

# SPHERLS

1.0

Generated by Doxygen 1.7.6.1

Fri Dec 7 2012 12:48:46



# Contents

<b>1</b>	<b>Boundary Conditions</b>	<b>1</b>
<b>2</b>	<b>Todo List</b>	<b>9</b>
<b>3</b>	<b>Class Index</b>	<b>11</b>
3.1	Class List . . . . .	11
<b>4</b>	<b>File Index</b>	<b>13</b>
4.1	File List . . . . .	13
<b>5</b>	<b>Class Documentation</b>	<b>15</b>
5.1	eos Class Reference . . . . .	15
5.1.1	Detailed Description . . . . .	16
5.1.2	Constructor & Destructor Documentation . . . . .	16
5.1.2.1	eos . . . . .	16
5.1.2.2	eos . . . . .	16
5.1.2.3	eos . . . . .	17
5.1.2.4	~eos . . . . .	17
5.1.3	Member Function Documentation . . . . .	17
5.1.3.1	dDRhoDP . . . . .	17
5.1.3.2	dGetEnergy . . . . .	17
5.1.3.3	dGetOpacity . . . . .	18
5.1.3.4	dGetPressure . . . . .	18
5.1.3.5	dSoundSpeed . . . . .	19
5.1.3.6	gamma1DelAdC_v . . . . .	19
5.1.3.7	getDlnPDlnTDlnPDlnPDEDT . . . . .	19
5.1.3.8	getEAndDTDE . . . . .	20

5.1.3.9	<a href="#">getEKappa</a>	20
5.1.3.10	<a href="#">getPAndDRhoDP</a>	20
5.1.3.11	<a href="#">getPEKappa</a>	21
5.1.3.12	<a href="#">getPEKappaGamma</a>	21
5.1.3.13	<a href="#">getPEKappaGammaCp</a>	22
5.1.3.14	<a href="#">getPKappaGamma</a>	22
5.1.3.15	<a href="#">operator=</a>	22
5.1.3.16	<a href="#">readAscii</a>	22
5.1.3.17	<a href="#">readBin</a>	23
5.1.3.18	<a href="#">readBobsAscii</a>	23
5.1.3.19	<a href="#">writeAscii</a>	23
5.1.3.20	<a href="#">writeBin</a>	23
5.1.4	<a href="#">Member Data Documentation</a>	24
5.1.4.1	<a href="#">dLogE</a>	24
5.1.4.2	<a href="#">dLogKappa</a>	24
5.1.4.3	<a href="#">dLogP</a>	24
5.1.4.4	<a href="#">dLogRhoDelta</a>	24
5.1.4.5	<a href="#">dLogRhoMin</a>	25
5.1.4.6	<a href="#">dLogTDelta</a>	25
5.1.4.7	<a href="#">dLogTMin</a>	25
5.1.4.8	<a href="#">dXMassFrac</a>	25
5.1.4.9	<a href="#">dYMassFrac</a>	25
5.1.4.10	<a href="#">nNumRho</a>	25
5.1.4.11	<a href="#">nNumT</a>	26
5.2	<a href="#">exception2 Class Reference</a>	26
5.3	<a href="#">Functions Class Reference</a>	26
5.3.1	<a href="#">Detailed Description</a>	27
5.3.2	<a href="#">Constructor &amp; Destructor Documentation</a>	27
5.3.2.1	<a href="#">Functions</a>	27
5.3.3	<a href="#">Member Data Documentation</a>	27
5.3.3.1	<a href="#">fpCalculateAveDensities</a>	27
5.3.3.2	<a href="#">fpCalculateDeltat</a>	27
5.3.3.3	<a href="#">fpCalculateNewAV</a>	28
5.3.3.4	<a href="#">fpCalculateNewDensities</a>	28

5.3.3.5	<a href="#">fpCalculateNewEddyVisc</a>	28
5.3.3.6	<a href="#">fpCalculateNewEnergies</a>	28
5.3.3.7	<a href="#">fpCalculateNewEOSVars</a>	28
5.3.3.8	<a href="#">fpCalculateNewGridVelocities</a>	28
5.3.3.9	<a href="#">fpCalculateNewRadii</a>	28
5.3.3.10	<a href="#">fpCalculateNewVelocities</a>	29
5.3.3.11	<a href="#">fpImplicitSolve</a>	29
5.3.3.12	<a href="#">fpModelWrite</a>	29
5.3.3.13	<a href="#">fpUpdateLocalBoundaryVelocitiesNewGrid</a>	29
5.3.3.14	<a href="#">fpWriteWatchZones</a>	29
5.4	Global Class Reference	29
5.4.1	Detailed Description	30
5.4.2	Constructor & Destructor Documentation	30
5.4.2.1	Global	30
5.4.3	Member Data Documentation	30
5.4.3.1	functions	30
5.4.3.2	grid	30
5.4.3.3	implicit	30
5.4.3.4	messPass	31
5.4.3.5	output	31
5.4.3.6	parameters	31
5.4.3.7	performance	31
5.4.3.8	procTop	31
5.4.3.9	time	31
5.5	Grid Class Reference	31
5.5.1	Detailed Description	33
5.5.2	Constructor & Destructor Documentation	35
5.5.2.1	Grid	35
5.5.3	Member Data Documentation	36
5.5.3.1	dLocalGridNew	36
5.5.3.2	dLocalGridOld	36
5.5.3.3	nCenIntOffset	37
5.5.3.4	nCotThetaIJK	38
5.5.3.5	nCotThetaIjp1halfK	38

5.5.3.6	nD	38
5.5.3.7	nDCosThetaJK	39
5.5.3.8	nDenAve	39
5.5.3.9	nDM	40
5.5.3.10	nDonorCellFrac	40
5.5.3.11	nDPhi	41
5.5.3.12	nDTheta	41
5.5.3.13	nE	41
5.5.3.14	nEddyVisc	42
5.5.3.15	nEndGhostUpdateExplicit	42
5.5.3.16	nEndGhostUpdateImplicit	43
5.5.3.17	nEndUpdateExplicit	43
5.5.3.18	nEndUpdateImplicit	44
5.5.3.19	nGamma	44
5.5.3.20	nGlobalGridDims	44
5.5.3.21	nGlobalGridPositionLocalGrid	44
5.5.3.22	nKappa	45
5.5.3.23	nLocalGridDims	45
5.5.3.24	nM	45
5.5.3.25	nNum1DZones	45
5.5.3.26	nNumDims	45
5.5.3.27	nNumGhostCells	46
5.5.3.28	nNumIntVars	46
5.5.3.29	nNumVars	46
5.5.3.30	nP	47
5.5.3.31	nPhi	47
5.5.3.32	nQ0	47
5.5.3.33	nQ1	48
5.5.3.34	nQ2	48
5.5.3.35	nR	49
5.5.3.36	nSinThetaJK	49
5.5.3.37	nSinThetaJp1halfK	50
5.5.3.38	nStartGhostUpdateExplicit	50
5.5.3.39	nStartGhostUpdateImplicit	51

5.5.3.40	<a href="#">nStartUpdateExplicit</a>	51
5.5.3.41	<a href="#">nStartUpdateImplicit</a>	51
5.5.3.42	<a href="#">nT</a>	52
5.5.3.43	<a href="#">nTheta</a>	52
5.5.3.44	<a href="#">nU</a>	52
5.5.3.45	<a href="#">nU0</a>	53
5.5.3.46	<a href="#">nV</a>	53
5.5.3.47	<a href="#">nVariables</a>	54
5.5.3.48	<a href="#">nW</a>	54
5.6	<a href="#">Implicit Class Reference</a>	55
5.6.1	<a href="#">Detailed Description</a>	55
5.6.2	<a href="#">Constructor &amp; Destructor Documentation</a>	56
5.6.2.1	<a href="#">Implicit</a>	56
5.6.3	<a href="#">Member Data Documentation</a>	56
5.6.3.1	<a href="#">dAverageRHS</a>	56
5.6.3.2	<a href="#">dCurrentRelTError</a>	56
5.6.3.3	<a href="#">dDerivativeStepFraction</a>	56
5.6.3.4	<a href="#">dMaxErrorInRHS</a>	56
5.6.3.5	<a href="#">dTolerance</a>	57
5.6.3.6	<a href="#">kspContext</a>	57
5.6.3.7	<a href="#">matCoeff</a>	57
5.6.3.8	<a href="#">nCurrentNumIterations</a>	57
5.6.3.9	<a href="#">nLocDer</a>	57
5.6.3.10	<a href="#">nLocFun</a>	57
5.6.3.11	<a href="#">nMaxNumIterations</a>	58
5.6.3.12	<a href="#">nMaxNumSolverIterations</a>	58
5.6.3.13	<a href="#">nNumDerPerRow</a>	58
5.6.3.14	<a href="#">nNumImplicitZones</a>	58
5.6.3.15	<a href="#">nNumRowsALocal</a>	58
5.6.3.16	<a href="#">nNumRowsALocalSB</a>	58
5.6.3.17	<a href="#">nTypeDer</a>	59
5.6.3.18	<a href="#">vecRHS</a>	59
5.6.3.19	<a href="#">vecscatTCorrections</a>	59
5.6.3.20	<a href="#">vecTCorrections</a>	59

5.6.3.21	vecTCorrectionsLocal	59
5.7	MessPass Class Reference	59
5.7.1	Detailed Description	60
5.7.2	Constructor & Destructor Documentation	60
5.7.2.1	MessPass	60
5.7.3	Member Data Documentation	60
5.7.3.1	requestRecv	60
5.7.3.2	requestSend	60
5.7.3.3	statusRecv	61
5.7.3.4	statusSend	61
5.7.3.5	typeRecvNewVar	61
5.7.3.6	typeRecvOldGrid	61
5.7.3.7	typeSendNewGrid	61
5.7.3.8	typeSendNewVar	61
5.8	Output Class Reference	62
5.8.1	Detailed Description	62
5.8.2	Constructor & Destructor Documentation	62
5.8.2.1	Output	62
5.8.3	Member Data Documentation	62
5.8.3.1	bDump	62
5.8.3.2	bPrint	63
5.8.3.3	dDumpFrequencyTime	63
5.8.3.4	dPrintFrequencyTime	63
5.8.3.5	dTimeLastDump	63
5.8.3.6	dTimeLastPrint	63
5.8.3.7	nDumpFrequencyStep	63
5.8.3.8	nNumTimeStepsSinceLastPrint	63
5.8.3.9	nPrintFrequencyStep	64
5.8.3.10	nPrintMode	64
5.8.3.11	ofWatchZoneFiles	64
5.8.3.12	sBaseOutputFileName	64
5.8.3.13	watchzoneList	64
5.9	Parameters Class Reference	65
5.9.1	Detailed Description	65



5.9.2	Constructor & Destructor Documentation . . . . .	66
5.9.2.1	Parameters . . . . .	66
5.9.3	Member Data Documentation . . . . .	66
5.9.3.1	bAdiabatic . . . . .	66
5.9.3.2	bDEDMClamp . . . . .	66
5.9.3.3	bEOSGammaLaw . . . . .	66
5.9.3.4	dA . . . . .	66
5.9.3.5	dAlpha . . . . .	67
5.9.3.6	dAVThreshold . . . . .	67
5.9.3.7	dDEDMClampMr . . . . .	67
5.9.3.8	dDEDMClampValue . . . . .	67
5.9.3.9	dDonorCellMin . . . . .	67
5.9.3.10	dDonorCellMultiplier . . . . .	68
5.9.3.11	dEddyViscosity . . . . .	68
5.9.3.12	dEDMClampTemperature . . . . .	68
5.9.3.13	dG . . . . .	68
5.9.3.14	dGamma . . . . .	68
5.9.3.15	dMaxConvectiveVelocity . . . . .	69
5.9.3.16	dMaxConvectiveVelocity_c . . . . .	69
5.9.3.17	dPi . . . . .	69
5.9.3.18	dPrt . . . . .	70
5.9.3.19	dSigma . . . . .	70
5.9.3.20	dTolerance . . . . .	70
5.9.3.21	eosTable . . . . .	70
5.9.3.22	nMaxIterations . . . . .	70
5.9.3.23	nTypeTurbulenceMod . . . . .	71
5.9.3.24	sDebugProfileOutput . . . . .	71
5.9.3.25	sEOSFileName . . . . .	71
5.10	Performance Class Reference . . . . .	71
5.10.1	Detailed Description . . . . .	71
5.10.2	Constructor & Destructor Documentation . . . . .	72
5.10.2.1	Performance . . . . .	72
5.10.3	Member Data Documentation . . . . .	72
5.10.3.1	dEndTimer . . . . .	72

5.10.3.2	dStartTimer	72
5.11	ProcTop Class Reference	72
5.11.1	Detailed Description	73
5.11.2	Constructor & Destructor Documentation	73
5.11.2.1	ProcTop	73
5.11.3	Member Data Documentation	73
5.11.3.1	nCoords	73
5.11.3.2	nNeighborRanks	73
5.11.3.3	nNumNeighbors	73
5.11.3.4	nNumProcs	74
5.11.3.5	nNumRadialNeighbors	74
5.11.3.6	nPeriodic	74
5.11.3.7	nProcDims	74
5.11.3.8	nRadialNeighborNeighborIDs	74
5.11.3.9	nRadialNeighborRanks	74
5.11.3.10	nRank	75
5.12	profileData Class Reference	75
5.12.1	Constructor & Destructor Documentation	76
5.12.1.1	profileData	76
5.12.2	Member Function Documentation	76
5.12.2.1	clear	76
5.12.2.2	nMaxNumZones	76
5.12.2.3	set	76
5.12.2.4	set	76
5.12.2.5	setMax	76
5.12.2.6	setMax	76
5.12.2.7	setMaxAbs	76
5.12.2.8	setMaxAbs	77
5.12.2.9	setSum	77
5.12.2.10	setSum	77
5.12.2.11	test	77
5.12.2.12	toFile	77
5.13	Time Class Reference	77
5.13.1	Detailed Description	78

5.13.2	Constructor & Destructor Documentation . . . . .	78
5.13.2.1	Time . . . . .	78
5.13.3	Member Data Documentation . . . . .	78
5.13.3.1	bVariableTimeStep . . . . .	78
5.13.3.2	dConstTimeStep . . . . .	78
5.13.3.3	dDelE_t_E_max . . . . .	79
5.13.3.4	dDelRho_t_Rho_max . . . . .	79
5.13.3.5	dDelT_t_T_max . . . . .	79
5.13.3.6	dDeltat_n . . . . .	79
5.13.3.7	dDeltat_nm1half . . . . .	79
5.13.3.8	dDeltat_np1half . . . . .	80
5.13.3.9	dEndTime . . . . .	80
5.13.3.10	dPerChange . . . . .	80
5.13.3.11	dt . . . . .	80
5.13.3.12	dTimeStepFactor . . . . .	80
5.13.3.13	nEndTimeStep . . . . .	81
5.13.3.14	nTimeStepIndex . . . . .	81
5.14	WatchZone Class Reference . . . . .	81
5.14.1	Detailed Description . . . . .	81
<b>6</b>	<b>File Documentation</b>	<b>83</b>
6.1	/home/cgeroux/Documents/WORK/SPHERLS_git/src/eos.cpp File Reference	83
6.1.1	Detailed Description . . . . .	83
6.2	/home/cgeroux/Documents/WORK/SPHERLS_git/src/eos.h File Reference	83
6.2.1	Detailed Description . . . . .	83
6.3	dataManipulation.cpp File Reference . . . . .	84
6.3.1	Detailed Description . . . . .	85
6.3.2	Function Documentation . . . . .	85
6.3.2.1	average3DTo1DBoundariesNew . . . . .	85
6.3.2.2	average3DTo1DBoundariesOld . . . . .	85
6.3.2.3	fin . . . . .	86
6.3.2.4	init . . . . .	86
6.3.2.5	initImplicitCalculation . . . . .	87
6.3.2.6	initUpdateLocalBoundaries . . . . .	88

