from the particular radial zone into the averaged density variable. This way it can be used exactly the same way in the 1D region as it is in the 3D region. This is done using the density in the new grid, and places the result into the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References `Grid::dLocalGridNew`, `Grid::nD`, `Grid::nDenAve`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nStartGhostUpdateExplicit`, and `Grid::nStartUpdateExplicit`.

Referenced by `setMainFunctions()`.

7.12.2.13 void calNewDenave__RT (Grid & *grid*)

This function calculates the horizontal average density in a 2D region from the new grid density and stores the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDCosThetaIJK`, `Grid::nDenAve`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nR`, `Grid::nStartGhostUpdateExplicit`, and `Grid::nStartUpdateExplicit`.

Referenced by `setMainFunctions()`.

7.12.2.14 void calNewDenave__RTP (Grid & *grid*)

This function calculates the horizontal average density in a 3D region from the new grid density and stores the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDCosThetaIJK, Grid::nDenAve, Grid::nDPhi, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

Referenced by setMainFunctions().

7.12.2.15 void calNewE_R_AD (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new adiabatic energies using terms in the radial direction.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

References Time::dDeltat_n, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

Referenced by setMainFunctions().

7.12.2.16 void calNewE_R_NA (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial direction and includes radiation diffusion terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nKappa, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, and Grid::nU0.

Referenced by setMainFunctions().

7.12.2.17 void calNewE_R_NA_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial direction and includes radiation diffusion terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nKappa, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, and Grid::nU0.

7.12.2.18 void calNewE_RT_AD (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new adiabatic energies using terms in the radial and theta directions.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.

← *procTop*

Boundary Conditions

grid.dLocalGridOld[grid.nE][i+1][j][k] is missing

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing using inner gradient for both

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIJP1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().

7.12.2.19 void calNewE_RT_NA (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial and theta directions and includes radiation diffusion terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0] using flux equals $2\sigma T^4$ at surface.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nKappa, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIJP1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().

7.12.2.20 void calNewE_RT_NA_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial and theta directions and includes radiation diffusion terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Setting energy at surface equal to energy in last zone.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0] using flux equals $2\sigma T^4$ at surface.

References Parameters::dAlpha, Parameters::dAlphaExtra, Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dPrt, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nKappa, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().

7.12.2.21 void calNewE_RTP_AD (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new adiabatic energies using terms in the radial, theta, and phi directions.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to zero.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

7.12.2.22 void calNewE_RTP_NA (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial, theta, and phi directions and includes radiation diffusion terms.

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated densities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop***

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at i.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nKappa, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

7.12.2.23 void calNewE_RTP_NA_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial, theta, and phi directions and includes radiation diffusion terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated densities

- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing W at $i+1$, assuming the same as at i

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at i .

Boundary Conditions

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

References `Parameters::dAlpha`, `Parameters::dAlphaExtra`, `Time::dDeltat_n`, `Parameters::dDonorFrac`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dPrt`, `Parameters::dSigma`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nE`, `Grid::nEddyVisc`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nKappa`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `ProcTop::nRank`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIjp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `setMainFunctions()`.

7.12.2.24 void calNewEddyVisc_None (Grid & *grid*, Parameters & *parameters*)

This function is a empty function used as a place holder when no eddy viscosity model is being used.

Parameters:

- ↔ *grid*
- ← *parameters*

Referenced by `setMainFunctions()`.

7.12.2.25 void calNewEddyVisc_R_CN (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity using a constant times the zoning with only the radial terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.12.2.26 void calNewEddyVisc_R_SM (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity with only the radial terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU.

7.12.2.27 void calNewEddyVisc_RT_CN (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity using a constant times the zoning with only the radial and theta terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

Referenced by setMainFunctions().

7.12.2.28 void calNewEddyVisc_RT_SM (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity with only the radial and theta terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().

7.12.2.29 void calNewEddyVisc_RTP_CN (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity using a constant times the zoning with only the radial, theta, and phi terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nSinThetaIJK, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

Referenced by setMainFunctions().

7.12.2.30 void calNewEddyVisc_RTP_SM (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity with only the radial, theta, and phi terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

Boundary Conditions

assuming that theta velocity is constant across surface

Boundary Conditions

assume phi velocity is constant across surface

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nCotThetaIJK, Grid::nD, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nSinThetaIJK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

7.12.2.31 void calNewP_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the pressure. It is calculated using the new values of quantities and places the result in the new grid. It uses a gamma law gas give in [dEOS_GL](#) to calculate the pressure.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculations.

← *parameters* contains parameters used in calculating the pressure, namely the adiabatic gamma that is used.

References dEOS_GL(), Grid::dLocalGridNew, Grid::nD, Grid::nE,
Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP,
Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

Referenced by setMainFunctions().

7.12.2.32 void calNewPEKappaGamma_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the Energy, pressure and opacity of a cell. It calculates it using the new vaules of quantities and places the result in the new grid.

Parameters:

↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation

← *parameters* contains parameters used in calculating the pressure.

References Grid::dLocalGridNew, Parameters::eosTable, eos::getPEKappaGamma(), Grid::nD, Grid::nE, Grid::nEndGhostUpdateImplicit, Grid::nEndUpdateImplicit, Grid::nGamma, Grid::nKappa, Grid::nP, Grid::nStartGhostUpdateImplicit, Grid::nStartUpdateImplicit, and Grid::nT.

Referenced by implicitSolve_R(), implicitSolve_RT(), and implicitSolve_RTP().

7.12.2.33 void calNewQ0_R_GL (Grid & *grid*, Parameters & *parameters*)

This funciton calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial component of the viscosity only. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation.

← *parameters* contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD,
Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nR,
Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU.

Referenced by setMainFunctions().

7.12.2.34 void calNewQ0_R_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for the radial component of the viscosity only. It uses a sound speed derived from a tabulated equaiton of state for the calculation.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridNew,
 Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit,
 Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nR,
 Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU.

Referenced by setMainFunctions().

7.12.2.35 void calNewQ0Q1_RT_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial and theta componenets of the viscosity. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculations.
- ← *parameters* contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
 Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1,
 Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit,
 Grid::nStartUpdateExplicit, Grid::nU, and Grid::nV.

Referenced by setMainFunctions().

7.12.2.36 void calNewQ0Q1_RT_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for two component of the viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridNew,
 Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit,
 Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR,
 Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit,
 Grid::nStartUpdateExplicit, Grid::nU, and Grid::nV.

Referenced by setMainFunctions().

7.12.2.37 void calNewQ0Q1Q2_RTP_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial, theta, and phi componenets of the viscosity. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculations.
- ← *parameters* contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
 Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0,
 Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK,
 Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and
 Grid::nW.

Referenced by setMainFunctions().

7.12.2.38 void calNewQ0Q1Q2_RTP_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for the three component of the viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridNew,
 Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit,
 Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2,
 Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit,
 Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

7.12.2.39 void calNewR (Grid & *grid*, Time & *time*)

This function calculates the radii, from the new radial grid velocities

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities
- ← *time* contains time information, e.g. time step, current time etc.

References Time::dDeltat_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU0.

Referenced by setMainFunctions().

7.12.2.40 void calNewTPKappaGamma_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the Temperature, pressure and opacity of a cell. It calculates it using the new vaules of quantities and places the result in the new grid.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← *parameters* contains parameters used in calculating the pressure.

References Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dTolerance, Parameters::eosTable, eos::getEAndDTDE(), eos::getPKappaGamma(), Grid::nD, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nKappa, Parameters::nMaxIterations, Grid::nP, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nT.

Referenced by setMainFunctions().

7.12.2.41 void calNewU0_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*)

This function calculates the radial grid velocity, it does so by considering only the radial terms

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial grid velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology
- ← *messPass*

Todo

At some point I will likely want to make this functon compatiabile with a 3D domain decomposition instead of a purely radial domain decomposition.

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, ProcTop::nCoords, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, ProcTop::nNumRadialNeighbors, Grid::nR, ProcTop::nRadialNeighborNeighborIDs, ProcTop::nRadialNeighborRanks, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, MessPass::typeRecvNewVar, and MessPass::typeSendNewVar.

Referenced by setMainFunctions().

7.12.2.42 void calNewU0_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*)

This function calculates the radial grid velocity, and does it by only considering the radial and theta terms

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial grid velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology
- ↔ *messPass* handles data needed for message passing

Todo

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Boundary Conditions

grid.dLocalGridOld[grid.nD][i+1][j][k] is missing

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, ProcTop::nCoords, Grid::nD, Grid::nDCosThetaIJK, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nLocalGridDims, Grid::nNumGhostCells, ProcTop::nNumRadialNeighbors, Grid::nR, ProcTop::nRadialNeighborNeighborIDs, ProcTop::nRadialNeighborRanks, ProcTop::nRank, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, MessPass::typeRecvNewVar, and MessPass::typeSendNewVar.

Referenced by setMainFunctions().

7.12.2.43 void calNewU0_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*)

This function calculates the radial grid velocity, and does it by considering all radial, theta and phi terms

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial grid velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology
- ↔ *messPass* handles data needed for message passing

Todo

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Boundary Conditions

grid.dLocalGridOld[grid.nD][i+1][j][k] is missing

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, ProcTop::nCoords, Grid::nD, Grid::nDCosThetaIJK, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nLocalGridDims, Grid::nNumGhostCells, ProcTop::nNumRadialNeighbors, Grid::nR, ProcTop::nRadialNeighborNeighborIDs, ProcTop::nRadialNeighborRanks, ProcTop::nRank, Grid::nSinThetaIj1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, Grid::nW, MessPass::typeRecvNewVar, and MessPass::typeSendNewVar.

Referenced by setMainFunctions().

7.12.2.44 void calNewU_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by only considering the radial terms.

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated radial velocities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology

Boundary Conditions

Missing grid.dLocalGridOld[grid.nD][nICen+1][j][k] in calculation of $\rho_{i+1/2}$, setting it to 0.0

Boundary Conditions

Missing grid.dLocalGridOld[grid.nP][nICen+1][j][k] in calculation of S_1 , setting it to -1.0*grid.dLocalGridOld[grid.nP][nICen][j][k].

Boundary Conditions

Missing grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of centered A_1 gradient, setting it to zero.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nU][i+1][j][k] and grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of upwind gradient, when moving inward. Using centered gradient instead.

References Parameters::dAlpha, Parameters::dAlphaExtra, Time::dDeltat_n, Parameters::dDonorFrac, Parameters::dG, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nM, Grid::nP, Grid::nQ0, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

Referenced by calNewVelocities_R().

7.12.2.45 void calNewU_R_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by only considering the radial terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2}$, setting it to 0.0

Boundary Conditions

missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` using velocity at i

Boundary Conditions

Assuming eddy viscosity outside model is zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to $-1.0 * \text{grid.dLocalGridOld[grid.nP][nICen][j][k]}$.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

References `Parameters::dAlpha`, `Parameters::dAlphaExtra`, `Time::dDeltat_n`, `Parameters::dDonorFrac`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`,
`Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDM`, `Grid::nEddyVisc`,
`Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nM`, `Grid::nP`, `Grid::nQ0`,
`Grid::nR`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, and
`Grid::nU0`.

Referenced by `calNewVelocities_R_LES()`.

7.12.2.46 `void calNewU_RT (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)`

This function calculates the radial velocity, and does it by only considering the radial and theta terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.

← *procTop* contains information about the processor topology

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

Boundary Conditions

assuming theta velocity is constant across surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0]` in calculation of $\langle \rho \rangle_{i+1/2}$, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to $-1.0 * dP_ijk_n$.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

References `Parameters::dAlpha`, `Parameters::dAlphaExtra`, `Time::dDeltat_n`, `Parameters::dDonorFrac`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDTheta`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nM`, `Grid::nP`, `Grid::nQ0`, `Grid::nR`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `calNewVelocities_RT()`.

7.12.2.47 void calNewU_RT_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by only considering the radial and theta terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

↔ *grid* contains the local grid, and will hold the newly updated radial velocities

← *parameters* various parameters needed for the calculation

← *time* contains time information, e.g. time step, current time etc.

← *procTop* contains information about the processor topology

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Boundary Conditions

Missing density outside of surface, setting it to zero.

Boundary Conditions

Missing density outside model, setting it to zero.

Boundary Conditions

assuming theta and phi velocity same outside star as inside.

Boundary Conditions

Assuming theta velocities are constant across surface.

Boundary Conditions

assuming that V at $i+1$ is equal to v at i .

Boundary Conditions

Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

Boundary Conditions

assume viscosity is zero outside the star.

Boundary Conditions

Missing mass outside model, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

References `Parameters::dAlpha`, `Parameters::dAlphaExtra`, `Time::dDeltat_n`, `Parameters::dDonorFrac`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaIJK`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDTheta`, `Grid::nEddyVisc`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nM`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIjplhalfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `calNewVelocities_RT_LES()`.

7.12.2.48 void calNewU_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by including all radial, theta and phi terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` in calculation of $u_{i+1,j,k}$ setting $u_{i+1,j,k} = u_{i+1/2,j,k}$.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nD][i+1][j][k]` in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

Boundary Conditions

assuming theta velocity is constant across the surface.

Boundary Conditions

assuming phi velocity is constant across the surface.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0]` in calculation of $\langle \rho \rangle_{i+1/2}$ setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it equal to [Parameters::dAlpha](#) `grid.dLocalGridOld[grid.nDM][nICen][0][0]`.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using [Parameters::dAlpha](#) `*grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

References `Parameters::dAlpha`, `Parameters::dAlphaExtra`, `Time::dDeltat_n`, `Parameters::dDonorFrac`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nM`, `Grid::nP`, `Grid::nQ0`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `calNewVelocities_RTP()`.

7.12.2.49 void calNewU_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by including all radial, theta and phi terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Boundary Conditions

Missing density outside of surface, setting it to zero.

Boundary Conditions

Missing density outside model, setting it to zero.

Boundary Conditions

assuming theta and phi velocity same outside star as inside.

Boundary Conditions

Assuming theta velocities are constant across surface.

Boundary Conditions

assuming that V at $i+1$ is equal to v at i .

Boundary Conditions

Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

Boundary Conditions

assume viscosity is zero outside the star.

Boundary Conditions

Missing mass outside model, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

References Parameters::dAlpha, Parameters::dAlphaExtra, Time::dDeltat_n, Parameters::dDonorFrac, Parameters::dG, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaIJK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nM, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by calNewVelocities_RTP_LES().

7.12.2.50 void calNewV_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the theta velocity, and does it by only considering the radial and theta terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

grid.dLocalGridOld[grid.nV][i+1][j+1][k] is missing

Boundary Conditions

missing upwind gradient, using centred gradient instead

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by calNewVelocities_RT().

7.12.2.51 void calNewV_RT_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the theta velocity, and does it by only considering the radial and theta terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Assuming density outside star is zero

Boundary Conditions

Assuming theta velocity is constant across surface.

Boundary Conditions

Assuming eddy viscosity is zero at surface.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaIjp1halfK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by calNewVelocities_RT_LES().

7.12.2.52 void calNewV_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the theta velocity, and does it by considering the radial, theta, and phi terms.

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated theta velocities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology

Boundary Conditions

Assuming theta and phi velocities are the same at the surface of the star as just inside the star.

Boundary Conditions

ussing cetnered gradient for upwind gradient outside star at surface.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaIjp1halfK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by calNewVelocities_RTP().

7.12.2.53 void calNewV_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the theta velocity, and does it by considering the radial, theta, and phi terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Assuming density outside star is zero

Boundary Conditions

Assuming theta velocity is constant across surface.

Boundary Conditions

Assuming eddy viscosity is zero at surface.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaIjp1halfK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by calNewVelocities_RTP_LES().

7.12.2.54 void calNewVelocities_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls a function that calculate the radial velocity. Calls the function [calNewU_R](#) to calculate radial velocity, including only radial terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

References calNewU_R().

Referenced by setMainFunctions().

7.12.2.55 void calNewVelocities_R_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls a function that calculate the radial velocity. Calls the function [calNewU_R](#) to calculate radial velocity, including only radial terms.

Parameters:

- ↔ ***grid*** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← ***parameters*** contains parameters used in the calculation of the new velocities.
- ← ***time*** contains time step information, current time step, and current time
- ← ***procTop*** contains processor topology information

References `calNewU_RT_LES()`.

7.12.2.56 void `calNewVelocities_RT` (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls two other functions that calculate the radial and theta velocities. Calls the two functions `calNewU_RT` and `calNewV_RT` to calculate radial and theta velocities, including both radial and theta terms.

Parameters:

- ↔ ***grid*** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← ***parameters*** contains parameters used in the calculation of the new velocities.
- ← ***time*** contains time step information, current time step, and current time
- ← ***procTop*** contains processor topology information

References `calNewU_RT()`, and `calNewV_RT()`.

Referenced by `setMainFunctions()`.

7.12.2.57 void `calNewVelocities_RT_LES` (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls two other functions that calculate the radial and theta velocities. Calls the two functions `calNewU_RT` and `calNewV_RT` to calculate radial and theta velocities, including both radial and theta terms.

Parameters:

- ↔ ***grid*** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← ***parameters*** contains parameters used in the calculation of the new velocities.
- ← ***time*** contains time step information, current time step, and current time
- ← ***procTop*** contains processor topology information

References `calNewU_RT_LES()`, and `calNewV_RT_LES()`.

Referenced by `setMainFunctions()`.

7.12.2.58 void calNewVelocities_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls three other functions that calculate the radial, theta and phi velocities. Calls the two functions [calNewU_RTP](#), [calNewV_RTP](#) and [calNewW_RTP](#) to calculate radial, theta, and phi velocities, including radial, theta, and phi terms.

Parameters:

- ↔ ***grid*** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← ***parameters*** contains parameters used in the calculation of the new velocities.
- ← ***time*** contains time step information, current time step, and current time
- ← ***procTop*** contains processor topology information

References [calNewU_RTP\(\)](#), [calNewV_RTP\(\)](#), and [calNewW_RTP\(\)](#).

Referenced by [setMainFunctions\(\)](#).

7.12.2.59 void calNewVelocities_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls three other functions that calculate the radial, theta and phi velocities. Calls the two functions [calNewU_RTP](#), [calNewV_RTP](#) and [calNewW_RTP](#) to calculate radial, theta, and phi velocities, including radial, theta, and phi terms.

Parameters:

- ↔ ***grid*** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← ***parameters*** contains parameters used in the calculation of the new velocities.
- ← ***time*** contains time step information, current time step, and current time
- ← ***procTop*** contains processor topology information

References [calNewU_RTP_LES\(\)](#), [calNewV_RTP_LES\(\)](#), and [calNewW_RTP_LES\(\)](#).

Referenced by [setMainFunctions\(\)](#).

7.12.2.60 void calNewW_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the phi velocity, and does it by only considering the radial, theta, and phi terms.

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated theta velocities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology

Boundary Conditions

missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` assuming that the phi velocity at the outter most interface is the same as the phi velocity in the center of the zone.

Boundary Conditions

missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` in outter most zone. This is needed to calculate the upwind gradient for donnor cell. The centered gradient is used instead when moving in the negative direction.

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `Grid::dLocalGridNew`,
`Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaIJK`, `Grid::nD`,
`Grid::nDenAve`, `Grid::nDM`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nEndGhostUpdateExplicit`,
`Grid::nEndUpdateExplicit`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`,
`Grid::nSinThetaIJK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`,
`Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `calNewVelocities_RTP()`.

7.12.2.61 `void calNewW_RTP_LES (Grid & grid, Parameters & parameters,
Time & time, ProcTop & procTop)`

This function calculates the phi velocity, and does it by only considering the radial, theta, and phi terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

assume theta and phi velocities are constant across surface

Boundary Conditions

assume eddy viscosity is zero at surface

Boundary Conditions

assume upwind gradient is the same as centered gradient across surface

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `Grid::dLocalGridNew`,
`Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaIJK`,
`Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nEddyVisc`,
`Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nP`, `Grid::nQ0`,
`Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIjp1halfK`,
`Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, `Grid::nV`,
and `Grid::nW`.

Referenced by `calNewVelocities_RTP_LES()`.

7.12.2.62 void calOldDenave__None (Grid & *grid*)

This function is a dummy function, and doesn't do anything. In the case of a 1D calculation the average density is undefined, and only the density is used. This is different from the case where the 1D region exists on the rank 0 processor, but the grid as a whole is really 2D or 3D. In which case [calOldDenave__R](#) should be used instead.

7.12.2.63 void calOldDenave__R (Grid & *grid*)

This function does nothing as the averaged density is not needed in 1D calculations.

Parameters:

↔ ***grid*** supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References `Grid::dLocalGridOld`, `Grid::nD`, `Grid::nDenAve`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndGhostUpdateImplicit`, `Grid::nEndUpdateExplicit`, `Grid::nEndUpdateImplicit`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartGhostUpdateImplicit`, `Grid::nStartUpdateExplicit`, and `Grid::nStartUpdateImplicit`.

Referenced by `initInternalVars()`.

7.12.2.64 void calOldDenave__RT (Grid & *grid*)

This function calculates the horizontal average density in a 2D region. This function differs from [calNewDenave__RT](#) in that it calculates the average density from the old grid density and stores the result in the old grid. While `calNewDenave__RT` calculates the average density from the new grid density and places the result in the new grid.

Parameters:

↔ ***grid*** supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References `Grid::dLocalGridOld`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDCosThetaIJK`, `Grid::nDenAve`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndGhostUpdateImplicit`, `Grid::nEndUpdateExplicit`, `Grid::nEndUpdateImplicit`, `Grid::nR`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartGhostUpdateImplicit`, `Grid::nStartUpdateExplicit`, and `Grid::nStartUpdateImplicit`.

Referenced by `initInternalVars()`.

7.12.2.65 void calOldDenave__RTP (Grid & *grid*)

This function calculates the horizontal average density in a 3D region. This function differs from [calNewDenave__RTP](#) in that it calculates the average density from the old grid density and stores the result in the old grid. While `calNewDenave__RTP` calculates the average density from the new grid density and places the result in the new grid.

Parameters:

↔ ***grid*** supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD,
 Grid::nDCosThetaIJK, Grid::nDenAve, Grid::nDPhi, Grid::nEndGhostUpdateExplicit,
 Grid::nEndGhostUpdateImplicit, Grid::nEndUpdateExplicit, Grid::nEndUpdateImplicit,
 Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartGhostUpdateImplicit,
 Grid::nStartUpdateExplicit, and Grid::nStartUpdateImplicit.

Referenced by `initInternalVars()`.

7.12.2.66 `void calOldEddyVisc_R_CN (Grid & grid, Parameters & parameters)`

Calculates the eddy viscosity using a constant times the zoning including only the radial terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld,
 Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nEddyVisc,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nNumGhostCells, Grid::nR,
 Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

Referenced by `initInternalVars()`.

7.12.2.67 `void calOldEddyVisc_R_SM (Grid & grid, Parameters & parameters)`

Calculates the eddy viscosity including only the radial terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution. It uses the Smagorinsky model for calculating the eddy viscosity.

Parameters:

- ↔ ***grid*** supplies the input for calculating the eddy viscosity.
- ← ***parameters*** contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridOld, Grid::nCenIntOffset,
 Grid::nD, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit,
 Grid::nNumGhostCells, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit,
 and Grid::nU.

Referenced by `initInternalVars()`.

7.12.2.68 `void calOldEddyVisc_RT_CN (Grid & grid, Parameters & parameters)`

Calculates the eddy viscosity using a constant times the zoning including only the radial and theta terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

Parameters:

- ↔ ***grid*** supplies the input for calculating the eddy viscosity.
- ← ***parameters*** contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Pa-
 rameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nDTheta, Grid::nEddyVisc,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nNumGhostCells, Grid::nR,
 Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

Referenced by `initInternalVars()`.

7.12.2.69 void calOldEddyVisc_RT_SM (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity including only the radial and theta terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution. It uses the Smagorinsky model for calculating the eddy viscosity.

Parameters:

- ↔ ***grid*** supplies the input for calculating the eddy viscosity.
- ← ***parameters*** contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nNumGhostCells, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by initInternalVars().

7.12.2.70 void calOldEddyVisc_RTP_CN (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity using a constant times the zoning including the radial, theta, and phi terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

Parameters:

- ↔ ***grid*** supplies the input for calculating the eddy viscosity.
- ← ***parameters*** contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nNumGhostCells, Grid::nR, Grid::nSinThetaIJK, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

Referenced by initInternalVars().

7.12.2.71 void calOldEddyVisc_RTP_SM (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity including the radial, theta, and phi terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution. It uses the Smagorinsky model for calculating the eddy viscosity.

Parameters:

- ↔ ***grid*** supplies the input for calculating the eddy viscosity.
- ← ***parameters*** contains parameters used in calculating the eddy viscosity.

Boundary Conditions

assuming that theta velocity is constant across surface

Boundary Conditions

assume phi velocity is constant across surface

References Parameters::dEddyViscosity, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nCotThetaIJK, Grid::nD, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nNumGhostCells, Grid::nR, Grid::nSinThetaIJK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by `initInternalVars()`.

7.12.2.72 void calOldP_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the pressure using a gamma law gas, calculate by `dEOS_GL`.

Parameters:

- ↔ ***grid*** supplies the input for calculating the pressure and also accepts the results of the pressure calculations
- ← ***parameters*** contains parameters used in calculating the pressure, namely the value of the adiabatic gamma

References dEOS_GL(), Grid::dLocalGridOld, Grid::nD, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

Referenced by `initInternalVars()`.

7.12.2.73 void calOldPEKappaGamma_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the pressure, energy, opacity, and adiabatic index of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. This function is used to initialize the internal variables pressure, energy and kappa, and is suitable for both 1D and 3D calculations.

Parameters:

- ↔ ***grid*** supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← ***parameters*** contains parameters used in calculating the pressure.

References Grid::dLocalGridOld, Parameters::eosTable, eos::getPEKappaGamma(), Grid::nD, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndGhostUpdateImplicit, Grid::nEndUpdateExplicit, Grid::nEndUpdateImplicit, Grid::nGamma, Grid::nKappa, Grid::nNumGhostCells, Grid::nP, Grid::nStartGhostUpdateExplicit, Grid::nStartGhostUpdateImplicit, Grid::nStartUpdateExplicit, Grid::nStartUpdateImplicit, and Grid::nT.

Referenced by `initInternalVars()`.

7.12.2.74 void calOldQ0_R_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the radial component of the viscosity only. This function is used when using a gamma law gas equation of state.

Parameters:

- ↔ ***grid*** supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← ***parameters*** contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
 Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nR,
 Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU.

Referenced by initInternalVars().

7.12.2.75 void calOldQ0_R_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for 1D viscosity only.

Parameters:

- ↔ ***grid*** supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← ***parameters*** contains parameters used in calculating the artificial viscosity.

Todo

should use new P and rho

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridOld,
 Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit,
 Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nR, Grid::nStartGhostUpdateExplicit,
 Grid::nStartUpdateExplicit, and Grid::nU.

Referenced by initInternalVars().

7.12.2.76 void calOldQ0Q1_RT_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the two components of the viscosity. This function is used when using a gamma law gas equation of state.

Parameters:

- ↔ ***grid*** supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← ***parameters*** contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
 Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDTheta,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1,
 Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit,
 Grid::nStartUpdateExplicit, Grid::nU, and Grid::nV.

Referenced by `initInternalVars()`.

7.12.2.77 `void calOldQ0Q1_RT_TEOS (Grid & grid, Parameters & parameters)`

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for two components of the viscosity. This function is used when using a tabulated equation of state.

Parameters:

- ↔ ***grid*** supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← ***parameters*** contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridOld,
 Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit,
 Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nSinThetaIJK,
 Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit,
 Grid::nU, and Grid::nV.

Referenced by `initInternalVars()`.

7.12.2.78 `void calOldQ0Q1Q2_RTP_GL (Grid & grid, Parameters & parameters)`

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the three components of the viscosity. This function is used when using a gamma law gas equation of state.

Parameters:

- ↔ ***grid*** supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← ***parameters*** contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
 Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDPhi, Grid::nDTheta,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0,
 Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK,
 Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and
 Grid::nW.

Referenced by `initInternalVars()`.

7.12.2.79 `void calOldQ0Q1Q2_RTP_TEOS (Grid & grid, Parameters & parameters)`

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the three components of the viscosity. This function is used when using a tabulated equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

Referenced by initInternalVars().

7.12.2.80 double dEOS_GL (double *dRho*, double *dE*, Parameters *parameters*)

Calculates the pressure from the energy and density using a γ -law gas.

Parameters:

- ← *dRho* the density of a cell
- ← *dE* the energy of a cell
- ← *parameters* contains various parameters, including γ needed to calculate the pressure.

Returns:

the pressure

This version of [dEOS_GL](#) uses the same value of γ through out the model. The equation of state is given by $\rho(\gamma - 1)E$.

References Parameters::dGamma.

Referenced by calNewP_GL(), and calOldP_GL().

7.12.2.81 double dImplicitEnergyFunction_None (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This is an empty function, that isn't even called when no implicit solution is needed. This safe guards against future addition which may need to call an empty function when no implicit solve is being done.

Referenced by setMainFunctions().