6.3.2.7	modelRead . . . . .	89
6.3.2.8	modelWrite_GL . . . . .	89
6.3.2.9	modelWrite_TEOS . . . . .	90
6.3.2.10	setDEDMClamp . . . . .	90
6.3.2.11	setupLocalGrid . . . . .	91
6.3.2.12	updateLocalBoundaries . . . . .	91
6.3.2.13	updateLocalBoundariesNewGrid . . . . .	92
6.3.2.14	updateLocalBoundaryVelocitiesNewGrid_R . . . . .	92
6.3.2.15	updateLocalBoundaryVelocitiesNewGrid_RT . . . . .	93
6.3.2.16	updateLocalBoundaryVelocitiesNewGrid_RTP . . . . .	93
6.3.2.17	updateNewGridWithOld . . . . .	93
6.3.2.18	updateOldGrid . . . . .	93
6.4	dataManipulation.h File Reference . . . . .	94
6.4.1	Detailed Description . . . . .	94
6.4.2	Function Documentation . . . . .	95
6.4.2.1	average3DTo1DBoundariesNew . . . . .	95
6.4.2.2	average3DTo1DBoundariesOld . . . . .	95
6.4.2.3	fin . . . . .	95
6.4.2.4	init . . . . .	96
6.4.2.5	initImplicitCalculation . . . . .	97
6.4.2.6	initUpdateLocalBoundaries . . . . .	98
6.4.2.7	modelRead . . . . .	99
6.4.2.8	modelWrite_GL . . . . .	99
6.4.2.9	modelWrite_TEOS . . . . .	100
6.4.2.10	setDEDMClamp . . . . .	100
6.4.2.11	setupLocalGrid . . . . .	101
6.4.2.12	updateLocalBoundaries . . . . .	101
6.4.2.13	updateLocalBoundariesNewGrid . . . . .	102
6.4.2.14	updateLocalBoundaryVelocitiesNewGrid_R . . . . .	102
6.4.2.15	updateLocalBoundaryVelocitiesNewGrid_RT . . . . .	102
6.4.2.16	updateLocalBoundaryVelocitiesNewGrid_RTP . . . . .	103
6.4.2.17	updateNewGridWithOld . . . . .	103
6.4.2.18	updateOldGrid . . . . .	103
6.5	dataMonitoring.cpp File Reference . . . . .	103

6.5.1	Detailed Description	104
6.5.2	Function Documentation	104
6.5.2.1	finWatchZones	104
6.5.2.2	initWatchZones	104
6.5.2.3	writeWatchZones_R_GL	105
6.5.2.4	writeWatchZones_R_TEOS	105
6.5.2.5	writeWatchZones_RT_GL	106
6.5.2.6	writeWatchZones_RT_TEOS	106
6.5.2.7	writeWatchZones_RTP_GL	107
6.5.2.8	writeWatchZones_RTP_TEOS	107
6.6	dataMonitoring.h File Reference	107
6.6.1	Detailed Description	108
6.6.2	Function Documentation	108
6.6.2.1	finWatchZones	108
6.6.2.2	initWatchZones	108
6.6.2.3	writeWatchZones_R_GL	109
6.6.2.4	writeWatchZones_R_TEOS	109
6.6.2.5	writeWatchZones_RT_GL	110
6.6.2.6	writeWatchZones_RT_TEOS	110
6.6.2.7	writeWatchZones_RTP_GL	111
6.6.2.8	writeWatchZones_RTP_TEOS	111
6.7	global.cpp File Reference	111
6.7.1	Detailed Description	112
6.8	global.h File Reference	112
6.8.1	Detailed Description	112
6.8.2	Define Documentation	113
6.8.2.1	DEBUG_EQUATIONS	113
6.8.2.2	DEDEM_CLAMP	113
6.8.2.3	DUMP_VERSION	113
6.8.2.4	SEDOV	113
6.8.2.5	SIGNEGDEN	113
6.8.2.6	SIGNEGENG	113
6.8.2.7	SIGNEGTEMP	114
6.8.2.8	TRACKMAXSOLVERERROR	114

6.8.2.9	VISCOUS_ENERGY_EQ	114
6.9	main.cpp File Reference	114
6.9.1	Detailed Description	114
6.9.2	Function Documentation	114
6.9.2.1	main	115
6.9.2.2	signalHandler	116
6.10	main.h File Reference	116
6.10.1	Detailed Description	116
6.10.2	Function Documentation	116
6.10.2.1	main	116
6.10.2.2	signalHandler	118
6.11	physEquations.cpp File Reference	118
6.11.1	Detailed Description	122
6.11.2	Function Documentation	122
6.11.2.1	calDelt_CONST	122
6.11.2.2	calDelt_R_GL	122
6.11.2.3	calDelt_R_TEOS	123
6.11.2.4	calDelt_RT_GL	124
6.11.2.5	calDelt_RT_TEOS	124
6.11.2.6	calDelt_RTP_GL	125
6.11.2.7	calDelt_RTP_TEOS	125
6.11.2.8	calNewD_R	126
6.11.2.9	calNewD_RT	126
6.11.2.10	calNewD_RTP	127
6.11.2.11	calNewDenave_None	127
6.11.2.12	calNewDenave_R	128
6.11.2.13	calNewDenave_RT	128
6.11.2.14	calNewDenave_RTP	128
6.11.2.15	calNewE_R_AD	129
6.11.2.16	calNewE_R_NA	129
6.11.2.17	calNewE_R_NA_LES	130
6.11.2.18	calNewE_RT_AD	130
6.11.2.19	calNewE_RT_NA	131
6.11.2.20	calNewE_RT_NA_LES	132

6.11.2.21 calNewE_RTP_AD . . . . .	133
6.11.2.22 calNewE_RTP_NA . . . . .	133
6.11.2.23 calNewE_RTP_NA_LES . . . . .	134
6.11.2.24 calNewEddyVisc_None . . . . .	135
6.11.2.25 calNewEddyVisc_R_CN . . . . .	135
6.11.2.26 calNewEddyVisc_R_SM . . . . .	135
6.11.2.27 calNewEddyVisc_RT_CN . . . . .	136
6.11.2.28 calNewEddyVisc_RT_SM . . . . .	136
6.11.2.29 calNewEddyVisc_RTP_CN . . . . .	136
6.11.2.30 calNewEddyVisc_RTP_SM . . . . .	137
6.11.2.31 calNewP_GL . . . . .	137
6.11.2.32 calNewPEKappaGamma_TEOS . . . . .	138
6.11.2.33 calNewQ0_R_GL . . . . .	138
6.11.2.34 calNewQ0_R_TEOS . . . . .	138
6.11.2.35 calNewQ0Q1_RT_GL . . . . .	139
6.11.2.36 calNewQ0Q1_RT_TEOS . . . . .	139
6.11.2.37 calNewQ0Q1Q2_RTP_GL . . . . .	140
6.11.2.38 calNewQ0Q1Q2_RTP_TEOS . . . . .	140
6.11.2.39 calNewR . . . . .	141
6.11.2.40 calNewTPKappaGamma_TEOS . . . . .	141
6.11.2.41 calNewU0_R . . . . .	141
6.11.2.42 calNewU0_RT . . . . .	142
6.11.2.43 calNewU0_RTP . . . . .	142
6.11.2.44 calNewU_R . . . . .	143
6.11.2.45 calNewU_R_LES . . . . .	144
6.11.2.46 calNewU_RT . . . . .	145
6.11.2.47 calNewU_RT_LES . . . . .	145
6.11.2.48 calNewU_RTP . . . . .	146
6.11.2.49 calNewU_RTP_LES . . . . .	147
6.11.2.50 calNewV_RT . . . . .	148
6.11.2.51 calNewV_RT_LES . . . . .	149
6.11.2.52 calNewV_RTP . . . . .	150
6.11.2.53 calNewV_RTP_LES . . . . .	150
6.11.2.54 calNewVelocities_R . . . . .	151

6.11.2.55 calNewVelocities_R_LES . . . . .	151
6.11.2.56 calNewVelocities_RT . . . . .	152
6.11.2.57 calNewVelocities_RT_LES . . . . .	152
6.11.2.58 calNewVelocities_RTP . . . . .	152
6.11.2.59 calNewVelocities_RTP_LES . . . . .	153
6.11.2.60 calNewW_RTP . . . . .	153
6.11.2.61 calNewW_RTP_LES . . . . .	154
6.11.2.62 calOldDenave_None . . . . .	155
6.11.2.63 calOldDenave_R . . . . .	155
6.11.2.64 calOldDenave_RT . . . . .	155
6.11.2.65 calOldDenave_RTP . . . . .	156
6.11.2.66 calOldEddyVisc_R_CN . . . . .	156
6.11.2.67 calOldEddyVisc_R_SM . . . . .	156
6.11.2.68 calOldEddyVisc_RT_CN . . . . .	157
6.11.2.69 calOldEddyVisc_RT_SM . . . . .	157
6.11.2.70 calOldEddyVisc_RTP_CN . . . . .	157
6.11.2.71 calOldEddyVisc_RTP_SM . . . . .	158
6.11.2.72 calOldP_GL . . . . .	158
6.11.2.73 calOldPEKappaGamma_TEOS . . . . .	159
6.11.2.74 calOldQ0_R_GL . . . . .	159
6.11.2.75 calOldQ0_R_TEOS . . . . .	159
6.11.2.76 calOldQ0Q1_RT_GL . . . . .	160
6.11.2.77 calOldQ0Q1_RT_TEOS . . . . .	160
6.11.2.78 calOldQ0Q1Q2_RTP_GL . . . . .	161
6.11.2.79 calOldQ0Q1Q2_RTP_TEOS . . . . .	161
6.11.2.80 dEOS_GL . . . . .	162
6.11.2.81 dET4 . . . . .	162
6.11.2.82 dImplicitEnergyFunction_None . . . . .	162
6.11.2.83 dImplicitEnergyFunction_R . . . . .	162
6.11.2.84 dImplicitEnergyFunction_R_LES . . . . .	163
6.11.2.85 dImplicitEnergyFunction_R_LES_SB . . . . .	164
6.11.2.86 dImplicitEnergyFunction_R_SB . . . . .	165
6.11.2.87 dImplicitEnergyFunction_RT . . . . .	165
6.11.2.88 dImplicitEnergyFunction_RT_LES . . . . .	166



6.11.2.89 dImplicitEnergyFunction_RT_LES_SB . . . . .	167
6.11.2.90 dImplicitEnergyFunction_RT_SB . . . . .	168
6.11.2.91 dImplicitEnergyFunction_RTP . . . . .	169
6.11.2.92 dImplicitEnergyFunction_RTP_LES . . . . .	170
6.11.2.93 dImplicitEnergyFunction_RTP_LES_SB . . . . .	171
6.11.2.94 dImplicitEnergyFunction_RTP_SB . . . . .	173
6.11.2.95 implicitSolve_None . . . . .	174
6.11.2.96 implicitSolve_R . . . . .	174
6.11.2.97 implicitSolve_RT . . . . .	174
6.11.2.98 implicitSolve_RTP . . . . .	175
6.11.2.99 initDonorFracAndMaxConVel_R_GL . . . . .	175
6.11.2.100 initDonorFracAndMaxConVel_R_TEOS . . . . .	176
6.11.2.101 initDonorFracAndMaxConVel_RT_GL . . . . .	176
6.11.2.102 initDonorFracAndMaxConVel_RT_TEOS . . . . .	176
6.11.2.103 initDonorFracAndMaxConVel_RTP_GL . . . . .	177
6.11.2.104 initDonorFracAndMaxConVel_RTP_TEOS . . . . .	177
6.11.2.105 nitInternalVars . . . . .	177
6.11.2.106 setInternalVarInf . . . . .	178
6.11.2.107 setMainFunctions . . . . .	179
6.12 physEquations.h File Reference . . . . .	180
6.12.1 Detailed Description . . . . .	184
6.12.2 Function Documentation . . . . .	184
6.12.2.1 calDelt_CONST . . . . .	184
6.12.2.2 calDelt_R_GL . . . . .	184
6.12.2.3 calDelt_R_TEOS . . . . .	184
6.12.2.4 calDelt_RT_GL . . . . .	185
6.12.2.5 calDelt_RT_TEOS . . . . .	186
6.12.2.6 calDelt_RTP_GL . . . . .	186
6.12.2.7 calDelt_RTP_TEOS . . . . .	187
6.12.2.8 calNewD_R . . . . .	187
6.12.2.9 calNewD_RT . . . . .	188
6.12.2.10 calNewD_RTP . . . . .	188
6.12.2.11 calNewDenave_None . . . . .	189
6.12.2.12 calNewDenave_R . . . . .	189

6.12.2.13 calNewDenave_RT . . . . .	189
6.12.2.14 calNewDenave_RTP . . . . .	190
6.12.2.15 calNewE_R_AD . . . . .	190
6.12.2.16 calNewE_R_NA . . . . .	191
6.12.2.17 calNewE_R_NA_LES . . . . .	191
6.12.2.18 calNewE_RT_AD . . . . .	192
6.12.2.19 calNewE_RT_NA . . . . .	193
6.12.2.20 calNewE_RT_NA_LES . . . . .	193
6.12.2.21 calNewE_RTP_AD . . . . .	194
6.12.2.22 calNewE_RTP_NA . . . . .	195
6.12.2.23 calNewE_RTP_NA_LES . . . . .	195
6.12.2.24 calNewEddyVisc_None . . . . .	196
6.12.2.25 calNewEddyVisc_R_CN . . . . .	197
6.12.2.26 calNewEddyVisc_R_SM . . . . .	197
6.12.2.27 calNewEddyVisc_RT_CN . . . . .	197
6.12.2.28 calNewEddyVisc_RT_SM . . . . .	198
6.12.2.29 calNewEddyVisc_RTP_CN . . . . .	198
6.12.2.30 calNewEddyVisc_RTP_SM . . . . .	198
6.12.2.31 calNewP_GL . . . . .	199
6.12.2.32 calNewPEKappaGamma_TEOS . . . . .	199
6.12.2.33 calNewQ0_R_GL . . . . .	200
6.12.2.34 calNewQ0_R_TEOS . . . . .	200
6.12.2.35 calNewQ0Q1_RT_GL . . . . .	200
6.12.2.36 calNewQ0Q1_RT_TEOS . . . . .	201
6.12.2.37 calNewQ0Q1Q2_RTP_GL . . . . .	201
6.12.2.38 calNewQ0Q1Q2_RTP_TEOS . . . . .	202
6.12.2.39 calNewR . . . . .	202
6.12.2.40 calNewTPKappaGamma_TEOS . . . . .	203
6.12.2.41 calNewU0_R . . . . .	203
6.12.2.42 calNewU0_RT . . . . .	204
6.12.2.43 calNewU0_RTP . . . . .	204
6.12.2.44 calNewU_R . . . . .	205
6.12.2.45 calNewU_R_LES . . . . .	205
6.12.2.46 calNewU_RT . . . . .	206

6.12.2.47 calNewU_RT_LES . . . . .	207
6.12.2.48 calNewU_RTP . . . . .	208
6.12.2.49 calNewU_RTP_LES . . . . .	209
6.12.2.50 calNewV_RT . . . . .	210
6.12.2.51 calNewV_RT_LES . . . . .	211
6.12.2.52 calNewV_RTP . . . . .	211
6.12.2.53 calNewV_RTP_LES . . . . .	212
6.12.2.54 calNewVelocities_R . . . . .	213
6.12.2.55 calNewVelocities_R_LES . . . . .	213
6.12.2.56 calNewVelocities_RT . . . . .	213
6.12.2.57 calNewVelocities_RT_LES . . . . .	214
6.12.2.58 calNewVelocities_RTP . . . . .	214
6.12.2.59 calNewVelocities_RTP_LES . . . . .	215
6.12.2.60 calNewW_RTP . . . . .	215
6.12.2.61 calNewW_RTP_LES . . . . .	216
6.12.2.62 calOldDenave_None . . . . .	216
6.12.2.63 calOldDenave_R . . . . .	217
6.12.2.64 calOldDenave_RT . . . . .	217
6.12.2.65 calOldDenave_RTP . . . . .	217
6.12.2.66 calOldEddyVisc_R_CN . . . . .	218
6.12.2.67 calOldEddyVisc_R_SM . . . . .	218
6.12.2.68 calOldEddyVisc_RT_CN . . . . .	218
6.12.2.69 calOldEddyVisc_RT_SM . . . . .	219
6.12.2.70 calOldEddyVisc_RTP_CN . . . . .	219
6.12.2.71 calOldEddyVisc_RTP_SM . . . . .	220
6.12.2.72 calOldP_GL . . . . .	220
6.12.2.73 calOldPEKappaGamma_TEOS . . . . .	220
6.12.2.74 calOldQ0_R_GL . . . . .	221
6.12.2.75 calOldQ0_R_TEOS . . . . .	221
6.12.2.76 calOldQ0Q1_RT_GL . . . . .	222
6.12.2.77 calOldQ0Q1_RT_TEOS . . . . .	222
6.12.2.78 calOldQ0Q1Q2_RTP_GL . . . . .	223
6.12.2.79 calOldQ0Q1Q2_RTP_TEOS . . . . .	223
6.12.2.80 dEOS_GL . . . . .	223

6.12.2.81 dET4 . . . . .	224
6.12.2.82 dImplicitEnergyFunction_None . . . . .	224
6.12.2.83 dImplicitEnergyFunction_R . . . . .	224
6.12.2.84 dImplicitEnergyFunction_R_LES . . . . .	225
6.12.2.85 dImplicitEnergyFunction_R_LES_SB . . . . .	226
6.12.2.86 dImplicitEnergyFunction_R_SB . . . . .	226
6.12.2.87 dImplicitEnergyFunction_RT . . . . .	227
6.12.2.88 dImplicitEnergyFunction_RT_LES . . . . .	228
6.12.2.89 dImplicitEnergyFunction_RT_LES_SB . . . . .	229
6.12.2.90 dImplicitEnergyFunction_RT_SB . . . . .	230
6.12.2.91 dImplicitEnergyFunction_RTP . . . . .	231
6.12.2.92 dImplicitEnergyFunction_RTP_LES . . . . .	232
6.12.2.93 dImplicitEnergyFunction_RTP_LES_SB . . . . .	232
6.12.2.94 dImplicitEnergyFunction_RTP_SB . . . . .	234
6.12.2.95 implicitSolve_None . . . . .	235
6.12.2.96 implicitSolve_R . . . . .	235
6.12.2.97 implicitSolve_RT . . . . .	235
6.12.2.98 implicitSolve_RTP . . . . .	236
6.12.2.99 initDonorFracAndMaxConVel_R_GL . . . . .	236
6.12.2.100initDonorFracAndMaxConVel_R_TEOS . . . . .	237
6.12.2.101initDonorFracAndMaxConVel_RT_GL . . . . .	237
6.12.2.102initDonorFracAndMaxConVel_RT_TEOS . . . . .	237
6.12.2.103initDonorFracAndMaxConVel_RTP_GL . . . . .	238
6.12.2.104initDonorFracAndMaxConVel_RTP_TEOS . . . . .	238
6.12.2.105nitInternalVars . . . . .	238
6.12.2.106setInternalVarInf . . . . .	239
6.12.2.107setMainFunctions . . . . .	240
6.13 procTop.cpp File Reference . . . . .	241
6.13.1 Detailed Description . . . . .	241
6.14 procTop.h File Reference . . . . .	241
6.14.1 Detailed Description . . . . .	241
6.15 profileData.h File Reference . . . . .	241
6.15.1 Detailed Description . . . . .	242
6.16 time.cpp File Reference . . . . .	242

---

6.16.1 Detailed Description . . . . .	242
6.17 time.h File Reference . . . . .	242
6.17.1 Detailed Description . . . . .	242
6.18 watchzone.cpp File Reference . . . . .	242
6.18.1 Detailed Description . . . . .	242
6.19 watchzone.h File Reference . . . . .	242
6.19.1 Detailed Description . . . . .	243



# Chapter 1

## Boundary Conditions

**Member `calNewD_R` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `ProcTop &procTop`)**

doesn't allow mass flux through outer interface

**Member `calNewD_RT` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `ProcTop &procTop`)**

doesn't allow mass flux through outer interface

**Member `calNewD_RTP` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `ProcTop &procTop`)**

doesn't allow mass flux through outer interface

**Member `calNewE_R_NA` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `ProcTop &procTop`)**

Setting energy at surface equal to energy in last zone.

Upwind gradient in dA1 term should be zero as there is no flow into the star.

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

**Member `calNewE_R_NA_LES` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `ProcTop &procTop`)**

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.

`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.

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

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_RT_AD` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `ProcTop &procTop`)**

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

`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 (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)**

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.

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.

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

**Member calNewE\_RT\_NA\_LES (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)**

Missing  $\Delta M_r$  outside model using Parameters::dAlpha times  $\Delta M_r$  in the last zone instead.

Setting energy at surface equal to energy in last zone.

missing density outside model, setting it to zero

missing eddy viscosity outside the model setting it to zero

Upwind gradient in dA1 term should be zero as there is no flow into the star.

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

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 (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)**

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

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 (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)**

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.

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.

Missing grid.dLocalGridOld[grid.nT][i+1][0][0]

**Member calNewE\_RTP\_NA\_LES (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)**

Missing  $\Delta M_r$  outside model using Parameters::dAlpha times  $\Delta M_r$  in the last zone instead.

Missing W at i+1, assuming the same as at i

Setting energy at surface equal to energy in last zone.

missing density outside model, setting it to zero

missing eddy viscosity outside the model setting it to zero

Upwind gradient in dA1 term should be zero as there is no flow into the star.



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

missing energy outside the model, assuming it is the same as that in the last zone.

That causes this term to be zero.

**Member `calNewEddyVisc_RT_SM (Grid &grid, Parameters &parameters)`**

assuming that theta velocity is constant across surface

**Member `calNewEddyVisc_RTP_SM (Grid &grid, Parameters &parameters)`**

assuming that theta velocity is constant across surface

assume phi velocity is constant across surface

**Member `calNewU0_RT (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop, MessPass &messPass)`**

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

**Member `calNewU0_RTP (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop, MessPass &messPass)`**

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

**Member `calNewU_R (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)`**

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

Missing density outside model, setting it to zero.

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_R_LES (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)`**

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

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

Assuming eddy viscosity outside model is zero.

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

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

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

**Member `calNewU_RT (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)`**

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

assuming theta velocity is constant across surface

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

Missing `grid.dLocalGridOld[grid.nP][nlCen+1][j][k]` in calculation of  $S_1$ , setting it to  $-1.0 * dP\_ijk\_n$ .

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

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

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

**Member `calNewU_RT_LES` (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)**

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

Missing density outside of surface, setting it to zero.

Missing density outside model, setting it to zero.

assuming theta and phi velocity same outside star as inside.

Assuming theta velocities are constant across surface.

assuming that  $v_i$  at  $i+1$  is equal to  $v_i$  at  $i$ .

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.

assume viscosity is zero outside the star.

Missing mass outside model, setting it to zero.

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

**Member `calNewU_RTP` (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)**

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

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

assuming theta velocity is constant across the surface.

assuming phi velocity is constant across the surface.

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

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

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

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

---

**Member calNewU\_RTP\_LES (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)**

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.

Missing density outside of surface, setting it to zero.

Missing density outside model, setting it to zero.

assuming theta and phi velocity same outside star as inside.

Assuming theta velocities are constant across surface.

assuming that  $v_i$  at  $i+1$  is equal to  $v_i$  at  $i$ .

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.

assume viscosity is zero outside the star.

Missing mass outside model, setting it to zero.

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 &grid, Parameters &parameters, Time &time, ProcTop &procTop)**

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

missing upwind gradient, using centred gradient instead

**Member calNewV\_RT\_LES (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)**

Assuming density outside star is zero

Assuming theta velocity is constant across surface.

Assuming eddy viscosity is zero at surface.

**Member calNewV\_RTP (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)**

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

using centered gradient for upwind gradient outside star at surface.

**Member calNewV\_RTP\_LES (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)**

Assuming density outside star is zero

Assuming theta velocity is constant across surface.

Assuming eddy viscosity is zero at surface.

**Member calNewW\_RTP (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)**

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.

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` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `ProcTop &procTop`)**

assume theta and phi velocities are constant across surface

assume eddy viscosity is zero at surface

assume upwind gradient is the same as centered gradient across surface

**Member `calOldEddyVisc_RT_SM` (`Grid &grid`, `Parameters &parameters`)**

assuming that theta velocity is constant across surface

**Member `calOldEddyVisc_RTP_SM` (`Grid &grid`, `Parameters &parameters`)**

assuming that theta velocity is constant across surface

assume phi velocity is constant across surface

**Member `dImplicitEnergyFunction_R_LES_SB` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `double dTemps[]`, `int i`, `int j`, `int k`)**

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.

`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.

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

**Member `dImplicitEnergyFunction_R_SB` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `double dTemps[]`, `int i`, `int j`, `int k`)**

missing density outside model assuming it is zero

missing density outside model assuming it is zero

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

A1 upwind set to zero as no material is flowing into the star

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

**Member `dImplicitEnergyFunction_RT_LES_SB` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `double dTemps[]`, `int i`, `int j`, `int k`)**

Missing  $\Delta M_r$  outside model using `Parameters::dAlpha` times  $\Delta M_r$  in the last zone instead.

missing density outside model assuming it is zero

missing density outside model assuming it is zero

assuming V at ip1half is the same as V at i

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

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

A1 upwind set to zero as no material is flowing into the star

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

---

**Member `dImplicitEnergyFunction_RT_SB` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `double dTemps[]`, `int i`, `int j`, `int k`)**

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

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

**Member `dImplicitEnergyFunction_RTP_LES_SB` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `double dTemps[]`, `int i`, `int j`, `int k`)**

Missing  $\Delta M_r$  outside model using `Parameters::dAlpha` times  $\Delta M_r$  in the last zone instead.

missing density outside model assuming it is zero

missing desnity outside model assuming it is zero

assuming  $V$  at  $ip1half$  is the same as  $V$  at  $i$

assuming  $W$  at  $ip1half$  is the same as  $W$  at  $i$

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

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

A1 upwind set to zero as no material is flowing into the star

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

**Member `dImplicitEnergyFunction_RTP_SB` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `double dTemps[]`, `int i`, `int j`, `int k`)**

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

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

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



## Chapter 2

## Todo List

**Member `calNewU0_R` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `ProcTop &procTop`, `MessPass &messPass`)**

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` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `ProcTop &procTop`, `MessPass &messPass`)**

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` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `ProcTop &procTop`, `MessPass &messPass`)**

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 `dlImplicitEnergyFunction_R_LES` (`Grid &grid`, `Parameters &parameters`, `Time &time`, `double dTemps[]`, `int i`, `int j`, `int k`)**

this function should probably be turfed, the LES terms aren't needed in 1D. keeping it for now though.

**Member `initImplicitCalculation` (`Implicit &implicit`, `Grid &grid`, `ProcTop &procTop`, `int nNumArgs`, `char *cArgs[]`)**

`isFrom`, `isTo`, `matCoeff`, `vecTCorrections`, `vecTCorrections`, `vecRHS`, `vecTCorrections-Local`, `kspContext`, `vecscatTCorrections` all need to be destroyed before program finishes.

**Member `modelRead` (`std::string sFileName`, `ProcTop &procTop`, `Grid &grid`, `Time &time`, `Parameters &parameters`)**

At some point should get it working with only 1 processor

**Member `Time::nTimeStepIndex`**

should probably make this an unsigned variable, and perhaps also use the keyword `long` to help ensure there are enough values. Often need 7 decimal places.

**Member `updateLocalBoundaries` (`ProcTop &procTop`, `MessPass &messPass`, `Grid &grid`)**

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` (int nVar, `ProcTop` &procTop, `MessPass` &messPass, `Grid` &grid)**

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.



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">eos</a>	15
<a href="#">exception2</a>	26
<a href="#">Functions</a>	26
<a href="#">Global</a>	29
<a href="#">Grid</a>	31
<a href="#">Implicit</a>	55
<a href="#">MessPass</a>	59
<a href="#">Output</a>	62
<a href="#">Parameters</a>	65
<a href="#">Performance</a>	71
<a href="#">ProcTop</a>	72
<a href="#">profileData</a>	75
<a href="#">Time</a>	77
<a href="#">WatchZone</a>	81



## Chapter 4

# File Index

### 4.1 File List

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

/home/cgeroux/Documents/WORK/SPHERLS_git/src/eos.cpp . . . . .	83
/home/cgeroux/Documents/WORK/SPHERLS_git/src/eos.h . . . . .	83
/home/cgeroux/Documents/WORK/SPHERLS_git/src/exception2.h . . . . .	??
/home/cgeroux/Documents/WORK/SPHERLS_git/src/xmlFunctions.h . . . . .	??
dataManipulation.cpp . . . . .	84
dataManipulation.h . . . . .	94
dataMonitoring.cpp . . . . .	103
dataMonitoring.h . . . . .	107
fileExists.h . . . . .	??
global.cpp . . . . .	111
global.h . . . . .	112
main.cpp . . . . .	114
main.h . . . . .	116
physEquations.cpp . . . . .	118
physEquations.h . . . . .	180
procTop.cpp . . . . .	241
procTop.h . . . . .	241
profileData.h . . . . .	241
time.cpp . . . . .	242
time.h . . . . .	242
watchzone.cpp . . . . .	242
watchzone.h . . . . .	242



## Chapter 5

# Class Documentation

### 5.1 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)
- void [getDlnPDlnTDlnPDlnPDEDT](#) (double dT, double dRho, double &dDlnPDlnT, double &dDlnPDlnRho, double &dDEDT)

### Public Attributes

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

### 5.1.1 Detailed Description

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

### 5.1.2 Constructor & Destructor Documentation

#### 5.1.2.1 `eos::eos ( )`

Constructor, doesn't really do anything

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

#### 5.1.2.2 `eos::eos ( int nNumT, int nNumRho )`

Constructor, allocates memory for the 2D arrays

#### Parameters

in	<i>nNumT</i>	number of temperatures in the equaiton of state table
in	<i>nNumRho</i>	number of densities in the equaiton of state table

### 5.1.2.3 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.

### 5.1.2.4 eos::~~eos ( )

Destructor, deletes dynamic arrays

References dLogE, dLogKappa, dLogP, and nNumRho.

## 5.1.3 Member Function Documentation

### 5.1.3.1 double eos::dDRhoDP ( double dT, double dRho )

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

#### Parameters

in	$dT$	temperature at which the derivative is to be computed
in	$dRho$	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.

### 5.1.3.2 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

in	$dT$	temperature to interpolate to.
in	$dRho$	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_R_SB()`, `dImplicitEnergyFunction_RT()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RT_SB()`, `dImplicitEnergyFunction_RTP()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, and `dImplicitEnergyFunction_RTP_SB()`.

#### 5.1.3.3 `double eos::dGetOpacity ( double dT, double dRho )`

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

##### Parameters

<code>in</code>	<code>dT</code>	temperature to interpolate to.
<code>in</code>	<code>dRho</code>	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_R_SB()`, `dImplicitEnergyFunction_RT()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RT_SB()`, `dImplicitEnergyFunction_RTP()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, and `dImplicitEnergyFunction_RTP_SB()`.

#### 5.1.3.4 `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

<code>in</code>	<code>dT</code>	temperature to interpolate to.
<code>in</code>	<code>dRho</code>	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_R_SB()`, `dImplicitEnergyFunction_RT()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RT_SB()`, `dImplicitEnergyFunction_RTP()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, and `dImplicitEnergyFunction_RTP_SB()`.



EnergyFunction\_RT(), dImplicitEnergyFunction\_RT\_LES(), dImplicitEnergyFunction\_RT\_LES\_SB(), dImplicitEnergyFunction\_RT\_SB(), dImplicitEnergyFunction\_RTP(), dImplicitEnergyFunction\_RTP\_LES(), dImplicitEnergyFunction\_RTP\_LES\_SB(), and dImplicitEnergyFunction\_RTP\_SB().

#### 5.1.3.5 double eos::dSoundSpeed( double *dT*, double *dRho* )

This function calculates the adiabatic sound speed

##### Parameters

in	<i>dT</i>	temperature at which the derivative is to be computed
in	<i>dRho</i>	density at which the derivative is to be computed

##### Returns

the sound speed.

References dLogE, dLogP, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, n-NumRho, and nNumT.

#### 5.1.3.6 void eos::gamma1DelAdC\_v( double *dT*, double *dRho*, double & *dGamma1*, double & *dDelAd*, double & *dC\_v* )

This function calculates gamma1 and the adiabatic gradient

##### Parameters

in	<i>dT</i>	temperature at which the derivative is to be computed
in	<i>dRho</i>	density at which the derivative is to be computed
out	<i>dGamma1</i>	gamma1
out	<i>dDelAd</i>	adiabatic gradient
out	<i>dC_v</i>	specific heat at constant volume

References dLogE, dLogP, dLogRhoDelta, dLogRhoMin, dLogTDelta, dLogTMin, n-NumRho, and nNumT.