7.12.2.82 double dImplicitEnergyFunction_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n+1$, with each other in the non-adiabatic energy equation. The `_R` version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*

- ← *parameters*
- ← *time*
- ← **dTemps** dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$, dTemps[1]=dT_ip1jk_np1 is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, dTemps[2]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nQ0, Grid::nR, Grid::nT, Grid::nU, and Grid::nU0.

Referenced by setMainFunctions().

7.12.2.83 double dImplicitEnergyFunction_R_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The *_R* version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← **dTemps** dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$, dTemps[1]=dT_ip1jk_np1 is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, dTemps[2]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nQ0, Grid::nR, Grid::nT, Grid::nU, and Grid::nU0.

7.12.2.84 double dImplicitEnergyFunction_R_LES_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The *_R* version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for

calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "_SB" suffix ([dImplicitEnergyFunction_R](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$ and time $n + 1$, dTemps[1]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nR, Grid::nT, Grid::nU, and Grid::nU0.

7.12.2.85 double dImplicitEnergyFunction_R_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The _R version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "_SB" suffix ([dImplicitEnergyFunction_R](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$ and time $n + 1$, dTemps[1]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.

- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at *i*.

Boundary Conditions

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `da1`. Using the centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `eos::dGetEnergy()`, `eos::dGetOpacity()`, `eos::dGetPressure()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Parameters::eosTable`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDM`, `Grid::nE`, `Grid::nR`, `Grid::nT`, `Grid::nU`, and `Grid::nU0`.

Referenced by `setMainFunctions()`.

7.12.2.86 double dImplicitEnergyFunction_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RT` version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* `dTemps[0]=dT_ijk_np1` is the temperature at radial position (*i*, *j*, *k*) and time $n + 1$, `dTemps[1]=dT_ip1jk_np1` is the temperature at radial position ($i + 1$, *j*, *k*) and time $n + 1$, `dTemps[2]=dT_im1jk_np1` is the temperature at radial position ($i - 1$, *j*, *k*) and time $n + 1$, `dTemps[3]=dT_ijp1k_np1` is the temperature at radial position (*i*, $j + 1$, *k*) and time $n + 1$, `dTemps[4]=dT_ijm1k_np1` is the temperature at radial position (*i*, $j - 1$, *k*) and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `eos::dGetEnergy()`, `eos::dGetOpacity()`, `eos::dGetPressure()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Parameters::eosTable`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDTheta`, `Grid::nE`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIjp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `setMainFunctions()`.

7.12.2.87 double dImplicitEnergyFunction_RT_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RT` version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* `dTemps[0]=dT_ijk_np1` is the temperature at radial position (i, j, k) and time $n + 1$, `dTemps[1]=dT_ip1jk_np1` is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, `dTemps[2]=dT_im1jk_np1` is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, `dTemps[3]=dT_ijp1k_np1` is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, `dTemps[4]=dT_ijm1k_np1` is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `eos::dGetEnergy()`, `eos::dGetOpacity()`, `eos::dGetPressure()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dPrt`, `Parameters::dSigma`, `Parameters::eosTable`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDTheta`, `Grid::nE`, `Grid::nEddyVisc`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIjp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `setMainFunctions()`.

7.12.2.88 double dImplicitEnergyFunction_RT_LES_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RT` version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the `"_SB"` suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* `dTemps[0]=dT_ijk_np1` is the temperature at radial position (i, j, k) and time $n + 1$, `dTemps[1]=dT_ip1jk_np1` is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, `dTemps[2]=dT_im1jk_np1` is the temperature at radial position $(i - 1, j, k)$

and time $n + 1$, $dTemps[3]=dT_ijp1k_np1$ is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, $dTemps[4]=dT_ijm1k_np1$ is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.

← i is the radial index to evaluate the function at.

← j is the theta index to evaluate the function at.

← k is the phi index to evaluate the function at.

Boundary Conditions

assuming V at $ip1half$ is the same as V at i

Boundary Conditions

Assuming energy outside model is the same as the energy in the last zone inside the model.

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

References Parameters::dAlpha, Parameters::dAlphaExtra, Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dPrt, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nSinThetaLJK, Grid::nSinThetaIJp1halfK, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by `setMainFunctions()`.

7.12.2.89 double dImplicitEnergyFunction_RT_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RT` version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the `"_SB"` suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

← *grid*

← *parameters*

← *time*

← *dTemps* $dTemps[0]=dT_ijk_np1$ is the temperature at radial position (i, j, k) and time $n + 1$, $dTemps[1]=dT_ip1jk_np1$ is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, $dTemps[2]=dT_im1jk_np1$ is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, $dTemps[3]=dT_ijp1k_np1$ is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, $dTemps[4]=dT_ijm1k_np1$ is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.

- ← i is the radial index to evaluate the function at.
- ← j is the theta index to evaluate the function at.
- ← k is the phi index to evaluate the function at.

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `eos::dGetEnergy()`, `eos::dGetOpacity()`, `eos::dGetPressure()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Parameters::eosTable`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDTheta`, `Grid::nE`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIjp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `setMainFunctions()`.

7.12.2.90 `double dImplicitEnergyFunction RTP (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)`

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RTP` version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "`_SB`" suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps*, *dTemps[0]* = *dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps[1]* = *dT_ip1jk_np1* is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, *dTemps[2]* = *dT_im1jk_np1* is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, *dTemps[3]* = *dT_ijp1k_np1* is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, *dTemps[4]* = *dT_ijm1k_np1* is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, *dTemps[5]* = *dT_ijkp1_np1* is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, *dTemps[6]* = *dT_ijkm1_np1* is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.
- ← i is the radial index to evaluate the function at.
- ← j is the theta index to evaluate the function at.
- ← k is the phi index to evaluate the function at.

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `eos::dGetEnergy()`, `eos::dGetOpacity()`, `eos::dGetPressure()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Parameters::eosTable`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nE`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIjp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `setMainFunctions()`.

7.12.2.91 double dImplicitEnergyFunction_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RTP` version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the `"_SB"` suffix ([dImplicitEnergyFunction_RT](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps*, *dTemps*[0]=*dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps*[1]=*dT_ip1jk_np1* is the temperature at radial position ($i + 1, j, k$) and time $n + 1$, *dTemps*[2]=*dT_im1jk_np1* is the temperature at radial position ($i - 1, j, k$) and time $n + 1$, *dTemps*[3]=*dT_ijp1k_np1* is the temperature at radial position ($i, j + 1, k$) and time $n + 1$, *dTemps*[4]=*dT_ijm1k_np1* is the temperature at radial position ($i, j - 1, k$) and time $n + 1$, *dTemps*[5]=*dT_ijkp1_np1* is the temperature at radial position ($i, j, k + 1$) and time $n + 1$, *dTemps*[6]=*dT_ijkm1_np1* is the temperature at radial position ($i, j, k - 1$) and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dPrt, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nT, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

7.12.2.92 double dImplicitEnergyFunction_RTP_LES_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RTP` version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the `"_SB"` suffix ([dImplicitEnergyFunction_RT](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*

- ← **dTemps** dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$, dTemps[1]=dT_ip1jk_np1 is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, dTemps[2]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, dTemps[3]=dT_ijp1k_np1 is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, dTemps[4]=dT_ijm1k_np1 is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, dTemps[5]=dT_ijkp1_np1 is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, dTemps[6]=dT_ijkml_np1 is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.
- ← **i** is the radial index to evaluate the function at.
- ← **j** is the theta index to evaluate the function at.
- ← **k** is the phi index to evaluate the function at.

Boundary Conditions

assuming V at ip1half is the same as V at i

Boundary Conditions

assuming W at ip1half is the same as W at i

Boundary Conditions

Using $E_{\{i,j,k\}^{n+1/2}}$ for $E_{\{i+1/2,j,k\}^{n+1/2}}$

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0] using flux equals $2\sigma T^4$ at surface.

References Parameters::dAlpha, Parameters::dAlphaExtra, Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dPrt, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaJp1halfK, Grid::nT, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

7.12.2.93 double dImplicitEnergyFunction_RTP_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The _RTP version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "_SB" suffix ([dImplicitEnergyFunction_RT](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*

← *parameters*
 ← *time*
 ← **dTemps** dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$, dTemps[1]=dT_ip1jk_np1 is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, dTemps[2]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, dTemps[3]=dT_ijp1k_np1 is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, dTemps[4]=dT_ijm1k_np1 is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, dTemps[5]=dT_ijkp1_np1 is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, dTemps[6]=dT_ijkm1_np1 is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.
 ← *i* is the radial index to evaluate the function at.
 ← *j* is the theta index to evaluate the function at.
 ← *k* is the phi index to evaluate the function at.

Boundary Conditions

Using $E_{\{i,j,k\}^{n+1/2}}$ for $E_{\{i+1/2,j,k\}^{n+1/2}}$

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0] using flux equals $2\sigma T^4$ at surface.

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nT, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

7.12.2.94 void implicitSolve_None (Grid & *grid*, Implicit & *implicit*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*, Functions & *functions*)

This is an empty function, to be called when no implicit solution is needed. This allows the same code in the main program to be executed wheather or not an implicit solution is being preformed by setting the functon pointer to this functon if there is no implicit solution required.

Referenced by setMainFunctions().

7.12.2.95 void implicitSolve_R (Grid & *grid*, Implicit & *implicit*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*, Functions & *functions*)

This function solves for temperature corrections based on derivatives of the radial non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coefferient matrix. The discrepancy in the balance of the energy equation with the new

temperature, energy, pressure, and opacity are included as the right hand side of the system of equaitons. Solving this system of equaitons provides the corrections needed for the new temperature. This processes is then repeated until the corrections are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equaiton of state.

References `calNewPEKappaGamma_TEOS()`, `Implicit::dAverageRHS`, `Implicit::dCurrentRelTError`, `Implicit::dDerivativeStepFraction`, `Grid::dLocalGridNew`, `Implicit::dMaxErrorInRHS`, `Implicit::dTolerance`, `Functions::fpImplicitEnergyFunction`, `Functions::fpImplicitEnergyFunction_SB`, `Implicit::kspContext`, `Implicit::matCoeff`, `Implicit::nCurrentNumIterations`, `Implicit::nLocDer`, `Implicit::nLocFun`, `Implicit::nMaxNumIterations`, `Implicit::nMaxNumSolverIterations`, `Implicit::nNumDerPerRow`, `Implicit::nNumRowsALocal`, `Implicit::nNumRowsALocalSB`, `ProcTop::nRank`, `Grid::nT`, `Time::nTimeStepIndex`, `Implicit::nTypeDer`, `updateLocalBoundariesNewGrid()`, `Implicit::vecRHS`, `Implicit::vecscatTCorrections`, `Implicit::vecTCorrections`, and `Implicit::vecTCorrectionsLocal`.

Referenced by `setMainFunctions()`.

7.12.2.96 `void implicitSolve_RT (Grid & grid, Implicit & implicit, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass, Functions & functions)`

This function solves for temperature corrections based on derivatives of the radial-theta non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coeffecient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equaitons. Solving this system of equaitons provides the corrections needed for the new temperature. This processes is then repeated until the corrections are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equaiton of state.

References `calNewPEKappaGamma_TEOS()`, `Implicit::dAverageRHS`, `Implicit::dCurrentRelTError`, `Implicit::dDerivativeStepFraction`, `Grid::dLocalGridNew`, `Implicit::dMaxErrorInRHS`, `Implicit::dTolerance`, `Functions::fpImplicitEnergyFunction`, `Functions::fpImplicitEnergyFunction_SB`, `Implicit::kspContext`, `Implicit::matCoeff`, `Implicit::nCurrentNumIterations`, `Implicit::nLocDer`, `Implicit::nLocFun`, `Implicit::nMaxNumIterations`, `Implicit::nMaxNumSolverIterations`, `Implicit::nNumDerPerRow`, `Implicit::nNumRowsALocal`, `Implicit::nNumRowsALocalSB`, `ProcTop::nRank`, `Grid::nT`, `Time::nTimeStepIndex`, `Implicit::nTypeDer`, `updateLocalBoundariesNewGrid()`, `Implicit::vecRHS`, `Implicit::vecscatTCorrections`, `Implicit::vecTCorrections`, and `Implicit::vecTCorrectionsLocal`.

Referenced by `setMainFunctions()`.

7.12.2.97 `void implicitSolve_RTP (Grid & grid, Implicit & implicit, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass, Functions & functions)`

This function solves for temperature corrections based on derivatives of the radial-theta-phi non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coeffecient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equaitons. Solving this system of equaitons provides the corrections needed for the new

temperature. This processes is then repeated until the corrections are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equaiton of state.

References `calNewPEKappaGamma_TEOS()`, `Implicit::dAverageRHS`, `Implicit::dCurrentRelTError`, `Implicit::dDerivativeStepFraction`, `Grid::dLocalGridNew`, `Implicit::dMaxErrorInRHS`, `Implicit::dTolerance`, `Functions::fpImplicitEnergyFunction`, `Functions::fpImplicitEnergyFunction_SB`, `Implicit::kspContext`, `Implicit::matCoeff`, `Implicit::nCurrentNumIterations`, `Implicit::nLocDer`, `Implicit::nLocFun`, `Implicit::nMaxNumIterations`, `Implicit::nMaxNumSolverIterations`, `Implicit::nNumDerPerRow`, `Implicit::nNumRowsALocal`, `Implicit::nNumRowsALocalSB`, `ProcTop::nRank`, `Grid::nT`, `Time::nTimeStepIndex`, `Implicit::nTypeDer`, `updateLocalBoundariesNewGrid()`, `Implicit::vecRHS`, `Implicit::vecscatTCorrections`, `Implicit::vecTCorrections`, and `Implicit::vecTCorrectionsLocal`.

Referenced by `setMainFunctions()`.

7.12.2.98 `void initDonorFracAndMaxConVel_R_GL (Grid & grid, Parameters & parameters)`

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the fuction is for 1D, gamma law calculations.

References `Parameters::dDonorFrac`, `Parameters::dGamma`, `Grid::dLocalGridOld`, `Parameters::dMaxConvectiveVelocity`, `Parameters::dMaxConvectiveVelocity_c`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nP`, `Grid::nQ0`, `Grid::nStartUpdateExplicit`, `Grid::nU`, and `Grid::nU0`.

Referenced by `initInternalVars()`.

7.12.2.99 `void initDonorFracAndMaxConVel_R_TEOS (Grid & grid, Parameters & parameters)`

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the fuction is for 1D, tabulated equation of state calculations.

References `Parameters::dDonorFrac`, `Grid::dLocalGridOld`, `Parameters::dMaxConvectiveVelocity`, `Parameters::dMaxConvectiveVelocity_c`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nGamma`, `Grid::nP`, `Grid::nQ0`, `Grid::nStartUpdateExplicit`, `Grid::nU`, and `Grid::nU0`.

Referenced by `initInternalVars()`.

7.12.2.100 `void initDonorFracAndMaxConVel_RT_GL (Grid & grid, Parameters & parameters)`

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the fuction is for 2D, gamma law calculations.

References Parameters::dDonorFrac, Parameters::dGamma, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by initInternalVars().

7.12.2.101 void initDonorFracAndMaxConVel_RT_TEOS (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 2D, tabulated equation of state calculations.

References Parameters::dDonorFrac, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by initInternalVars().

7.12.2.102 void initDonorFracAndMaxConVel_RTP_GL (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 3D, gamma law calculations.

References Parameters::dDonorFrac, Parameters::dGamma, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by initInternalVars().

7.12.2.103 void initDonorFracAndMaxConVel_RTP_TEOS (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the function is for 3D, tabulated equation of state calculations.

References Parameters::dDonorFrac, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by initInternalVars().

7.12.2.104 void initInternalVars (Grid & *grid*, ProcTop & *procTop*, Parameters & *parameters*)

This function is used to set the initial values of the internal variables. While external variables are initialized from the starting model, internal variables are calculated at startup.

Parameters:

- ↔ ***grid*** supplies information needed for initializing internal variables as well as storing the initialized internal variables
- ← ***procTop*** contains information about processor topology
- ← ***parameters*** contains parameters used in initializing the internal variables.

Warning:

$\Delta\theta$, $\Delta\phi$, $\sin\theta_{i,j,k}$, $\Delta\cos\theta_{i,j,k}$, all don't have the first zone calculated. At the moment this is a ghost cell that doesn't matter, but it may become a problem if calculations require this quantity. This is an issue for quantities that aren't updated in time, as those that are will have boundary cells updated with periodic boundary conditions.

References Parameters: bEOSGammaLaw, calOldDenave_R(), calOldDenave_RT(), calOldDenave_RTP(), calOldEddyVisc_R_CN(), calOldEddyVisc_R_SM(), calOldEddyVisc_RT_CN(), calOldEddyVisc_RT_SM(), calOldEddyVisc_RTP_CN(), calOldEddyVisc_RTP_SM(), calOldP_GL(), calOldPEKappaGamma_TEOS(), calOldQ0_R_GL(), calOldQ0_R_TEOS(), calOldQ0Q1_RT_GL(), calOldQ0Q1_RT_TEOS(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), Grid::dLocalGridOld, initDonorFracAndMaxConVel_R_GL(), initDonorFracAndMaxConVel_R_TEOS(), initDonorFracAndMaxConVel_RT_GL(), initDonorFracAndMaxConVel_RT_TEOS(), initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), Grid::nCenIntOffset, Grid::nCotThetaIJK, Grid::nCotThetaIjp1halfK, Grid::nDCosThetaIJK, Grid::nDPhi, Grid::nDTheta, Grid::nLocalGridDims, Grid::nNumDims, Grid::nNumGhostCells, Grid::nPhi, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nTheta, and Parameters::nTypeTurbulenceMod.

Referenced by init().

7.12.2.105 void setInternalVarInf (Grid & *grid*, Parameters & *parameters*)

This function sets the information for internal variables. While external verabile information is derived from the starting model, internal variables infos are set in this function. In other words this function sets the values of [Grid::nVariables](#).

Parameters:

- ↔ ***grid*** supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.
- ← ***parameters*** is used when setting variable infos, since one needs to know if the code is calculating using a gamma law gas, or a tabulated equation of state.

References Parameters: bEOSGammaLaw, Grid::nCotThetaIJK, Grid::nCotThetaIjp1halfK, Grid::nDCosThetaIJK, Grid::nDenAve, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nGamma, Grid::nKappa, Grid::nNumDims, Grid::nNumIntVars, Grid::nNumVars, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Parameters::nTypeTurbulenceMod, and Grid::nVariables.

Referenced by modelRead().

7.12.2.106 void setMainFunctions (Functions & *functions*, ProcTop & *procTop*, Parameters & *parameters*, Grid & *grid*, Time & *time*, Implicit & *implicit*)

Used to set the functions that `main()` uses to evolve the input model.

Parameters:

- ***functions*** is of class `Functions` and is used to specify the functions called to calculate the evolution of the input model.
- ← ***procTop*** is of type `ProcTop`. `ProcTop::nRank` is used to set different functions based on processor rank. For instance processor rank 1 requires 1D versions of the equations.
- ← ***parameters*** is of class `Parameters`. It holds various constants and runtime parameters.
- ← ***grid*** of type `Grid`. This function requires the number of dimensions, specified by `Grid::nNumDims`.
- ← ***time*** of type `Time`. This function requires knowledge of the type of time setp being used, specified by `Time::bVariableTimeStep`.
- ← ***implicit*** of type `Implicit`. This function needs to know if there is an implicit region, specified when `Implicit::nNumImplicitZones`>0.

The functions are picked based on model geometry, and the physics requested or required by the input model, and the configuration file. The specific functions pointers that are set are described in the `Functions` class.

References `Parameters::bAdiabatic`, `Parameters::bEOSGammaLaw`, `Time::bVariableTimeStep`, `calDelt_CONST()`, `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `calNewD_R()`, `calNewD_RT()`, `calNewD_RTP()`, `calNewDenave_None()`, `calNewDenave_R()`, `calNewDenave_RT()`, `calNewDenave_RTP()`, `calNewE_R_AD()`, `calNewE_R_NA()`, `calNewE_RT_AD()`, `calNewE_RT_NA()`, `calNewE_RT_NA_LES()`, `calNewE_RTP_AD()`, `calNewE_RTP_NA()`, `calNewE_RTP_NA_LES()`, `calNewEddyVisc_None()`, `calNewEddyVisc_RT_CN()`, `calNewEddyVisc_RT_SM()`, `calNewEddyVisc_RTP_CN()`, `calNewEddyVisc_RTP_SM()`, `calNewP_GL()`, `calNewQ0_R_GL()`, `calNewQ0_R_TEOS()`, `calNewQ0Q1_RT_GL()`, `calNewQ0Q1_RT_TEOS()`, `calNewQ0Q1Q2_RTP_GL()`, `calNewQ0Q1Q2_RTP_TEOS()`, `calNewR()`, `calNewTPKappaGamma_TEOS()`, `calNewU0_R()`, `calNewU0_RT()`, `calNewU0_RTP()`, `calNewVelocities_R()`, `calNewVelocities_RT()`, `calNewVelocities_RT_LES()`, `calNewVelocities_RTP()`, `calNewVelocities_RTP_LES()`, `dImplicitEnergyFunction_None()`, `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RT_SB()`, `dImplicitEnergyFunction_RTP()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, `dImplicitEnergyFunction_RTP_SB()`, `Functions::fpCalculateAveDensities`, `Functions::fpCalculateDeltat`, `Functions::fpCalculateNewAV`, `Functions::fpCalculateNewDensities`, `Functions::fpCalculateNewEddyVisc`, `Functions::fpCalculateNewEnergies`, `Functions::fpCalculateNewEOSVars`, `Functions::fpCalculateNewGridVelocities`, `Functions::fpCalculateNewRadII`, `Functions::fpCalculateNewVelocities`, `Functions::fpImplicitEnergyFunction`, `Functions::fpImplicitEnergyFunction_SB`, `Functions::fpImplicitSolve`, `Functions::fpModelWrite`, `Functions::fpUpdateLocalBoundaryVelocitiesNewGrid`, `Functions::fpWriteWatchZones`, `implicitSolve_None()`, `implicitSolve_R()`, `implicitSolve_RT()`, `implicitSolve_RTP()`, `modelWrite_GL()`, `modelWrite_TEOS()`, `Grid::nNumDims`, `Implicit::nNumImplicitZones`, `ProcTop::nRank`, `Parameters::nTypeTurbulenceMod`, `updateLocalBoundaryVelocitiesNewGrid_R()`,

updateLocalBoundaryVelocitiesNewGrid_RT(), updateLocalBoundaryVelocitiesNewGrid_RTP(), writeWatchZones_R_GL(), writeWatchZones_R_TEOS(), writeWatchZones_RT_GL(), writeWatchZones_RT_TEOS(), writeWatchZones_RTP_GL(), and writeWatchZones_RTP_TEOS().

Referenced by main().

7.13 physEquations_HEAD.cpp File Reference

```
#include <cmath>
#include <sstream>
#include <signal.h>
#include "exception2.h"
#include "physEquations.h"
#include "dataManipulation.h"
#include "dataMonitoring.h"
#include "global.h"
#include <limits>
```

Functions

- void [setMainFunctions](#) ([Functions](#) &functions, [ProcTop](#) &procTop, [Parameters](#) ¶meters, [Grid](#) &grid, [Time](#) &time, [Implicit](#) &implicit)
- void [setInternalVarInf](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [initInternalVars](#) ([Grid](#) &grid, [ProcTop](#) &procTop, [Parameters](#) ¶meters)
- void [calNewVelocities_R](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_R_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RT_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RTP_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_R](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_R_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RT_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RTP_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RT_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)

- void `calNewV_RTP_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewW_RTP` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewW_RTP_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewU0_R` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewU0_RT` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewU0_RTP` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewR` (`Grid` &grid, `Time` &time)
- void `calNewD_R` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewD_RT` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewD_RTP` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_R_AD` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_R_NA` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_R_NA_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RT_AD` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RT_NA` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RT_NA_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RTP_AD` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RTP_NA` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RTP_NA_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewDenave_None` (`Grid` &grid)
- void `calNewDenave_R` (`Grid` &grid)
- void `calNewDenave_RT` (`Grid` &grid)
- void `calNewDenave_RTP` (`Grid` &grid)
- void `calNewP_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewTPKappaGamma_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewPEKappaGamma_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0_R_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0_R_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0Q1_RT_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0Q1_RT_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0Q1Q2_RTP_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0Q1Q2_RTP_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewEddyVisc_None` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewEddyVisc_R_CN` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewEddyVisc_RT_CN` (`Grid` &grid, `Parameters` ¶meters)

- void `calNewEddyVisc_RTP_CN` (Grid &grid, Parameters ¶meters)
- void `calNewEddyVisc_R_SM` (Grid &grid, Parameters ¶meters)
- void `calNewEddyVisc_RT_SM` (Grid &grid, Parameters ¶meters)
- void `calNewEddyVisc_RTP_SM` (Grid &grid, Parameters ¶meters)
- void `calOldDenave_None` (Grid &grid)
- void `calOldDenave_R` (Grid &grid)
- void `calOldDenave_RT` (Grid &grid)
- void `calOldDenave_RTP` (Grid &grid)
- void `calOldP_GL` (Grid &grid, Parameters ¶meters)
- void `calOldPEKappaGamma_TEOS` (Grid &grid, Parameters ¶meters)
- void `calOldQ0_R_GL` (Grid &grid, Parameters ¶meters)
- void `calOldQ0_R_TEOS` (Grid &grid, Parameters ¶meters)
- void `calOldQ0Q1_RT_GL` (Grid &grid, Parameters ¶meters)
- void `calOldQ0Q1_RT_TEOS` (Grid &grid, Parameters ¶meters)
- void `calOldQ0Q1Q2_RTP_GL` (Grid &grid, Parameters ¶meters)
- void `calOldQ0Q1Q2_RTP_TEOS` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_R_CN` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RT_CN` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RTP_CN` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_R_SM` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RT_SM` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RTP_SM` (Grid &grid, Parameters ¶meters)
- void `calDelt_R_GL` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_R_TEOS` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RT_GL` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RT_TEOS` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RTP_GL` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RTP_TEOS` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_CONST` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `implicitSolve_None` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- void `implicitSolve_R` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- void `implicitSolve_RT` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- void `implicitSolve_RTP` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- double `dImplicitEnergyFunction_None` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R_SB` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)

- double [dImplicitEnergyFunction_RT](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RT_SB](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RTP](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RTP_SB](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_R_LES](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_R_LES_SB](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RT_LES](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RT_LES_SB](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RTP_LES](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dImplicitEnergyFunction_RTP_LES_SB](#) (Grid &grid, [Parameters](#) ¶meters, [Time](#) &time, double dTemps[], int i, int j, int k)
- double [dEOS_GL](#) (double dRho, double dE, [Parameters](#) parameters)
- void [initDonorFracAndMaxConVel_R_GL](#) (Grid &grid, [Parameters](#) ¶meters)
- void [initDonorFracAndMaxConVel_R_TEOS](#) (Grid &grid, [Parameters](#) ¶meters)
- void [initDonorFracAndMaxConVel_RT_GL](#) (Grid &grid, [Parameters](#) ¶meters)
- void [initDonorFracAndMaxConVel_RT_TEOS](#) (Grid &grid, [Parameters](#) ¶meters)
- void [initDonorFracAndMaxConVel_RTP_GL](#) (Grid &grid, [Parameters](#) ¶meters)
- void [initDonorFracAndMaxConVel_RTP_TEOS](#) (Grid &grid, [Parameters](#) ¶meters)

7.13.1 Detailed Description

This file is used to specify the functions which contain physics. This includes conservation equations, equation of state, etc.. It also sets function pointers for these functions, so that [main\(\)](#) will know which functions to call. This implementation also allows the functions called to calculate, for example new densities, to be different depending on the processor. This allows one processor to handle the 1D region and other processors to handle a 3D region.

7.13.2 Function Documentation

7.13.2.1 void calDelt_CONST (Grid & *grid*, [Parameters](#) & *parameters*, [Time](#) & *time*, [ProcTop](#) & *procTop*)

This function is used when a constant tie step is desired.

References [Time::dConstTimeStep](#), [Time::dDeltat_n](#), [Time::dDeltat_nm1half](#), [Time::dDeltat_np1half](#), [Time::dt](#), and [Time::nTimeStepIndex](#).

7.13.2.2 void calDelt_R_GL (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial direction only and is compatible with a gamma law gas EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelE_t_E_max, Time::dDelRho_t_Rho_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Parameters::dGamma, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartUpdateExplicit, Time::nTimeStepIndex, Grid::nU, and Grid::nU0.

7.13.2.3 void calDelt_R_TEOS (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial direction only and is compatible with a tabulated EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelRho_t_Rho_max, Time::dDelT_t_T_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartUpdateExplicit, Grid::nT, Time::nTimeStepIndex, Grid::nU, and Grid::nU0.

7.13.2.4 void calDelt_RT_GL (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial and theta directions only and is compatible with a gamma law gas EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelE_t_E_max, Time::dDelRho_t_Rho_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Parameters::dGamma, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, ProcTop::nRank, Grid::nStartUpdateExplicit, Time::nTimeStepIndex, Grid::nU, Grid::nU0, and Grid::nV.

7.13.2.5 void calDelt_RT_TEOS (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial and theta directions and is compatible with a tabulated EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelRho_t_Rho_max, Time::dDelT_t_T_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, ProcTop::nRank, Grid::nStartUpdateExplicit, Grid::nT, Time::nTimeStepIndex, Grid::nU, Grid::nU0, and Grid::nV.