#### 5.1.3.7 void eos::getDlnPDlnTDlnPDlnPDEDT( double *dT*, double *dRho*, double & *dDlnPDlnT*, double & *dDlnPDlnRho*, double & *dDEDT* )

This function calculates various partial derivatives

##### Parameters

in	<i>dT</i>	temperature at which the derivative is to be computed
in	<i>dRho</i>	density at which the derivative is to be computed
out	<i>dDlnPDlnT</i>	derivative of ln(P) w.r.t. ln(T)

out	$d\ln P/d\ln \rho$	derivative of $\ln(P)$ w.r.t. $\ln(\rho)$
out	$dE/dT$	derivative of temperature w.r.t. energy at constant density

References  $d\log E$ ,  $d\log \kappa$ ,  $d\log P$ ,  $d\log \rho_{\Delta}$ ,  $d\log \rho_{\min}$ ,  $d\log T_{\Delta}$ ,  $d\log T_{\min}$ ,  $n_{\text{NumRho}}$ , and  $n_{\text{NumT}}$ .

#### 5.1.3.8 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

in	$dT$	temperature at which the derivative is to be computed
in	$d\rho$	density at which the derivative is to be computed
out	$dE$	energy at $dT$ and $d\rho$
out	$dDTDE$	derivative of temperature w.r.t. energy at constant density

References  $d\log E$ ,  $d\log \rho_{\Delta}$ ,  $d\log \rho_{\min}$ ,  $d\log T_{\Delta}$ ,  $d\log T_{\min}$ ,  $n_{\text{NumRho}}$ , and  $n_{\text{NumT}}$ .

Referenced by `calNewTPKappaGamma_TEOS()`.

#### 5.1.3.9 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  $d\rho$  are not in log space.

##### Parameters

in	$dT$	temperature to interpolate to.
in	$d\rho$	density to interpolate to.
out	$dE$	energy at $dT$ and $d\rho$ .
out	$d\kappa$	opacity at $dT$ and $d\rho$ .

References  $d\log E$ ,  $d\log \kappa$ ,  $d\log \rho_{\Delta}$ ,  $d\log \rho_{\min}$ ,  $d\log T_{\Delta}$ ,  $d\log T_{\min}$ ,  $n_{\text{NumRho}}$ , and  $n_{\text{NumT}}$ .

#### 5.1.3.10 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

in	$dT$	temperature at which the derivative is to be computed
in	$dRho$	density at which the derivative is to be computed
out	$dP$	pressure at $dT$ and $dRho$
out	$dDRhoDP$	derivative of density w.r.t. pressure at conatant temperature

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

**5.1.3.11** `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

in	$dT$	temperature to interpolate to.
in	$dRho$	density to interpolate to.
out	$dP$	pressure at $dT$ and $dRho$ .
out	$dE$	energy at $dT$ and $dRho$ .
out	$dKappa$	opacity at $dT$ and $dRho$ .

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

**5.1.3.12** `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

in	$dT$	temperature to interpolate to.
in	$dRho$	density to interpolate to.
out	$dP$	pressure at $dT$ and $dRho$ .
out	$dE$	energy at $dT$ and $dRho$ .
out	$dKappa$	opacity at $dT$ and $dRho$ .
out	$dGamma$	adiabatic index at $dT$ and $dRho$ .

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

Referenced by `calNewPEKappaGamma_TEOS()`, and `calOldPEKappaGamma_TEOS()`.

**5.1.3.13** `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

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

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

**5.1.3.14** `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

in	dT	temperature to interpolate to.
in	dRho	density to interpolate to.
out	dP	pressure at dT and dRho.
out	dKappa	opacity at dT and dRho.
out	dGamma	adiabatic index at dT and dRho.

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

Referenced by calNewTPKappaGamma\_TEOS().

**5.1.3.15** `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.

**5.1.3.16** `void eos::readAscii( std::string sFileName )`

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

## Parameters

in	<i>sFileName</i>	name of the equation of state file to read from.
----	------------------	--

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

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

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

## Parameters

in	<i>sFileName</i>	name of the equation of state file to read from.
----	------------------	--

Referenced by init().

#### 5.1.3.18 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

in	<i>sFileName</i>	name of the equation of state file to read from.
----	------------------	--

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

#### 5.1.3.19 void eos::writeAscii ( std::string *sFileName* )

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

## Parameters

in	<i>sFileName</i>	name of the file to write the equation of state to.
----	------------------	---

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

#### 5.1.3.20 void eos::writeBin ( std::string *sFileName* )

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

## Parameters

in	<i>sFileName</i>	name of the file to write the equaiton of state to.
----	------------------	---

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

## 5.1.4 Member Data Documentation

### 5.1.4.1 `double** eos::dLogE`

2D array of log10 energies. `dLogE[i][j]` gives the log10 energy at log10 density of `eos::dLogRhoDelta*i+eos::dLogRhoMin`, and at log10 temperature of `eos::dLogTDelta*j+eos::dLogTMin`.

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

### 5.1.4.2 `double** eos::dLogKappa`

2D array of log10 opacities. `dLogKappa[i][j]` gives the log10 opacity at log10 density of `eos::dLogRhoDelta*i+eos::dLogRhoMin`, and at log10 temperature of `eos::dLogTDelta*j+eos::dLogTMin`.

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

### 5.1.4.3 `double** eos::dLogP`

2D array of log10 pressures. `dLogP[i][j]` gives the log10 pressure at log10 density of `eos::dLogRhoDelta*i+eos::dLogRhoMin`, and at log10 temperature of `eos::dLogTDelta*j+eos::dLogTMin`.

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

### 5.1.4.4 `double eos::dLogRhoDelta`

Increment of the density between table entries in log10.

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

**5.1.4.5 double eos::dLogRhoMin**

Minimum density of the table in log10.

Referenced by dDRhoDP(), dGetEnergy(), dGetOpacity(), dGetPressure(), dSoundSpeed(), eos(), gamma1DelAdC\_v(), getDlnPDlnTDlnPDlnPDEDT(), getEAndDTDE(), getEKappa(), getPAndDRhoDP(), getPEKappa(), getPEKappaGamma(), getPEKappaGammaCp(), getPKappaGamma(), operator=(), readAscii(), readBobsAscii(), writeAscii(), and writeBin().

**5.1.4.6 double eos::dLogTDelta**

Increment of the temperature between table entries in log10.

Referenced by dDRhoDP(), dGetEnergy(), dGetOpacity(), dGetPressure(), dSoundSpeed(), eos(), gamma1DelAdC\_v(), getDlnPDlnTDlnPDlnPDEDT(), getEAndDTDE(), getEKappa(), getPAndDRhoDP(), getPEKappa(), getPEKappaGamma(), getPEKappaGammaCp(), getPKappaGamma(), operator=(), readAscii(), readBobsAscii(), writeAscii(), and writeBin().

**5.1.4.7 double eos::dLogTMin**

Minimum temperature of the table in log10.

Referenced by dDRhoDP(), dGetEnergy(), dGetOpacity(), dGetPressure(), dSoundSpeed(), eos(), gamma1DelAdC\_v(), getDlnPDlnTDlnPDlnPDEDT(), getEAndDTDE(), getEKappa(), getPAndDRhoDP(), getPEKappa(), getPEKappaGamma(), getPEKappaGammaCp(), getPKappaGamma(), operator=(), readAscii(), readBobsAscii(), writeAscii(), and writeBin().

**5.1.4.8 double eos::dXMassFrac**

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

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

**5.1.4.9 double eos::dYMassFrac**

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

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

**5.1.4.10 int eos::nNumRho**

Number of densities in the equation of state table

Referenced by dDRhoDP(), dGetEnergy(), dGetOpacity(), dGetPressure(), dSoundSpeed(), eos(), gamma1DelAdC\_v(), getDlnPDlnTDlnPDlnPDEDT(), getEAndDTDE(),

getEKappa(), getPAndDRhoDP(), getPEKappa(), getPEKappaGamma(), getPEKappaGammaCp(), getPKappaGamma(), operator=(), readAscii(), readBobsAscii(), writeAscii(), writeBin(), and ~eos().

#### 5.1.4.11 int eos::nNumT

Number of temperatures in the equation of state table

Referenced by dDRhoDP(), dGetEnergy(), dGetOpacity(), dGetPressure(), dSoundSpeed(), eos(), gamma1DelAdC\_v(), getDlnPDlnTDlnPDlnPDEDT(), getEAndDTDE(), getEKappa(), getPAndDRhoDP(), getPEKappa(), getPEKappaGamma(), getPEKappaGammaCp(), getPKappaGamma(), operator=(), readAscii(), readBobsAscii(), writeAscii(), and writeBin().

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

- [/home/cgeroux/Documents/WORK/SPHERLS\\_git/src/eos.h](#)
- [/home/cgeroux/Documents/WORK/SPHERLS\\_git/src/eos.cpp](#)

## 5.2 exception2 Class Reference

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

- [/home/cgeroux/Documents/WORK/SPHERLS\\_git/src/exception2.h](#)
- [/home/cgeroux/Documents/WORK/SPHERLS\\_git/src/exception2.cpp](#)

## 5.3 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 &)

### 5.3.1 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.

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 Functions::Functions( )

Constructor for the class [Functions](#).

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

### 5.3.3 Member Data Documentation

#### 5.3.3.1 void(\* Functions::fpCalculateAveDensities)(Grid &)

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

Referenced by [Functions\(\)](#), [main\(\)](#), and [setMainFunctions\(\)](#).

#### 5.3.3.2 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\(\)](#).

**5.3.3.3 void(\* Functions::fpCalculateNewAV)(Grid &, Parameters &)**

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

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

**5.3.3.4 void(\* Functions::fpCalculateNewDensities)(Grid &, Parameters &, Time &, ProcTop &)**

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

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

**5.3.3.5 void(\* Functions::fpCalculateNewEddyVisc)(Grid &, Parameters &)**

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

Referenced by main(), and setMainFunctions().

**5.3.3.6 void(\* Functions::fpCalculateNewEnergies)(Grid &, Parameters &, Time &, ProcTop &)**

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

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

**5.3.3.7 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().

**5.3.3.8 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().

**5.3.3.9 void(\* Functions::fpCalculateNewRadii)(Grid &, Time &)**

Function pointer to the function used to calculate new radii.

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

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

Function pointer to the function used to calculate new velocities.

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

**5.3.3.11** `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()`.

**5.3.3.12** `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()`.

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

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

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

**5.3.3.14** `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()`.

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

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

## 5.4 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

### 5.4.1 Detailed Description

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

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 `Global::Global( )`

Constructor for the class [Global](#).

### 5.4.3 Member Data Documentation

#### 5.4.3.1 Functions `Global::functions`

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

Referenced by `main()`.

#### 5.4.3.2 Grid `Global::grid`

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

Referenced by `main()`.

#### 5.4.3.3 Implicit `Global::implicit`

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

Referenced by `main()`.

#### 5.4.3.4 MessPass Global::messPass

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

Referenced by `main()`.

#### 5.4.3.5 Output Global::output

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

Referenced by `main()`.

#### 5.4.3.6 Parameters Global::parameters

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

Referenced by `main()`.

#### 5.4.3.7 Performance Global::performance

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

Referenced by `main()`.

#### 5.4.3.8 ProcTop Global::procTop

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

Referenced by `main()`.

#### 5.4.3.9 Time Global::time

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

Referenced by `main()`.

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

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

## 5.5 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 nSinThetaJp1halfK](#)
- [int nCotThetaJp1halfK](#)
- [int nCotThetaIJK](#)
- [int nDCosThetaIJK](#)
- [int nEddyVisc](#)
- [int nDonorCellFrac](#)
- [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]

### 5.5.1 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)																																																												
<table><tr><th>Variable</th><th>Index</th></tr><tr><td>nM</td><td>0</td></tr><tr><td>nDM</td><td>1</td></tr><tr><td>nR</td><td>2</td></tr><tr><td>nD</td><td>3</td></tr><tr><td>nU</td><td>4</td></tr><tr><td>nU0</td><td>5</td></tr><tr><td>nE</td><td>6</td></tr></table>	Variable	Index	nM	0	nDM	1	nR	2	nD	3	nU	4	nU0	5	nE	6	<table><tr><th>Variable</th><th>Index</th></tr><tr><td>nM</td><td>0</td></tr><tr><td>nTheta</td><td>1</td></tr><tr><td>nDM</td><td>2</td></tr><tr><td>nR</td><td>3</td></tr><tr><td>nD</td><td>4</td></tr><tr><td>nU</td><td>5</td></tr><tr><td>nU0</td><td>6</td></tr><tr><td>nV</td><td>7</td></tr><tr><td>nE</td><td>8</td></tr></table>	Variable	Index	nM	0	nTheta	1	nDM	2	nR	3	nD	4	nU	5	nU0	6	nV	7	nE	8	<table><tr><th>Variable</th><th>Index</th></tr><tr><td>nM</td><td>0</td></tr><tr><td>nTheta</td><td>1</td></tr><tr><td>nPhi</td><td>2</td></tr><tr><td>nDM</td><td>3</td></tr><tr><td>nR</td><td>4</td></tr><tr><td>nD</td><td>5</td></tr><tr><td>nU</td><td>6</td></tr><tr><td>nU0</td><td>7</td></tr><tr><td>nV</td><td>8</td></tr><tr><td>nW</td><td>9</td></tr><tr><td>nE</td><td>10</td></tr></table>	Variable	Index	nM	0	nTheta	1	nPhi	2	nDM	3	nR	4	nD	5	nU	6	nU0	7	nV	8	nW	9	nE	10
Variable	Index																																																													
nM	0																																																													
nDM	1																																																													
nR	2																																																													
nD	3																																																													
nU	4																																																													
nU0	5																																																													
nE	6																																																													
Variable	Index																																																													
nM	0																																																													
nTheta	1																																																													
nDM	2																																																													
nR	3																																																													
nD	4																																																													
nU	5																																																													
nU0	6																																																													
nV	7																																																													
nE	8																																																													
Variable	Index																																																													
nM	0																																																													
nTheta	1																																																													
nPhi	2																																																													
nDM	3																																																													
nR	4																																																													
nD	5																																																													
nU	6																																																													
nU0	7																																																													
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)																																																												
<table><tr><th>Variable</th><th>Index</th></tr><tr><td>nM</td><td>0</td></tr><tr><td>nDM</td><td>1</td></tr><tr><td>nR</td><td>2</td></tr><tr><td>nD</td><td>3</td></tr><tr><td>nU</td><td>4</td></tr><tr><td>nU0</td><td>5</td></tr><tr><td>nT</td><td>6</td></tr></table>	Variable	Index	nM	0	nDM	1	nR	2	nD	3	nU	4	nU0	5	nT	6	<table><tr><th>Variable</th><th>Index</th></tr><tr><td>nM</td><td>0</td></tr><tr><td>nTheta</td><td>1</td></tr><tr><td>nDM</td><td>2</td></tr><tr><td>nR</td><td>3</td></tr><tr><td>nD</td><td>4</td></tr><tr><td>nU</td><td>5</td></tr><tr><td>nU0</td><td>6</td></tr><tr><td>nV</td><td>7</td></tr><tr><td>nT</td><td>8</td></tr></table>	Variable	Index	nM	0	nTheta	1	nDM	2	nR	3	nD	4	nU	5	nU0	6	nV	7	nT	8	<table><tr><th>Variable</th><th>Index</th></tr><tr><td>nM</td><td>0</td></tr><tr><td>nTheta</td><td>1</td></tr><tr><td>nPhi</td><td>2</td></tr><tr><td>nDM</td><td>3</td></tr><tr><td>nR</td><td>4</td></tr><tr><td>nD</td><td>5</td></tr><tr><td>nU</td><td>6</td></tr><tr><td>nU0</td><td>7</td></tr><tr><td>nV</td><td>8</td></tr><tr><td>nW</td><td>9</td></tr><tr><td>nT</td><td>10</td></tr></table>	Variable	Index	nM	0	nTheta	1	nPhi	2	nDM	3	nR	4	nD	5	nU	6	nU0	7	nV	8	nW	9	nT	10
Variable	Index																																																													
nM	0																																																													
nDM	1																																																													
nR	2																																																													
nD	3																																																													
nU	4																																																													
nU0	5																																																													
nT	6																																																													
Variable	Index																																																													
nM	0																																																													
nTheta	1																																																													
nDM	2																																																													
nR	3																																																													
nD	4																																																													
nU	5																																																													
nU0	6																																																													
nV	7																																																													
nT	8																																																													
Variable	Index																																																													
nM	0																																																													
nTheta	1																																																													
nPhi	2																																																													
nDM	3																																																													
nR	4																																																													
nD	5																																																													
nU	6																																																													
nU0	7																																																													
nV	8																																																													
nW	9																																																													
nT	10																																																													

Internal variables with GL gas equation of state:

1D (nNumIntVars=2)	2D (nNumIntVars=8)																										
<table> <tr> <th>Variable</th><th>Index</th></tr> <tr> <td>nP</td><td>nNumVars+0</td></tr> <tr> <td>nQ0</td><td>nNumVars+1</td></tr> </table>	Variable	Index	nP	nNumVars+0	nQ0	nNumVars+1	<table> <tr> <th>Variable</th><th>Index</th></tr> <tr> <td>nP</td><td>nNumVars+0</td></tr> <tr> <td>nQ0</td><td>nNumVars+1</td></tr> <tr> <td>nDenAve</td><td>nNumVars+2</td></tr> <tr> <td>nDCosThetaJK</td><td>nNumVars+3</td></tr> <tr> <td>nQ1</td><td>nNumVars+4</td></tr> <tr> <td>nDTheta</td><td>nNumVars+5</td></tr> <tr> <td>nSinThetaJK</td><td>nNumVars+6</td></tr> <tr> <td>nSinThetaJp1halfK</td><td>nNumVars+7</td></tr> </table>	Variable	Index	nP	nNumVars+0	nQ0	nNumVars+1	nDenAve	nNumVars+2	nDCosThetaJK	nNumVars+3	nQ1	nNumVars+4	nDTheta	nNumVars+5	nSinThetaJK	nNumVars+6	nSinThetaJp1halfK	nNumVars+7		
Variable	Index																										
nP	nNumVars+0																										
nQ0	nNumVars+1																										
Variable	Index																										
nP	nNumVars+0																										
nQ0	nNumVars+1																										
nDenAve	nNumVars+2																										
nDCosThetaJK	nNumVars+3																										
nQ1	nNumVars+4																										
nDTheta	nNumVars+5																										
nSinThetaJK	nNumVars+6																										
nSinThetaJp1halfK	nNumVars+7																										
3D (nNumIntVars=12)																											
<table> <tr> <th>Variable</th><th>Index</th></tr> <tr> <td>nP</td><td>nNumVars+0</td></tr> <tr> <td>nQ0</td><td>nNumVars+1</td></tr> <tr> <td>nDenAve</td><td>nNumVars+2</td></tr> <tr> <td>nDPhi</td><td>nNumVars+3</td></tr> <tr> <td>nDCosThetaJK</td><td>nNumVars+4</td></tr> <tr> <td>nQ1</td><td>nNumVars+5</td></tr> <tr> <td>nDTheta</td><td>nNumVars+6</td></tr> <tr> <td>nSinThetaJK</td><td>nNumVars+7</td></tr> <tr> <td>nSinThetaJp1halfK</td><td>nNumVars+8</td></tr> <tr> <td>nCotThetaJK</td><td>nNumVars+9</td></tr> <tr> <td>nCotThetaJp1halfK</td><td>nNumVars+10</td></tr> <tr> <td>nQ2</td><td>nNumVars+11</td></tr> </table>	Variable	Index	nP	nNumVars+0	nQ0	nNumVars+1	nDenAve	nNumVars+2	nDPhi	nNumVars+3	nDCosThetaJK	nNumVars+4	nQ1	nNumVars+5	nDTheta	nNumVars+6	nSinThetaJK	nNumVars+7	nSinThetaJp1halfK	nNumVars+8	nCotThetaJK	nNumVars+9	nCotThetaJp1halfK	nNumVars+10	nQ2	nNumVars+11	
Variable	Index																										
nP	nNumVars+0																										
nQ0	nNumVars+1																										
nDenAve	nNumVars+2																										
nDPhi	nNumVars+3																										
nDCosThetaJK	nNumVars+4																										
nQ1	nNumVars+5																										
nDTheta	nNumVars+6																										
nSinThetaJK	nNumVars+7																										
nSinThetaJp1halfK	nNumVars+8																										
nCotThetaJK	nNumVars+9																										
nCotThetaJp1halfK	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.

## 5.5.2 Constructor & Destructor Documentation

### 5.5.2.1 Grid::Grid ( )

sets how many zones out from the 1D-multi-D boundary that theta/phi velocities are not updated and thus kept at zero. Constructor for the class `Grid`.

References `dLocalGridNew`, `dLocalGridOld`, `nCenIntOffset`, `nCotThetaIJK`, `nCotThetaIjp1halfK`, `nD`, `nDCosThetaIJK`, `nDenAve`, `nDM`, `nDonorCellFrac`, `nDPhi`, `nDTheta`, `nE`, `nEddyVisc`, `nEndGhostUpdateExplicit`, `nEndGhostUpdateImplicit`, `nEndUpdateExplicit`, `nEndUpdateImplicit`, `nGamma`, `nGlobalGridDims`, `nKappa`, `nLocalGridDims`, `nM`, `nP`,

nPhi, nQ0, nQ1, nQ2, nR, nSinThetaJK, nSinThetaJp1halfK, nStartGhostUpdate-Explicit, nStartGhostUpdateImplicit, nStartUpdateExplicit, nStartUpdateImplicit, nT, nTheta, nU, nU0, nV, nVariables, and nW.

### 5.5.3 Member Data Documentation

#### 5.5.3.1 `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()`.

#### 5.5.3.2 `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_RT_CN()`, `calNewEddyVisc_RT_SM()`, `calNewEddyVisc_RTP_CN()`, `calNewEddyVisc_RTP_SM()`, `calNewQ0Q1_RT_GL()`, `calNewQ0Q1_RT_TEOS()`, `calNewQ0Q1Q2_RTP_GL()`, `calNewQ0Q1Q2_RTP_TEOS()`, `calNewR()`, `calNewTPKappaGamma_TEOS()`, `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()`, `setDEDMClamp()`, `setupLocalGrid()`, `updateLocalBoundaries()`, `updateNewGridWithOld()`, `updateOldGrid()`, `writeWatchZones_R_GL()`, `writeWatchZones_R_TEOS()`, `writeWatchZones_RT_GL()`, `writeWatchZones_RT_TEOS()`, and `writeWatchZones_RTP_TEOS()`.

### 5.5.3.3 `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_T-`

EOS(), calNewQ0Q1Q2\_RTP\_GL(), calNewQ0Q1Q2\_RTP\_TEOS(), 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(), setDEDMClamp(), setupLocalGrid(), writeWatchZones\_R\_GL(), writeWatchZones\_R\_TEOS(), writeWatchZones\_RT\_GL(), writeWatchZones\_RT\_TEOS(), writeWatchZones\_RTP\_GL(), and writeWatchZones\_RTP\_TEOS().

#### 5.5.3.4 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().

#### 5.5.3.5 int Grid::nCotThetaJp1halfK

Index of  $\cot \theta$  at  $\theta$  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().

#### 5.5.3.6 int Grid::nD

Index of  $p$  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\_S-

M(), 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\_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(), 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().

#### 5.5.3.7 int Grid::nDCosThetaJK

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\(\)](#).

#### 5.5.3.8 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\\_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\\_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\\_R\(\)](#), [dImplicitEnergyFunction\\_R\\_LES\(\)](#), [dImplicitEnergyFunction\\_R\\_SB\(\)](#), [dImplicitEnergyFunction\\_RT\(\)](#), [dImplicitEnergyFunction\\_RT\\_LES\(\)](#), [dImplicitEnergyFunction\\_RT\\_LES\\_SB\(\)](#), [dImplicitEnergyFunction\\_RT\\_SB\(\)](#), [dImplicitEnergyFunction\\_RTP\(\)](#), [dImplicitEnergyFunction\\_RTP\\_L](#)

ES(), dImplicitEnergyFunction\_RTP\_LES\_SB(), dImplicitEnergyFunction\_RTP\_SB(), Grid(), main(), modelRead(), setInternalVarInf(), writeWatchZones\_RT\_GL(), writeWatchZones\_RT\_TEOS(), writeWatchZones\_RTP\_GL(), and writeWatchZones\_RTP\_TEOS().

#### 5.5.3.9 int Grid::nDM

Index of  $\delta M$  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(), setDEDMClamp(), writeWatchZones\_R\_GL(), writeWatchZones\_R\_TEOS(), writeWatchZones\_RT\_GL(), writeWatchZones\_RT\_TEOS(), writeWatchZones\_RTP\_GL(), and writeWatchZones\_RTP\_TEOS().

#### 5.5.3.10 int Grid::nDonorCellFrac

Index of the amount of donor cell to use at that particular radial zone. It is defined at zone centers, and is an internal grid variable and is included in the count of [Grid::nNumIntVars](#).

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\_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(), 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(), and setInternalVarInf().

## 5.5.3.11 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()`, `dImplicitEnergyFunction_RTP()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, `dImplicitEnergyFunction_RTP_SB()`, `Grid()`, `initInternalVars()`, `modelRead()`, and `setInternalVarInf()`.

## 5.5.3.12 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()`, `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()`, `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()`.

## 5.5.3.13 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()`, `dImplicitEnergy-`



Function\_RT\_SB(), dImplicitEnergyFunction\_RTP(), dImplicitEnergyFunction\_RTP\_LES(), dImplicitEnergyFunction\_RTP\_LES\_SB(), dImplicitEnergyFunction\_RTP\_SB(), Grid(), initUpdateLocalBoundaries(), modelRead(), setDEDMClamp(), setInternalVar-Inf(), writeWatchZones\_R\_GL(), writeWatchZones\_R\_TEOS(), writeWatchZones\_RT\_GL(), writeWatchZones\_RT\_TEOS(), writeWatchZones\_RTP\_GL(), and writeWatchZones\_RTP\_TEOS().

#### 5.5.3.14 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(), main(), modelRead(), setInternalVar-Inf(), writeWatchZones\_RT\_GL(), writeWatchZones\_RT\_TEOS(), writeWatchZones\_RTP\_GL(), and writeWatchZones\_RTP\_TEOS().

#### 5.5.3.15 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(), calOld-



EddyVisc\_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().

#### 5.5.3.16 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().

#### 5.5.3.17 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(), setDEDMClamp(), and updateOldGrid().

### 5.5.3.18 `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()`, `setDEDMClamp()`, and `updateOldGrid()`.

### 5.5.3.19 `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()`.

### 5.5.3.20 `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  $\theta$  and  $\phi$  dimensions are set to 1 or just the  $\phi$  dimensions is set to 1 depending on the number of dimensions. The  $r$ ,  $\theta$  and  $\phi$  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()`.

### 5.5.3.21 `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 `calNewD_R()`, `calNewD_RT()`, `calNewD_RTP()`, `calNewE_R_NA()`, `calNewE_RT_NA_LES()`, `calNewE_RTP_NA_LES()`, `calNewU_R()`, `calNewU_RT_LES()`, `calNewU_RTP_LES()`, `calNewV_RT_LES()`, `calNewV_RTP_LES()`, `calNewW_RTP_LES()`, `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_LES()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, `initUpdateLocalBoundaries()`, and `setupLocalGrid()`.

## 5.5.3.22 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()`.

## 5.5.3.23 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()`.

## 5.5.3.24 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()`, `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, `Grid()`, `modelRead()`, and `setDEDMClamp()`.

## 5.5.3.25 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 `init()`, `initUpdateLocalBoundaries()`, `initWatchZones()`, `modelRead()`, `modelWrite_GL()`, `modelWrite_TEOS()`, and `setupLocalGrid()`.

## 5.5.3.26 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()`, `setDEDMClamp()`, `setInternalVarInf()`, `setMainFunctions()`, `setupLocalGrid()`, and `updateNewGridWithOld()`.

#### 5.5.3.27 `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 `calNewD_R()`, `calNewD_RT()`, `calNewD_RTP()`, `calNewE_R_NA()`, `calNewE_RT_NA_LES()`, `calNewE RTP_NA_LES()`, `calNewU0_RT()`, `calNewU0_RTP()`, `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()`, `calOldPEKappaGamma_TEOS()`, `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_LES()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction RTP_LES()`, `dImplicitEnergyFunction RTP_LES_SB()`, `initImplicitCalculation()`, `initInternalVars()`, `initUpdateLocalBoundaries()`, `initWatchZones()`, `modelRead()`, `modelWrite_GL()`, `modelWrite_TEOS()`, `setDEDMClamp()`, `setupLocalGrid()`, and `updateNewGridWithOld()`.

#### 5.5.3.28 `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()`.

#### 5.5.3.29 `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()`.

5.5.3.30 `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()`.

5.5.3.31 `int Grid::nPhi`

Index of  $\phi$  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()`.

5.5.3.32 `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()`, `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()`.

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(), setInternalVarInf(), writeWatchZones\_R\_GL(), writeWatchZones\_R\_TEOS(), writeWatchZones\_RT\_GL(), writeWatchZones\_RT\_TEOS(), writeWatchZones\_RTP\_GL(), and writeWatchZones\_RTP\_TEOS().

#### 5.5.3.33 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(), initDonorFracAndMaxConVel\_RT\_TEOS(), initDonorFracAndMaxConVel\_RTP\_GL(), initDonorFracAndMaxConVel\_RTP\_TEOS(), modelRead(), setInternalVarInf(), writeWatchZones\_RT\_GL(), writeWatchZones\_RT\_TEOS(), writeWatchZones\_RTP\_GL(), and writeWatchZones\_RTP\_TEOS().

#### 5.5.3.34 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().

\_RTP\_TEOS().

#### 5.5.3.35 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\\_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\(\)](#).

#### 5.5.3.36 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().

#### 5.5.3.37 int Grid::nSinThetalJp1halfK

Index of  $\sin \theta$  at  $\theta$  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(), 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().

#### 5.5.3.38 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().



### 5.5.3.39 `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()`.

### 5.5.3.40 `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_RT_P()`, `calNewW_RTP_LES()`, `calOldDenave_R()`, `calOldDenave_RT()`, `calOldDenave_RT_P()`, `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()`, `setDEDMClamp()`, and `updateOldGrid()`.

### 5.5.3.41 `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()`, `setDEDMClamp()`, and `updateOldGrid()`.

#### 5.5.3.42 `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()`, `setDEDMClamp()`, `writeWatchZones_R_TEOS()`, `writeWatchZones_RT_TEOS()`, and `writeWatchZones_RTP_TEOS()`.

#### 5.5.3.43 `int Grid::nTheta`

Index of  $\theta$  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()`.

#### 5.5.3.44 `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().

#### 5.5.3.45 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\_R\_SM(), 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().

#### 5.5.3.46 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(), 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(), initUpdateLocalBoundaries(), modelRead(), updateLocalBoundaryVelocitiesNewGrid\_RT(), updateLocalBoundaryVelocitiesNewGrid\_RTP(), writeWatchZones\_RT\_GL(), writeWatchZones\_RT\_TEOS(), writeWatchZones\_RTP\_GL(), and writeWatchZones\_RTP\_TEOS().

#### 5.5.3.47 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][1]` 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()`.

#### 5.5.3.48 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()`, `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_RT-`

P\_SB(), Grid(), initDonorFracAndMaxConVel\_RTP\_GL(), initDonorFracAndMaxConVel\_RTP\_TEOS(), initUpdateLocalBoundaries(), modelRead(), updateLocalBoundaryVelocitiesNewGrid\_RTP(), writeWatchZones\_RTP\_GL(), and writeWatchZones\_RTP\_TEOS().

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

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

## 5.6 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 [dCurrentRelTErr](#)
- int [nCurrentNumIterations](#)
- int [nMaxNumSolverIterations](#)
- double [dMaxErrorInRHS](#)
- double [dAverageRHS](#)

### 5.6.1 Detailed Description

This class holds data required for the implicit calculation.

## 5.6.2 Constructor & Destructor Documentation

### 5.6.2.1 Implicit::Implicit ( )

constructor the the class [Implicit](#).

References [dCurrentRelTError](#), [dDerivativeStepFraction](#), [dMaxErrorInRHS](#), [dTolerance](#), [nCurrentNumIterations](#), [nLocDer](#), [nLocFun](#), [nMaxNumIterations](#), [nMaxNumSolverIterations](#), [nNumDerPerRow](#), [nNumImplicitZones](#), [nNumRowsALocal](#), [nNumRowsALocalSB](#), and [nTypeDer](#).