7.13.2.6 void calDelt_RTP_GL (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial, theta and phi directions only and is compatible with a gamma law gas EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities

- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelE_t_E_max, Time::dDelRho_t_Rho_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Parameters::dGamma, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nStartUpdateExplicit, Time::nTimeStepIndex, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.13.2.7 void calDelt_RTP_TEOS (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial, theta and phi directions and is compatible with a tabulated EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelRho_t_Rho_max, Time::dDelT_t_T_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nStartUpdateExplicit, Grid::nT, Time::nTimeStepIndex, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.13.2.8 void calNewD_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new densities using terms in the radial direction only

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.

- ← ***procTop*** contains information about the processor topology, uses [ProcTop::nRank](#) when reporting negative densities

Boundary Conditions

doesn't allow mass flux through outter interface

References Time::dDeltat_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld,
Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit,
Grid::nR, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit,
Grid::nU, and Grid::nU0.

7.13.2.9 void calNewD_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new densities using terms in the radial and theta directions

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated densities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology, uses [ProcTop::nRank](#) when reporting negative densities

Boundary Conditions

doesn't allow mass flux through outter interface

References Time::dDeltat_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld,
Grid::nCenIntOffset, Grid::nD, Grid::nDCosThetaIJK, Grid::nEndGhostUpdateExplicit,
Grid::nEndUpdateExplicit, Grid::nR, ProcTop::nRank, Grid::nSinThetaIjp1halfK,
Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and
Grid::nV.

7.13.2.10 void calNewD_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new densities using terms in the radial, theta, and phi directions

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated densities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology, uses [ProcTop::nRank](#) when reporting negative densities

Boundary Conditions

doesn't allow mass flux through outter interface

References Time::dDeltat_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDCosThetaIJK, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, ProcTop::nRank, Grid::nSinThetaIj1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.13.2.11 void calNewDenave_None (Grid & *grid*)

This function is a dummy function, and doesn't do anything. In the case of a 1D calculation the average density is undefined, and only the density is used. This is different from the case where the 1D region exists on the rank 0 processor, but the grid as a whole is really 2D or 3D. In which case [calNewDenave_R](#) should be used instead.

Parameters:

↔ *grid*

7.13.2.12 void calNewDenave_R (Grid & *grid*)

This function calculates the horizontal average density in a 3\1D region. This really just copies the density from the particular radial zone into the averaged density variable. This way it can be used exactly the same way in the 1D region as it is in the 3D region. This is done using the density in the new grid, and places the result into the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References Grid::dLocalGridNew, Grid::nD, Grid::nDenAve, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.13.2.13 void calNewDenave_RT (Grid & *grid*)

This function calculates the horizontal average density in a 2D region from the new grid density and stores the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDCosThetaIJK, Grid::nDenAve, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.13.2.14 void calNewDenave_RTP (Grid & *grid*)

This function calculates the horizontal average density in a 3D region from the new grid density and stores the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDCosThetaIJK, Grid::nDenAve, Grid::nDPhi, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.13.2.15 void calNewE_R_AD (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new adiabatic energies using terms in the radial direction.

Parameters:

↔ *grid* contains the local grid, and will hold the newly updated densities
 ← *parameters* various parameters needed for the calculation
 ← *time* contains time information, e.g. time step, current time etc.
 ← *procTop*

References Time::dDeltat_n, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

7.13.2.16 void calNewE_R_NA (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial direction and includes radiation diffusion terms.

Parameters:

↔ *grid* contains the local grid, and will hold the newly updated densities
 ← *parameters* various parameters needed for the calculation
 ← *time* contains time information, e.g. time step, current time etc.
 ← *procTop*

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nKappa, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, and Grid::nU0.

7.13.2.17 void calNewE_R_NA_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial direction and includes radiation diffusion terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nKappa, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, and Grid::nU0.

7.13.2.18 void calNewE_RT_AD (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new adiabatic energies using terms in the radial and theta directions.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

grid.dLocalGridOld[grid.nE][i+1][j][k] is missing

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing using inner gradient for both

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

7.13.2.19 void calNewE_RT_NA (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial and theta directions and includes radiation diffusion terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0] using flux equals $2\sigma T^4$ at surface.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nKappa, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

7.13.2.20 void calNewE_RT_NA_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial and theta directions and includes radiation diffusion terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Setting energy at surface equal to energy in last zone.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nKappa, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

7.13.2.21 `void calNewE_RTP_AD (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)`

This function calculates new adiabatic energies using terms in the radial, theta, and phi directions.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to zero.

Boundary Conditions

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.13.2.22 void calNewE_RTP_NA (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial, theta, and phi directions and includes radiation diffusion terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at `i`.

Boundary Conditions

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew,
Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Grid::nCenIntOffset,
Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nE,
Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nKappa, Grid::nP,
Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK,
Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit,
Grid::nT, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.13.2.23 void calNewE_RTP_NA_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial, theta, and phi directions and includes radiation diffusion terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing `W` at `i+1`, assuming the same as at `i`

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at `i`.

Boundary Conditions

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `Grid::dLocalGridNew`,
`Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Grid::nCenIntOffset`, `Grid::nD`,
`Grid::nDenAve`, `Grid::nDM`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nE`, `Grid::nEddyVisc`,
`Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nKappa`, `Grid::nP`,
`Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `ProcTop::nRank`, `Grid::nSinThetaIJK`,
`Grid::nSinThetaIjp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`,
`Grid::nT`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

7.13.2.24 void calNewEddyVisc_None (Grid & *grid*, Parameters & *parameters*)

This function is a empty function used as a place holder when no eddy viscosity model is being used.

Parameters:

↔ *grid*

← *parameters*

7.13.2.25 void calNewEddyVisc_R_CN (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity using a constant times the zoning with only the radial terms.

Parameters:

↔ *grid* supplies the input for calculating the eddy viscosity.

← *parameters* contains parameters used in calculating the eddy viscosity.

References `Parameters::dEddyViscosity`, `Grid::dLocalGridNew`, `Parameters::dMaxConvectiveVelocity`,
`Grid::nCenIntOffset`, `Grid::nEddyVisc`,
`Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nR`,
`Grid::nStartGhostUpdateExplicit`, and `Grid::nStartUpdateExplicit`.

7.13.2.26 void calNewEddyVisc_R_SM (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity with only the radial terms.

Parameters:

- ↔ **grid** supplies the input for calculating the eddy viscosity.
- ← **parameters** contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU.

7.13.2.27 void calNewEddyVisc_RT_CN (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity using a constant times the zoning with only the radial and theta terms.

Parameters:

- ↔ **grid** supplies the input for calculating the eddy viscosity.
- ← **parameters** contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.13.2.28 void calNewEddyVisc_RT_SM (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity with only the radial and theta terms.

Parameters:

- ↔ **grid** supplies the input for calculating the eddy viscosity.
- ← **parameters** contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

7.13.2.29 void calNewEddyVisc_RTP_CN (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity using a constant times the zoning with only the radial, theta, and phi terms.

Parameters:

- ↔ **grid** supplies the input for calculating the eddy viscosity.
- ← **parameters** contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nSinThetaIJK, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.13.2.30 void calNewEddyVisc_RTP_SM (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity with only the radial, theta, and phi terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

Boundary Conditions

assuming that theta velocity is constant across surface

Boundary Conditions

assume phi velocity is constant across surface

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nCotThetaIJK, Grid::nD, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nSinThetaIJK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.13.2.31 void calNewP_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the pressure. It is calculated using the new values of quantities and places the result in the new grid. It uses a gamma law as given in [dEOS_GL](#) to calculate the pressure.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculations.
- ← *parameters* contains parameters used in calculating the pressure, namely the adiabatic gamma that is used.

References dEOS_GL(), Grid::dLocalGridNew, Grid::nD, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.13.2.32 void calNewPEKappaGamma_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the Energy, pressure and opacity of a cell. It calculates it using the new values of quantities and places the result in the new grid.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← *parameters* contains parameters used in calculating the pressure.

References Grid::dLocalGridNew, Parameters::eosTable, eos::getPEKappaGamma(), Grid::nD, Grid::nE, Grid::nEndGhostUpdateImplicit, Grid::nEndUpdateImplicit, Grid::nGamma, Grid::nKappa, Grid::nP, Grid::nStartGhostUpdateImplicit, Grid::nStartUpdateImplicit, and Grid::nT.

7.13.2.33 void calNewQ0_R_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial component of the viscosity only. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ ***grid*** supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation.
- ← ***parameters*** contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD,
Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nR,
Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU.

7.13.2.34 void calNewQ0_R_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for the radial component of the viscosity only. It uses a sound speed derived from a tabulated equation of state for the calculation.

Parameters:

- ↔ ***grid*** supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← ***parameters*** contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridNew,
Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit,
Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nR,
Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU.

7.13.2.35 void calNewQ0Q1_RT_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial and theta components of the viscosity. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ ***grid*** supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculations.
- ← ***parameters*** contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD,
Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1,
Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit,
Grid::nStartUpdateExplicit, Grid::nU, and Grid::nV.

7.13.2.36 void calNewQ0Q1_RT_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for two component of the viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIJP1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nV.

7.13.2.37 void calNewQ0Q1Q2_RTP_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial, theta, and phi componenets of the viscosity. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculations.
- ← *parameters* contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIJP1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

7.13.2.38 void calNewQ0Q1Q2_RTP_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for the three component of the viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridNew,
 Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit,
 Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2,
 Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit,
 Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

7.13.2.39 void calNewR (Grid & *grid*, Time & *time*)

This function calculates the radii, from the new radial grid velocities

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated radial velocities
- ← ***time*** contains time information, e.g. time step, current time etc.

References Time::dDeltat_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR,
 Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU0.

7.13.2.40 void calNewTPKappaGamma_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the Temperature, pressure and opacity of a cell. It calculates it using the new vaules of quantities and places the result in the new grid.

Parameters:

- ↔ ***grid*** supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← ***parameters*** contains parameters used in calculating the pressure.

References Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dTolerance,
 Parameters::eosTable, eos::getEAndDTDE(), eos::getPKappaGamma(), Grid::nD,
 Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma,
 Grid::nKappa, Parameters::nMaxIterations, Grid::nP, Grid::nStartGhostUpdateExplicit,
 Grid::nStartUpdateExplicit, and Grid::nT.

7.13.2.41 void calNewU0_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*)

This function calculates the radial grid velocity, it does so by considering only the radial terms

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated radial grid velocities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology
- ← ***messPass***

Todo

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, ProcTop::nCoords, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, ProcTop::nNumRadialNeighbors, Grid::nR, ProcTop::nRadialNeighborNeighborIDs, ProcTop::nRadialNeighborRanks, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, MessPass::typeRecvNewVar, and MessPass::typeSendNewVar.

7.13.2.42 void calNewU0_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*)

This function calculates the radial grid velocity, and does it by only considering the radial and theta terms

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial grid velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology
- ↔ *messPass* handles data needed for message passing

Todo

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Boundary Conditions

grid.dLocalGridOld[grid.nD][i+1][j][k] is missing

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, ProcTop::nCoords, Grid::nD, Grid::nDCosThetaIJK, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, ProcTop::nNumRadialNeighbors, Grid::nR, ProcTop::nRadialNeighborNeighborIDs, ProcTop::nRadialNeighborRanks, ProcTop::nRank, Grid::nSinThetaIj1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, MessPass::typeRecvNewVar, and MessPass::typeSendNewVar.

7.13.2.43 void calNewU0_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*)

This function calculates the radial grid velocity, and does it by considering all radial, theta and phi terms

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial grid velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.

- ← *procTop* contains information about the processor topology
- ↔ *messPass* handles data needed for message passing

Todo

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Boundary Conditions

grid.dLocalGridOld[grid.nD][i+1][j][k] is missing

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, ProcTop::nCoords, Grid::nD, Grid::nDCosThetaIJK, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nLocalGridDims, Grid::nNumGhostCells, ProcTop::nNumRadialNeighbors, Grid::nR, ProcTop::nRadialNeighborNeighborIDs, ProcTop::nRadialNeighborRanks, ProcTop::nRank, Grid::nSinThetaIj1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, Grid::nW, MessPass::typeRecvNewVar, and MessPass::typeSendNewVar.

7.13.2.44 void calNewU_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by only considering the radial terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Missing grid.dLocalGridOld[grid.nD][nICen+1][j][k] in calculation of $\rho_{i+1/2}$, setting it to 0.0

Boundary Conditions

Missing grid.dLocalGridOld[grid.nP][nICen+1][j][k] in calculation of S_1 , setting it to $-1.0 \times \text{grid.dLocalGridOld[grid.nP][nICen][j][k]}$.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of centered A_1 gradient, setting it to zero.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nU][i+1][j][k] and grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of upwind gradient, when moving inward. Using centered gradient instead.

References Parameters::dAlpha, Parameters::dAlphaExtra, Time::dDeltat_n, Parameters::dDonorFrac, Parameters::dG, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nM, Grid::nP, Grid::nQ0, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

7.13.2.45 void calNewU_R_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by only considering the radial terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2}$, setting it to 0.0

Boundary Conditions

missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` using velocity at i

Boundary Conditions

Assuming eddy viscosity outside model is zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to $-1.0 * \text{grid.dLocalGridOld[grid.nP][nICen][j][k]}$.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

References Parameters::dAlpha, Parameters::dAlphaExtra, Time::dDeltat_n, Parameters::dDonorFrac, Parameters::dG, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nM, Grid::nP, Grid::nQ0, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

7.13.2.46 void calNewU_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by only considering the radial and theta terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities

- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

Boundary Conditions

assuming theta velocity is constant across surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0]` in calculation of $\langle \rho \rangle_{i+1/2}$, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to $-1.0 * dP_ijk_n$.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

References `Parameters::dAlpha`, `Parameters::dAlphaExtra`, `Time::dDeltat_n`, `Parameters::dDonorFrac`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDTheta`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nM`, `Grid::nP`, `Grid::nQ0`, `Grid::nR`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

7.13.2.47 void calNewU_RT_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by only considering the radial and theta terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities

- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Boundary Conditions

Missing density outside of surface, setting it to zero.

Boundary Conditions

Missing density outside model, setting it to zero.

Boundary Conditions

assuming theta and phi velocity same outside star as inside.

Boundary Conditions

Assuming theta velocities are constant across surface.

Boundary Conditions

assuming that V at $i+1$ is equal to v at i .

Boundary Conditions

Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

Boundary Conditions

assume viscosity is zero outside the star.

Boundary Conditions

Missing mass outside model, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

References `Parameters::dAlpha`, `Parameters::dAlphaExtra`, `Time::dDeltat_n`, `Parameters::dDonorFrac`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaIJK`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDTheta`, `Grid::nEddyVisc`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nM`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIjp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

7.13.2.48 void calNewU_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by including all radial, theta and phi terms.

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated radial velocities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology

Boundary Conditions

missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` in calculation of $u_{i+1,j,k}$ setting $u_{i+1,j,k} = u_{i+1/2,j,k}$.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nD][i+1][j][k]` in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

Boundary Conditions

assuming theta velocity is constant across the surface.

Boundary Conditions

assuming phi velocity is constant across the surface.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0]` in calculation of $\langle \rho \rangle_{i+1/2}$ setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it equal to [Parameters::dAlpha](#) `grid.dLocalGridOld[grid.nDM][nICen][0][0]`.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using [Parameters::dAlpha](#) * `grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

References [Parameters::dAlpha](#), [Parameters::dAlphaExtra](#), [Time::dDeltat_n](#), [Parameters::dDonorFrac](#), [Parameters::dG](#), [Grid::dLocalGridNew](#), [Grid::dLocalGridOld](#), [Parameters::dPi](#), [Grid::nCenIntOffset](#), [Grid::nD](#), [Grid::nDenAve](#), [Grid::nDM](#), [Grid::nDPhi](#), [Grid::nDTheta](#), [Grid::nEndGhostUpdateExplicit](#), [Grid::nEndUpdateExplicit](#), [Grid::nM](#), [Grid::nP](#), [Grid::nQ0](#), [Grid::nR](#), [Grid::nSinThetaIJK](#), [Grid::nStartGhostUpdateExplicit](#), [Grid::nStartUpdateExplicit](#), [Grid::nU](#), [Grid::nU0](#), [Grid::nV](#), and [Grid::nW](#).

7.13.2.49 void calNewU_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by including all radial, theta and phi terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Boundary Conditions

Missing density outside of surface, setting it to zero.

Boundary Conditions

Missing density outside model, setting it to zero.

Boundary Conditions

assuming theta and phi velocity same outside star as inside.

Boundary Conditions

Assuming theta velocities are constant across surface.

Boundary Conditions

assuming that V at $i+1$ is equal to v at i .

Boundary Conditions

Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

Boundary Conditions

assume viscosity is zero outside the star.

Boundary Conditions

Missing mass outside model, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

References Parameters::dAlpha, Parameters::dAlphaExtra, Time::dDeltat_n, Parameters::dDonorFrac, Parameters::dG, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaIJK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nM, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIj1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.13.2.50 void calNewV_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the theta velocity, and does it by only considering the radial and theta terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

grid.dLocalGridOld[grid.nV][i+1][j+1][k] is missing

Boundary Conditions

missing upwind gradient, using centred gradient instead

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

7.13.2.51 void calNewV_RT_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the theta velocity, and does it by only considering the radial and theta terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Assuming density outside star is zero

Boundary Conditions

Assuming theta velocity is constant across surface.

Boundary Conditions

Assuming eddy viscosity is zero at surface.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaIjp1halfK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

7.13.2.52 void calNewV_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the theta velocity, and does it by considering the radial, theta, and phi terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Assuming theta and phi velocities are the same at the surface of the star as just inside the star.

Boundary Conditions

ussing cetnered gradient for upwind gradient outside star at surface.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaIjp1halfK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.13.2.53 void calNewV_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the theta velocity, and does it by considering the radial, theta, and phi terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities

- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Assuming density outside star is zero

Boundary Conditions

Assuming theta velocity is constant across surface.

Boundary Conditions

Assuming eddy viscosity is zero at surface.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaIjp1halfK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.13.2.54 void calNewVelocities_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls a function that calculate the radial velocity. Calls the function [calNewU_R](#) to calculate radial velocity, including only radial terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

References calNewU_R().

7.13.2.55 void calNewVelocities_R_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls a function that calculate the radial velocity. Calls the function [calNewU_R](#) to calculate radial velocity, including only radial terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

References calNewU_R_LES().

7.13.2.56 void calNewVelocities_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls two other functions that calculate the radial and theta velocities. Calls the two functions [calNewU_RT](#) and [calNewV_RT](#) to calculate radial and theta velocities, including both radial and theta terms.

Parameters:

- ↔ ***grid*** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← ***parameters*** contains parameters used in the calculation of the new velocities.
- ← ***time*** contains time step information, current time step, and current time
- ← ***procTop*** contains processor topology information

References [calNewU_RT\(\)](#), and [calNewV_RT\(\)](#).

7.13.2.57 void calNewVelocities_RT_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls two other functions that calculate the radial and theta velocities. Calls the two functions [calNewU_RT](#) and [calNewV_RT](#) to calculate radial and theta velocities, including both radial and theta terms.

Parameters:

- ↔ ***grid*** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← ***parameters*** contains parameters used in the calculation of the new velocities.
- ← ***time*** contains time step information, current time step, and current time
- ← ***procTop*** contains processor topology information

References [calNewU_RT_LES\(\)](#), and [calNewV_RT_LES\(\)](#).

7.13.2.58 void calNewVelocities_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls three other functions that calculate the radial, theta and phi velocities. Calls the two functions [calNewU_RTP](#), [calNewV_RTP](#) and [calNewW_RTP](#) to calculate radial, theta, and phi velocities, including radial, theta, and phi terms.

Parameters:

- ↔ ***grid*** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← ***parameters*** contains parameters used in the calculation of the new velocities.
- ← ***time*** contains time step information, current time step, and current time
- ← ***procTop*** contains processor topology information

References [calNewU_RTP\(\)](#), [calNewV_RTP\(\)](#), and [calNewW_RTP\(\)](#).

7.13.2.59 void calNewVelocities_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls three other functions that calculate the radial, theta and phi velocities. Calls the two functions [calNewU_RTP](#), [calNewV_RTP](#) and [calNewW_RTP](#) to calculate radial, theta, and phi velocities, including radial, theta, and phi terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

References [calNewU_RTP_LES\(\)](#), [calNewV_RTP_LES\(\)](#), and [calNewW_RTP_LES\(\)](#).

7.13.2.60 void calNewW_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the phi velocity, and does it by only considering the radial, theta, and phi terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` assuming that the phi velocity at the outer most interface is the same as the phi velocity in the center of the zone.

Boundary Conditions

missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` in outer most zone. This is needed to calculate the upwind gradient for donor cell. The centered gradient is used instead when moving in the negative direction.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaIJK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.13.2.61 void calNewW_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the phi velocity, and does it by only considering the radial, theta, and phi terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

assume theta and phi velocities are constant across surface

Boundary Conditions

assume eddy viscosity is zero at surface

Boundary Conditions

assume upwind gradient is the same as centered gradient across surface

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaIJK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.13.2.62 void calOldDenave _None (Grid & *grid*)

This function is a dummy function, and doesn't do anything. In the case of a 1D calculation the average density is undefined, and only the density is used. This is different from the case where the 1D region exists on the rank 0 processor, but the grid as a whole is really 2D or 3D. In which case [calOldDenave _R](#) should be used instead.

7.13.2.63 void calOldDenave _R (Grid & *grid*)

This function does nothing as the averaged density is not needed in 1D calculations.

Parameters:

- ↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References Grid::dLocalGridOld, Grid::nD, Grid::nDenAve, Grid::nEndGhostUpdateExplicit, Grid::nEndGhostUpdateImplicit, Grid::nEndUpdateExplicit, Grid::nEndUpdateImplicit, Grid::nStartGhostUpdateExplicit, Grid::nStartGhostUpdateImplicit, Grid::nStartUpdateExplicit, and Grid::nStartUpdateImplicit.

7.13.2.64 void calOldDenave _RT (Grid & *grid*)

This function calculates the horizontal average density in a 2D region. This function differs from [calNewDenave _RT](#) in that it calculates the average density from the old grid density and stores the result in the old grid. While [calNewDenave _RT](#) calculates the average density from the new grid density and places the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD,
 Grid::nDCosThetaIJK, Grid::nDenAve, Grid::nEndGhostUpdateExplicit,
 Grid::nEndGhostUpdateImplicit, Grid::nEndUpdateExplicit, Grid::nEndUpdateImplicit,
 Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartGhostUpdateImplicit,
 Grid::nStartUpdateExplicit, and Grid::nStartUpdateImplicit.

7.13.2.65 void calOldDenave_RTP (Grid & *grid*)

This function calculates the horizontal average density in a 3D region. This function differs from [calNewDenave_RTP](#) in that it calculates the average density from the old grid density and stores the result in the old grid. While [calNewDenave_RTP](#) calculates the average density from the new grid density and places the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD,
 Grid::nDCosThetaIJK, Grid::nDenAve, Grid::nDPhi, Grid::nEndGhostUpdateExplicit,
 Grid::nEndGhostUpdateImplicit, Grid::nEndUpdateExplicit, Grid::nEndUpdateImplicit,
 Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartGhostUpdateImplicit,
 Grid::nStartUpdateExplicit, and Grid::nStartUpdateImplicit.

7.13.2.66 void calOldEddyVisc_R_CN (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity using a constant times the zoning including only the radial terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld,
 Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nEddyVisc,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nNumGhostCells, Grid::nR,
 Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.13.2.67 void calOldEddyVisc_R_SM (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity including only the radial terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution. It uses the Smagorinsky model for calculating the eddy viscosity.

Parameters:

↔ *grid* supplies the input for calculating the eddy viscosity.

← *parameters* contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridOld, Grid::nCenIntOffset,
 Grid::nD, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit,
 Grid::nNumGhostCells, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit,
 and Grid::nU.

7.13.2.68 void calOldEddyVisc_RT_CN (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity using a constant times the zoning including only the radial and theta terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

Parameters:

- ↔ ***grid*** supplies the input for calculating the eddy viscosity.
- ← ***parameters*** contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nNumGhostCells, Grid::nR, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.13.2.69 void calOldEddyVisc_RT_SM (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity including only the radial and theta terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution. It uses the Smagorinsky model for calculating the eddy viscosity.

Parameters:

- ↔ ***grid*** supplies the input for calculating the eddy viscosity.
- ← ***parameters*** contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nNumGhostCells, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

7.13.2.70 void calOldEddyVisc_RTP_CN (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity using a constant times the zoning including the radial, theta, and phi terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

Parameters:

- ↔ ***grid*** supplies the input for calculating the eddy viscosity.
- ← ***parameters*** contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nNumGhostCells, Grid::nR, Grid::nSinThetaIJK, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.13.2.71 void calOldEddyVisc_RTP_SM (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity including the radial, theta, and phi terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution. It uses the Smagorinsky model for calculating the eddy viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

Boundary Conditions

assuming that theta velocity is constant across surface

Boundary Conditions

assume phi velocity is constant across surface

References Parameters::dEddyViscosity, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nCotThetaIJK, Grid::nD, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nNumGhostCells, Grid::nR, Grid::nSinThetaIJK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.13.2.72 void calOldP_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the pressure using a gamma law gas, calculate by [dEOS_GL](#).

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the results of the pressure calculations
- ← *parameters* contains parameters used in calculating the pressure, namely the value of the adiabatic gamma

References dEOS_GL(), Grid::dLocalGridOld, Grid::nD, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.13.2.73 void calOldPEKappaGamma_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the pressure, energy, opacity, and adiabatic index of a cell. It calculates it using the old values of quantities and places the result in the old grid. This function is used to initialize the internal variables pressure, energy and kappa, and is suitable for both 1D and 3D calculations.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation

← *parameters* contains parameters used in calculating the pressure.

References Grid::dLocalGridOld, Parameters::eosTable, eos::getPEKappaGamma(),
 Grid::nD, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndGhostUpdateImplicit,
 Grid::nEndUpdateExplicit, Grid::nEndUpdateImplicit, Grid::nGamma,
 Grid::nKappa, Grid::nNumGhostCells, Grid::nP, Grid::nStartGhostUpdateExplicit,
 Grid::nStartGhostUpdateImplicit, Grid::nStartUpdateExplicit, Grid::nStartUpdateImplicit,
 and Grid::nT.

7.13.2.74 void calOldQ0_R_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the radial component of the viscosity only. This function is used when using a gamma law gas equation of state.

Parameters:

↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation

← *parameters* contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
 Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nR,
 Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU.

7.13.2.75 void calOldQ0_R_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for 1D viscosity only.

Parameters:

↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation

← *parameters* contains parameters used in calculating the artificial viscosity.

Todo

should use new P and rho

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridOld,
 Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit,
 Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nR, Grid::nStartGhostUpdateExplicit,
 Grid::nStartUpdateExplicit, and Grid::nU.

7.13.2.76 void calOldQ0Q1_RT_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the two components of the viscosity. This function is used when using a gamma law gas equation of state.

Parameters:

- ↔ **grid** supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← **parameters** contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
 Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDTheta,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1,
 Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and
 Grid::nV.

7.13.2.77 void calOldQ0Q1_RT_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for two components of the viscosity. This function is used when using a tabulated equation of state.

Parameters:

- ↔ **grid** supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← **parameters** contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridOld,
 Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit,
 Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nSinThetaIJK,
 Grid::nSinThetaIJP1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit,
 Grid::nU, and Grid::nV.

7.13.2.78 void calOldQ0Q1Q2_RTP_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the three components of the viscosity. This function is used when using a gamma law gas equation of state.

Parameters:

- ↔ **grid** supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← **parameters** contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
 Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDPhi, Grid::nDTheta,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0,
 Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nStartGhostUpdateExplicit,
 Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

7.13.2.79 void calOldQ0Q1Q2_RTP_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the three components of the viscosity. This function is used when using a tabulated equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

7.13.2.80 double dEOS_GL (double dRho, double dE, Parameters parameters)

Calculates the pressure from the energy and density using a γ -law gas.

Parameters:

- ← *dRho* the density of a cell
- ← *dE* the energy of a cell
- ← *parameters* contains various parameters, including γ needed to calculate the pressure.

Returns:

the pressure

This version of [dEOS_GL](#) uses the same value of γ through out the model. The equation of state is given by $\rho(\gamma - 1)E$.

References Parameters::dGamma.

7.13.2.81 double dImplicitEnergyFunction_None (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)

This is an empty function, that isn't even called when no implicit solution is needed. This safe guards against future addition which may need to call an empty function when no implicit solve is being done.

7.13.2.82 double dImplicitEnergyFunction_R (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)

This function is used to determine the agreement of the updated values at $n+1$, with each other in the non-adiabatic energy equation. The `_R` version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*

- ← **dTemps** dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$, dTemps[1]=dT_ip1jk_np1 is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, dTemps[2]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.
- ← **i** is the radial index to evaluate the function at.
- ← **j** is the theta index to evaluate the function at.
- ← **k** is the phi index to evaluate the function at.

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nQ0, Grid::nR, Grid::nT, Grid::nU, and Grid::nU0.