## 5.6.3 Member Data Documentation

### 5.6.3.1 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\(\)](#).

### 5.6.3.2 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\(\)](#).

### 5.6.3.3 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\(\)](#).

### 5.6.3.4 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 coefferent 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\(\)](#).

### 5.6.3.5 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().

### 5.6.3.6 KSP Implicit::kspContext

PETSc solver context.

Referenced by implicitSolve\_R(), implicitSolve\_RT(), implicitSolve\_RTP(), and initImplicitCalculation().

### 5.6.3.7 Mat Implicit::matCoeff

Parallel coefficient matrix (spread across all processors)

Referenced by implicitSolve\_R(), implicitSolve\_RT(), implicitSolve\_RTP(), and initImplicitCalculation().

### 5.6.3.8 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().

### 5.6.3.9 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().

### 5.6.3.10 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().

### 5.6.3.11 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\(\)](#).

### 5.6.3.12 int Implicit::nMaxNumSolverIterations

If [TRACKMAXSOLVERERROR](#) set to 1, then this will be the current maximum number of iterations required for the linear equation 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\(\)](#).

### 5.6.3.13 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\(\)](#).

### 5.6.3.14 int Implicit::nNumImplicitZones

The number of zones in the region near the surface which should use 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\(\)](#).

### 5.6.3.15 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\(\)](#).

### 5.6.3.16 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\(\)](#).



**5.6.3.17 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()`.

**5.6.3.18 Vec Implicit::vecRHS**

RHS vector (spread across all processors)

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

**5.6.3.19 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()`.

**5.6.3.20 Vec Implicit::vecTCorrections**

Temperature corrections solution vector (spread across all processors)

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

**5.6.3.21 Vec Implicit::vecTCorrectionsLocal**

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

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

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

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

**5.7 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](#)

### 5.7.1 Detailed Description

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

### 5.7.2 Constructor & Destructor Documentation

#### 5.7.2.1 `MessPass::MessPass( )`

Constructor for class [MessPass](#).

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

### 5.7.3 Member Data Documentation

#### 5.7.3.1 `MPI::Request* MessPass::requestRecv`

Message handles.

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

#### 5.7.3.2 `MPI::Request* MessPass::requestSend`

Message handles.

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

### 5.7.3.3 MPI::Status\* MessPass::statusRecv

Message status.

Referenced by `initUpdateLocalBoundaries()`, `MessPass()`, `updateLocalBoundaries()`, and `updateLocalBoundariesNewGrid()`.

### 5.7.3.4 MPI::Status\* MessPass::statusSend

Message status.

Referenced by `initUpdateLocalBoundaries()`, `MessPass()`, and `updateLocalBoundaries()`.

### 5.7.3.5 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()`.

### 5.7.3.6 MPI::Datatype\* MessPass::typeRecvOldGrid

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

Referenced by `initUpdateLocalBoundaries()`, `MessPass()`, and `updateLocalBoundaries()`.

### 5.7.3.7 MPI::Datatype\* MessPass::typeSendNewGrid

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

Referenced by `initUpdateLocalBoundaries()`, `MessPass()`, and `updateLocalBoundaries()`.

### 5.7.3.8 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()`.

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

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

## 5.8 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](#)

### 5.8.1 Detailed Description

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

### 5.8.2 Constructor & Destructor Documentation

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

Constructor for this class.

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

### 5.8.3 Member Data Documentation

#### 5.8.3.1 [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()`.

#### 5.8.3.2 bool `Output::bPrint`

Should status updates be printed to the screen.

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

#### 5.8.3.3 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()`.

#### 5.8.3.4 double `Output::dPrintFrequencyTime`

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

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

#### 5.8.3.5 double `Output::dTimeLastDump`

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

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

#### 5.8.3.6 double `Output::dTimeLastPrint`

Simulation time when last status was printed.

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

#### 5.8.3.7 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()`.

#### 5.8.3.8 int `Output::nNumTimeStepsSinceLastPrint`

The number of time steps since the last model dump.

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

#### 5.8.3.9 `int Output::nPrintFrequencyStep`

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

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

#### 5.8.3.10 `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()`.

#### 5.8.3.11 `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()`.

#### 5.8.3.12 `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()`.

#### 5.8.3.13 `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()`.

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

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

## 5.9 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 [dDonorCellMultiplier](#)
- double [dDonorCellMin](#)
- double [dAlpha](#)
- double [dTolerance](#)
- int [nMaxIterations](#)
- double [dEddyViscosity](#)
- double [dMaxConvectiveVelocity](#)
- double [dMaxConvectiveVelocity\\_c](#)
- double [dPrt](#)
- double [dEDMClampValue](#)
- double [dEDMClampMr](#)
- double [dEDMClampTemperature](#)
- bool [bEDMClamp](#)
- std::string [sDebugProfileOutput](#)

### 5.9.1 Detailed Description

This class holds parameters and constants used for calculation.

## 5.9.2 Constructor & Destructor Documentation

### 5.9.2.1 `Parameters::Parameters( )`

Constructor for the class [Parameters](#)

References `bDEDMClamp`, `dA`, `dAlpha`, `dAVThreshold`, `dDEDMClampMr`, `dDEDMClampValue`, `dDonorCellMin`, `dEddyViscosity`, `dEDMClampTemperature`, `dG`, `dMaxConvectiveVelocity`, `dMaxConvectiveVelocity_c`, `dPi`, `dPrt`, and `dSigma`.

## 5.9.3 Member Data Documentation

### 5.9.3.1 `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()`.

### 5.9.3.2 `bool Parameters::bDEDMClamp`

Specifies if a DEDM clamp should be used. This should only be used when starting from a model with out any sizable convection. It could give undesirable results if used when starting a calculation from a model with already established convection.

Referenced by `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, `init()`, and `Parameters()`.

### 5.9.3.3 `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()`.

### 5.9.3.4 `double Parameters::dA`

Artificial viscosity parameter, reasonable values range from 0 to  $\sim 3$ .

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()`.



### 5.9.3.5 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()`.

### 5.9.3.6 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()`.

### 5.9.3.7 double Parameters::dDEDMClampMr

The mass above which the DEDM clamp is applied.

Referenced by `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, `init()`, `Parameters()`, and `setDEDMClamp()`.

### 5.9.3.8 double Parameters::dDEDMClampValue

The value to use for DEDM in energy conservation equation when [Parameters::bDEDMClamp](#) is true.

Referenced by `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RTP_LES()`, `dImplicitEnergyFunction_RTP_LES_SB()`, `init()`, `Parameters()`, and `setDEDMClamp()`.

### 5.9.3.9 double Parameters::dDonorCellMin

The minimum amount of donor cell allowed. Set in constructor, [Parameters::Parameters](#)

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()`, `init`

DonorFracAndMaxConVel\_RT\_TEOS(), initDonorFracAndMaxConVel\_RTP\_GL(), initDonorFracAndMaxConVel\_RTP\_TEOS(), Parameters(), and setDEDMClamp().

#### 5.9.3.10 double Parameters::dDonorCellMultiplier

Multiplier used to determine the fraction of the sound speed at which donor cell is full. e.g. a value of 1.0 means the donor cell will be full when the convective velocity is equal to the sound speed. A value of 0.5 will mean that it will be full donor cell when the convective velocity is twice the sound speed. A value of 2.0 will mean that it will use full donor cell when the convective velocity is half the sound speed.

Referenced by calDelt\_R\_GL(), calDelt\_R\_TEOS(), calDelt\_RT\_GL(), calDelt\_RT\_TEOS(), calDelt\_RTP\_GL(), calDelt\_RTP\_TEOS(), init(), initDonorFracAndMaxConVel\_R\_GL(), initDonorFracAndMaxConVel\_R\_TEOS(), initDonorFracAndMaxConVel\_RT\_GL(), initDonorFracAndMaxConVel\_RT\_TEOS(), initDonorFracAndMaxConVel\_RTP\_GL(), and initDonorFracAndMaxConVel\_RTP\_TEOS().

#### 5.9.3.11 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().

#### 5.9.3.12 double Parameters::dEDMClampTemperature

The temperature at which to chose [Parameters::dDEDMClampMr](#) from the stating model.

Referenced by init(), Parameters(), and setDEDMClamp().

#### 5.9.3.13 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().

#### 5.9.3.14 double Parameters::dGamma

The adiabatic  $\gamma$ , 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()`.

#### 5.9.3.15 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()`.

#### 5.9.3.16 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 `Parameters()`.

#### 5.9.3.17 double Parameters::dPi

The value of  $\pi$ .

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()`, `Parameters()`, `writeWatchZones_R_GL()`, `writeWatchZones_R_TEOS()`, `writeWatchZones_RT_GL()`, `writeWatchZones_RT_TEOS()`, `writeWatchZones_RTP_GL()`, and `writeWatchZones_RTP_TEOS()`.

**5.9.3.18 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 LUV11 Large Eddy Simulation Model" April 15, 1991 a Lawrence Livermore National Laboratory 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()`.

**5.9.3.19 double Parameters::dSigma**

The value of  $\sigma$ , 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()`.

**5.9.3.20 double Parameters::dTolerance**

Amount of error to tolerate when calculating temperature from the equation of state.

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

**5.9.3.21 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()`.

**5.9.3.22 int Parameters::nMaxIterations**

The maximum number of iterations to try to get the the relative error in the temperature below `parameters::dTolerance`.

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

### 5.9.3.23 `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()`.

### 5.9.3.24 `std::string Parameters::sDebugProfileOutput`

output file name for debugging profile, only used if `DEBUG_EQUATIONS` is set to 1

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

### 5.9.3.25 `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()`.

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

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

## 5.10 Performance Class Reference

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

### Public Member Functions

- [Performance\(\)](#)

### Public Attributes

- double [dStartTimer](#)
- double [dEndTimer](#)

### 5.10.1 Detailed Description

This class manages information pertaining to performance analysis of the code.

## 5.10.2 Constructor & Destructor Documentation

### 5.10.2.1 `Performance::Performance ( )`

Constructor for the class [Performance](#).

References `dEndTimer`, and `dStartTimer`.

## 5.10.3 Member Data Documentation

### 5.10.3.1 `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()`.

### 5.10.3.2 `double Performance::dStartTimer`

The time that the code timer was started.

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

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

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

## 5.11 ProcTop Class Reference

```
#include <procTop.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](#)

### 5.11.1 Detailed Description

This class manages information which pertains to the processor topology.

### 5.11.2 Constructor & Destructor Documentation

#### 5.11.2.1 ProcTop::ProcTop( )

Constructor for class [ProcTop](#).

References [nCoords](#), [nNeighborRanks](#), [nNumNeighbors](#), [nNumRadialNeighbors](#), [nPeriodic](#), [nProcDims](#), [nRadialNeighborNeighborIDs](#), and [nRadialNeighborRanks](#).

### 5.11.3 Member Data Documentation

#### 5.11.3.1 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\(\)](#), [setupLocalGrid\(\)](#), and [profileData::toFile\(\)](#).

#### 5.11.3.2 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\(\)](#).

#### 5.11.3.3 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\(\)](#).

#### 5.11.3.4 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()`, `setupLocalGrid()`, and `profileData::toFile()`.

#### 5.11.3.5 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()`, `ProcTop()`, and `profileData::toFile()`.

#### 5.11.3.6 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 integer 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()`.

#### 5.11.3.7 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()`.

#### 5.11.3.8 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()`.

#### 5.11.3.9 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\(\)](#), [ProcTop\(\)](#), and [profileData::ToFile\(\)](#).

### 5.11.3.10 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\\_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\(\)](#), [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\(\)](#), [setD-EDMClamp\(\)](#), [setMainFunctions\(\)](#), [setupLocalGrid\(\)](#), [profileData::ToFile\(\)](#), [updateLocalBoundaries\(\)](#), [updateLocalBoundariesNewGrid\(\)](#), and [updateNewGridWithOld\(\)](#).

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

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

## 5.12 profileData Class Reference

### Public Member Functions

- void [set](#) (std::string sName, unsigned int nZone, double dValue)
- void [set](#) (std::string sName, unsigned int nZone, int nValue)
- void [setSum](#) (std::string sName, unsigned int nZone, double dValue)
- void [setSum](#) (std::string sName, unsigned int nZone, int nValue)
- void [setMax](#) (std::string sName, unsigned int nZone, double dValue)
- void [setMax](#) (std::string sName, unsigned int nZone, int nValue)
- void [setMaxAbs](#) (std::string sName, unsigned int nZone, double dValue)
- void [setMaxAbs](#) (std::string sName, unsigned int nZone, int nValue)
- void [ToFile](#) (std::string sFileName, [Time](#) time, [ProcTop](#) procTop)
- void [clear](#) ()
- unsigned int [nMaxNumZones](#) ()
- bool [test](#) ()
- [profileData](#) ()

## 5.12.1 Constructor & Destructor Documentation

### 5.12.1.1 `profileData::profileData( )`

Constructor for class

## 5.12.2 Member Function Documentation

### 5.12.2.1 `void profileData::clear( )`

Resets values to their initial values. It doesn't free any memory.

### 5.12.2.2 `unsigned int profileData::nMaxNumZones( )`

Returns the maximum number of zones found under a key

Referenced by `toFile()`.

### 5.12.2.3 `void profileData::set( std::string sName, unsigned int nZone, double dValue )`

Sets a new bit of data to `dValue`, identified by `sName` in radial zone `nZone`.

### 5.12.2.4 `void profileData::set( std::string sName, unsigned int nZone, int nValue )`

Sets a new bit of data to `nValue`, identified by `sName` in radial zone `nZone`.

### 5.12.2.5 `void profileData::setMax( std::string sName, unsigned int nZone, double dValue )`

If the value is already set it will set it to which ever is largest, the current value or the new value I am trying to set it to

Referenced by `test()`.

### 5.12.2.6 `void profileData::setMax( std::string sName, unsigned int nZone, int nValue )`

If the value is already set it will set it to which ever is largest, the current value or the new value.

### 5.12.2.7 `void profileData::setMaxAbs( std::string sName, unsigned int nZone, double dValue )`

If the value is already set it will set it to which ever has the largest absolute value, the current value or the new value.

5.12.2.8 `void profileData::setMaxAbs( std::string sName, unsigned int nZone, int nValue )`

If the value is already set it will set it to which ever has the largest absolute value, the current value or the new value.

5.12.2.9 `void profileData::setSum( std::string sName, unsigned int nZone, double dValue )`

If the value is already set it will add to it

Referenced by test().

5.12.2.10 `void profileData::setSum( std::string sName, unsigned int nZone, int nValue )`

If the value is already set it will add to it

5.12.2.11 `bool profileData::test( )`

Runs a series of tests to insure that the functions are doing what they should be. - Returns true if all tests passed, returns false other wise.

References setMax(), and setSum().

5.12.2.12 `void profileData::toFile( std::string sFileName, Time time, ProcTop procTop )`

Prints the data to a file in the same format as the radial profiles generated by SPHERL-Sanal

References Time::dt, ProcTop::nCoords, nMaxNumZones(), ProcTop::nNumProcs, ProcTop::nNumRadialNeighbors, ProcTop::nRadialNeighborRanks, and ProcTop::nRank.

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

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

## 5.13 Time Class Reference

```
#include <time.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](#)

### 5.13.1 Detailed Description

This class manages information which pertains to time variables.

### 5.13.2 Constructor & Destructor Documentation

#### 5.13.2.1 `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](#).

### 5.13.3 Member Data Documentation

#### 5.13.3.1 `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\(\)](#).

#### 5.13.3.2 `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\(\)](#).

5.13.3.3 `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()`.

5.13.3.4 `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()`.

5.13.3.5 `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()`.

5.13.3.6 `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()`, `calNewU_R()`, `calNewU_R_LES()`, `calNewU_RT()`, `calNewU_RT_LES()`, `calNewU_RTP()`, `calNewU_RT_P_LES()`, `calNewV_RT()`, `calNewV_RT_LES()`, `calNewV_RTP()`, `calNewV_RTP_LES()`, `calNewW_RTP()`, `calNewW_RTP_LES()`, `modelRead()`, and `Time()`.

5.13.3.7 `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()`.

### 5.13.3.8 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()`, `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()`, `calNewR()`, `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()`, `fin()`, `main()`, `modelRead()`, `modelWrite_TEOS()`, and `Time()`.