7.13.2.83 double dImplicitEnergyFunction_R_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The **_R** version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← **grid**
- ← **parameters**
- ← **time**
- ← **dTemps** dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$, dTemps[1]=dT_ip1jk_np1 is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, dTemps[2]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.
- ← **i** is the radial index to evaluate the function at.
- ← **j** is the theta index to evaluate the function at.
- ← **k** is the phi index to evaluate the function at.

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nQ0, Grid::nR, Grid::nT, Grid::nU, and Grid::nU0.

7.13.2.84 double dImplicitEnergyFunction_R_LES_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The **_R** version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "**_SB**" suffix ([dImplicitEnergyFunction_R](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$ and time $n + 1$, dTemps[1]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nR, Grid::nT, Grid::nU, and Grid::nU0.

7.13.2.85 double dImplicitEnergyFunction_R_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The _R version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "_SB" suffix ([dImplicitEnergyFunction_R](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$ and time $n + 1$, dTemps[1]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.

← k is the phi index to evaluate the function at.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i .

Boundary Conditions

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `eos::dGetEnergy()`, `eos::dGetOpacity()`, `eos::dGetPressure()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Parameters::eosTable`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDM`, `Grid::nE`, `Grid::nR`, `Grid::nT`, `Grid::nU`, and `Grid::nU0`.

7.13.2.86 `double dImplicitEnergyFunction_RT (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)`

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RT` version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

← *grid*

← *parameters*

← *time*

← *dTemps* `dTemps[0]=dT_ijk_np1` is the temperature at radial position (i, j, k) and time $n + 1$, `dTemps[1]=dT_ip1jk_np1` is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, `dTemps[2]=dT_im1jk_np1` is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, `dTemps[3]=dT_ijp1k_np1` is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, `dTemps[4]=dT_ijm1k_np1` is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.

← i is the radial index to evaluate the function at.

← j is the theta index to evaluate the function at.

← k is the phi index to evaluate the function at.

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `eos::dGetEnergy()`, `eos::dGetOpacity()`, `eos::dGetPressure()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Parameters::eosTable`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDTheta`, `Grid::nE`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIJp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

7.13.2.87 double dImplicitEnergyFunction_RT_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The _RT version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$, dTemps[1]=dT_ip1jk_np1 is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, dTemps[2]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, dTemps[3]=dT_ijp1k_np1 is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, dTemps[4]=dT_ijm1k_np1 is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nSinThetaLJK, Grid::nSinThetaIJp1halfK, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

7.13.2.88 double dImplicitEnergyFunction_RT_LES_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The _RT version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "_SB" suffix (dImplicitEnergyFunction_RT) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$, dTemps[1]=dT_ip1jk_np1 is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, dTemps[2]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, dTemps[3]=dT_ijp1k_np1 is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, dTemps[4]=dT_ijm1k_np1 is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.

- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `eos::dGetEnergy()`, `eos::dGetOpacity()`, `eos::dGetPressure()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Parameters::eosTable`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDTheta`, `Grid::nE`, `Grid::nEddyVisc`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIjp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

7.13.2.89 `double dImplicitEnergyFunction_RT_SB (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)`

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RT` version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the `"_SB"` suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* `dTemps[0]=dT_ijk_np1` is the temperature at radial position (i, j, k) and time $n + 1$, `dTemps[1]=dT_ip1jk_np1` is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, `dTemps[2]=dT_im1jk_np1` is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, `dTemps[3]=dT_ijp1k_np1` is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, `dTemps[4]=dT_ijm1k_np1` is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nE, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

7.13.2.90 double dImplicitEnergyFunction_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The _RTP version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "_SB" suffix ([dImplicitEnergyFunction_RT](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps*, *dTemps*[0]=*dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps*[1]=*dT_ip1jk_np1* is the temperature at radial position ($i + 1, j, k$) and time $n + 1$, *dTemps*[2]=*dT_im1jk_np1* is the temperature at radial position ($i - 1, j, k$) and time $n + 1$, *dTemps*[3]=*dT_ijp1k_np1* is the temperature at radial position ($i, j + 1, k$) and time $n + 1$, *dTemps*[4]=*dT_ijm1k_np1* is the temperature at radial position ($i, j - 1, k$) and time $n + 1$, *dTemps*[5]=*dT_ijkp1_np1* is the temperature at radial position ($i, j, k + 1$) and time $n + 1$, *dTemps*[6]=*dT_ijkm1_np1* is the temperature at radial position ($i, j, k - 1$) and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nT, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.13.2.91 double dImplicitEnergyFunction_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The _RTP version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "_SB" suffix ([dImplicitEnergyFunction_RT](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps*, *dTemps[0]=dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps[1]=dT_ip1jk_np1* is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, *dTemps[2]=dT_im1jk_np1* is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, *dTemps[3]=dT_ijp1k_np1* is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, *dTemps[4]=dT_ijm1k_np1* is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, *dTemps[5]=dT_ijkp1_np1* is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, *dTemps[6]=dT_ijkm1_np1* is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nT, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.13.2.92 double dImplicitEnergyFunction_RTP_LES_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The _RTP version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "_SB" suffix ([dImplicitEnergyFunction_RT](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* *dTemps[0]=dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps[1]=dT_ip1jk_np1* is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, *dTemps[2]=dT_im1jk_np1* is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, *dTemps[3]=dT_ijp1k_np1* is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, *dTemps[4]=dT_ijm1k_np1* is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, *dTemps[5]=dT_ijkp1_np1* is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, *dTemps[6]=dT_ijkm1_np1* is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.

← k is the phi index to evaluate the function at.

Boundary Conditions

assuming V at $ip1half$ is the same as V at i

Boundary Conditions

assuming W at $ip1half$ is the same as W at i

Boundary Conditions

Using $E_{\{i,j,k\}^{n+1/2}}$ for $E_{\{i+1/2,j,k\}^{n+1/2}}$

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `eos::dGetEnergy()`, `eos::dGetOpacity()`, `eos::dGetPressure()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Parameters::eosTable`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nE`, `Grid::nEddyVisc`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIjp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

7.13.2.93 double dImplicitEnergyFunction_RTP_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RTP` version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the `"_SB"` suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

← *grid*

← *parameters*

← *time*

← *dTemps* `dTemps[0]`=`dT_ijk_np1` is the temperature at radial position (i, j, k) and time $n + 1$, `dTemps[1]`=`dT_ip1jk_np1` is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, `dTemps[2]`=`dT_im1jk_np1` is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, `dTemps[3]`=`dT_ijp1k_np1` is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, `dTemps[4]`=`dT_ijm1k_np1` is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, `dTemps[5]`=`dT_ijkp1_np1` is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, `dTemps[6]`=`dT_ijkm1_np1` is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.

← i is the radial index to evaluate the function at.

← j is the theta index to evaluate the function at.

← k is the phi index to evaluate the function at.

Boundary Conditions

Using $E_{\{i,j,k\}^{n+1/2}}$ for $E_{\{i+1/2,j,k\}^{n+1/2}}$

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `eos::dGetEnergy()`, `eos::dGetOpacity()`, `eos::dGetPressure()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Parameters::eosTable`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nE`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIjp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

7.13.2.94 `void implicitSolve_None (Grid & grid, Implicit & implicit, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass, Functions & functions)`

This is an empty function, to be called when no implicit solution is needed. This allows the same code in the main program to be executed wheather or not an implicit solution is being preformed by setting the functon pointer to this functon if there is no implicit solution required.

7.13.2.95 `void implicitSolve_R (Grid & grid, Implicit & implicit, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass, Functions & functions)`

This function solves for temperature corrections based on derivatives of the radial non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coeeficient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equaitons. Solving this system of equaitons provides the corrections needed for the new temperature. This processes is then repeated until the corrections are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equaiton of state.

References `calNewPEKappaGamma_TEOS()`, `Implicit::dAverageRHS`, `Implicit::dCurrentRelTError`, `Implicit::dDerivativeStepFraction`, `Grid::dLocalGridNew`, `Implicit::dMaxErrorInRHS`, `Implicit::dTolerance`, `Functions::fpImplicitEnergyFunction`, `Functions::fpImplicitEnergyFunction_SB`, `Implicit::kspContext`, `Implicit::matCoeff`, `Implicit::nCurrentNumIterations`, `Implicit::nLocDer`, `Implicit::nLocFun`, `Implicit::nMaxNumIterations`, `Implicit::nMaxNumSolverIterations`, `Implicit::nNumDerPerRow`, `Implicit::nNumRowsALocal`, `Implicit::nNumRowsALocalSB`, `ProcTop::nRank`, `Grid::nT`, `Time::nTimeStepIndex`, `Implicit::nTypeDer`, `updateLocalBoundariesNewGrid()`, `Implicit::vecRHS`, `Implicit::vecscatTCorrections`, `Implicit::vecTCorrections`, and `Implicit::vecTCorrectionsLocal`.

7.13.2.96 void implicitSolve_RT (Grid & *grid*, Implicit & *implicit*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*, Functions & *functions*)

This function solves for temperature corrections based on derivatives of the radial-theta non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coefficient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equations. Solving this system of equations provides the corrections needed for the new temperature. This process is then repeated until the corrections are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equation of state.

References calNewPEKappaGamma_TEOS(), Implicit::dAverageRHS, Implicit::dCurrentRelTError, Implicit::dDerivativeStepFraction, Grid::dLocalGridNew, Implicit::dMaxErrorInRHS, Implicit::dTolerance, Functions::fpImplicitEnergyFunction, Functions::fpImplicitEnergyFunction_SB, Implicit::kspContext, Implicit::matCoeff, Implicit::nCurrentNumIterations, Implicit::nLocDer, Implicit::nLocFun, Implicit::nMaxNumIterations, Implicit::nMaxNumSolverIterations, Implicit::nNumDerPerRow, Implicit::nNumRowsALocal, Implicit::nNumRowsALocalSB, ProcTop::nRank, Grid::nT, Time::nTimeStepIndex, Implicit::nTypeDer, updateLocalBoundariesNewGrid(), Implicit::vecRHS, Implicit::vecscatTCorrections, Implicit::vecTCorrections, and Implicit::vecTCorrectionsLocal.

7.13.2.97 void implicitSolve_RTP (Grid & *grid*, Implicit & *implicit*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*, Functions & *functions*)

This function solves for temperature corrections based on derivatives of the radial-theta-phi non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coefficient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equations. Solving this system of equations provides the corrections needed for the new temperature. This process is then repeated until the corrections are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equation of state.

References calNewPEKappaGamma_TEOS(), Implicit::dAverageRHS, Implicit::dCurrentRelTError, Implicit::dDerivativeStepFraction, Grid::dLocalGridNew, Implicit::dMaxErrorInRHS, Implicit::dTolerance, Functions::fpImplicitEnergyFunction, Functions::fpImplicitEnergyFunction_SB, Implicit::kspContext, Implicit::matCoeff, Implicit::nCurrentNumIterations, Implicit::nLocDer, Implicit::nLocFun, Implicit::nMaxNumIterations, Implicit::nMaxNumSolverIterations, Implicit::nNumDerPerRow, Implicit::nNumRowsALocal, Implicit::nNumRowsALocalSB, ProcTop::nRank, Grid::nT, Time::nTimeStepIndex, Implicit::nTypeDer, updateLocalBoundariesNewGrid(), Implicit::vecRHS, Implicit::vecscatTCorrections, Implicit::vecTCorrections, and Implicit::vecTCorrectionsLocal.

7.13.2.98 void initDonorFracAndMaxConVel_R_GL (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection.

tion terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the fuction is for 1D, gamma law calculations.

References Parameters::dDonorFrac, Parameters::dGamma, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

7.13.2.99 void initDonorFracAndMaxConVel_R_TEOS (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the fuction is for 1D, tabulated equation of state calculations.

References Parameters::dDonorFrac, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

7.13.2.100 void initDonorFracAndMaxConVel_RT_GL (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the fuction is for 2D, gamma law calculations.

References Parameters::dDonorFrac, Parameters::dGamma, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

7.13.2.101 void initDonorFracAndMaxConVel_RT_TEOS (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the fuction is for 2D, tabulated equation of state calculations.

References Parameters::dDonorFrac, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

7.13.2.102 void initDonorFracAndMaxConVel_RTP_GL (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity

parameter. This version of the fuction is for 3D, gamma law calculations.

References Parameters::dDonorFrac, Parameters::dGamma, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.13.2.103 void initDonorFracAndMaxConVel_RTP_TEOS (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the fuction is for 3D, tabulated equation of state calculations.

References Parameters::dDonorFrac, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.13.2.104 void initInternalVars (Grid & *grid*, ProcTop & *procTop*, Parameters & *parameters*)

This function function is used to set the initial values of the internal variables. While external variables are initialized from the starting model, internal variables are calculated at startup.

Parameters:

- ↔ *grid* supplies information needed for initilizing internal variables as well as storing the initilized internal variables
- ← *procTop* contians information about processor topology
- ← *parameters* contains parameters used in initializing the internal variables.

Warning:

$\Delta\theta$, $\Delta\phi$, $\sin\theta_{i,j,k}$, $\Delta\cos\theta_{i,j,k}$, all don't have the first zone calculated. At the moment this is a ghost cell that doesn't matter, but it may become a problem if calculations require this quantity. This is an issue for quantities that aren't updated in time, as those that are will have boundary cells updated with periodic boundary conditions.

References Parameters::bEOSGammaLaw, calOldDenave_R(), calOldDenave_RT(), calOldDenave_RTP(), calOldEddyVisc_R_CN(), calOldEddyVisc_R_SM(), calOldEddyVisc_RT_CN(), calOldEddyVisc_RT_SM(), calOldEddyVisc_RTP_CN(), calOldEddyVisc_RTP_SM(), calOldP_GL(), calOldPEKappaGamma_TEOS(), calOldQ0_R_GL(), calOldQ0_R_TEOS(), calOldQ0Q1_RT_GL(), calOldQ0Q1_RT_TEOS(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), Grid::dLocalGridOld, initDonorFracAndMaxConVel_R_GL(), initDonorFracAndMaxConVel_R_TEOS(), initDonorFracAndMaxConVel_RT_GL(), initDonorFracAndMaxConVel_RT_TEOS(), initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), Grid::nCenIntOffset, Grid::nCotThetaIJK, Grid::nCotThetaIjplhalfK, Grid::nDCosThetaIJK, Grid::nDPhi, Grid::nDTheta, Grid::nLocalGridDims, Grid::nNumDims, Grid::nNumGhostCells, Grid::nPhi, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIjplhalfK, Grid::nTheta, and Parameters::nTypeTurbulenceMod.

7.13.2.105 void setInternalVarInf (Grid & *grid*, Parameters & *parameters*)

This function sets the information for internal variables. While external verabile information is derived from the starting model, internal variables infos are set in this function. In other words this function sets the values of [Grid::nVariables](#).

Parameters:

- ↔ ***grid*** supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.
- ← ***parameters*** is used when setting variable infos, since one needs to know if the code is calculating using a gamma law gas, or a tabulated equation of state.

References [Parameters::bEOSGammaLaw](#), [Grid::nCotThetaIJK](#), [Grid::nCotThetaIjp1halfK](#), [Grid::nDCosThetaIJK](#), [Grid::nDenAve](#), [Grid::nDPhi](#), [Grid::nDTheta](#), [Grid::nE](#), [Grid::nEddyVisc](#), [Grid::nGamma](#), [Grid::nKappa](#), [Grid::nNumDims](#), [Grid::nNumIntVars](#), [Grid::nNumVars](#), [Grid::nP](#), [Grid::nQ0](#), [Grid::nQ1](#), [Grid::nQ2](#), [Grid::nSinThetaIJK](#), [Grid::nSinThetaIjp1halfK](#), [Parameters::nTypeTurbulenceMod](#), and [Grid::nVariables](#).

7.13.2.106 void setMainFunctions (Functions & *functions*, ProcTop & *procTop*, Parameters & *parameters*, Grid & *grid*, Time & *time*, Implicit & *implicit*)

Used to set the functions that [main\(\)](#) uses to evolve the input model.

Parameters:

- ***functions*** is of class [Functions](#) and is used to specify the functions called to calculate the evolution of the input model.
- ← ***procTop*** is of type [ProcTop](#). [ProcTop::nRank](#) is used to set different functions based on processor rank. For instance processor rank 1 requires 1D versions of the equations.
- ← ***parameters*** is of class [Parameters](#). It holds various constants and runtime parameters.
- ← ***grid*** of type [Grid](#). This function requires the number of dimensions, specified by [Grid::nNumDims](#).
- ← ***time*** of type [Time](#). This function requires knowledge of the type of time setp being used, specified by [Time::bVariableTimeStep](#).
- ← ***implicit*** of type [Implicit](#). This function needs to know if there is an implicit region, specified when [Implicit::nNumImplicitZones](#)>0.

The functions are picked based on model geometry, and the physics requested or required by the input model, and the configuration file. The specific functions pointers that are set are described in the [Functions](#) class.

References [Parameters::bAdiabatic](#), [Parameters::bEOSGammaLaw](#), [Time::bVariableTimeStep](#), [calDelt_CONST\(\)](#), [calDelt_R_GL\(\)](#), [calDelt_R_TEOS\(\)](#), [calDelt_RT_GL\(\)](#), [calDelt_RT_TEOS\(\)](#), [calDelt_RTP_GL\(\)](#), [calDelt_RTP_TEOS\(\)](#), [calNewD_R\(\)](#), [calNewD_RT\(\)](#), [calNewD_RTP\(\)](#), [calNewDenave_None\(\)](#), [calNewDenave_R\(\)](#), [calNewDenave_RT\(\)](#), [calNewDenave_RTP\(\)](#), [calNewE_R_AD\(\)](#), [calNewE_R_NA\(\)](#), [calNewE_RT_AD\(\)](#), [calNewE_RT_NA\(\)](#), [calNewE_RT_NA_LES\(\)](#), [calNewE_RTP_AD\(\)](#), [calNewE_RTP_NA\(\)](#), [calNewE_RTP_NA_LES\(\)](#), [calNewEddyVisc_None\(\)](#), [calNewEddyVisc_RT_CN\(\)](#), [calNewEddyVisc_RT_SM\(\)](#), [calNewEddyVisc_RTP_CN\(\)](#),

```

calNewEddyVisc_RTP_SM(),    calNewP_GL(),    calNewQ0_R_GL(),    calNewQ0_R_
TEOS(),    calNewQ0Q1_RT_GL(),    calNewQ0Q1_RT_TEOS(),    calNewQ0Q1Q2_RTP_
GL(),    calNewQ0Q1Q2_RTP_TEOS(),    calNewR(),    calNewTPKappaGamma_TEOS(),
calNewU0_R(), calNewU0_RT(), calNewU0_RTP(), calNewVelocities_R(), calNewVelocities_
RT(),    calNewVelocities_RT_LES(),    calNewVelocities_RTP(),    calNewVelocities_
RTP_LES(),    dImplicitEnergyFunction_None(),    dImplicitEnergyFunction_R(),
dImplicitEnergyFunction_R_SB(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_
RT_LES(),    dImplicitEnergyFunction_RT_LES_SB(),    dImplicitEnergyFunction_
RT_SB(),    dImplicitEnergyFunction_RTP(),    dImplicitEnergyFunction_RTP_LES(),
dImplicitEnergyFunction_RTP_LES_SB(),    dImplicitEnergyFunction_RTP_SB(),    Func-
tions::fpCalculateAveDensities,    Functions::fpCalculateDeltat,    Functions::fpCalculateNewAV,
Functions::fpCalculateNewDensities,    Functions::fpCalculateNewEddyVisc,    Func-
tions::fpCalculateNewEnergies,    Functions::fpCalculateNewEOSVars,    Func-
tions::fpCalculateNewGridVelocities,    Functions::fpCalculateNewRadii,    Func-
tions::fpCalculateNewVelocities,    Functions::fpImplicitEnergyFunction,
Functions::fpImplicitEnergyFunction_SB, Functions::fpImplicitSolve, Functions::fpModelWrite,
Functions::fpUpdateLocalBoundaryVelocitiesNewGrid,    Functions::fpWriteWatchZones,
implicitSolve_None(),    implicitSolve_R(),    implicitSolve_RT(),    implicitSolve_RTP(),
modelWrite_GL(), modelWrite_TEOS(), Grid::nNumDims, Implicit::nNumImplicitZones, Proc-
Top::nRank, Parameters::nTypeTurbulenceMod, updateLocalBoundaryVelocitiesNewGrid_R(),
updateLocalBoundaryVelocitiesNewGrid_RT(),    updateLocalBoundaryVelocitiesNewGrid_
RTP(),    writeWatchZones_R_GL(),    writeWatchZones_R_TEOS(),    writeWatchZones_RT_
GL(),    writeWatchZones_RT_TEOS(),    writeWatchZones_RTP_GL(),    and    writeWatchZones_
RTP_TEOS().

```

7.14 physEquations_new.cpp File Reference

```
#include <cmath>
#include <sstream>
#include <signal.h>
#include "exception2.h"
#include "physEquations.h"
#include "dataManipulation.h"
#include "dataMonitoring.h"
#include "global.h"
#include <limits>
```

Functions

- void [setMainFunctions](#) ([Functions](#) &functions, [ProcTop](#) &procTop, [Parameters](#) ¶meters, [Grid](#) &grid, [Time](#) &time, [Implicit](#) &implicit)
- void [setInternalVarInf](#) ([Grid](#) &grid, [Parameters](#) ¶meters)
- void [initInternalVars](#) ([Grid](#) &grid, [ProcTop](#) &procTop, [Parameters](#) ¶meters)
- void [calNewVelocities_R](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_R_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RT_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities_RTP_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_R](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_R_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RT_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU_RTP_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RT](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RT_LES](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewV_RTP](#) ([Grid](#) &grid, [Parameters](#) ¶meters, [Time](#) &time, [ProcTop](#) &procTop)

- void `calNewV_RTP_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewW_RTP` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewW_RTP_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewU0_R` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewU0_RT` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewU0_RTP` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewR` (`Grid` &grid, `Time` &time)
- void `calNewD_R` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewD_RT` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewD_RTP` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_R_AD` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_R_NA` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_R_NA_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RT_AD` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RT_NA` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RT_NA_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RTP_AD` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RTP_NA` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RTP_NA_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, `ProcTop` &procTop)
- void `calNewDenave_None` (`Grid` &grid)
- void `calNewDenave_R` (`Grid` &grid)
- void `calNewDenave_RT` (`Grid` &grid)
- void `calNewDenave_RTP` (`Grid` &grid)
- void `calNewP_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewTPKappaGamma_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewPEKappaGamma_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0_R_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0_R_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0Q1_RT_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0Q1_RT_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0Q1Q2_RTP_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewQ0Q1Q2_RTP_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewEddyVisc_None` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewEddyVisc_R_CN` (`Grid` &grid, `Parameters` ¶meters)
- void `calNewEddyVisc_RT_CN` (`Grid` &grid, `Parameters` ¶meters)

- void `calNewEddyVisc_RTP_CN` (Grid &grid, Parameters ¶meters)
- void `calNewEddyVisc_R_SM` (Grid &grid, Parameters ¶meters)
- void `calNewEddyVisc_RT_SM` (Grid &grid, Parameters ¶meters)
- void `calNewEddyVisc_RTP_SM` (Grid &grid, Parameters ¶meters)
- void `calOldDenave_None` (Grid &grid)
- void `calOldDenave_R` (Grid &grid)
- void `calOldDenave_RT` (Grid &grid)
- void `calOldDenave_RTP` (Grid &grid)
- void `calOldP_GL` (Grid &grid, Parameters ¶meters)
- void `calOldPEKappaGamma_TEOS` (Grid &grid, Parameters ¶meters)
- void `calOldQ0_R_GL` (Grid &grid, Parameters ¶meters)
- void `calOldQ0_R_TEOS` (Grid &grid, Parameters ¶meters)
- void `calOldQ0Q1_RT_GL` (Grid &grid, Parameters ¶meters)
- void `calOldQ0Q1_RT_TEOS` (Grid &grid, Parameters ¶meters)
- void `calOldQ0Q1Q2_RTP_GL` (Grid &grid, Parameters ¶meters)
- void `calOldQ0Q1Q2_RTP_TEOS` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_R_CN` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RT_CN` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RTP_CN` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_R_SM` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RT_SM` (Grid &grid, Parameters ¶meters)
- void `calOldEddyVisc_RTP_SM` (Grid &grid, Parameters ¶meters)
- void `calDelt_R_GL` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_R_TEOS` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RT_GL` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RT_TEOS` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RTP_GL` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_RTP_TEOS` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `calDelt_CONST` (Grid &grid, Parameters ¶meters, Time &time, ProcTop &procTop)
- void `implicitSolve_None` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- void `implicitSolve_R` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- void `implicitSolve_RT` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- void `implicitSolve_RTP` (Grid &grid, Implicit &implicit, Parameters ¶meters, Time &time, ProcTop &procTop, MessPass &messPass, Functions &functions)
- double `dImplicitEnergyFunction_None` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R_SB` (Grid &grid, Parameters ¶meters, Time &time, double dTemps[], int i, int j, int k)

- double `dImplicitEnergyFunction_RT` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RT_SB` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RTP` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RTP_SB` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R_LES_SB` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RT_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RT_LES_SB` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RTP_LES` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_RTP_LES_SB` (`Grid` &grid, `Parameters` ¶meters, `Time` &time, double dTemps[], int i, int j, int k)
- double `dEOS_GL` (double dRho, double dE, `Parameters` parameters)
- void `initDonorFracAndMaxConVel_R_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `initDonorFracAndMaxConVel_R_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `initDonorFracAndMaxConVel_RT_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `initDonorFracAndMaxConVel_RT_TEOS` (`Grid` &grid, `Parameters` ¶meters)
- void `initDonorFracAndMaxConVel_RTP_GL` (`Grid` &grid, `Parameters` ¶meters)
- void `initDonorFracAndMaxConVel_RTP_TEOS` (`Grid` &grid, `Parameters` ¶meters)

7.14.1 Detailed Description

This file is used to specify the functions which contain physics. This includes conservation equations, equation of state, etc.. It also sets function pointers for these functions, so that `main()` will know which functions to call. This implementation also allows the functions called to calculate, for example new densities, to be different depending on the processor. This allows one processor to handle the 1D region and other processors to handle a 3D region.

7.14.2 Function Documentation

7.14.2.1 void calDelt_CONST (`Grid` & *grid*, `Parameters` & *parameters*, `Time` & *time*, `ProcTop` & *procTop*)

This function is used when a constant tie step is desired.

References `Time::dConstTimeStep`, `Time::dDeltat_n`, `Time::dDeltat_nm1half`, `Time::dDeltat_np1half`, `Time::dt`, and `Time::nTimeStepIndex`.

7.14.2.2 void calDelt_R_GL (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial direction only and is compatible with a gamma law gas EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelE_t_E_max, Time::dDelRho_t_Rho_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Parameters::dGamma, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartUpdateExplicit, Time::nTimeStepIndex, Grid::nU, and Grid::nU0.

7.14.2.3 void calDelt_R_TEOS (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial direction only and is compatible with a tabulated EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelRho_t_Rho_max, Time::dDelT_t_T_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartUpdateExplicit, Grid::nT, Time::nTimeStepIndex, Grid::nU, and Grid::nU0.

7.14.2.4 void calDelt_RT_GL (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial and theta directions only and is compatible with a gamma law gas EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelE_t_E_max, Time::dDelRho_t_Rho_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Parameters::dGamma, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, ProcTop::nRank, Grid::nStartUpdateExplicit, Time::nTimeStepIndex, Grid::nU, Grid::nU0, and Grid::nV.

7.14.2.5 void calDelt_RT_TEOS (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial and theta directions and is compatible with a tabulated EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelRho_t_Rho_max, Time::dDelT_t_T_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, ProcTop::nRank, Grid::nStartUpdateExplicit, Grid::nT, Time::nTimeStepIndex, Grid::nU, Grid::nU0, and Grid::nV.

7.14.2.6 void calDelt_RTP_GL (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial, theta and phi directions only and is compatible with a gamma law gas EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities

- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelE_t_E_max, Time::dDelRho_t_Rho_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Parameters::dGamma, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nStartUpdateExplicit, Time::nTimeStepIndex, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.14.2.7 void calDelt_RTP_TEOS (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the time step by considering the sound crossing time in the radial, theta and phi directions and is compatible with a tabulated EOS.

Parameters:

- ← *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ↔ *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology. This function uses [ProcTop::nRank](#) to pass messages.

References Time::dDelRho_t_Rho_max, Time::dDelT_t_T_max, Time::dDeltat_n, Time::dDeltat_nm1half, Time::dDeltat_np1half, Time::dDelUmU0_t_UmU0_max, Time::dDelV_t_V_max, Time::dDelW_t_W_max, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nStartUpdateExplicit, Grid::nT, Time::nTimeStepIndex, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.14.2.8 void calNewD_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new densities using terms in the radial direction only

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.

- ← **procTop** contains information about the processor topology, uses [ProcTop::nRank](#) when reporting negative densities

Boundary Conditions

doesn't allow mass flux through outter interface

References Time::dDeltat_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

7.14.2.9 void calNewD_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new densities using terms in the radial and theta directions

Parameters:

- ↔ **grid** contains the local grid, and will hold the newly updated densities
- ← **parameters** various parameters needed for the calculation
- ← **time** contains time information, e.g. time step, current time etc.
- ← **procTop** contains information about the processor topology, uses [ProcTop::nRank](#) when reporting negative densities

Boundary Conditions

doesn't allow mass flux through outter interface

References Time::dDeltat_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDCosThetaIJK, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, ProcTop::nRank, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

7.14.2.10 void calNewD_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new densities using terms in the radial, theta, and phi directions

Parameters:

- ↔ **grid** contains the local grid, and will hold the newly updated densities
- ← **parameters** various parameters needed for the calculation
- ← **time** contains time information, e.g. time step, current time etc.
- ← **procTop** contains information about the processor topology, uses [ProcTop::nRank](#) when reporting negative densities

Boundary Conditions

doesn't allow mass flux through outter interface

References Time::dDeltat_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDCosThetaIJK, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, ProcTop::nRank, Grid::nSinThetaIj1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.14.2.11 void calNewDenave_None (Grid & *grid*)

This function is a dummy function, and doesn't do anything. In the case of a 1D calculation the average density is undefined, and only the density is used. This is different from the case where the 1D region exists on the rank 0 processor, but the grid as a whole is really 2D or 3D. In which case [calNewDenave_R](#) should be used instead.

Parameters:

↔ *grid*

7.14.2.12 void calNewDenave_R (Grid & *grid*)

This function calculates the horizontal average density in a 3D region. This really just copies the density from the particular radial zone into the averaged density variable. This way it can be used exactly the same way in the 1D region as it is in the 3D region. This is done using the density in the new grid, and places the result into the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References Grid::dLocalGridNew, Grid::nD, Grid::nDenAve, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.14.2.13 void calNewDenave_RT (Grid & *grid*)

This function calculates the horizontal average density in a 2D region from the new grid density and stores the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDCosThetaIJK, Grid::nDenAve, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.14.2.14 void calNewDenave_RTP (Grid & *grid*)

This function calculates the horizontal average density in a 3D region from the new grid density and stores the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDCosThetaIJK, Grid::nDenAve, Grid::nDPhi, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.14.2.15 void calNewE_R_AD (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new adiabatic energies using terms in the radial direction.

Parameters:

↔ *grid* contains the local grid, and will hold the newly updated densities
 ← *parameters* various parameters needed for the calculation
 ← *time* contains time information, e.g. time step, current time etc.
 ← *procTop*

References Time::dDeltat_n, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

7.14.2.16 void calNewE_R_NA (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial direction and includes radiation diffusion terms.

Parameters:

↔ *grid* contains the local grid, and will hold the newly updated densities
 ← *parameters* various parameters needed for the calculation
 ← *time* contains time information, e.g. time step, current time etc.
 ← *procTop*

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nKappa, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, and Grid::nU0.

7.14.2.17 void calNewE_R_NA_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial direction and includes radiation diffusion terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nKappa, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, and Grid::nU0.

7.14.2.18 void calNewE_RT_AD (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new adiabatic energies using terms in the radial and theta directions.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

grid.dLocalGridOld[grid.nE][i+1][j][k] is missing

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing using inner gradient for both

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

7.14.2.19 void calNewE_RT_NA (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial and theta directions and includes radiation diffusion terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0] using flux equals $2\sigma T^4$ at surface.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nKappa, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

7.14.2.20 void calNewE_RT_NA_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial and theta directions and includes radiation diffusion terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Setting energy at surface equal to energy in last zone.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

References Parameters::dAlpha, Parameters::dAlphaExtra, Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dPrt, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nKappa, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

7.14.2.21 `void calNewE_RTP_AD (Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop)`

This function calculates new adiabatic energies using terms in the radial, theta, and phi directions.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to zero.

Boundary Conditions

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.14.2.22 void calNewE_RTP_NA (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial, theta, and phi directions and includes radiation diffusion terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at *i*.

Boundary Conditions

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew,
Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Grid::nCenIntOffset,
Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nE,
Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nKappa, Grid::nP,
Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK,
Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit,
Grid::nT, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.14.2.23 void calNewE_RTP_NA_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates new non-adiabatic energies using terms in the radial, theta, and phi directions and includes radiation diffusion terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated densities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop*

Boundary Conditions

Missing *W* at *i*+1, assuming the same as at *i*

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nE][i+1][j][k]` in calculation of $E_{i+1/2,j,k}$ setting it equal to the value at `i`.

Boundary Conditions

`grid.dLocalGridOld[grid.nDM][i+1][0][0]` and `grid.dLocalGridOld[grid.nE][i+1][j][k]` missing in the calculation of upwind gradient in `dA1`. Using the centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]`

References `Parameters::dAlpha`, `Parameters::dAlphaExtra`, `Time::dDeltat_n`, `Parameters::dDonorFrac`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dPrt`, `Parameters::dSigma`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nE`, `Grid::nEddyVisc`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nKappa`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `ProcTop::nRank`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIjp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

7.14.2.24 void calNewEddyVisc_None (Grid & *grid*, Parameters & *parameters*)

This function is a empty function used as a place holder when no eddy viscosity model is being used.

Parameters:

- ↔ *grid*
- ← *parameters*

7.14.2.25 void calNewEddyVisc_R_CN (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity using a constant times the zoning with only the radial terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

References `Parameters::dEddyViscosity`, `Grid::dLocalGridNew`, `Parameters::dMaxConvectiveVelocity`, `Grid::nCenIntOffset`, `Grid::nEddyVisc`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nR`, `Grid::nStartGhostUpdateExplicit`, and `Grid::nStartUpdateExplicit`.

7.14.2.26 void calNewEddyVisc_R_SM (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity with only the radial terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU.

7.14.2.27 void calNewEddyVisc_RT_CN (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity using a constant times the zoning with only the radial and theta terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.14.2.28 void calNewEddyVisc_RT_SM (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity with only the radial and theta terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

7.14.2.29 void calNewEddyVisc_RTP_CN (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity using a constant times the zoning with only the radial, theta, and phi terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nSinThetaIJK, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.14.2.30 void calNewEddyVisc_RTP_SM (Grid & *grid*, Parameters & *parameters*)

This function calculates the eddy viscosity with only the radial, theta, and phi terms.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

Boundary Conditions

assuming that theta velocity is constant across surface

Boundary Conditions

assume phi velocity is constant across surface

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nCotThetaIJK, Grid::nD, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nSinThetaIJK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.14.2.31 void calNewP_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the pressure. It is calculated using the new values of quantities and places the result in the new grid. It uses a gamma law as given in [dEOS_GL](#) to calculate the pressure.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculations.
- ← *parameters* contains parameters used in calculating the pressure, namely the adiabatic gamma that is used.

References dEOS_GL(), Grid::dLocalGridNew, Grid::nD, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.14.2.32 void calNewPEKappaGamma_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the Energy, pressure and opacity of a cell. It calculates it using the new values of quantities and places the result in the new grid.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← *parameters* contains parameters used in calculating the pressure.

References Grid::dLocalGridNew, Parameters::eosTable, eos::getPEKappaGamma(), Grid::nD, Grid::nE, Grid::nEndGhostUpdateImplicit, Grid::nEndUpdateImplicit, Grid::nGamma, Grid::nKappa, Grid::nP, Grid::nStartGhostUpdateImplicit, Grid::nStartUpdateImplicit, and Grid::nT.

7.14.2.33 void calNewQ0_R_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial component of the viscosity only. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ ***grid*** supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation.
- ← ***parameters*** contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
 Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nR,
 Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU.

7.14.2.34 void calNewQ0_R_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for the radial component of the viscosity only. It uses a sound speed derived from a tabulated equation of state for the calculation.

Parameters:

- ↔ ***grid*** supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← ***parameters*** contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridNew,
 Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit,
 Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nR,
 Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU.

7.14.2.35 void calNewQ0Q1_RT_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial and theta components of the viscosity. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ ***grid*** supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculations.
- ← ***parameters*** contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
 Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1,
 Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit,
 Grid::nStartUpdateExplicit, Grid::nU, and Grid::nV.

7.14.2.36 void calNewQ0Q1_RT_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for two component of the viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridNew,
Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit,
Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR,
Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit,
Grid::nStartUpdateExplicit, Grid::nU, and Grid::nV.

7.14.2.37 void calNewQ0Q1Q2_RTP_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial, theta, and phi componenets of the viscosity. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculations.
- ← *parameters* contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD,
Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0,
Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK,
Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and
Grid::nW.

7.14.2.38 void calNewQ0Q1Q2_RTP_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for the three component of the viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

7.14.2.39 void calNewR (Grid & *grid*, Time & *time*)

This function calculates the radii, from the new radial grid velocities

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated radial velocities
- ← ***time*** contains time information, e.g. time step, current time etc.

References Time::dDeltat_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU0.

7.14.2.40 void calNewTPKappaGamma_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the Temperature, pressure and opacity of a cell. It calculates it using the new vaules of quantities and places the result in the new grid.

Parameters:

- ↔ ***grid*** supplies the input for calculating the pressure and also accepts the result of the pressure calculation
- ← ***parameters*** contains parameters used in calculating the pressure.

References Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dTolerance, Parameters::eosTable, eos::getEAndDTDE(), eos::getPKappaGamma(), Grid::nD, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nKappa, Parameters::nMaxIterations, Grid::nP, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nT.

7.14.2.41 void calNewU0_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*)

This function calculates the radial grid velocity, it does so by considering only the radial terms

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated radial grid velocities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology
- ← ***messPass***

Todo

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, ProcTop::nCoords, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, ProcTop::nNumRadialNeighbors, Grid::nR, ProcTop::nRadialNeighborNeighborIDs, ProcTop::nRadialNeighborRanks, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, MessPass::typeRecvNewVar, and MessPass::typeSendNewVar.

7.14.2.42 void calNewU0_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*)

This function calculates the radial grid velocity, and does it by only considering the radial and theta terms

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial grid velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology
- ↔ *messPass* handles data needed for message passing

Todo

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Boundary Conditions

grid.dLocalGridOld[grid.nD][i+1][j][k] is missing

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, ProcTop::nCoords, Grid::nD, Grid::nDCosThetaIJK, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nLocalGridDims, Grid::nNumGhostCells, ProcTop::nNumRadialNeighbors, Grid::nR, ProcTop::nRadialNeighborNeighborIDs, ProcTop::nRadialNeighborRanks, ProcTop::nRank, Grid::nSinThetaIp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, MessPass::typeRecvNewVar, and MessPass::typeSendNewVar.

7.14.2.43 void calNewU0_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*)

This function calculates the radial grid velocity, and does it by considering all radial, theta and phi terms

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial grid velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.

← *procTop* contains information about the processor topology

↔ *messPass* handles data needed for message passing

Todo

At some point I will likely want to make this function compatible with a 3D domain decomposition instead of a purely radial domain decomposition.

Boundary Conditions

grid.dLocalGridOld[grid.nD][i+1][j][k] is missing

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, ProcTop::nCoords, Grid::nD, Grid::nDCosThetaIJK, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nLocalGridDims, Grid::nNumGhostCells, ProcTop::nNumRadialNeighbors, Grid::nR, ProcTop::nRadialNeighborNeighborIDs, ProcTop::nRadialNeighborRanks, ProcTop::nRank, Grid::nSinThetaIj1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, Grid::nW, MessPass::typeRecvNewVar, and MessPass::typeSendNewVar.

7.14.2.44 void calNewU_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by only considering the radial terms.

Parameters:

↔ *grid* contains the local grid, and will hold the newly updated radial velocities

← *parameters* various parameters needed for the calculation

← *time* contains time information, e.g. time step, current time etc.

← *procTop* contains information about the processor topology

Boundary Conditions

Missing grid.dLocalGridOld[grid.nD][nICen+1][j][k] in calculation of $\rho_{i+1/2}$, setting it to 0.0

Boundary Conditions

Missing grid.dLocalGridOld[grid.nP][nICen+1][j][k] in calculation of S_1 , setting it to $-1.0 \cdot \text{grid.dLocalGridOld[grid.nP][nICen][j][k]}$.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of centered A_1 gradient, setting it to zero.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nU][i+1][j][k] and grid.dLocalGridOld[grid.nDM][nICen+1][0][0] in calculation of upwind gradient, when moving inward. Using centered gradient instead.

References Parameters::dAlpha, Parameters::dAlphaExtra, Time::dDeltat_n, Parameters::dDonorFrac, Parameters::dG, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nM, Grid::nP, Grid::nQ0, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

7.14.2.45 void calNewU_R_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by only considering the radial terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2}$, setting it to 0.0

Boundary Conditions

missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` using velocity at i

Boundary Conditions

Assuming eddy viscosity outside model is zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to $-1.0 * \text{grid.dLocalGridOld[grid.nP][nICen][j][k]}$.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

References Parameters::dAlpha, Parameters::dAlphaExtra, Time::dDeltat_n, Parameters::dDonorFrac, Parameters::dG, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nM, Grid::nP, Grid::nQ0, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

7.14.2.46 void calNewU_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by only considering the radial and theta terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities

- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nD][nICen+1][j][k]` in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

Boundary Conditions

assuming theta velocity is constant across surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0]` in calculation of $\langle \rho \rangle_{i+1/2}$, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nP][nICen+1][j][k]` in calculation of S_1 , setting it to $-1.0 * dP_ijk_n$.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

References `Parameters::dAlpha`, `Parameters::dAlphaExtra`, `Time::dDeltat_n`, `Parameters::dDonorFrac`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDTheta`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nM`, `Grid::nP`, `Grid::nQ0`, `Grid::nR`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

7.14.2.47 void calNewU_RT_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by only considering the radial and theta terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities

- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Boundary Conditions

Missing density outside of surface, setting it to zero.

Boundary Conditions

Missing density outside model, setting it to zero.

Boundary Conditions

assuming theta and phi velocity same outside star as inside.

Boundary Conditions

Assuming theta velocities are constant across surface.

Boundary Conditions

assuming that V at $i+1$ is equal to v at i .

Boundary Conditions

Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

Boundary Conditions

assume viscosity is zero outside the star.

Boundary Conditions

Missing mass outside model, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

References `Parameters::dAlpha`, `Parameters::dAlphaExtra`, `Time::dDeltat_n`, `Parameters::dDonorFrac`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaIJK`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDTheta`, `Grid::nEddyVisc`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nM`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIjp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

7.14.2.48 void calNewU_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by including all radial, theta and phi terms.

Parameters:

- ↔ ***grid*** contains the local grid, and will hold the newly updated radial velocities
- ← ***parameters*** various parameters needed for the calculation
- ← ***time*** contains time information, e.g. time step, current time etc.
- ← ***procTop*** contains information about the processor topology

Boundary Conditions

missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` in calculation of $u_{i+1,j,k}$ setting $u_{i+1,j,k} = u_{i+1/2,j,k}$.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nD][i+1][j][k]` in calculation of $\rho_{i+1/2,j,k}$, setting it to zero.

Boundary Conditions

assuming theta velocity is constant across the surface.

Boundary Conditions

assuming phi velocity is constant across the surface.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDenAve][nICen+1][0][0]` in calculation of $\langle \rho \rangle_{i+1/2}$ setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of centered A_1 gradient, setting it equal to [Parameters::dAlpha](#) `grid.dLocalGridOld[grid.nDM][nICen][0][0]`.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using [Parameters::dAlpha](#) `*grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

References `Parameters::dAlpha`, `Parameters::dAlphaExtra`, `Time::dDeltat_n`, `Parameters::dDonorFrac`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nM`, `Grid::nP`, `Grid::nQ0`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

7.14.2.49 void calNewU_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the radial velocity, and does it by including all radial, theta and phi terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated radial velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nDM][i+1][0][0]` in calculation of S_1 using `Parameters::dAlpha * grid.dLocalGridOld[grid.nDM][nICen][0][0]` instead.

Boundary Conditions

Missing density outside of surface, setting it to zero.

Boundary Conditions

Missing density outside model, setting it to zero.

Boundary Conditions

assuming theta and phi velocity same outside star as inside.

Boundary Conditions

Assuming theta velocities are constant across surface.

Boundary Conditions

assuming that V at $i+1$ is equal to v at i .

Boundary Conditions

Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

Boundary Conditions

assume viscosity is zero outside the star.

Boundary Conditions

Missing mass outside model, setting it to zero.

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nU][i+1][j][k]` and `grid.dLocalGridOld[grid.nDM][nICen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

References Parameters::dAlpha, Parameters::dAlphaExtra, Time::dDeltat_n, Parameters::dDonorFrac, Parameters::dG, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaIJK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nM, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIj1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.14.2.50 void calNewV_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the theta velocity, and does it by only considering the radial and theta terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

grid.dLocalGridOld[grid.nV][i+1][j+1][k] is missing

Boundary Conditions

missing upwind gradient, using centred gradient instead

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

7.14.2.51 void calNewV_RT_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the theta velocity, and does it by only considering the radial and theta terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Assuming density outside star is zero

Boundary Conditions

Assuming theta velocity is constant across surface.

Boundary Conditions

Assuming eddy viscosity is zero at surface.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaIjp1halfK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

7.14.2.52 void calNewV_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the theta velocity, and does it by considering the radial, theta, and phi terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Assuming theta and phi velocities are the same at the surface of the star as just inside the star.

Boundary Conditions

ussing cetnered gradient for upwind gradient outside star at surface.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaIjp1halfK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.14.2.53 void calNewV_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the theta velocity, and does it by considering the radial, theta, and phi terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities

- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

Assuming density outside star is zero

Boundary Conditions

Assuming theta velocity is constant across surface.

Boundary Conditions

Assuming eddy viscosity is zero at surface.

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaIjp1halfK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.14.2.54 void calNewVelocities_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls a function that calculate the radial velocity. Calls the function [calNewU_R](#) to calculate radial velocity, including only radial terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

References calNewU_R().

7.14.2.55 void calNewVelocities_R_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls a function that calculate the radial velocity. Calls the function [calNewU_R](#) to calculate radial velocity, including only radial terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

References calNewU_R_LES().

7.14.2.56 void calNewVelocities_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls two other functions that calculate the radial and theta velocities. Calls the two functions [calNewU_RT](#) and [calNewV_RT](#) to calculate radial and theta velocities, including both radial and theta terms.

Parameters:

- ↔ ***grid*** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← ***parameters*** contains parameters used in the calculation of the new velocities.
- ← ***time*** contains time step information, current time step, and current time
- ← ***procTop*** contains processor topology information

References [calNewU_RT\(\)](#), and [calNewV_RT\(\)](#).

7.14.2.57 void calNewVelocities_RT_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls two other functions that calculate the radial and theta velocities. Calls the two functions [calNewU_RT](#) and [calNewV_RT](#) to calculate radial and theta velocities, including both radial and theta terms.

Parameters:

- ↔ ***grid*** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← ***parameters*** contains parameters used in the calculation of the new velocities.
- ← ***time*** contains time step information, current time step, and current time
- ← ***procTop*** contains processor topology information

References [calNewU_RT_LES\(\)](#), and [calNewV_RT_LES\(\)](#).

7.14.2.58 void calNewVelocities_RTP (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls three other functions that calculate the radial, theta and phi velocities. Calls the two functions [calNewU_RTP](#), [calNewV_RTP](#) and [calNewW_RTP](#) to calculate radial, theta, and phi velocities, including radial, theta, and phi terms.

Parameters:

- ↔ ***grid*** contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← ***parameters*** contains parameters used in the calculation of the new velocities.
- ← ***time*** contains time step information, current time step, and current time
- ← ***procTop*** contains processor topology information

References [calNewU_RTP\(\)](#), [calNewV_RTP\(\)](#), and [calNewW_RTP\(\)](#).

7.14.2.59 void calNewVelocities_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function simply calls three other functions that calculate the radial, theta and phi velocities. Calls the two functions [calNewU_RTP](#), [calNewV_RTP](#) and [calNewW_RTP](#) to calculate radial, theta, and phi velocities, including radial, theta, and phi terms.

Parameters:

- ↔ *grid* contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
- ← *parameters* contains parameters used in the calculation of the new velocities.
- ← *time* contains time step information, current time step, and current time
- ← *procTop* contains processor topology information

References [calNewU_RTP_LES\(\)](#), [calNewV_RTP_LES\(\)](#), and [calNewW_RTP_LES\(\)](#).

7.14.2.60 void calNewW_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the phi velocity, and does it by only considering the radial, theta, and phi terms.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` assuming that the phi velocity at the outer most interface is the same as the phi velocity in the center of the zone.

Boundary Conditions

missing `grid.dLocalGridOld[grid.nW][i+1][j][k]` in outer most zone. This is needed to calculate the upwind gradient for donor cell. The centered gradient is used instead when moving in the negative direction.

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaIJK`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

7.14.2.61 void calNewW_RTP_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*)

This function calculates the phi velocity, and does it by only considering the radial, theta, and phi terms. It also includes the terms for including real viscosity, used in the LES.

Parameters:

- ↔ *grid* contains the local grid, and will hold the newly updated theta velocities
- ← *parameters* various parameters needed for the calculation
- ← *time* contains time information, e.g. time step, current time etc.
- ← *procTop* contains information about the processor topology

Boundary Conditions

assume theta and phi velocities are constant across surface

Boundary Conditions

assume eddy viscosity is zero at surface

Boundary Conditions

assume upwind gradient is the same as centered gradient across surface

References Time::dDeltat_n, Parameters::dDonorFrac, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaIJK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.14.2.62 void calOldDenave_None (Grid & *grid*)

This function is a dummy function, and doesn't do anything. In the case of a 1D calculation the average density is undefined, and only the density is used. This is different from the case where the 1D region exists on the rank 0 processor, but the grid as a whole is really 2D or 3D. In which case [calOldDenave_R](#) should be used instead.

7.14.2.63 void calOldDenave_R (Grid & *grid*)

This function does nothing as the averaged density is not needed in 1D calculations.

Parameters:

- ↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References Grid::dLocalGridOld, Grid::nD, Grid::nDenAve, Grid::nEndGhostUpdateExplicit, Grid::nEndGhostUpdateImplicit, Grid::nEndUpdateExplicit, Grid::nEndUpdateImplicit, Grid::nStartGhostUpdateExplicit, Grid::nStartGhostUpdateImplicit, Grid::nStartUpdateExplicit, and Grid::nStartUpdateImplicit.

7.14.2.64 void calOldDenave_RT (Grid & *grid*)

This function calculates the horizontal average density in a 2D region. This function differs from [calNewDenave_RT](#) in that it calculates the average density from the old grid density and stores the result in the old grid. While [calNewDenave_RT](#) calculates the average density from the new grid density and places the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD,
 Grid::nDCosThetaIJK, Grid::nDenAve, Grid::nEndGhostUpdateExplicit,
 Grid::nEndGhostUpdateImplicit, Grid::nEndUpdateExplicit, Grid::nEndUpdateImplicit,
 Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartGhostUpdateImplicit,
 Grid::nStartUpdateExplicit, and Grid::nStartUpdateImplicit.

7.14.2.65 void calOldDenave_RTP (Grid & *grid*)

This function calculates the horizontal average density in a 3D region. This function differs from [calNewDenave_RTP](#) in that it calculates the average density from the old grid density and stores the result in the old grid. While [calNewDenave_RTP](#) calculates the average density from the new grid density and places the result in the new grid.

Parameters:

↔ *grid* supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.

References Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD,
 Grid::nDCosThetaIJK, Grid::nDenAve, Grid::nDPhi, Grid::nEndGhostUpdateExplicit,
 Grid::nEndGhostUpdateImplicit, Grid::nEndUpdateExplicit, Grid::nEndUpdateImplicit,
 Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartGhostUpdateImplicit,
 Grid::nStartUpdateExplicit, and Grid::nStartUpdateImplicit.

7.14.2.66 void calOldEddyVisc_R_CN (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity using a constant times the zoning including only the radial terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld,
 Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nEddyVisc,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nNumGhostCells, Grid::nR,
 Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.14.2.67 void calOldEddyVisc_R_SM (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity including only the radial terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution. It uses the Smagorinsky model for calculating the eddy viscosity.

Parameters:

↔ *grid* supplies the input for calculating the eddy viscosity.

← *parameters* contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridOld, Grid::nCenIntOffset,
 Grid::nD, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit,
 Grid::nNumGhostCells, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit,
 and Grid::nU.

7.14.2.68 void calOldEddyVisc_RT_CN (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity using a constant times the zoning including only the radial and theta terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nNumGhostCells, Grid::nR, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.14.2.69 void calOldEddyVisc_RT_SM (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity including only the radial and theta terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution. It uses the Smagorinsky model for calculating the eddy viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nNumGhostCells, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

7.14.2.70 void calOldEddyVisc_RTP_CN (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity using a constant times the zoning including the radial, theta, and phi terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nNumGhostCells, Grid::nR, Grid::nSinThetaIJK, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.14.2.71 void calOldEddyVisc_RTP_SM (Grid & *grid*, Parameters & *parameters*)

Calculates the eddy viscosity including the radial, theta, and phi terms. It puts the result into the old grid. This function is used to initialize the eddy viscosity when the code begins execution. It uses the Smagorinsky model for calculating the eddy viscosity.

Parameters:

- ↔ *grid* supplies the input for calculating the eddy viscosity.
- ← *parameters* contains parameters used in calculating the eddy viscosity.

Boundary Conditions

assuming that theta velocity is constant across surface

Boundary Conditions

assume phi velocity is constant across surface

References Parameters::dEddyViscosity, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nCotThetaIJK, Grid::nD, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nNumGhostCells, Grid::nR, Grid::nSinThetaIJK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.14.2.72 void calOldP_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the pressure using a gamma law gas, calculate by [dEOS_GL](#).

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the results of the pressure calculations
- ← *parameters* contains parameters used in calculating the pressure, namely the value of the adiabatic gamma

References dEOS_GL(), Grid::dLocalGridOld, Grid::nD, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

7.14.2.73 void calOldPEKappaGamma_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the pressure, energy, opacity, and adiabatic index of a cell. It calculates it using the old values of quantities and places the result in the old grid. This function is used to initialize the internal variables pressure, energy and kappa, and is suitable for both 1D and 3D calculations.

Parameters:

- ↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation

← *parameters* contains parameters used in calculating the pressure.

References Grid::dLocalGridOld, Parameters::eosTable, eos::getPEKappaGamma(),
 Grid::nD, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndGhostUpdateImplicit,
 Grid::nEndUpdateExplicit, Grid::nEndUpdateImplicit, Grid::nGamma,
 Grid::nKappa, Grid::nNumGhostCells, Grid::nP, Grid::nStartGhostUpdateExplicit,
 Grid::nStartGhostUpdateImplicit, Grid::nStartUpdateExplicit, Grid::nStartUpdateImplicit,
 and Grid::nT.

7.14.2.74 void calOldQ0_R_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the radial component of the viscosity only. This function is used when using a gamma law gas equation of state.

Parameters:

↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation

← *parameters* contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
 Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nR,
 Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU.

7.14.2.75 void calOldQ0_R_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for 1D viscosity only.

Parameters:

↔ *grid* supplies the input for calculating the pressure and also accepts the result of the pressure calculation

← *parameters* contains parameters used in calculating the artificial viscosity.

Todo

should use new P and rho

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridOld,
 Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit,
 Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nR, Grid::nStartGhostUpdateExplicit,
 Grid::nStartUpdateExplicit, and Grid::nU.

7.14.2.76 void calOldQ0Q1_RT_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the two components of the viscosity. This function is used when using a gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
 Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDTheta,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1,
 Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit,
 Grid::nStartUpdateExplicit, Grid::nU, and Grid::nV.

7.14.2.77 void calOldQ0Q1_RT_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for two components of the viscosity. This function is used when using a tabulated equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridOld,
 Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit,
 Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nSinThetaIJK,
 Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit,
 Grid::nU, and Grid::nV.

7.14.2.78 void calOldQ0Q1Q2_RTP_GL (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old vaules of quantities and places the result in the old grid. It does this for the three components of the viscosity. This function is used when using a gamma law gas equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma,
 Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDPhi, Grid::nDTheta,
 Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0,
 Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK,
 Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and
 Grid::nW.

7.14.2.79 void calOldQ0Q1Q2_RTP_TEOS (Grid & *grid*, Parameters & *parameters*)

This function calculates the artificial viscosity of a cell. It calculates it using the old values of quantities and places the result in the old grid. It does this for the three components of the viscosity. This function is used when using a tabulated equation of state.

Parameters:

- ↔ *grid* supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
- ← *parameters* contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

7.14.2.80 double dEOS_GL (double *dRho*, double *dE*, Parameters *parameters*)

Calculates the pressure from the energy and density using a γ -law gas.

Parameters:

- ← *dRho* the density of a cell
- ← *dE* the energy of a cell
- ← *parameters* contains various parameters, including γ needed to calculate the pressure.

Returns:

the pressure

This version of [dEOS_GL](#) uses the same value of γ through out the model. The equation of state is given by $\rho(\gamma - 1)E$.

References Parameters::dGamma.

7.14.2.81 double dImplicitEnergyFunction_None (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This is an empty function, that isn't even called when no implicit solution is needed. This safe guards against future addition which may need to call an empty function when no implicit solve is being done.

7.14.2.82 double dImplicitEnergyFunction_R (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n+1$, with each other in the non-adiabatic energy equation. The `_R` version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$, dTemps[1]=dT_ip1jk_np1 is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, dTemps[2]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nQ0, Grid::nR, Grid::nT, Grid::nU, and Grid::nU0.