### 5.13.3.9 double Time::dEndTime

The end time of the current calculation in seconds.

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

### 5.13.3.10 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()`.

### 5.13.3.11 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()`, `profileData::toFile()`, `writeWatchZones_R_GL()`, `writeWatchZones_R_TEOS()`, `writeWatchZones_RT_GL()`, `writeWatchZones_RT_TEOS()`, `writeWatchZones_RTP_GL()`, and `writeWatchZones_RTP_TEOS()`.

### 5.13.3.12 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\\_np1half](#).

Referenced by `calDelt_R_GL()`, `calDelt_R_TEOS()`, `calDelt_RT_GL()`, `calDelt_RT_TEOS()`, `calDelt_RTP_GL()`, `calDelt_RTP_TEOS()`, `init()`, and `Time()`.

#### 5.13.3.13 `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()`.

#### 5.13.3.14 `int Time::nTimeStepIndex`

An index indicating the current time step. An index of zero corresponds to a `Time::dt=0`.

**Todo** should probably make this an unsigned variable, and perhaps also use the keyword `long` to help ensure there are enough values. Often need 7 decimal places.

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()`, `setDEDMClamp()`, `Time()`, `writeWatchZones_R_GL()`, `writeWatchZones_R_TEOS()`, `writeWatchZones_RT_GL()`, `writeWatchZones_RT_TEOS()`, `writeWatchZones_RTP_GL()`, and `writeWatchZones_RTP_TEOS()`.

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

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

## 5.14 WatchZone Class Reference

```
#include <watchzone.h>
```

### 5.14.1 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 6

# File Documentation

### 6.1 /home/cgeroux/Documents/WORK/SPHERLS\_git/src/eos.cpp File Reference

```
#include <string> #include <fstream> #include <sstream>
#include <iostream> #include <cmath> #include <stdlib.-
h> #include "eos.h" #include "exception2.h"
```

#### 6.1.1 Detailed Description

Implements the eos (equation of state) class defined in [eos.h](#)

### 6.2 /home/cgeroux/Documents/WORK/SPHERLS\_git/src/eos.h File Reference

```
#include <string> #include "exception2.h"
```

#### Classes

- class [eos](#)

#### 6.2.1 Detailed Description

Header file for [eos.cpp](#)

### 6.3 dataManipulation.cpp File Reference

```
#include <cmath> #include <sstream> #include <fstream>
#include <iomanip> #include <vector> #include <fcntl.h>
#include "dataManipulation.h" #include "global.h" ×
#include "xmlFunctions.h" #include "exception2.h" #include
"dataMonitoring.h" #include "physEquations.h" #include
<string> #include "fileExists.h"
```

#### Functions

- void [init](#) ([ProcTop](#) &procTop, [Grid](#) &grid, [Output](#) &output, [Time](#) &time, [Parameters](#) &parameters, [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](#) &parameters, [Functions](#) &functions, [Performance](#) &performance, [Implicit](#) &implicit)
- void [modelWrite\\_GL](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) &parameters)
- void [modelWrite\\_TEOS](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) &parameters)
- void [modelRead](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) &parameters)
- 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[])
- void [setDEDMClamp](#) ([Parameters](#) &parameters, [Time](#) &time, [Grid](#) &grid, [ProcTop](#) &procTop)

### 6.3.1 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.

### 6.3.2 Function Documentation

#### 6.3.2.1 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

in,out	<i>grid</i>	supplies the information for calculating the averages and receives the averages.
in	<i>nVar</i>	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\(\)](#).

#### 6.3.2.2 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

in,out	<i>grid</i>	supplies the information for calculating the averages and receives 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\(\)](#).

**6.3.2.3 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

in	<i>bWrite-Current-StateToFile</i>	is a bool value which indicates wheather or not to write out current model state.
in	<i>time</i>	
in	<i>output</i>	
in	<i>procTop</i>	
in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>functions</i>	
in	<i>performance</i>	
in	<i>implicit</i>	

References Implicit::dAverageRHS, Implicit::dCurrentRelTError, Time::dDelRho\_t\_Rho\_max, Time::dDelT\_t\_T\_max, Time::dDeltat\_np1half, 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, Output::sBaseOutputFileName, and Parameters::sDebugProfileOutput.

Referenced by main().

**6.3.2.4 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](#)
- Initalizes the watchZones, i.e. figure out which processors have which watch zones, opens the files and prints headers.

## Parameters

out	<i>procTop</i>	all parts of this stucture are set, and do not change though-out the rest of the calculation.
out	<i>grid</i>	through the function <a href="#">modelRead</a> the function <a href="#">setupLocalGrid</a> is called to allocate memory for the grid, and set sizes of it.
out	<i>output</i>	
out	<i>time</i>	
out	<i>parameters</i>	
out	<i>messPass</i>	
out	<i>performance</i>	
out	<i>implicit</i>	
in	<i>argc</i>	
in	<i>argv</i>	

References `Parameters::bAdiabatic`, `Parameters::bDEDMClamp`, `Output::bDump`, `Parameters::bEOSGammaLaw`, `Output::bPrint`, `Time::bVariableTimeStep`, `Parameters::dA`, `Parameters::dAVThreshold`, `Time::dConstTimeStep`, `Parameters::dDEDMClampMr`, `Parameters::dDEDMClampValue`, `Implicit::dDerivativeStepFraction`, `Parameters::dDonorCellMultiplier`, `Output::dDumpFrequencyTime`, `Parameters::dEddyViscosity`, `Parameters::dEDMClampTemperature`, `Time::dEndTime`, `Time::dPerChange`, `Output::dPrintFrequencyTime`, `Performance::dStartTimer`, `Time::dt`, `Output::dTimeLastDump`, `Output::dTimeLastPrint`, `Time::dTimeStepFactor`, `Parameters::dTolerance`, `Implicit::dTolerance`, `Parameters::eosTable`, `initImplicitCalculation()`, `initInternalVars()`, `initUpdateLocalBoundaries()`, `initWatchZones()`, `modelRead()`, `Output::nDumpFrequencyStep`, `Time::nEndTimeStep`, `Grid::nGlobalGridDims`, `Parameters::nMaxIterations`, `Implicit::nMaxNumIterations`, `Grid::nNum1DZones`, `Implicit::nNumImplicitZones`, `ProcTop::nNumProcs`, `Output::nPrintFrequencyStep`, `Output::nPrintMode`, `ProcTop::nProcDims`, `ProcTop::nRank`, `Parameters::nTypeTurbulenceMod`, `eos::readBin()`, `Output::sBaseOutputFileName`, `Parameters::sDebugProfileOutput`, `Parameters::sEOSFileName`, and `setDEDMClamp()`.

Referenced by `main()`.

### 6.3.2.5 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

in, out	<i>implicit</i>	
in	<i>grid</i>	size information of the grid is used
in	<i>procTop</i>	
in	<i>nNumArgs</i>	number of command line arguments, PETSc wants them
in	<i>cArgs</i>	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()`.

#### 6.3.2.6 `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`
- `MessPass::statusSend`
- `MessPass::statusRecv`

#### Parameters

in, out	<i>procTop</i>	
in, out	<i>grid</i>	
in, out	<i>messPass</i>	
in, out	<i>implicit</i>	

References `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Grid::nCenIntOffset`, `ProcTop::nCoords`, `Grid::nE`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndGhostUpdateImplicit`, `Grid::nEndUpdateExplicit`, `Grid::nEndUpdateImplicit`, `Grid::nGlobalGridDims`, `Grid::nGlobalGridPositionLocalGrid`, `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::nV, Grid::nVariables, Grid::nW, MessPass::requestRecv, MessPass::requestSend, MessPass::statusRecv, MessPass::statusSend, MessPass::typeRecvNewVar, MessPass::typeRecvOldGrid, MessPass::typeSendNewGrid, and MessPass::typeSendNewVar.

Referenced by init().

#### 6.3.2.7 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

in	<i>sFileName</i>	name of the file containing the model to be read in
out	<i>procTop</i>	
out	<i>grid</i>	
out	<i>time</i>	
out	<i>parameters</i>	

**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::nDonorCellFrac, 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().

#### 6.3.2.8 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

in	<i>sFileName</i>	base name of the output files
in	<i>procTop</i>	
in	<i>grid</i>	
in	<i>time</i>	
in	<i>parameters</i>	

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::nNum1D-Zones, Grid::nNumGhostCells, Grid::nNumVars, ProcTop::nPeriodic, ProcTop::nProcDims, ProcTop::nRank, Time::nTimeStepIndex, and Grid::nVariables.

Referenced by setMainFunctions().

#### 6.3.2.9 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

in	<i>sFileName</i>	base name of the output files
in	<i>procTop</i>	
in	<i>grid</i>	
in	<i>time</i>	
in	<i>parameters</i>	

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::nNum1D-Zones, Grid::nNumGhostCells, Grid::nNumVars, ProcTop::nPeriodic, ProcTop::nProcDims, ProcTop::nRank, Time::nTimeStepIndex, Grid::nVariables, and Parameters::sEOSFileName.

Referenced by setMainFunctions().

#### 6.3.2.10 void setDEDMClamp ( Parameters & *parameters*, Time & *time*, Grid & *grid*, ProcTop & *procTop* )

This function sets the DEDM clamp if starting from an initial model, otherwise it throws an exception.

References Parameters::dDEDMClampMr, Parameters::dDEDMClampValue, Parameters::dDonorCellMin, Parameters::dEDMClampTemperature, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nDM, Grid::nE, Grid::nEndUpdateExplicit, Grid::nEndUpdateImplicit, Grid::nM, Grid::nNumDims, Grid::nNumGhostCells, ProcTop::nRank,



Grid::nStartUpdateExplicit, Grid::nStartUpdateImplicit, Grid::nT, and Time::nTimeStep-Index.

Referenced by init().

#### 6.3.2.11 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

in, out	<i>procTop</i>	contains information about the processor topology
in, out	<i>grid</i>	contains information about gird

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().

#### 6.3.2.12 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

in	<i>procTop</i>	
in	<i>messPass</i>	
in, out	<i>grid</i>	

**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().

#### 6.3.2.13 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

in	<i>procTop</i>	
in	<i>messPass</i>	
in, out	<i>grid</i>	

**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\(\)](#).

#### 6.3.2.14 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\(\)](#).

**6.3.2.15** 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()`.

**6.3.2.16** 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()`.

**6.3.2.17** void `updateNewGridWithOld`( `Grid & grid`, `ProcTop & procTop` )

Copies the contents of the old grid to the new grid including ghost cells.

#### Parameters

<code>in, out</code>	<code>grid</code>	
<code>in</code>	<code>procTop</code>	

References `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Grid::nLocalGridDims`, `Grid::nNumDims`, `Grid::nNumGhostCells`, `Grid::nNumIntVars`, `Grid::nNumVars`, `ProcTop::nRank`, and `Grid::nVariables`.

Referenced by `main()`.

**6.3.2.18** void `updateOldGrid`( `ProcTop & procTop`, `Grid & grid` )

Updates the old grid with the new grid, not including boundaries.

#### Parameters

<code>in</code>	<code>procTop</code>	
<code>in, out</code>	<code>grid</code>	

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()`.

## 6.4 dataManipulation.h File Reference

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

### Functions

- void [init](#) ([ProcTop](#) &procTop, [Grid](#) &grid, [Output](#) &output, [Time](#) &time, [Parameters](#) &parameters, [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](#) &parameters, [Functions](#) &functions, [Performance](#) &performance, [Implicit](#) &implicit)
- void [modelWrite\\_GL](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) &parameters)
- void [modelWrite\\_TEOS](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) &parameters)
- void [modelRead](#) (std::string sFileName, [ProcTop](#) &procTop, [Grid](#) &grid, [Time](#) &time, [Parameters](#) &parameters)
- 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[])
- void [setDEDMClamp](#) ([Parameters](#) &parameters, [Time](#) &time, [Grid](#) &grid, [ProcTop](#) &procTop)

### 6.4.1 Detailed Description

Header file for [dataManipulation.cpp](#)

## 6.4.2 Function Documentation

### 6.4.2.1 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

<i>in, out</i>	<i>grid</i>	supplies the information for calculating the averages and receives the averages.
<i>in</i>	<i>nVar</i>	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\(\)](#).

### 6.4.2.2 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

<i>in, out</i>	<i>grid</i>	supplies the information for calculating the averages and receives 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\(\)](#).

### 6.4.2.3 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 writing out run time.

## Parameters

in	<i>bWrite-Current-StateToFile</i>	is a bool value which indicates wheather or not to write out current model state.
in	<i>time</i>	
in	<i>output</i>	
in	<i>procTop</i>	
in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>functions</i>	
in	<i>performance</i>	
in	<i>implicit</i>	

References Implicit::dAverageRHS, Implicit::dCurrentRelTErrors, Time::dDelRho\_t\_Rho\_max, Time::dDelT\_t\_T\_max, Time::dDeltat\_np1half, 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, Output::sBaseOutputFileName, and Parameters::sDebugProfileOutput.

Referenced by main().

**6.4.2.4 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](#)
- Initalizes the watchZones, i.e. figure out which processors have which watch zones, opens the files and prints headers.

## Parameters

out	<i>procTop</i>	all parts of this stucture are set, and do not change thoughout the rest of the calculation.
out	<i>grid</i>	through the function <a href="#">modelRead</a> the function <a href="#">setupLocalGrid</a> is called to allocate memory for the grid, and set sizes of it.

out	<i>output</i>	
out	<i>time</i>	
out	<i>parameters</i>	
out	<i>messPass</i>	
out	<i>performance</i>	
out	<i>implicit</i>	
in	<i>argc</i>	
in	<i>argv</i>	

References Parameters::bAdiabatic, Parameters::bDEDMClamp, Output::bDump, - Parameters::bEOSGammaLaw, Output::bPrint, Time::bVariableTimeStep, Parameters::dA, Parameters::dAVThreshold, Time::dConstTimeStep, Parameters::dDEDMClampMr, Parameters::dDEDMClampValue, Implicit::dDerivativeStepFraction, Parameters::dDonorCellMultiplier, Output::dDumpFrequencyTime, Parameters::dEddyViscosity, Parameters::dEDMClampTemperature, Time::dEndTime, Time::dPerChange, Output::dPrintFrequencyTime, Performance::dStartTimer, Time::dt, Output::dTimeLastDump, Output::dTimeLastPrint, Time::dTimeStepFactor, Parameters::dTolerance, Implicit::dTolerance, Parameters::eosTable, initImplicitCalculation(), initInternalVars(), initUpdateLocalBoundaries(), initWatchZones(), modelRead(), Output::nDumpFrequencyStep, Time::nEndTimeStep, Grid::nGlobalGridDims, Parameters::nMaxIterations, Implicit::nMaxNumIterations, Grid::nNum1DZones, Implicit::nNumImplicitZones, ProcTop::nNumProcs, Output::nPrintFrequencyStep, Output::nPrintMode, ProcTop::nProcDims, ProcTop::nRank, Parameters::nTypeTurbulenceMod, eos::readBin(), Output::sBaseOutputFileName, Parameters::sDebugProfileOutput, Parameters::sEOSFileName, and setDEDMClamp().

Referenced by main().

#### 6.4.2.5 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

in,out	<i>implicit</i>	
in	<i>grid</i>	size information of the grid is used
in	<i>procTop</i>	
in	<i>nNumArgs</i>	number of command line arguments, PETSc wants them
in	<i>cArgs</i>	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()`.

#### 6.4.2.6 `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 memory for:

- `MessPass::requestSend`
- `MessPass::requestRecv`
- `MessPass::statusSend`
- `MessPass::statusRecv`

#### Parameters

in, out	<i>procTop</i>	
in, out	<i>grid</i>	
in, out	<i>messPass</i>	
in, out	<i>implicit</i>	

References `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Grid::nCenIntOffset`, `ProcTop::nCoords`, `Grid::nE`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndGhostUpdateImplicit`, `Grid::nEndUpdateExplicit`, `Grid::nEndUpdateImplicit`, `Grid::nGlobalGridDims`, `Grid::nGlobalGridPositionLocalGrid`, `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::nV`, `Grid::nVariables`, `Grid::nW`, `MessPass::requestRecv`, `MessPass::requestSend`, `MessPass::statusRecv`, `MessPass::statusSend`, `MessPass::typeRecvNewVar`, `MessPass::typeRecvOldGrid`, `MessPass::typeSendNewGrid`, and `MessPass::typeSendNewVar`.