7.14.2.83 double dImplicitEnergyFunction_R_LES (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The *_R* version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$, dTemps[1]=dT_ip1jk_np1 is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, dTemps[2]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nQ0, Grid::nR, Grid::nT, Grid::nU, and Grid::nU0.

7.14.2.84 double dImplicitEnergyFunction_R_LES_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The *_R* version of the function contains only the radial

terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "_SB" suffix ([dImplicitEnergyFunction_R](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$ and time $n + 1$, dTemps[1]=dT_im1jk_np1 is the temperature at radial position ($i - 1, j, k$) and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nEddyVisc, Grid::nR, Grid::nT, Grid::nU, and Grid::nU0.

7.14.2.85 double dImplicitEnergyFunction_R_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The _R version of the function contains only the radial terms, and should be used for purely radial calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "_SB" suffix ([dImplicitEnergyFunction_R](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*

- ← **dTemps** dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$ and time $n + 1$, dTemps[1]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$.
- ← **i** is the radial index to evaluate the function at.
- ← **j** is the theta index to evaluate the function at.
- ← **k** is the phi index to evaluate the function at.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nE][i+1][j][k] in calculation of $E_{i+1/2,j,k}$ setting it equal to value at i.

Boundary Conditions

grid.dLocalGridOld[grid.nDM][i+1][0][0] and grid.dLocalGridOld[grid.nE][i+1][j][k] missing in the calculation of upwind gradient in dA1. Using the centered gradient instead.

Boundary Conditions

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nR, Grid::nT, Grid::nU, and Grid::nU0.

7.14.2.86 double dImplicitEnergyFunction_RT (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The _RT version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← **grid**
- ← **parameters**
- ← **time**
- ← **dTemps** dTemps[0]=dT_ijk_np1 is the temperature at radial position (i, j, k) and time $n + 1$, dTemps[1]=dT_ip1jk_np1 is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, dTemps[2]=dT_im1jk_np1 is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, dTemps[3]=dT_ijp1k_np1 is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, dTemps[4]=dT_ijm1k_np1 is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.
- ← **i** is the radial index to evaluate the function at.
- ← **j** is the theta index to evaluate the function at.
- ← **k** is the phi index to evaluate the function at.

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDTheta, Grid::nE, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIJp1halfK, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

7.14.2.87 `double dImplicitEnergyFunction_RT_LES (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)`

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RT` version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* `dTemps[0]=dT_ijk_np1` is the temperature at radial position (i, j, k) and time $n + 1$, `dTemps[1]=dT_ip1jk_np1` is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, `dTemps[2]=dT_im1jk_np1` is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, `dTemps[3]=dT_ijp1k_np1` is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, `dTemps[4]=dT_ijm1k_np1` is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `eos::dGetEnergy()`, `eos::dGetOpacity()`, `eos::dGetPressure()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dPrt`, `Parameters::dSigma`, `Parameters::eosTable`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDTheta`, `Grid::nE`, `Grid::nEddyVisc`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIjp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

7.14.2.88 `double dImplicitEnergyFunction_RT_LES_SB (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)`

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RT` version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the `"_SB"` suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* `dTemps[0]=dT_ijk_np1` is the temperature at radial position (i, j, k) and time $n + 1$, `dTemps[1]=dT_ip1jk_np1` is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, `dTemps[2]=dT_im1jk_np1` is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, `dTemps[3]=dT_ijp1k_np1` is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, `dTemps[4]=dT_ijm1k_np1` is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.

- ← i is the radial index to evaluate the function at.
- ← j is the theta index to evaluate the function at.
- ← k is the phi index to evaluate the function at.

Boundary Conditions

assuming V at $ip1half$ is the same as V at i

Boundary Conditions

Assuming energy outside model is the same as the energy in the last zone inside the model.

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

References `Parameters::dAlpha`, `Parameters::dAlphaExtra`, `Time::dDeltat_n`, `Parameters::dDonorFrac`, `eos::dGetEnergy()`, `eos::dGetOpacity()`, `eos::dGetPressure()`,
`Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dPrt`, `Parameters::dSigma`, `Parameters::eosTable`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`,
`Grid::nDTheta`, `Grid::nE`, `Grid::nEddyVisc`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaIJK`,
`Grid::nSinThetaIJp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

7.14.2.89 `double dImplicitEnergyFunction_RT_SB (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)`

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RT` version of the function contains only the radial and theta terms, and should be used for radial-theta calculations. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the `"_SB"` suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* `dTemps[0]`=`dT_ijk_np1` is the temperature at radial position (i, j, k) and time $n + 1$, `dTemps[1]`=`dT_ip1jk_np1` is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, `dTemps[2]`=`dT_im1jk_np1` is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, `dTemps[3]`=`dT_ijp1k_np1` is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, `dTemps[4]`=`dT_ijm1k_np1` is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$.
- ← i is the radial index to evaluate the function at.
- ← j is the theta index to evaluate the function at.
- ← k is the phi index to evaluate the function at.

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `eos::dGetEnergy()`, `eos::dGetOpacity()`, `eos::dGetPressure()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Parameters::eosTable`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDTheta`, `Grid::nE`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIjp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

7.14.2.90 `double dImplicitEnergyFunction_RTP (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)`

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RTP` version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "`_SB`" suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps*, *dTemps*[0]=*dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps*[1]=*dT_ip1jk_np1* is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, *dTemps*[2]=*dT_im1jk_np1* is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, *dTemps*[3]=*dT_ijp1k_np1* is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, *dTemps*[4]=*dT_ijm1k_np1* is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, *dTemps*[5]=*dT_ijkp1_np1* is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, *dTemps*[6]=*dT_ijkm1_np1* is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `eos::dGetEnergy()`, `eos::dGetOpacity()`, `eos::dGetPressure()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Parameters::eosTable`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nE`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIjp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

7.14.2.91 `double dImplicitEnergyFunction_RTP_LES (Grid & grid, Parameters & parameters, Time & time, double dTemps[], int i, int j, int k)`

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RTP` version of the function contains terms for all

three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "_SB" suffix ([dImplicitEnergyFunction_RT](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps*, *dTemps[0]*=*dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps[1]*=*dT_ip1jk_np1* is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, *dTemps[2]*=*dT_im1jk_np1* is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, *dTemps[3]*=*dT_ijp1k_np1* is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, *dTemps[4]*=*dT_ijm1k_np1* is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, *dTemps[5]*=*dT_ijkp1_np1* is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, *dTemps[6]*=*dT_ijkm1_np1* is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.
- ← *i* is the radial index to evaluate the function at.
- ← *j* is the theta index to evaluate the function at.
- ← *k* is the phi index to evaluate the function at.

References Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dPrt, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nT, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.14.2.92 double dImplicitEnergyFunction_RTP_LES_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The _RTP version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the "_SB" suffix ([dImplicitEnergyFunction_RT](#)) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* *dTemps[0]*=*dT_ijk_np1* is the temperature at radial position (i, j, k) and time $n + 1$, *dTemps[1]*=*dT_ip1jk_np1* is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, *dTemps[2]*=*dT_im1jk_np1* is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, *dTemps[3]*=*dT_ijp1k_np1* is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, *dTemps[4]*=*dT_ijm1k_np1* is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, *dTemps[5]*=*dT_ijkp1_np1* is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, *dTemps[6]*=*dT_ijkm1_np1* is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.

- ← i is the radial index to evaluate the function at.
- ← j is the theta index to evaluate the function at.
- ← k is the phi index to evaluate the function at.

Boundary Conditions

assuming V at $ip1half$ is the same as V at i

Boundary Conditions

assuming W at $ip1half$ is the same as W at i

Boundary Conditions

Using $E_{\{i,j,k\}^{n+1/2}}$ for $E_{\{i+1/2,j,k\}^{n+1/2}}$

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0]` using flux equals $2\sigma T^4$ at surface.

References Parameters::dAlpha, Parameters::dAlphaExtra, Time::dDeltat_n, Parameters::dDonorFrac, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dPrt, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIjp1halfK, Grid::nT, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.14.2.93 double dImplicitEnergyFunction_RTP_SB (Grid & *grid*, Parameters & *parameters*, Time & *time*, double *dTemps*[], int *i*, int *j*, int *k*)

This function is used to determine the agreement of the updated values at $n + 1$, with each other in the non-adiabatic energy equation. The `_RTP` version of the function contains terms for all three directions, and should be used for calculations involving all three directions. This function can also be used for calculating numerical derivatives by varying the input temperatures. This function differs from the version without the `"_SB"` suffix (`dImplicitEnergyFunction_RT`) in that it is tailored to the surface boundary region.

Parameters:

- ← *grid*
- ← *parameters*
- ← *time*
- ← *dTemps* `dTemps[0]=dT_ijk_np1` is the temperature at radial position (i, j, k) and time $n + 1$, `dTemps[1]=dT_ip1jk_np1` is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$, `dTemps[2]=dT_im1jk_np1` is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$, `dTemps[3]=dT_ijp1k_np1` is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$, `dTemps[4]=dT_ijm1k_np1` is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$, `dTemps[5]=dT_ijkp1_np1` is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$, `dTemps[6]=dT_ijkm1_np1` is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$.

- ← i is the radial index to evaluate the function at.
- ← j is the theta index to evaluate the function at.
- ← k is the phi index to evaluate the function at.

Boundary Conditions

Using $E_{\{i,j,k\}^{n+1/2}}$ for $E_{\{i+1/2,j,k\}^{n+1/2}}$

Boundary Conditions

Using centered gradient for upwind gradient when motion is into the star at the surface

Boundary Conditions

Missing `grid.dLocalGridOld[grid.nT][i+1][0][0]` using flux equals $2\sigma T^4$ at surface.

References `Time::dDeltat_n`, `Parameters::dDonorFrac`, `eos::dGetEnergy()`, `eos::dGetOpacity()`, `eos::dGetPressure()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Parameters::eosTable`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nE`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIJP1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

7.14.2.94 `void implicitSolve_None (Grid & grid, Implicit & implicit, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass, Functions & functions)`

This is an empty function, to be called when no implicit solution is needed. This allows the same code in the main program to be executed wheather or not an implicit solution is being preformed by setting the funciton pointer to this funciton if there is no implicit solution required.

7.14.2.95 `void implicitSolve_R (Grid & grid, Implicit & implicit, Parameters & parameters, Time & time, ProcTop & procTop, MessPass & messPass, Functions & functions)`

This function solves for temperature corrections based on derivatives of the radial non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coefferient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equaitons. Solving this system of equaitons provides the corrections needed for the new temperature. This processes is then repeated until the corrections are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equaiton of state.

References `calNewPEKappaGamma_TEOS()`, `Implicit::dAverageRHS`, `Implicit::dCurrentRelTError`, `Implicit::dDerivativeStepFraction`, `Grid::dLocalGridNew`, `Implicit::dMaxErrorInRHS`, `Implicit::dTolerance`, `Functions::fpImplicitEnergyFunction`, `Functions::fpImplicitEnergyFunction_SB`, `Implicit::kspContext`, `Implicit::matCoeff`, `Implicit::nCurrentNumIterations`, `Implicit::nLocDer`, `Implicit::nLocFun`, `Implicit::nMaxNumIterations`, `Implicit::nMaxNumSolverIterations`, `Implicit::nNumDerPerRow`, `Implicit::nNumRowsALocal`, `Implicit::nNumRowsALocalSB`, `ProcTop::nRank`, `Grid::nT`, `Time::nTimeStepIndex`, `Implicit::nTypeDer`, `updateLocalBoundariesNewGrid()`, `Implicit::vecRHS`, `Implicit::vecscatTCorrections`, `Implicit::vecTCorrections`, and `Implicit::vecTCorrectionsLocal`.

7.14.2.96 void implicitSolve_RT (Grid & *grid*, Implicit & *implicit*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*, Functions & *functions*)

This function solves for temperature corrections based on derivatives of the radial-theta non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coefficient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equations. Solving this system of equations provides the corrections needed for the new temperature. This process is then repeated until the corrections are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equation of state.

References calNewPEKappaGamma_TEOS(), Implicit::dAverageRHS, Implicit::dCurrentRelTError, Implicit::dDerivativeStepFraction, Grid::dLocalGridNew, Implicit::dMaxErrorInRHS, Implicit::dTolerance, Functions::fpImplicitEnergyFunction, Functions::fpImplicitEnergyFunction_SB, Implicit::kspContext, Implicit::matCoeff, Implicit::nCurrentNumIterations, Implicit::nLocDer, Implicit::nLocFun, Implicit::nMaxNumIterations, Implicit::nMaxNumSolverIterations, Implicit::nNumDerPerRow, Implicit::nNumRowsALocal, Implicit::nNumRowsALocalSB, ProcTop::nRank, Grid::nT, Time::nTimeStepIndex, Implicit::nTypeDer, updateLocalBoundariesNewGrid(), Implicit::vecRHS, Implicit::vecscatTCorrections, Implicit::vecTCorrections, and Implicit::vecTCorrectionsLocal.

7.14.2.97 void implicitSolve_RTP (Grid & *grid*, Implicit & *implicit*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop*, MessPass & *messPass*, Functions & *functions*)

This function solves for temperature corrections based on derivatives of the radial-theta-phi non-adiabatic energy equation with respect to the new temperature. It then uses these derivatives as entries in the coefficient matrix. The discrepancy in the balance of the energy equation with the new temperature, energy, pressure, and opacity are included as the right hand side of the system of equations. Solving this system of equations provides the corrections needed for the new temperature. This process is then repeated until the corrections are small. At this point the new temperature is used to update the energy, pressure, and opacity in the new grid via the equation of state.

References calNewPEKappaGamma_TEOS(), Implicit::dAverageRHS, Implicit::dCurrentRelTError, Implicit::dDerivativeStepFraction, Grid::dLocalGridNew, Implicit::dMaxErrorInRHS, Implicit::dTolerance, Functions::fpImplicitEnergyFunction, Functions::fpImplicitEnergyFunction_SB, Implicit::kspContext, Implicit::matCoeff, Implicit::nCurrentNumIterations, Implicit::nLocDer, Implicit::nLocFun, Implicit::nMaxNumIterations, Implicit::nMaxNumSolverIterations, Implicit::nNumDerPerRow, Implicit::nNumRowsALocal, Implicit::nNumRowsALocalSB, ProcTop::nRank, Grid::nT, Time::nTimeStepIndex, Implicit::nTypeDer, updateLocalBoundariesNewGrid(), Implicit::vecRHS, Implicit::vecscatTCorrections, Implicit::vecTCorrections, and Implicit::vecTCorrectionsLocal.

7.14.2.98 void initDonorFracAndMaxConVel_R_GL (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advec-

tion terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the fuction is for 1D, gamma law calculations.

References Parameters::dDonorFrac, Parameters::dGamma, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

7.14.2.99 void initDonorFracAndMaxConVel_R_TEOS (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the fuction is for 1D, tabulated equation of state calculations.

References Parameters::dDonorFrac, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

7.14.2.100 void initDonorFracAndMaxConVel_RT_GL (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the fuction is for 2D, gamma law calculations.

References Parameters::dDonorFrac, Parameters::dGamma, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

7.14.2.101 void initDonorFracAndMaxConVel_RT_TEOS (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the fuction is for 2D, tabulated equation of state calculations.

References Parameters::dDonorFrac, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

7.14.2.102 void initDonorFracAndMaxConVel_RTP_GL (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity

parameter. This version of the fuction is for 3D, gamma law calculations.

References Parameters::dDonorFrac, Parameters::dGamma, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.14.2.103 void initDonorFracAndMaxConVel_RTP_TEOS (Grid & *grid*, Parameters & *parameters*)

Initializes the donor fraction, and the maximum convective velocity when starting a calculation. The donor fraction is used to determine the amount of upwinded donor cell to use in advection terms. The maximum convective velocity is used for calculation of constant eddy viscosity parameter. This version of the fuction is for 3D, tabulated equation of state calculations.

References Parameters::dDonorFrac, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Parameters::dMaxConvectiveVelocity_c, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

7.14.2.104 void initInternalVars (Grid & *grid*, ProcTop & *procTop*, Parameters & *parameters*)

This function function is used to set the initial values of the internal variables. While external variables are initialized from the starting model, internal variables are calculated at startup.

Parameters:

- ↔ *grid* supplies information needed for initilizing internal variables as well as storing the initilized internal variables
- ← *procTop* contians information about processor topology
- ← *parameters* contains parameters used in initializing the internal variables.

Warning:

$\Delta\theta$, $\Delta\phi$, $\sin\theta_{i,j,k}$, $\Delta\cos\theta_{i,j,k}$, all don't have the first zone calculated. At the moment this is a ghost cell that doesn't matter, but it may become a problem if calculations require this quantity. This is an issue for quantities that aren't updated in time, as those that are will have boundary cells updated with periodic boundary conditions.

References Parameters::bEOSGammaLaw, calOldDenave_R(), calOldDenave_RT(), calOldDenave_RTP(), calOldEddyVisc_R_CN(), calOldEddyVisc_R_SM(), calOldEddyVisc_RT_CN(), calOldEddyVisc_RT_SM(), calOldEddyVisc_RTP_CN(), calOldEddyVisc_RTP_SM(), calOldP_GL(), calOldPEKappaGamma_TEOS(), calOldQ0_R_GL(), calOldQ0_R_TEOS(), calOldQ0Q1_RT_GL(), calOldQ0Q1_RT_TEOS(), calOldQ0Q1Q2_RTP_GL(), calOldQ0Q1Q2_RTP_TEOS(), Grid::dLocalGridOld, initDonorFracAndMaxConVel_R_GL(), initDonorFracAndMaxConVel_R_TEOS(), initDonorFracAndMaxConVel_RT_GL(), initDonorFracAndMaxConVel_RT_TEOS(), initDonorFracAndMaxConVel_RTP_GL(), initDonorFracAndMaxConVel_RTP_TEOS(), Grid::nCenIntOffset, Grid::nCotThetaIJK, Grid::nCotThetaIjplhalfK, Grid::nDCosThetaIJK, Grid::nDPhi, Grid::nDTheta, Grid::nLocalGridDims, Grid::nNumDims, Grid::nNumGhostCells, Grid::nPhi, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaIjplhalfK, Grid::nTheta, and Parameters::nTypeTurbulenceMod.

7.14.2.105 void setInternalVarInf (Grid & *grid*, Parameters & *parameters*)

This function sets the information for internal variables. While external verabile information is derived from the starting model, internal variables infos are set in this function. In other words this function sets the values of [Grid::nVariables](#).

Parameters:

- ↔ ***grid*** supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.
- ← ***parameters*** is used when setting variable infos, since one needs to know if the code is calculating using a gamma law gas, or a tabulated equation of state.

References [Parameters::bEOSGammaLaw](#), [Grid::nCotThetaIJK](#), [Grid::nCotThetaIjp1halfK](#), [Grid::nDCosThetaIJK](#), [Grid::nDenAve](#), [Grid::nDPhi](#), [Grid::nDTheta](#), [Grid::nE](#), [Grid::nEddyVisc](#), [Grid::nGamma](#), [Grid::nKappa](#), [Grid::nNumDims](#), [Grid::nNumIntVars](#), [Grid::nNumVars](#), [Grid::nP](#), [Grid::nQ0](#), [Grid::nQ1](#), [Grid::nQ2](#), [Grid::nSinThetaIJK](#), [Grid::nSinThetaIjp1halfK](#), [Parameters::nTypeTurbulenceMod](#), and [Grid::nVariables](#).

7.14.2.106 void setMainFunctions (Functions & *functions*, ProcTop & *procTop*, Parameters & *parameters*, Grid & *grid*, Time & *time*, Implicit & *implicit*)

Used to set the functions that [main\(\)](#) uses to evolve the input model.

Parameters:

- ***functions*** is of class [Functions](#) and is used to specify the functions called to calculate the evolution of the input model.
- ← ***procTop*** is of type [ProcTop](#). [ProcTop::nRank](#) is used to set different functions based on processor rank. For instance processor rank 1 requires 1D versions of the equations.
- ← ***parameters*** is of class [Parameters](#). It holds various constants and runtime parameters.
- ← ***grid*** of type [Grid](#). This function requires the number of dimensions, specified by [Grid::nNumDims](#).
- ← ***time*** of type [Time](#). This function requires knowledge of the type of time setp being used, specified by [Time::bVariableTimeStep](#).
- ← ***implicit*** of type [Implicit](#). This function needs to know if there is an implicit region, specified when [Implicit::nNumImplicitZones](#)>0.

The functions are picked based on model geometry, and the physics requested or required by the input model, and the configuration file. The specific functions pointers that are set are described in the [Functions](#) class.

References [Parameters::bAdiabatic](#), [Parameters::bEOSGammaLaw](#), [Time::bVariableTimeStep](#), [calDelt_CONST\(\)](#), [calDelt_R_GL\(\)](#), [calDelt_R_TEOS\(\)](#), [calDelt_RT_GL\(\)](#), [calDelt_RT_TEOS\(\)](#), [calDelt_RTP_GL\(\)](#), [calDelt_RTP_TEOS\(\)](#), [calNewD_R\(\)](#), [calNewD_RT\(\)](#), [calNewD_RTP\(\)](#), [calNewDenave_None\(\)](#), [calNewDenave_R\(\)](#), [calNewDenave_RT\(\)](#), [calNewDenave_RTP\(\)](#), [calNewE_R_AD\(\)](#), [calNewE_R_NA\(\)](#), [calNewE_RT_AD\(\)](#), [calNewE_RT_NA\(\)](#), [calNewE_RT_NA_LES\(\)](#), [calNewE_RTP_AD\(\)](#), [calNewE_RTP_NA\(\)](#), [calNewE_RTP_NA_LES\(\)](#), [calNewEddyVisc_None\(\)](#), [calNewEddyVisc_RT_CN\(\)](#), [calNewEddyVisc_RT_SM\(\)](#), [calNewEddyVisc_RTP_CN\(\)](#),

```

calNewEddyVisc_RTP_SM(),    calNewP_GL(),    calNewQ0_R_GL(),    calNewQ0_R_
TEOS(),    calNewQ0Q1_RT_GL(),    calNewQ0Q1_RT_TEOS(),    calNewQ0Q1Q2_RTP_
GL(),    calNewQ0Q1Q2_RTP_TEOS(),    calNewR(),    calNewTPKappaGamma_TEOS(),
calNewU0_R(), calNewU0_RT(), calNewU0_RTP(), calNewVelocities_R(), calNewVelocities_
RT(),    calNewVelocities_RT_LES(),    calNewVelocities_RTP(),    calNewVelocities_
RTP_LES(),    dImplicitEnergyFunction_None(),    dImplicitEnergyFunction_R(),
dImplicitEnergyFunction_R_SB(), dImplicitEnergyFunction_RT(), dImplicitEnergyFunction_
RT_LES(),    dImplicitEnergyFunction_RT_LES_SB(),    dImplicitEnergyFunction_
RT_SB(),    dImplicitEnergyFunction_RTP(),    dImplicitEnergyFunction_RTP_LES(),
dImplicitEnergyFunction_RTP_LES_SB(),    dImplicitEnergyFunction_RTP_SB(),    Func-
tions::fpCalculateAveDensities,    Functions::fpCalculateDeltat,    Functions::fpCalculateNewAV,
Functions::fpCalculateNewDensities,    Functions::fpCalculateNewEddyVisc,    Func-
tions::fpCalculateNewEnergies,    Functions::fpCalculateNewEOSVars,    Func-
tions::fpCalculateNewGridVelocities,    Functions::fpCalculateNewRadii,    Func-
tions::fpCalculateNewVelocities,    Functions::fpImplicitEnergyFunction,
Functions::fpImplicitEnergyFunction_SB, Functions::fpImplicitSolve, Functions::fpModelWrite,
Functions::fpUpdateLocalBoundaryVelocitiesNewGrid,    Functions::fpWriteWatchZones,
implicitSolve_None(),    implicitSolve_R(),    implicitSolve_RT(),    implicitSolve_RTP(),
modelWrite_GL(), modelWrite_TEOS(), Grid::nNumDims, Implicit::nNumImplicitZones, Proc-
Top::nRank, Parameters::nTypeTurbulenceMod, updateLocalBoundaryVelocitiesNewGrid_R(),
updateLocalBoundaryVelocitiesNewGrid_RT(),    updateLocalBoundaryVelocitiesNewGrid_
RTP(),    writeWatchZones_R_GL(),    writeWatchZones_R_TEOS(),    writeWatchZones_RT_
GL(),    writeWatchZones_RT_TEOS(),    writeWatchZones_RTP_GL(),    and    writeWatchZones_
RTP_TEOS().

```

7.15 userguide.h File Reference

7.15.1 Detailed Description

Contains the text for the "Using and Modifying SPHERLS" section of this manual.

7.16 watchzone.cpp File Reference

```
#include "watchzone.h"  
#include "exception2.h"  
#include <sstream>
```

7.16.1 Detailed Description

This file holds the implementation of the watchzone class.

7.17 watchzone.h File Reference

```
#include <string>
#include <fstream>
```

Classes

- class [WatchZone](#)

7.17.1 Detailed Description

This file holds the definition of the watchzone class.

Index

- ~eos
 - eos, [50](#)
 - /home/cgeroux/SPHERLS/src/eos.cpp, [111](#)
 - /home/cgeroux/SPHERLS/src/eos.h, [112](#)
- average3DTo1DBoundariesNew
 - dataManipulation.cpp, [114](#)
 - dataManipulation.h, [123](#)
- average3DTo1DBoundariesOld
 - dataManipulation.cpp, [114](#)
 - dataManipulation.h, [124](#)
- bAdiabatic
 - Parameters, [96](#)
- bDump
 - Output, [93](#)
- bEOSGammaLaw
 - Parameters, [96](#)
- bFileExists
 - dataMonitoring.cpp, [132](#)
 - dataMonitoring.h, [137](#)
- bPrint
 - Output, [93](#)
- bVariableTimeStep
 - Time, [108](#)
- calDelt_CONST
 - physEquations.cpp, [152](#)
 - physEquations.h, [198](#)
 - physEquations_HEAD.cpp, [254](#)
 - physEquations_new.cpp, [307](#)
- calDelt_R_GL
 - physEquations.cpp, [152](#)
 - physEquations.h, [198](#)
 - physEquations_HEAD.cpp, [254](#)
 - physEquations_new.cpp, [307](#)
- calDelt_R_TEOS
 - physEquations.cpp, [153](#)
 - physEquations.h, [199](#)
 - physEquations_HEAD.cpp, [255](#)
 - physEquations_new.cpp, [308](#)
- calDelt_RT_GL
 - physEquations.cpp, [153](#)
 - physEquations.h, [199](#)
 - physEquations_HEAD.cpp, [255](#)
- physEquations_new.cpp, [308](#)
- calDelt_RT_TEOS
 - physEquations.cpp, [153](#)
 - physEquations.h, [200](#)
 - physEquations_HEAD.cpp, [256](#)
 - physEquations_new.cpp, [309](#)
- calDelt_RTP_GL
 - physEquations.cpp, [154](#)
 - physEquations.h, [200](#)
 - physEquations_HEAD.cpp, [256](#)
 - physEquations_new.cpp, [309](#)
- calDelt_RTP_TEOS
 - physEquations.cpp, [154](#)
 - physEquations.h, [201](#)
 - physEquations_HEAD.cpp, [257](#)
 - physEquations_new.cpp, [310](#)
- calNewD_R
 - physEquations.cpp, [154](#)
 - physEquations.h, [201](#)
 - physEquations_HEAD.cpp, [257](#)
 - physEquations_new.cpp, [310](#)
- calNewD_RT
 - physEquations.cpp, [155](#)
 - physEquations.h, [202](#)
 - physEquations_HEAD.cpp, [258](#)
 - physEquations_new.cpp, [311](#)
- calNewD_RTP
 - physEquations.cpp, [155](#)
 - physEquations.h, [202](#)
 - physEquations_HEAD.cpp, [258](#)
 - physEquations_new.cpp, [311](#)
- calNewDenave_None
 - physEquations.cpp, [156](#)
 - physEquations.h, [202](#)
 - physEquations_HEAD.cpp, [259](#)
 - physEquations_new.cpp, [312](#)
- calNewDenave_R
 - physEquations.cpp, [156](#)
 - physEquations.h, [203](#)
 - physEquations_HEAD.cpp, [259](#)
 - physEquations_new.cpp, [312](#)
- calNewDenave_RT
 - physEquations.cpp, [156](#)
 - physEquations.h, [203](#)
 - physEquations_HEAD.cpp, [259](#)

- physEquations_new.cpp, 312
- calNewDenave_RTP
 - physEquations.cpp, 156
 - physEquations.h, 203
 - physEquations_HEAD.cpp, 259
 - physEquations_new.cpp, 312
- calNewE_R_AD
 - physEquations.cpp, 157
 - physEquations.h, 204
 - physEquations_HEAD.cpp, 260
 - physEquations_new.cpp, 313
- calNewE_R_NA
 - physEquations.cpp, 157
 - physEquations.h, 204
 - physEquations_HEAD.cpp, 260
 - physEquations_new.cpp, 313
- calNewE_R_NA_LES
 - physEquations.cpp, 157
 - physEquations.h, 205
 - physEquations_HEAD.cpp, 261
 - physEquations_new.cpp, 314
- calNewE_RT_AD
 - physEquations.cpp, 158
 - physEquations.h, 205
 - physEquations_HEAD.cpp, 261
 - physEquations_new.cpp, 314
- calNewE_RT_NA
 - physEquations.cpp, 158
 - physEquations.h, 206
 - physEquations_HEAD.cpp, 262
 - physEquations_new.cpp, 315
- calNewE_RT_NA_LES
 - physEquations.cpp, 159
 - physEquations.h, 206
 - physEquations_HEAD.cpp, 262
 - physEquations_new.cpp, 315
- calNewE_RTP_AD
 - physEquations.cpp, 159
 - physEquations.h, 207
 - physEquations_HEAD.cpp, 263
 - physEquations_new.cpp, 316
- calNewE_RTP_NA
 - physEquations.cpp, 160
 - physEquations.h, 208
 - physEquations_HEAD.cpp, 263
 - physEquations_new.cpp, 316
- calNewE_RTP_NA_LES
 - physEquations.cpp, 160
 - physEquations.h, 208
 - physEquations_HEAD.cpp, 264
 - physEquations_new.cpp, 317
- calNewEddyVisc_None
 - physEquations.cpp, 161
 - physEquations.h, 209
- physEquations_HEAD.cpp, 265
- physEquations_new.cpp, 318
- calNewEddyVisc_R_CN
 - physEquations.cpp, 161
 - physEquations.h, 209
 - physEquations_HEAD.cpp, 265
 - physEquations_new.cpp, 318
- calNewEddyVisc_R_SM
 - physEquations.cpp, 162
 - physEquations.h, 210
 - physEquations_HEAD.cpp, 265
 - physEquations_new.cpp, 318
- calNewEddyVisc_RT_CN
 - physEquations.cpp, 162
 - physEquations.h, 210
 - physEquations_HEAD.cpp, 266
 - physEquations_new.cpp, 319
- calNewEddyVisc_RT_SM
 - physEquations.cpp, 162
 - physEquations.h, 210
 - physEquations_HEAD.cpp, 266
 - physEquations_new.cpp, 319
- calNewEddyVisc_RTP_CN
 - physEquations.cpp, 162
 - physEquations.h, 210
 - physEquations_HEAD.cpp, 266
 - physEquations_new.cpp, 319
- calNewEddyVisc_RTP_SM
 - physEquations.cpp, 163
 - physEquations.h, 211
 - physEquations_HEAD.cpp, 266
 - physEquations_new.cpp, 319
- calNewP_GL
 - physEquations.cpp, 163
 - physEquations.h, 211
 - physEquations_HEAD.cpp, 267
 - physEquations_new.cpp, 320
- calNewPEKappaGamma_TEOS
 - physEquations.cpp, 163
 - physEquations.h, 212
 - physEquations_HEAD.cpp, 267
 - physEquations_new.cpp, 320
- calNewQ0_R_GL
 - physEquations.cpp, 163
 - physEquations.h, 212
 - physEquations_HEAD.cpp, 267
 - physEquations_new.cpp, 320
- calNewQ0_R_TEOS
 - physEquations.cpp, 164
 - physEquations.h, 212
 - physEquations_HEAD.cpp, 268
 - physEquations_new.cpp, 321
- calNewQ0Q1_RT_GL
 - physEquations.cpp, 164

- physEquations.h, 213
- physEquations_HEAD.cpp, 268
- physEquations_new.cpp, 321
- calNewQ0Q1_RT_TEOS
 - physEquations.cpp, 164
 - physEquations.h, 213
 - physEquations_HEAD.cpp, 268
 - physEquations_new.cpp, 321
- calNewQ0Q1Q2_RTP_GL
 - physEquations.cpp, 165
 - physEquations.h, 213
 - physEquations_HEAD.cpp, 269
 - physEquations_new.cpp, 322
- calNewQ0Q1Q2_RTP_TEOS
 - physEquations.cpp, 165
 - physEquations.h, 214
 - physEquations_HEAD.cpp, 269
 - physEquations_new.cpp, 322
- calNewR
 - physEquations.cpp, 165
 - physEquations.h, 214
 - physEquations_HEAD.cpp, 270
 - physEquations_new.cpp, 323
- calNewTPKappaGamma_TEOS
 - physEquations.cpp, 165
 - physEquations.h, 215
 - physEquations_HEAD.cpp, 270
 - physEquations_new.cpp, 323
- calNewU0_R
 - physEquations.cpp, 166
 - physEquations.h, 215
 - physEquations_HEAD.cpp, 270
 - physEquations_new.cpp, 323
- calNewU0_RT
 - physEquations.cpp, 166
 - physEquations.h, 215
 - physEquations_HEAD.cpp, 271
 - physEquations_new.cpp, 324
- calNewU0_RTP
 - physEquations.cpp, 167
 - physEquations.h, 216
 - physEquations_HEAD.cpp, 271
 - physEquations_new.cpp, 324
- calNewU_R
 - physEquations.cpp, 167
 - physEquations.h, 217
 - physEquations_HEAD.cpp, 272
 - physEquations_new.cpp, 325
- calNewU_R_LES
 - physEquations.cpp, 168
 - physEquations.h, 217
 - physEquations_HEAD.cpp, 272
 - physEquations_new.cpp, 325
- calNewU_RT
 - physEquations.cpp, 168
 - physEquations.h, 218
 - physEquations_HEAD.cpp, 273
 - physEquations_new.cpp, 326
- calNewU_RT_LES
 - physEquations.cpp, 169
 - physEquations.h, 219
 - physEquations_HEAD.cpp, 274
 - physEquations_new.cpp, 327
- calNewU_RTP
 - physEquations.cpp, 170
 - physEquations.h, 220
 - physEquations_HEAD.cpp, 275
 - physEquations_new.cpp, 328
- calNewU_RTP_LES
 - physEquations.cpp, 171
 - physEquations.h, 221
 - physEquations_HEAD.cpp, 276
 - physEquations_new.cpp, 329
- calNewV_RT
 - physEquations.cpp, 172
 - physEquations.h, 223
 - physEquations_HEAD.cpp, 278
 - physEquations_new.cpp, 331
- calNewV_RT_LES
 - physEquations.cpp, 173
 - physEquations.h, 223
 - physEquations_HEAD.cpp, 278
 - physEquations_new.cpp, 331
- calNewV_RTP
 - physEquations.cpp, 173
 - physEquations.h, 224
 - physEquations_HEAD.cpp, 279
 - physEquations_new.cpp, 332
- calNewV_RTP_LES
 - physEquations.cpp, 174
 - physEquations.h, 224
 - physEquations_HEAD.cpp, 279
 - physEquations_new.cpp, 332
- calNewVelocities_R
 - physEquations.cpp, 174
 - physEquations.h, 225
 - physEquations_HEAD.cpp, 280
 - physEquations_new.cpp, 333
- calNewVelocities_R_LES
 - physEquations.cpp, 175
 - physEquations.h, 225
 - physEquations_HEAD.cpp, 280
 - physEquations_new.cpp, 333
- calNewVelocities_RT
 - physEquations.cpp, 175
 - physEquations.h, 226
 - physEquations_HEAD.cpp, 280
 - physEquations_new.cpp, 333

- calNewVelocities_RT_LES
 - physEquations.cpp, 175
 - physEquations.h, 226
 - physEquations_HEAD.cpp, 281
 - physEquations_new.cpp, 334
- calNewVelocities_RTP
 - physEquations.cpp, 176
 - physEquations.h, 226
 - physEquations_HEAD.cpp, 281
 - physEquations_new.cpp, 334
- calNewVelocities_RTP_LES
 - physEquations.cpp, 176
 - physEquations.h, 227
 - physEquations_HEAD.cpp, 281
 - physEquations_new.cpp, 334
- calNewW_RTP
 - physEquations.cpp, 176
 - physEquations.h, 227
 - physEquations_HEAD.cpp, 282
 - physEquations_new.cpp, 335
- calNewW_RTP_LES
 - physEquations.cpp, 177
 - physEquations.h, 228
 - physEquations_HEAD.cpp, 282
 - physEquations_new.cpp, 335
- calOldDenave_None
 - physEquations.cpp, 177
 - physEquations.h, 228
 - physEquations_HEAD.cpp, 283
 - physEquations_new.cpp, 336
- calOldDenave_R
 - physEquations.cpp, 177
 - physEquations.h, 229
 - physEquations_HEAD.cpp, 283
 - physEquations_new.cpp, 336
- calOldDenave_RT
 - physEquations.cpp, 178
 - physEquations.h, 229
 - physEquations_HEAD.cpp, 283
 - physEquations_new.cpp, 336
- calOldDenave_RTP
 - physEquations.cpp, 178
 - physEquations.h, 229
 - physEquations_HEAD.cpp, 284
 - physEquations_new.cpp, 337
- calOldEddyVisc_R_CN
 - physEquations.cpp, 178
 - physEquations.h, 230
 - physEquations_HEAD.cpp, 284
 - physEquations_new.cpp, 337
- calOldEddyVisc_R_SM
 - physEquations.cpp, 178
 - physEquations.h, 230
 - physEquations_HEAD.cpp, 284
- physEquations_new.cpp, 337
- calOldEddyVisc_RT_CN
 - physEquations.cpp, 179
 - physEquations.h, 230
 - physEquations_HEAD.cpp, 284
 - physEquations_new.cpp, 337
- calOldEddyVisc_RT_SM
 - physEquations.cpp, 179
 - physEquations.h, 230
 - physEquations_HEAD.cpp, 285
 - physEquations_new.cpp, 338
- calOldEddyVisc_RTP_CN
 - physEquations.cpp, 179
 - physEquations.h, 231
 - physEquations_HEAD.cpp, 285
 - physEquations_new.cpp, 338
- calOldEddyVisc_RTP_SM
 - physEquations.cpp, 179
 - physEquations.h, 231
 - physEquations_HEAD.cpp, 285
 - physEquations_new.cpp, 338
- calOldP_GL
 - physEquations.cpp, 180
 - physEquations.h, 232
 - physEquations_HEAD.cpp, 286
 - physEquations_new.cpp, 339
- calOldPEKappaGamma_TEOS
 - physEquations.cpp, 180
 - physEquations.h, 232
 - physEquations_HEAD.cpp, 286
 - physEquations_new.cpp, 339
- calOldQ0_R_GL
 - physEquations.cpp, 180
 - physEquations.h, 232
 - physEquations_HEAD.cpp, 287
 - physEquations_new.cpp, 340
- calOldQ0_R_TEOS
 - physEquations.cpp, 181
 - physEquations.h, 233
 - physEquations_HEAD.cpp, 287
 - physEquations_new.cpp, 340
- calOldQ0Q1_RT_GL
 - physEquations.cpp, 181
 - physEquations.h, 233
 - physEquations_HEAD.cpp, 287
 - physEquations_new.cpp, 340
- calOldQ0Q1_RT_TEOS
 - physEquations.cpp, 181
 - physEquations.h, 234
 - physEquations_HEAD.cpp, 288
 - physEquations_new.cpp, 341
- calOldQ0Q1Q2_RTP_GL
 - physEquations.cpp, 181
 - physEquations.h, 234

- physEquations_HEAD.cpp, 288
- physEquations_new.cpp, 341
- calOldQ0Q1Q2_RTP_TEOS
 - physEquations.cpp, 182
 - physEquations.h, 234
 - physEquations_HEAD.cpp, 288
 - physEquations_new.cpp, 341
- dA
 - Parameters, 97
- dAlpha
 - Parameters, 98
- dataManipulation.cpp, 113
 - average3DTolDBoundariesNew, 114
 - average3DTolDBoundariesOld, 114
 - fin, 115
 - init, 115
 - initImplicitCalculation, 116
 - initUpdateLocalBoundaries, 117
 - modelRead, 117
 - modelWrite_GL, 118
 - modelWrite_TEOS, 119
 - setupLocalGrid, 119
 - updateLocalBoundaries, 119
 - updateLocalBoundariesNewGrid, 120
 - updateLocalBoundaryVelocitiesNewGrid_-R, 120
 - updateLocalBoundaryVelocitiesNewGrid_-RT, 121
 - updateLocalBoundaryVelocitiesNewGrid_-RTP, 121
 - updateNewGridWithOld, 121
 - updateOldGrid, 121
- dataManipulation.h, 123
 - average3DTolDBoundariesNew, 123
 - average3DTolDBoundariesOld, 124
 - fin, 124
 - init, 125
 - initImplicitCalculation, 126
 - initUpdateLocalBoundaries, 126
 - modelRead, 127
 - modelWrite_GL, 128
 - modelWrite_TEOS, 128
 - setupLocalGrid, 129
 - updateLocalBoundaries, 129
 - updateLocalBoundariesNewGrid, 130
 - updateLocalBoundaryVelocitiesNewGrid_-R, 130
 - updateLocalBoundaryVelocitiesNewGrid_-RT, 130
 - updateLocalBoundaryVelocitiesNewGrid_-RTP, 131
 - updateNewGridWithOld, 131
 - updateOldGrid, 131
- dataMonitoring.cpp, 132
 - bFileExists, 132
 - finWatchZones, 133
 - initWatchZones, 133
 - writeWatchZones_R_GL, 133
 - writeWatchZones_R_TEOS, 134
 - writeWatchZones_RT_GL, 134
 - writeWatchZones_RT_TEOS, 135
 - writeWatchZones_RTP_GL, 135
 - writeWatchZones_RTP_TEOS, 135
- dataMonitoring.h, 137
 - bFileExists, 137
 - finWatchZones, 137
 - initWatchZones, 138
 - writeWatchZones_R_GL, 138
 - writeWatchZones_R_TEOS, 138
 - writeWatchZones_RT_GL, 139
 - writeWatchZones_RT_TEOS, 139
 - writeWatchZones_RTP_GL, 140
 - writeWatchZones_RTP_TEOS, 140
- dAverageRHS
 - Implicit, 89
- dAVThreshold
 - Parameters, 98
- dConstTimeStep
 - Time, 108
- dCurrentRelTErr
 - Implicit, 88
- dDelE_t_E_max
 - Time, 109
- dDelRho_t_Rho_max
 - Time, 108
- dDelT_t_T_max
 - Time, 109
- dDeltat_n
 - Time, 107
- dDeltat_nm1half
 - Time, 107
- dDeltat_np1half
 - Time, 106
- dDerivativeStepFraction
 - Implicit, 88
- dDonorFrac
 - Parameters, 98
- dDRhoDP
 - eos, 53
- dDumpFrequencyTime
 - Output, 93
- dEddyViscosity
 - Parameters, 99
- dEndTime
 - Time, 107
- dEndTimer
 - Performance, 101

- dEOS_GL
 - physEquations.cpp, [182](#)
 - physEquations.h, [235](#)
 - physEquations_HEAD.cpp, [289](#)
 - physEquations_new.cpp, [342](#)
- dG
 - Parameters, [97](#)
- dGamma
 - Parameters, [97](#)
- dGetEnergy
 - eos, [52](#)
- dGetOpacity
 - eos, [53](#)
- dGetPressure
 - eos, [52](#)
- dImplicitEnergyFunction_None
 - physEquations.cpp, [182](#)
 - physEquations.h, [235](#)
 - physEquations_HEAD.cpp, [289](#)
 - physEquations_new.cpp, [342](#)
- dImplicitEnergyFunction_R
 - physEquations.cpp, [182](#)
 - physEquations.h, [235](#)
 - physEquations_HEAD.cpp, [289](#)
 - physEquations_new.cpp, [342](#)
- dImplicitEnergyFunction_R_LES
 - physEquations.cpp, [183](#)
 - physEquations.h, [236](#)
 - physEquations_HEAD.cpp, [290](#)
 - physEquations_new.cpp, [343](#)
- dImplicitEnergyFunction_R_LES_SB
 - physEquations.cpp, [183](#)
 - physEquations.h, [236](#)
 - physEquations_HEAD.cpp, [290](#)
 - physEquations_new.cpp, [343](#)
- dImplicitEnergyFunction_R_SB
 - physEquations.cpp, [184](#)
 - physEquations.h, [237](#)
 - physEquations_HEAD.cpp, [291](#)
 - physEquations_new.cpp, [344](#)
- dImplicitEnergyFunction_RT
 - physEquations.cpp, [185](#)
 - physEquations.h, [238](#)
 - physEquations_HEAD.cpp, [292](#)
 - physEquations_new.cpp, [345](#)
- dImplicitEnergyFunction_RT_LES
 - physEquations.cpp, [185](#)
 - physEquations.h, [238](#)
 - physEquations_HEAD.cpp, [292](#)
 - physEquations_new.cpp, [345](#)
- dImplicitEnergyFunction_RT_LES_SB
 - physEquations.cpp, [186](#)
 - physEquations.h, [239](#)
 - physEquations_HEAD.cpp, [293](#)
- physEquations_new.cpp, [346](#)
- dImplicitEnergyFunction_RT_SB
 - physEquations.cpp, [187](#)
 - physEquations.h, [240](#)
 - physEquations_HEAD.cpp, [294](#)
 - physEquations_new.cpp, [347](#)
- dImplicitEnergyFunction_RTP
 - physEquations.cpp, [188](#)
 - physEquations.h, [241](#)
 - physEquations_HEAD.cpp, [295](#)
 - physEquations_new.cpp, [348](#)
- dImplicitEnergyFunction_RTP_LES
 - physEquations.cpp, [188](#)
 - physEquations.h, [241](#)
 - physEquations_HEAD.cpp, [295](#)
 - physEquations_new.cpp, [348](#)
- dImplicitEnergyFunction_RTP_LES_SB
 - physEquations.cpp, [189](#)
 - physEquations.h, [242](#)
 - physEquations_HEAD.cpp, [296](#)
 - physEquations_new.cpp, [349](#)
- dImplicitEnergyFunction_RTP_SB
 - physEquations.cpp, [190](#)
 - physEquations.h, [243](#)
 - physEquations_HEAD.cpp, [297](#)
 - physEquations_new.cpp, [350](#)
- dLocalGridNew
 - Grid, [79](#)
- dLocalGridOld
 - Grid, [80](#)
- dLogE
 - eos, [58](#)
- dLogKappa
 - eos, [58](#)
- dLogP
 - eos, [58](#)
- dLogRhoDelta
 - eos, [57](#)
- dLogRhoMin
 - eos, [57](#)
- dLogTDelta
 - eos, [58](#)
- dLogTMin
 - eos, [57](#)
- dMaxConvectiveVelocity
 - Parameters, [99](#)
- dMaxConvectiveVelocity_c
 - Parameters, [99](#)
- dMaxErrorInRHS
 - Implicit, [89](#)
- dPerChange
 - Time, [108](#)
- dPi
 - Parameters, [96](#)

- dPrintFrequencyTime
 - Output, [94](#)
- dPrt
 - Parameters, [99](#)
- dSigma
 - Parameters, [96](#)
- dSoundSpeed
 - eos, [53](#)
- dStartTimer
 - Performance, [101](#)
- dt
 - Time, [107](#)
- dTimeLastDump
 - Output, [93](#)
- dTimeLastPrint
 - Output, [94](#)
- dTimeStepFactor
 - Time, [108](#)
- dTolerance
 - Implicit, [87](#)
 - Parameters, [98](#)
- DUMP_VERSION
 - global.h, [142](#)
- dXMassFrac
 - eos, [57](#)
- dYMassFrac
 - eos, [57](#)
- eos, [49](#)
 - ~eos, [50](#)
 - dDRhoDP, [53](#)
 - dGetEnergy, [52](#)
 - dGetOpacity, [53](#)
 - dGetPressure, [52](#)
 - dLogE, [58](#)
 - dLogKappa, [58](#)
 - dLogP, [58](#)
 - dLogRhoDelta, [57](#)
 - dLogRhoMin, [57](#)
 - dLogTDelta, [58](#)
 - dLogTMin, [57](#)
 - dSoundSpeed, [53](#)
 - dXMassFrac, [57](#)
 - dYMassFrac, [57](#)
 - eos, [50](#)
 - gamma1DelAdC_v, [55](#)
 - getEAndDTDE, [56](#)
 - getEKappa, [54](#)
 - getPAndDRhoDP, [56](#)
 - getPEKappa, [54](#)
 - getPEKappaGamma, [54](#)
 - getPEKappaGammaCp, [55](#)
 - getPKappaGamma, [55](#)
 - nNumRho, [56](#)
 - nNumT, [57](#)
 - operator=, [51](#)
 - readAscii, [51](#)
 - readBin, [51](#)
 - readBobsAscii, [51](#)
 - writeAscii, [51](#)
 - writeBin, [52](#)
- eosTable
 - Parameters, [97](#)
- fin
 - dataManipulation.cpp, [115](#)
 - dataManipulation.h, [124](#)
- finWatchZones
 - dataMonitoring.cpp, [133](#)
 - dataMonitoring.h, [137](#)
- fpCalculateAveDensities
 - Functions, [60](#)
- fpCalculateDeltat
 - Functions, [60](#)
- fpCalculateNewAV
 - Functions, [61](#)
- fpCalculateNewDensities
 - Functions, [60](#)
- fpCalculateNewEddyVisc
 - Functions, [61](#)
- fpCalculateNewEnergies
 - Functions, [60](#)
- fpCalculateNewEOSVars
 - Functions, [60](#)
- fpCalculateNewGridVelocities
 - Functions, [60](#)
- fpCalculateNewRadii
 - Functions, [60](#)
- fpCalculateNewVelocities
 - Functions, [60](#)
- fpImplicitSolve
 - Functions, [61](#)
- fpModelWrite
 - Functions, [61](#)
- fpUpdateLocalBoundaryVelocitiesNewGrid
 - Functions, [61](#)
- fpWriteWatchZones
 - Functions, [61](#)
- Functions, [59](#)
 - fpCalculateAveDensities, [60](#)
 - fpCalculateDeltat, [60](#)
 - fpCalculateNewAV, [61](#)
 - fpCalculateNewDensities, [60](#)
 - fpCalculateNewEddyVisc, [61](#)
 - fpCalculateNewEnergies, [60](#)
 - fpCalculateNewEOSVars, [60](#)
 - fpCalculateNewGridVelocities, [60](#)
 - fpCalculateNewRadii, [60](#)

- fpCalculateNewVelocities, 60
- fpImplicitSolve, 61
- fpModelWrite, 61
- fpUpdateLocalBoundaryVelocitiesNew-Grid, 61
- fpWriteWatchZones, 61
- Functions, 59
- functions
 - Global, 63
- gamma1DelAdC_v
 - eos, 55
- getEAndDTDE
 - eos, 56
- getEKappa
 - eos, 54
- getPAndDRhoDP
 - eos, 56
- getPEKappa
 - eos, 54
- getPEKappaGamma
 - eos, 54
- getPEKappaGammaCp
 - eos, 55
- getPKappaGamma
 - eos, 55
- Global, 62
 - functions, 63
 - Global, 62
 - grid, 63
 - implicit, 63
 - messPass, 62
 - output, 63
 - parameters, 63
 - performance, 63
 - procTop, 62
 - time, 63
- global.cpp, 141
- global.h, 142
 - DUMP_VERSION, 142
 - SEDOV, 143
 - SIGNEG DEN, 143
 - SIGNEG ENG, 143
 - SIGNEG TEMP, 143
 - TRACKMAXSOLVERERROR, 143
 - VISCOUS_ENERGY_EQ, 143
- Grid, 64
 - dLocalGridNew, 79
 - dLocalGridOld, 80
 - Grid, 67
 - nCenIntOffset, 83
 - nCotThetaIJK, 76
 - nCotThetaIJp1halfK, 76
 - nD, 69
 - nDCosThetaIJK, 77
 - nDenAve, 73
 - nDM, 68
 - nDPhi, 75
 - nDTheta, 75
 - nE, 72
 - nEddyVisc, 77
 - nEndGhostUpdateExplicit, 82
 - nEndGhostUpdateImplicit, 83
 - nEndUpdateExplicit, 81
 - nEndUpdateImplicit, 82
 - nGamma, 73
 - nGlobalGridDims, 78
 - nGlobalGridPositionLocalGrid, 84
 - nKappa, 73
 - nLocalGridDims, 79
 - nM, 68
 - nNum1DZones, 78
 - nNumDims, 77
 - nNumGhostCells, 78
 - nNumIntVars, 77
 - nNumVars, 77
 - nP, 72
 - nPhi, 68
 - nQ0, 74
 - nQ1, 74
 - nQ2, 75
 - nR, 68
 - nSinThetaIJK, 76
 - nSinThetaIJp1halfK, 76
 - nStartGhostUpdateExplicit, 82
 - nStartGhostUpdateImplicit, 83
 - nStartUpdateExplicit, 80
 - nStartUpdateImplicit, 81
 - nT, 72
 - nTheta, 68
 - nU, 70
 - nU0, 70
 - nV, 71
 - nVariables, 78
 - nW, 71
- grid
 - Global, 63
- Implicit, 85
 - dAverageRHS, 89
 - dCurrentRelTError, 88
 - dDerivativeStepFraction, 88
 - dMaxErrorInRHS, 89
 - dTolerance, 87
 - Implicit, 85
 - kspContext, 86
 - matCoeff, 86
 - nCurrentNumIterations, 88

- nLocDer, [88](#)
- nLocFun, [88](#)
- nMaxNumIterations, [87](#)
- nMaxNumSolverIterations, [88](#)
- nNumDerPerRow, [87](#)
- nNumImplicitZones, [86](#)
- nNumRowsALocal, [87](#)
- nNumRowsALocalSB, [87](#)
- nTypeDer, [87](#)
- vecRHS, [86](#)
- vecscatT Corrections, [87](#)
- vecT Corrections, [86](#)
- vecT CorrectionsLocal, [86](#)
- implicit
 - Global, [63](#)
- implicitSolve_None
 - physEquations.cpp, [191](#)
 - physEquations.h, [244](#)
 - physEquations_HEAD.cpp, [298](#)
 - physEquations_new.cpp, [351](#)
- implicitSolve_R
 - physEquations.cpp, [191](#)
 - physEquations.h, [244](#)
 - physEquations_HEAD.cpp, [298](#)
 - physEquations_new.cpp, [351](#)
- implicitSolve_RT
 - physEquations.cpp, [191](#)
 - physEquations.h, [245](#)
 - physEquations_HEAD.cpp, [298](#)
 - physEquations_new.cpp, [351](#)
- implicitSolve_RTP
 - physEquations.cpp, [192](#)
 - physEquations.h, [245](#)
 - physEquations_HEAD.cpp, [299](#)
 - physEquations_new.cpp, [352](#)
- init
 - dataManipulation.cpp, [115](#)
 - dataManipulation.h, [125](#)
- initDonorFracAndMaxConVel_R_GL
 - physEquations.cpp, [192](#)
 - physEquations.h, [246](#)
 - physEquations_HEAD.cpp, [299](#)
 - physEquations_new.cpp, [352](#)
- initDonorFracAndMaxConVel_R_TEOS
 - physEquations.cpp, [192](#)
 - physEquations.h, [246](#)
 - physEquations_HEAD.cpp, [300](#)
 - physEquations_new.cpp, [353](#)
- initDonorFracAndMaxConVel_RT_GL
 - physEquations.cpp, [192](#)
 - physEquations.h, [246](#)
 - physEquations_HEAD.cpp, [300](#)
 - physEquations_new.cpp, [353](#)
- initDonorFracAndMaxConVel_RT_TEOS
 - physEquations.cpp, [192](#)
 - physEquations.h, [247](#)
 - physEquations_HEAD.cpp, [300](#)
 - physEquations_new.cpp, [353](#)
- initImplicitCalculation
 - dataManipulation.cpp, [116](#)
 - dataManipulation.h, [126](#)
- initInternalVars
 - physEquations.cpp, [193](#)
 - physEquations.h, [247](#)
 - physEquations_HEAD.cpp, [301](#)
 - physEquations_new.cpp, [354](#)
- initUpdateLocalBoundaries
 - dataManipulation.cpp, [117](#)
 - dataManipulation.h, [126](#)
- initWatchZones
 - dataMonitoring.cpp, [133](#)
 - dataMonitoring.h, [138](#)
- kspContext
 - Implicit, [86](#)
- main
 - main.cpp, [144](#)
 - main.h, [147](#)
- main.cpp, [144](#)
 - main, [144](#)
 - signalHandler, [146](#)
- main.h, [147](#)
 - main, [147](#)
 - signalHandler, [148](#)
- matCoeff
 - Implicit, [86](#)
- MessPass, [90](#)
 - MessPass, [90](#)
 - requestRecv, [91](#)
 - requestSend, [91](#)
 - statusRecv, [91](#)
 - statusSend, [91](#)
 - typeRecvNewVar, [91](#)
 - typeRecvOldGrid, [90](#)
 - typeSendNewGrid, [90](#)
 - typeSendNewVar, [91](#)
- messPass

- Global, [62](#)
- modelRead
 - dataManipulation.cpp, [117](#)
 - dataManipulation.h, [127](#)
- modelWrite_GL
 - dataManipulation.cpp, [118](#)
 - dataManipulation.h, [128](#)
- modelWrite_TEOS
 - dataManipulation.cpp, [119](#)
 - dataManipulation.h, [128](#)
- nCenIntOffset
 - Grid, [83](#)
- nCoords
 - ProcTop, [104](#)
- nCotThetaIJK
 - Grid, [76](#)
- nCotThetaIJP1halfK
 - Grid, [76](#)
- nCurrentNumIterations
 - Implicit, [88](#)
- nD
 - Grid, [69](#)
- nDCosThetaIJK
 - Grid, [77](#)
- nDenAve
 - Grid, [73](#)
- nDM
 - Grid, [68](#)
- nDPhi
 - Grid, [75](#)
- nDTheta
 - Grid, [75](#)
- nDumpFrequencyStep
 - Output, [92](#)
- nE
 - Grid, [72](#)
- nEddyVisc
 - Grid, [77](#)
- nEndGhostUpdateExplicit
 - Grid, [82](#)
- nEndGhostUpdateImplicit
 - Grid, [83](#)
- nEndTimeStep
 - Time, [108](#)
- nEndUpdateExplicit
 - Grid, [81](#)
- nEndUpdateImplicit
 - Grid, [82](#)
- nGamma
 - Grid, [73](#)
- nGlobalGridDims
 - Grid, [78](#)
- nGlobalGridPositionLocalGrid
 - Grid, [84](#)
- nKappa
 - Grid, [73](#)
- nLocalGridDims
 - Grid, [79](#)
- nLocDer
 - Implicit, [88](#)
- nLocFun
 - Implicit, [88](#)
- nM
 - Grid, [68](#)
- nMaxIterations
 - Parameters, [98](#)
- nMaxNumIterations
 - Implicit, [87](#)
- nMaxNumSolverIterations
 - Implicit, [88](#)
- nNeighborRanks
 - ProcTop, [104](#)
- nNum1DZones
 - Grid, [78](#)
- nNumDerPerRow
 - Implicit, [87](#)
- nNumDims
 - Grid, [77](#)
- nNumGhostCells
 - Grid, [78](#)
- nNumImplicitZones
 - Implicit, [86](#)
- nNumInt Vars
 - Grid, [77](#)
- nNumNeighbors
 - ProcTop, [104](#)
- nNumProcs
 - ProcTop, [103](#)
- nNumRadialNeighbors
 - ProcTop, [105](#)
- nNumRho
 - eos, [56](#)
- nNumRowsALocal
 - Implicit, [87](#)
- nNumRowsALocalSB
 - Implicit, [87](#)
- nNumT
 - eos, [57](#)
- nNumTimeStepsSinceLastPrint
 - Output, [93](#)
- nNumVars
 - Grid, [77](#)
- nP
 - Grid, [72](#)
- nPeriodic
 - ProcTop, [104](#)
- nPhi

- Grid, [68](#)
- nPrintFrequencyStep
 - Output, [94](#)
- nPrintMode
 - Output, [93](#)
- nProcDims
 - ProcTop, [103](#)
- nQ0
 - Grid, [74](#)
- nQ1
 - Grid, [74](#)
- nQ2
 - Grid, [75](#)
- nR
 - Grid, [68](#)
- nRadialNeighborNeighborIDs
 - ProcTop, [105](#)
- nRadialNeighborRanks
 - ProcTop, [105](#)
- nRank
 - ProcTop, [104](#)
- nSinThetaLJK
 - Grid, [76](#)
- nSinThetaJp1halfK
 - Grid, [76](#)
- nStartGhostUpdateExplicit
 - Grid, [82](#)
- nStartGhostUpdateImplicit
 - Grid, [83](#)
- nStartUpdateExplicit
 - Grid, [80](#)
- nStartUpdateImplicit
 - Grid, [81](#)
- nT
 - Grid, [72](#)
- nTheta
 - Grid, [68](#)
- nTimeStepIndex
 - Time, [108](#)
- nTypeDer
 - Implicit, [87](#)
- nTypeTurbulenceMod
 - Parameters, [96](#)
- nU
 - Grid, [70](#)
- nU0
 - Grid, [70](#)
- nV
 - Grid, [71](#)
- nVariables
 - Grid, [78](#)
- nW
 - Grid, [71](#)

- ofWatchZoneFiles
 - Output, [94](#)
- operator=
 - eos, [51](#)
- Output, [92](#)
 - bDump, [93](#)
 - bPrint, [93](#)
 - dDumpFrequencyTime, [93](#)
 - dPrintFrequencyTime, [94](#)
 - dTimeLastDump, [93](#)
 - dTimeLastPrint, [94](#)
 - nDumpFrequencyStep, [92](#)
 - nNumTimeStepsSinceLastPrint, [93](#)
 - nPrintFrequencyStep, [94](#)
 - nPrintMode, [93](#)
 - ofWatchZoneFiles, [94](#)
 - Output, [92](#)
 - sBaseOutputFileName, [93](#)
 - watchzoneList, [94](#)
- output
 - Global, [63](#)
- Parameters, [95](#)
 - bAdiabatic, [96](#)
 - bEOSGammaLaw, [96](#)
 - dA, [97](#)
 - dAlpha, [98](#)
 - dAVThreshold, [98](#)
 - dDonorFrac, [98](#)
 - dEddyViscosity, [99](#)
 - dG, [97](#)
 - dGamma, [97](#)
 - dMaxConvectiveVelocity, [99](#)
 - dMaxConvectiveVelocity_c, [99](#)
 - dPi, [96](#)
 - dPrt, [99](#)
 - dSigma, [96](#)
 - dTolerance, [98](#)
 - eosTable, [97](#)
 - nMaxIterations, [98](#)
 - nTypeTurbulenceMod, [96](#)
 - Parameters, [95](#)
 - sEOSFileName, [97](#)
- parameters
 - Global, [63](#)
- Performance, [101](#)
 - dEndTimer, [101](#)
 - dStartTimer, [101](#)
 - Performance, [101](#)
- performance
 - Global, [63](#)
- physEquations.cpp, [149](#)
 - calDelt_CONST, [152](#)
 - calDelt_R_GL, [152](#)

- calDelt_R_TEOS, [153](#)
- calDelt_RT_GL, [153](#)
- calDelt_RT_TEOS, [153](#)
- calDelt_RTP_GL, [154](#)
- calDelt_RTP_TEOS, [154](#)
- calNewD_R, [154](#)
- calNewD_RT, [155](#)
- calNewD_RTP, [155](#)
- calNewDenave_None, [156](#)
- calNewDenave_R, [156](#)
- calNewDenave_RT, [156](#)
- calNewDenave_RTP, [156](#)
- calNewE_R_AD, [157](#)
- calNewE_R_NA, [157](#)
- calNewE_R_NA_LES, [157](#)
- calNewE_RT_AD, [158](#)
- calNewE_RT_NA, [158](#)
- calNewE_RT_NA_LES, [159](#)
- calNewE_RTP_AD, [159](#)
- calNewE_RTP_NA, [160](#)
- calNewE_RTP_NA_LES, [160](#)
- calNewEddyVisc_None, [161](#)
- calNewEddyVisc_R_CN, [161](#)
- calNewEddyVisc_R_SM, [162](#)
- calNewEddyVisc_RT_CN, [162](#)
- calNewEddyVisc_RT_SM, [162](#)
- calNewEddyVisc_RTP_CN, [162](#)
- calNewEddyVisc_RTP_SM, [163](#)
- calNewP_GL, [163](#)
- calNewPEKappaGamma_TEOS, [163](#)
- calNewQ0_R_GL, [163](#)
- calNewQ0_R_TEOS, [164](#)
- calNewQ0Q1_RT_GL, [164](#)
- calNewQ0Q1_RT_TEOS, [164](#)
- calNewQ0Q1Q2_RTP_GL, [165](#)
- calNewQ0Q1Q2_RTP_TEOS, [165](#)
- calNewR, [165](#)
- calNewTPKappaGamma_TEOS, [165](#)
- calNewU0_R, [166](#)
- calNewU0_RT, [166](#)
- calNewU0_RTP, [167](#)
- calNewU_R, [167](#)
- calNewU_R_LES, [168](#)
- calNewU_RT, [168](#)
- calNewU_RT_LES, [169](#)
- calNewU_RTP, [170](#)
- calNewU_RTP_LES, [171](#)
- calNewV_RT, [172](#)
- calNewV_RT_LES, [173](#)
- calNewV_RTP, [173](#)
- calNewV_RTP_LES, [174](#)
- calNewVelocities_R, [174](#)
- calNewVelocities_R_LES, [175](#)
- calNewVelocities_RT, [175](#)
- calNewVelocities_RT_LES, [175](#)
- calNewVelocities_RTP, [176](#)
- calNewVelocities_RTP_LES, [176](#)
- calNewW_RTP, [176](#)
- calNewW_RTP_LES, [177](#)
- calOldDenave_None, [177](#)
- calOldDenave_R, [177](#)
- calOldDenave_RT, [178](#)
- calOldDenave_RTP, [178](#)
- calOldEddyVisc_R_CN, [178](#)
- calOldEddyVisc_R_SM, [178](#)
- calOldEddyVisc_RT_CN, [179](#)
- calOldEddyVisc_RT_SM, [179](#)
- calOldEddyVisc_RTP_CN, [179](#)
- calOldEddyVisc_RTP_SM, [179](#)
- calOldP_GL, [180](#)
- calOldPEKappaGamma_TEOS, [180](#)
- calOldQ0_R_GL, [180](#)
- calOldQ0_R_TEOS, [181](#)
- calOldQ0Q1_RT_GL, [181](#)
- calOldQ0Q1_RT_TEOS, [181](#)
- calOldQ0Q1Q2_RTP_GL, [181](#)
- calOldQ0Q1Q2_RTP_TEOS, [182](#)
- dEOS_GL, [182](#)
- dImplicitEnergyFunction_None, [182](#)
- dImplicitEnergyFunction_R, [182](#)
- dImplicitEnergyFunction_R_LES, [183](#)
- dImplicitEnergyFunction_R_LES_SB, [183](#)
- dImplicitEnergyFunction_R_SB, [184](#)
- dImplicitEnergyFunction_RT, [185](#)
- dImplicitEnergyFunction_RT_LES, [185](#)
- dImplicitEnergyFunction_RT_LES_SB, [186](#)
- dImplicitEnergyFunction_RT_SB, [187](#)
- dImplicitEnergyFunction_RTP, [188](#)
- dImplicitEnergyFunction_RTP_LES, [188](#)
- dImplicitEnergyFunction_RTP_LES_SB, [189](#)
- dImplicitEnergyFunction_RTP_SB, [190](#)
- implicitSolve_None, [191](#)
- implicitSolve_R, [191](#)
- implicitSolve_RT, [191](#)
- implicitSolve_RTP, [192](#)
- initDonorFracAndMaxConVel_R_GL, [192](#)
- initDonorFracAndMaxConVel_R_TEOS, [192](#)
- initDonorFracAndMaxConVel_RT_GL, [192](#)
- initDonorFracAndMaxConVel_RT_-TEOS, [192](#)
- initDonorFracAndMaxConVel_RTP_GL, [193](#)

- initDonorFracAndMaxConVel_RTP_-TEOS, 193
- initInternalVars, 193
- setInternalVarInf, 193
- setMainFunctions, 194
- physEquations.h, 195
- calDelt_CONST, 198
- calDelt_R_GL, 198
- calDelt_R_TEOS, 199
- calDelt_RT_GL, 199
- calDelt_RT_TEOS, 200
- calDelt_RTP_GL, 200
- calDelt_RTP_TEOS, 201
- calNewD_R, 201
- calNewD_RT, 202
- calNewD_RTP, 202
- calNewDenave_None, 202
- calNewDenave_R, 203
- calNewDenave_RT, 203
- calNewDenave_RTP, 203
- calNewE_R_AD, 204
- calNewE_R_NA, 204
- calNewE_R_NA_LES, 205
- calNewE_RT_AD, 205
- calNewE_RT_NA, 206
- calNewE_RT_NA_LES, 206
- calNewE_RTP_AD, 207
- calNewE_RTP_NA, 208
- calNewE_RTP_NA_LES, 208
- calNewEddyVisc_None, 209
- calNewEddyVisc_R_CN, 209
- calNewEddyVisc_R_SM, 210
- calNewEddyVisc_RT_CN, 210
- calNewEddyVisc_RT_SM, 210
- calNewEddyVisc_RTP_CN, 210
- calNewEddyVisc_RTP_SM, 211
- calNewP_GL, 211
- calNewPEKappaGamma_TEOS, 212
- calNewQ0_R_GL, 212
- calNewQ0_R_TEOS, 212
- calNewQ0Q1_RT_GL, 213
- calNewQ0Q1_RT_TEOS, 213
- calNewQ0Q1Q2_RTP_GL, 213
- calNewQ0Q1Q2_RTP_TEOS, 214
- calNewR, 214
- calNewTPKappaGamma_TEOS, 215
- calNewU0_R, 215
- calNewU0_RT, 215
- calNewU0_RTP, 216
- calNewU_R, 217
- calNewU_R_LES, 217
- calNewU_RT, 218
- calNewU_RT_LES, 219
- calNewU_RTP, 220
- calNewU_RTP_LES, 221
- calNewV_RT, 223
- calNewV_RT_LES, 223
- calNewV_RTP, 224
- calNewV_RTP_LES, 224
- calNewVelocities_R, 225
- calNewVelocities_R_LES, 225
- calNewVelocities_RT, 226
- calNewVelocities_RT_LES, 226
- calNewVelocities_RTP, 226
- calNewVelocities_RTP_LES, 227
- calNewW_RTP, 227
- calNewW_RTP_LES, 228
- calOldDenave_None, 228
- calOldDenave_R, 229
- calOldDenave_RT, 229
- calOldDenave_RTP, 229
- calOldEddyVisc_R_CN, 230
- calOldEddyVisc_R_SM, 230
- calOldEddyVisc_RT_CN, 230
- calOldEddyVisc_RT_SM, 230
- calOldEddyVisc_RTP_CN, 231
- calOldEddyVisc_RTP_SM, 231
- calOldP_GL, 232
- calOldPEKappaGamma_TEOS, 232
- calOldQ0_R_GL, 232
- calOldQ0_R_TEOS, 233
- calOldQ0Q1_RT_GL, 233
- calOldQ0Q1_RT_TEOS, 234
- calOldQ0Q1Q2_RTP_GL, 234
- calOldQ0Q1Q2_RTP_TEOS, 234
- dEOS_GL, 235
- dImplicitEnergyFunction_None, 235
- dImplicitEnergyFunction_R, 235
- dImplicitEnergyFunction_R_LES, 236
- dImplicitEnergyFunction_R_LES_SB, 236
- dImplicitEnergyFunction_R_SB, 237
- dImplicitEnergyFunction_RT, 238
- dImplicitEnergyFunction_RT_LES, 238
- dImplicitEnergyFunction_RT_LES_SB, 239
- dImplicitEnergyFunction_RT_SB, 240
- dImplicitEnergyFunction_RTP, 241
- dImplicitEnergyFunction_RTP_LES, 241
- dImplicitEnergyFunction_RTP_LES_SB, 242
- dImplicitEnergyFunction_RTP_SB, 243
- implicitSolve_None, 244
- implicitSolve_R, 244
- implicitSolve_RT, 245
- implicitSolve_RTP, 245
- initDonorFracAndMaxConVel_R_GL, 246

- initDonorFracAndMaxConVel_R_TEOS, [246](#)
- initDonorFracAndMaxConVel_RT_GL, [246](#)
- initDonorFracAndMaxConVel_RT_-TEOS, [247](#)
- initDonorFracAndMaxConVel_RTP_GL, [247](#)
- initDonorFracAndMaxConVel_RTP_-TEOS, [247](#)
- initInternalVars, [247](#)
- setInternalVarInf, [248](#)
- setMainFunctions, [248](#)
- physEquations_HEAD.cpp, [251](#)
- calDelt_CONST, [254](#)
- calDelt_R_GL, [254](#)
- calDelt_R_TEOS, [255](#)
- calDelt_RT_GL, [255](#)
- calDelt_RT_TEOS, [256](#)
- calDelt_RTP_GL, [256](#)
- calDelt_RTP_TEOS, [257](#)
- calNewD_R, [257](#)
- calNewD_RT, [258](#)
- calNewD_RTP, [258](#)
- calNewDenave_None, [259](#)
- calNewDenave_R, [259](#)
- calNewDenave_RT, [259](#)
- calNewDenave_RTP, [259](#)
- calNewE_R_AD, [260](#)
- calNewE_R_NA, [260](#)
- calNewE_R_NA_LES, [261](#)
- calNewE_RT_AD, [261](#)
- calNewE_RT_NA, [262](#)
- calNewE_RT_NA_LES, [262](#)
- calNewE_RTP_AD, [263](#)
- calNewE_RTP_NA, [263](#)
- calNewE_RTP_NA_LES, [264](#)
- calNewEddyVisc_None, [265](#)
- calNewEddyVisc_R_CN, [265](#)
- calNewEddyVisc_R_SM, [265](#)
- calNewEddyVisc_RT_CN, [266](#)
- calNewEddyVisc_RT_SM, [266](#)
- calNewEddyVisc_RTP_CN, [266](#)
- calNewEddyVisc_RTP_SM, [266](#)
- calNewP_GL, [267](#)
- calNewPEKappaGamma_TEOS, [267](#)
- calNewQ0_R_GL, [267](#)
- calNewQ0_R_TEOS, [268](#)
- calNewQ0Q1_RT_GL, [268](#)
- calNewQ0Q1_RT_TEOS, [268](#)
- calNewQ0Q1Q2_RTP_GL, [269](#)
- calNewQ0Q1Q2_RTP_TEOS, [269](#)
- calNewR, [270](#)
- calNewTPKappaGamma_TEOS, [270](#)
- calNewU0_R, [270](#)
- calNewU0_RT, [271](#)
- calNewU0_RTP, [271](#)
- calNewU_R, [272](#)
- calNewU_R_LES, [272](#)
- calNewU_RT, [273](#)
- calNewU_RT_LES, [274](#)
- calNewU_RTP, [275](#)
- calNewU_RTP_LES, [276](#)
- calNewV_RT, [278](#)
- calNewV_RT_LES, [278](#)
- calNewV_RTP, [279](#)
- calNewV_RTP_LES, [279](#)
- calNewVelocities_R, [280](#)
- calNewVelocities_R_LES, [280](#)
- calNewVelocities_RT, [280](#)
- calNewVelocities_RT_LES, [281](#)
- calNewVelocities_RTP, [281](#)
- calNewVelocities_RTP_LES, [281](#)
- calNewW_RTP, [282](#)
- calNewW_RTP_LES, [282](#)
- calOldDenave_None, [283](#)
- calOldDenave_R, [283](#)
- calOldDenave_RT, [283](#)
- calOldDenave_RTP, [284](#)
- calOldEddyVisc_R_CN, [284](#)
- calOldEddyVisc_R_SM, [284](#)
- calOldEddyVisc_RT_CN, [284](#)
- calOldEddyVisc_RT_SM, [285](#)
- calOldEddyVisc_RTP_CN, [285](#)
- calOldEddyVisc_RTP_SM, [285](#)
- calOldP_GL, [286](#)
- calOldPEKappaGamma_TEOS, [286](#)
- calOldQ0_R_GL, [287](#)
- calOldQ0_R_TEOS, [287](#)
- calOldQ0Q1_RT_GL, [287](#)
- calOldQ0Q1_RT_TEOS, [288](#)
- calOldQ0Q1Q2_RTP_GL, [288](#)
- calOldQ0Q1Q2_RTP_TEOS, [288](#)
- dEOS_GL, [289](#)
- dImplicitEnergyFunction_None, [289](#)
- dImplicitEnergyFunction_R, [289](#)
- dImplicitEnergyFunction_R_LES, [290](#)
- dImplicitEnergyFunction_R_LES_SB, [290](#)
- dImplicitEnergyFunction_R_SB, [291](#)
- dImplicitEnergyFunction_RT, [292](#)
- dImplicitEnergyFunction_RT_LES, [292](#)
- dImplicitEnergyFunction_RT_LES_SB, [293](#)
- dImplicitEnergyFunction_RT_SB, [294](#)
- dImplicitEnergyFunction_RTP, [295](#)
- dImplicitEnergyFunction_RTP_LES, [295](#)

- dImplicitEnergyFunction_RTP_LES_SB, 296
- dImplicitEnergyFunction_RTP_SB, 297
- implicitSolve_None, 298
- implicitSolve_R, 298
- implicitSolve_RT, 298
- implicitSolve_RTP, 299
- initDonorFracAndMaxConVel_R_GL, 299
- initDonorFracAndMaxConVel_R_TEOS, 300
- initDonorFracAndMaxConVel_RT_GL, 300
- initDonorFracAndMaxConVel_RT_-TEOS, 300
- initDonorFracAndMaxConVel_RTP_GL, 300
- initDonorFracAndMaxConVel_RTP_-TEOS, 301
- initInternalVars, 301
- setInternalVarInf, 301
- setMainFunctions, 302
- physEquations_new.cpp, 304
- calDelt_CONST, 307
- calDelt_R_GL, 307
- calDelt_R_TEOS, 308
- calDelt_RT_GL, 308
- calDelt_RT_TEOS, 309
- calDelt_RTP_GL, 309
- calDelt_RTP_TEOS, 310
- calNewD_R, 310
- calNewD_RT, 311
- calNewD_RTP, 311
- calNewDenave_None, 312
- calNewDenave_R, 312
- calNewDenave_RT, 312
- calNewDenave_RTP, 312
- calNewE_R_AD, 313
- calNewE_R_NA, 313
- calNewE_R_NA_LES, 314
- calNewE_RT_AD, 314
- calNewE_RT_NA, 315
- calNewE_RT_NA_LES, 315
- calNewE_RTP_AD, 316
- calNewE_RTP_NA, 316
- calNewE_RTP_NA_LES, 317
- calNewEddyVisc_None, 318
- calNewEddyVisc_R_CN, 318
- calNewEddyVisc_R_SM, 318
- calNewEddyVisc_RT_CN, 319
- calNewEddyVisc_RT_SM, 319
- calNewEddyVisc_RTP_CN, 319
- calNewEddyVisc_RTP_SM, 319
- calNewP_GL, 320
- calNewPEKappaGamma_TEOS, 320
- calNewQ0_R_GL, 320
- calNewQ0_R_TEOS, 321
- calNewQ0Q1_RT_GL, 321
- calNewQ0Q1_RT_TEOS, 321
- calNewQ0Q1Q2_RTP_GL, 322
- calNewQ0Q1Q2_RTP_TEOS, 322
- calNewR, 323
- calNewTPKappaGamma_TEOS, 323
- calNewU0_R, 323
- calNewU0_RT, 324
- calNewU0_RTP, 324
- calNewU_R, 325
- calNewU_R_LES, 325
- calNewU_RT, 326
- calNewU_RT_LES, 327
- calNewU_RTP, 328
- calNewU_RTP_LES, 329
- calNewV_RT, 331
- calNewV_RT_LES, 331
- calNewV_RTP, 332
- calNewV_RTP_LES, 332
- calNewVelocities_R, 333
- calNewVelocities_R_LES, 333
- calNewVelocities_RT, 333
- calNewVelocities_RT_LES, 334
- calNewVelocities_RTP, 334
- calNewVelocities_RTP_LES, 334
- calNewW_RTP, 335
- calNewW_RTP_LES, 335
- calOldDenave_None, 336
- calOldDenave_R, 336
- calOldDenave_RT, 336
- calOldDenave_RTP, 337
- calOldEddyVisc_R_CN, 337
- calOldEddyVisc_R_SM, 337
- calOldEddyVisc_RT_CN, 337
- calOldEddyVisc_RT_SM, 338
- calOldEddyVisc_RTP_CN, 338
- calOldEddyVisc_RTP_SM, 338
- calOldP_GL, 339
- calOldPEKappaGamma_TEOS, 339
- calOldQ0_R_GL, 340
- calOldQ0_R_TEOS, 340
- calOldQ0Q1_RT_GL, 340
- calOldQ0Q1_RT_TEOS, 341
- calOldQ0Q1Q2_RTP_GL, 341
- calOldQ0Q1Q2_RTP_TEOS, 341
- dEOS_GL, 342
- dImplicitEnergyFunction_None, 342
- dImplicitEnergyFunction_R, 342
- dImplicitEnergyFunction_R_LES, 343
- dImplicitEnergyFunction_R_LES_SB, 343
- dImplicitEnergyFunction_R_SB, 344

- dImplicitEnergyFunction_RT, 345
- dImplicitEnergyFunction_RT_LES, 345
- dImplicitEnergyFunction_RT_LES_SB, 346
- dImplicitEnergyFunction_RT_SB, 347
- dImplicitEnergyFunction_RTP, 348
- dImplicitEnergyFunction_RTP_LES, 348
- dImplicitEnergyFunction_RTP_LES_SB, 349
- dImplicitEnergyFunction_RTP_SB, 350
- implicitSolve_None, 351
- implicitSolve_R, 351
- implicitSolve_RT, 351
- implicitSolve_RTP, 352
- initDonorFracAndMaxConVel_R_GL, 352
- initDonorFracAndMaxConVel_R_TEOS, 353
- initDonorFracAndMaxConVel_RT_GL, 353
- initDonorFracAndMaxConVel_RT_-TEOS, 353
- initDonorFracAndMaxConVel_RTP_GL, 353
- initDonorFracAndMaxConVel_RTP_-TEOS, 354
- initInternalVars, 354
- setInternalVarInf, 354
- setMainFunctions, 355
- ProcTop, 103
 - nCoords, 104
 - nNeighborRanks, 104
 - nNumNeighbors, 104
 - nNumProcs, 103
 - nNumRadialNeighbors, 105
 - nPeriodic, 104
 - nProcDims, 103
 - nRadialNeighborNeighborIDs, 105
 - nRadialNeighborRanks, 105
 - nRank, 104
 - ProcTop, 103
- procTop
 - Global, 62
- readAscii
 - eos, 51
- readBin
 - eos, 51
- readBobsAscii
 - eos, 51
- requestRecv
 - MessPass, 91
- requestSend
 - MessPass, 91
- sBaseOutputFileName
 - Output, 93
- SEDOV
 - global.h, 143
- sEOSFileName
 - Parameters, 97
- setInternalVarInf
 - physEquations.cpp, 193
 - physEquations.h, 248
 - physEquations_HEAD.cpp, 301
 - physEquations_new.cpp, 354
- setMainFunctions
 - physEquations.cpp, 194
 - physEquations.h, 248
 - physEquations_HEAD.cpp, 302
 - physEquations_new.cpp, 355
- setupLocalGrid
 - dataManipulation.cpp, 119
 - dataManipulation.h, 129
- signalHandler
 - main.cpp, 146
 - main.h, 148
- SIGNEG DEN
 - global.h, 143
- SIGNEG ENG
 - global.h, 143
- SIGNEG TEMP
 - global.h, 143
- statusRecv
 - MessPass, 91
- statusSend
 - MessPass, 91
- Time, 106
 - bVariableTimeStep, 108
 - dConstTimeStep, 108
 - dDelE_t_E_max, 109
 - dDelRho_t_Rho_max, 108
 - dDelT_t_T_max, 109
 - dDeltat_n, 107
 - dDeltat_nm1half, 107
 - dDeltat_np1half, 106
 - dEndTime, 107
 - dPerChange, 108
 - dt, 107
 - dTimeStepFactor, 108
 - nEndTimeStep, 108
 - nTimeStepIndex, 108
 - Time, 106
- time
 - Global, 63
- TRACKMAXSOLVERERROR
 - global.h, 143
- typeRecvNewVar

- MessPass, [91](#)
- typeRecvOldGrid
 - MessPass, [90](#)
- typeSendNewGrid
 - MessPass, [90](#)
- typeSendNewVar
 - MessPass, [91](#)
- updateLocalBoundaries
 - dataManipulation.cpp, [119](#)
 - dataManipulation.h, [129](#)
- updateLocalBoundariesNewGrid
 - dataManipulation.cpp, [120](#)
 - dataManipulation.h, [130](#)
- updateLocalBoundaryVelocitiesNewGrid_R
 - dataManipulation.cpp, [120](#)
 - dataManipulation.h, [130](#)
- updateLocalBoundaryVelocitiesNewGrid_RT
 - dataManipulation.cpp, [121](#)
 - dataManipulation.h, [130](#)
- updateLocalBoundaryVelocitiesNewGrid_RTP
 - dataManipulation.cpp, [121](#)
 - dataManipulation.h, [131](#)
- updateNewGridWithOld
 - dataManipulation.cpp, [121](#)
 - dataManipulation.h, [131](#)
- updateOldGrid
 - dataManipulation.cpp, [121](#)
 - dataManipulation.h, [131](#)
- userguide.h, [357](#)
- vecRHS
 - Implicit, [86](#)
- vecscatTCorrections
 - Implicit, [87](#)
- vecTCorrections
 - Implicit, [86](#)
- vecTCorrectionsLocal
 - Implicit, [86](#)
- VISCOUS_ENERGY_EQ
 - global.h, [143](#)
- WatchZone, [110](#)
- watchzone.cpp, [358](#)
- watchzone.h, [359](#)
- watchzoneList
 - Output, [94](#)
- writeAscii
 - eos, [51](#)
- writeBin
 - eos, [52](#)
- writeWatchZones_R_GL
 - dataMonitoring.cpp, [133](#)
 - dataMonitoring.h, [138](#)
- writeWatchZones_R_TEOS
 - dataMonitoring.cpp, [134](#)
 - dataMonitoring.h, [138](#)
- writeWatchZones_RT_GL
 - dataMonitoring.cpp, [134](#)
 - dataMonitoring.h, [139](#)
- writeWatchZones_RT_TEOS
 - dataMonitoring.cpp, [135](#)
 - dataMonitoring.h, [139](#)
- writeWatchZones_RTP_GL
 - dataMonitoring.cpp, [135](#)
 - dataMonitoring.h, [140](#)
- writeWatchZones_RTP_TEOS
 - dataMonitoring.cpp, [135](#)
 - dataMonitoring.h, [140](#)