Referenced by `init()`.

**6.4.2.7** `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

in	<i>sFileName</i>	name of the file containing the model to be read in
out	<i>procTop</i>	
out	<i>grid</i>	
out	<i>time</i>	
out	<i>parameters</i>	

**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::nCotThetaJp1halfK`, `Grid::nD`, `Grid::nDCosThetaIJK`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `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::nSinThetaJp1halfK`, `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()`.

**6.4.2.8** `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 it's 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

in	<i>sFileName</i>	base name of the output files
in	<i>procTop</i>	
in	<i>grid</i>	
in	<i>time</i>	
in	<i>parameters</i>	

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::nNum1D-Zones, Grid::nNumGhostCells, Grid::nNumVars, ProcTop::nPeriodic, ProcTop::nProcDims, ProcTop::nRank, Time::nTimeStepIndex, and Grid::nVariables.

Referenced by setMainFunctions().

#### 6.4.2.9 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

in	<i>sFileName</i>	base name of the output files
in	<i>procTop</i>	
in	<i>grid</i>	
in	<i>time</i>	
in	<i>parameters</i>	

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::nNum1D-Zones, Grid::nNumGhostCells, Grid::nNumVars, ProcTop::nPeriodic, ProcTop::nProcDims, ProcTop::nRank, Time::nTimeStepIndex, Grid::nVariables, and Parameters::sEOSFileName.

Referenced by setMainFunctions().

#### 6.4.2.10 void setDEDMClamp ( Parameters & *parameters*, Time & *time*, Grid & *grid*, ProcTop & *procTop* )

This function sets the DEDM clamp if starting from an initial model, otherwise it throws an exception.

References Parameters::dDEDMClampMr, Parameters::dDEDMClampValue, Parameters::dDonorCellMin, Parameters::dEDMClampTemperature, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nDM, Grid::nE, Grid::nEndUpdateExplicit, Grid::nEndUpdateImplicit, Grid::nM, Grid::nNumDims, Grid::nNumGhostCells, ProcTop::nRank, Grid::nStartUpdateExplicit, Grid::nStartUpdateImplicit, Grid::nT, and Time::nTimeStepIndex.

Referenced by init().

#### 6.4.2.11 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

in, out	<i>procTop</i>	contains information about the processor topology
in, out	<i>grid</i>	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\(\)](#).

#### 6.4.2.12 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

in	<i>procTop</i>	
in	<i>messPass</i>	
in, out	<i>grid</i>	

**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\(\)](#).

#### 6.4.2.13 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

in	<i>procTop</i>	
in	<i>messPass</i>	
in, out	<i>grid</i>	

**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()`.

#### 6.4.2.14 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()`.

#### 6.4.2.15 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()`.

**6.4.2.16** 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()`.

**6.4.2.17** void `updateNewGridWithOld( Grid & grid, ProcTop & procTop )`

Copies the contents of the old grid to the new grid including ghost cells.

Parameters

in, out	<i>grid</i>	
in	<i>procTop</i>	

References Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nLocalGridDims, Grid::nNumDims, Grid::nNumGhostCells, Grid::nNumIntVars, Grid::nNumVars, ProcTop::nRank, and Grid::nVariables.

Referenced by `main()`.

**6.4.2.18** void `updateOldGrid( ProcTop & procTop, Grid & grid )`

Updates the old grid with the new grid, not including boundaries.

Parameters

in	<i>procTop</i>	
in, out	<i>grid</i>	

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()`.

## 6.5 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" #include "fileExists.h"
```

## Functions

- void [initWatchZones](#) (XMLNode xParent, [ProcTop](#) &procTop, [Grid](#) &grid, [Output](#) &output, [Parameters](#) &parameters, [Time](#) &time)
- void [writeWatchZones\\_R\\_GL](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones\\_R\\_TEOS](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones\\_RT\\_GL](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones\\_RT\\_TEOS](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones\\_RTP\\_GL](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones\\_RTP\\_TEOS](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [finWatchZones](#) ([Output](#) &output)

### 6.5.1 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.

### 6.5.2 Function Documentation

#### 6.5.2.1 void [finWatchZones](#) ( [Output](#) & *output* )

Closes the files opened for writing out the watchzones

##### Parameters

<i>in</i>	<i>output</i>	
-----------	---------------	--

References [Output::ofWatchZoneFiles](#), and [Output::watchzoneList](#).

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

#### 6.5.2.2 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

in	<i>xParent</i>	
in	<i>procTop</i>	
in	<i>grid</i>	
in, out	<i>output</i>	
in	<i>parameters</i>	
in	<i>time</i>	

References Parameters::bEOSGammaLaw, 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().

#### 6.5.2.3 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

in, out	<i>output</i>	
in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>procTop</i>	

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().

#### 6.5.2.4 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

in, out	<i>output</i>	
in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>procTop</i>	

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().

#### 6.5.2.5 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

in, out	output	
in	grid	
in	parameters	
in	time	
in	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().

#### 6.5.2.6 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

in, out	output	
in	grid	
in	parameters	
in	time	
in	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().



### 6.5.2.7 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

in, out	<i>output</i>	
in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>procTop</i>	

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().

### 6.5.2.8 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

in, out	<i>output</i>	
in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>procTop</i>	

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().

## 6.6 dataMonitoring.h File Reference

```
#include <string> #include "xmlParser.h" #include "global.-h"
```

## Functions

- void [initWatchZones](#) (XMLNode xParent, [ProcTop](#) &procTop, [Grid](#) &grid, [Output](#) &output, [Parameters](#) &parameters, [Time](#) &time)
- void [writeWatchZones\\_R\\_GL](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones\\_R\\_TEOS](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones\\_RT\\_GL](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones\\_RT\\_TEOS](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones\\_RTP\\_GL](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [writeWatchZones\\_RTP\\_TEOS](#) ([Output](#) &output, [Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [finWatchZones](#) ([Output](#) &output)

### 6.6.1 Detailed Description

Header file for [dataMonitoring.cpp](#)

### 6.6.2 Function Documentation

#### 6.6.2.1 void [finWatchZones](#) ( [Output](#) & *output* )

Closes the files opened for writing out the watchzones

##### Parameters

<i>in</i>	<i>output</i>	
-----------	---------------	--

References [Output::ofWatchZoneFiles](#), and [Output::watchzoneList](#).

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

#### 6.6.2.2 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

<i>in</i>	<i>xParent</i>	
<i>in</i>	<i>procTop</i>	
<i>in</i>	<i>grid</i>	

<i>in, out</i>	<i>output</i>	
<i>in</i>	<i>parameters</i>	
<i>in</i>	<i>time</i>	

References Parameters::bEOSGammaLaw, 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().

#### 6.6.2.3 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

<i>in, out</i>	<i>output</i>	
<i>in</i>	<i>grid</i>	
<i>in</i>	<i>parameters</i>	
<i>in</i>	<i>time</i>	
<i>in</i>	<i>procTop</i>	

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().

#### 6.6.2.4 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

<i>in, out</i>	<i>output</i>	
<i>in</i>	<i>grid</i>	
<i>in</i>	<i>parameters</i>	
<i>in</i>	<i>time</i>	
<i>in</i>	<i>procTop</i>	

References Grid::dLocalGridOld, Parameters::dPi, Time::dt, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nP, Grid::nQ0, Grid::nR, Grid::nT, Time::nTimeStep-

Index, Grid::nU, Grid::nU0, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

#### 6.6.2.5 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

in, out	output	
in	grid	
in	parameters	
in	time	
in	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().

#### 6.6.2.6 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

in, out	output	
in	grid	
in	parameters	
in	time	
in	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().

### 6.6.2.7 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

in, out	<i>output</i>	
in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>procTop</i>	

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().

### 6.6.2.8 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

in, out	<i>output</i>	
in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>procTop</i>	

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().

## 6.7 global.cpp File Reference

```
#include "global.h"
```

### 6.7.1 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 initialize the default values of various parameters.

## 6.8 global.h File Reference

```
#include <vector> #include <mpi.h> #include "watchzone.-  
h" #include "eos.h" #include "petscksp.h" #include <csignal> ×  
#include <limits> #include "profileData.h" #include "proc-  
Top.h" #include "time.h"
```

### Classes

- class [MessPass](#)
- class [Grid](#)
- class [Parameters](#)
- class [Output](#)
- class [Performance](#)
- class [Implicit](#)
- class [Functions](#)
- class [Global](#)

### Defines

- #define [SIGNEDEN](#) 0
- #define [SIGNEDENG](#) 0
- #define [SIGNEDTEMP](#) 0
- #define [TRACKMAXSOLVERERROR](#) 0
- #define [SEDOV](#) 0
- #define [VISCOUS\\_ENERGY\\_EQ](#) 1
- #define [DUMP\\_VERSION](#) 1
- #define [DEBUG\\_EQUATIONS](#) 0
- #define [DEDEM\\_CLAMP](#) 1

### 6.8.1 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.

## 6.8.2 Define Documentation

### 6.8.2.1 #define DEBUG\_EQUATIONS 0

If 1 will write out in the form of a profile file, all the horizontal maximum values of all terms in all equations.

Referenced by `dImplicitEnergyFunction_R()`, `dImplicitEnergyFunction_R_SB()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RTP_LES()`, and `dImplicitEnergyFunction_RTP_LES_SB()`.

### 6.8.2.2 #define DEDEM\_CLAMP 1

If 1 a clamp on the DEDM gradient will be used to limit how large DE/DM becomes in the advection term in the energy equation.

### 6.8.2.3 #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()`.

### 6.8.2.4 #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.

### 6.8.2.5 #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.

### 6.8.2.6 #define SIGNEGENG 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.

### 6.8.2.7 #define **SIGNEGTEMP** 0

Raise signal on calculation of negative temperature if set to 1, else don't raise 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 useful output upon negative energies, and will simply abort the program.

### 6.8.2.8 #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 whether the solver is working accurately this is very handy to turn on.

### 6.8.2.9 #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

## 6.9 main.cpp File Reference

```
#include <mpi.h> #include <sstream> #include <string> ×
#include <fstream> #include <cmath> #include <vector>
#include <algorithm> #include <iomanip> #include <csignal> ×
#include <fcntl.h> #include "main.h" #include "global.h" ×
#include "watchzone.h" #include "exception2.h" #include
"xmlParser.h" #include "xmlFunctions.h" #include "data-
Manipulation.h" #include "dataMonitoring.h" #include "phys-
Equations.h"
```

### Functions

- int [main](#) (int argc, char \*argv[])
- void [signalHandler](#) (int nSig)

### 6.9.1 Detailed Description

This file contains the main function which is the driver for SPHERLS.

### 6.9.2 Function Documentation



6.9.2.1 `int main ( int argc, char * argv[] )`

Main driving function of SPHERLS.

## Parameters

<code>in</code>	<code>argc</code>	number of arguments passed from the command line
<code>in</code>	<code>argv</code>	array of character strings of size <code>argc</code> 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),  $\theta$ -velocity (V) and the  $\phi$ -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`, `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`, `Grid::nDenAve`, `Output::nDumpFrequencyStep`, `Grid::nEddyVisc`, `Time::nEndTimeStep`, `Grid::nGamma`, `Implicit::nMaxNumSolverIterations`, `Grid::nNumDims`, `Implicit::nNumImplicitZones`, `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`, `Parameters::sDebugProfileOutput`, `setMainFunctions()`, `signalHandler()`, `Global::time`, `updateLocalBoundaries()`, `updateLocalBoundariesNewGrid()`, and `updateNewGridWithOld()`.

#### 6.9.2.2 void signalHandler ( int nSig )

Used for catching signals.

Referenced by `main()`.

## 6.10 main.h File Reference

### Functions

- void [signalHandler](#) (int nSig)
- int [main](#) (int argc, char \*argv[])

#### 6.10.1 Detailed Description

Header file for [main.cpp](#)

#### 6.10.2 Function Documentation

##### 6.10.2.1 int main ( int argc, char \* argv[] )

Main driving function of SPHERLS.

## Parameters

in	<i>argc</i>	number of arguments passed from the command line
in	<i>argv</i>	array of character strings of size <i>argc</i> 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, 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, Grid::nDenAve, Output::nDumpFrequencyStep, Grid::nEddyVisc, Time::nEndTimeStep, Grid::nGamma, Implicit::nMaxNumSolverIterations, Grid::nNumDims, Implicit::nNumImplicitZones, 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, Parameters::sDebugProfileOutput, setMainFunctions(), signalHandler(), Global::time, updateLocalBoundaries(), updateLocalBoundariesNewGrid(), and updateNewGridWithOld().

#### 6.10.2.2 void signalHandler ( int nSig )

Used for catching signals.

Referenced by main().

## 6.11 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> #include "profile-
Data.h"
```

### Functions

- void [setMainFunctions](#) (Functions &functions, ProcTop &procTop, Parameters &parameters, Grid &grid, Time &time, Implicit &implicit)
- void [setInternalVarInf](#) (Grid &grid, Parameters &parameters)
- void [initInternalVars](#) (Grid &grid, ProcTop &procTop, Parameters &parameters)
- void [calNewVelocities\\_R](#) (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)
- void [calNewVelocities\\_R\\_LES](#) (Grid &grid, Parameters &parameters, Time &time, ProcTop &procTop)

- void `calNewVelocities_RT` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewVelocities_RT_LES` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewVelocities_RTP` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewVelocities_RTP_LES` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewU_R` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewU_R_LES` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewU_RT` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewU_RT_LES` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewU_RTP` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewU_RTP_LES` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewV_RT` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewV_RT_LES` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewV_RTP` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewV_RTP_LES` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewW_RTP` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewW_RTP_LES` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewU0_R` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewU0_RT` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewU0_RTP` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewR` (`Grid` &grid, `Time` &time)
- void `calNewD_R` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewD_RT` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewD_RTP` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_R_AD` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)

- void [calNewE\\_R\\_NA](#) (Grid &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewE\\_R\\_NA\\_LES](#) (Grid &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewE\\_RT\\_AD](#) (Grid &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewE\\_RT\\_NA](#) (Grid &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewE\\_RT\\_NA\\_LES](#) (Grid &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewE\\_RTP\\_AD](#) (Grid &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewE\\_RTP\\_NA](#) (Grid &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewE\\_RTP\\_NA\\_LES](#) (Grid &grid, [Parameters](#) &parameters, [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](#) &parameters)
- void [calNewTPKappaGamma\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewPEKappaGamma\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewQ0\\_R\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewQ0\\_R\\_GL](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewQ0Q1\\_RT\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewQ0Q1\\_RT\\_GL](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewQ0Q1Q2\\_RTP\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewQ0Q1Q2\\_RTP\\_GL](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewEddyVisc\\_None](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewEddyVisc\\_R\\_CN](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewEddyVisc\\_RT\\_CN](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewEddyVisc\\_RTP\\_CN](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewEddyVisc\\_R\\_SM](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewEddyVisc\\_RT\\_SM](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewEddyVisc\\_RTP\\_SM](#) (Grid &grid, [Parameters](#) &parameters)
- 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](#) &parameters)
- void [calOldPEKappaGamma\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldQ0\\_R\\_GL](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldQ0\\_R\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldQ0Q1\\_RT\\_GL](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldQ0Q1\\_RT\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldQ0Q1Q2\\_RTP\\_GL](#) (Grid &grid, [Parameters](#) &parameters)

- void `calOldQ0Q1Q2_RTP_TEOS` (`Grid` &grid, `Parameters` &parameters)
- void `calOldEddyVisc_R_CN` (`Grid` &grid, `Parameters` &parameters)
- void `calOldEddyVisc_RT_CN` (`Grid` &grid, `Parameters` &parameters)
- void `calOldEddyVisc_RTP_CN` (`Grid` &grid, `Parameters` &parameters)
- void `calOldEddyVisc_R_SM` (`Grid` &grid, `Parameters` &parameters)
- void `calOldEddyVisc_RT_SM` (`Grid` &grid, `Parameters` &parameters)
- void `calOldEddyVisc_RTP_SM` (`Grid` &grid, `Parameters` &parameters)
- void `calDelt_R_GL` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calDelt_R_TEOS` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calDelt_RT_GL` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calDelt_RT_TEOS` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calDelt_RTP_GL` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calDelt_RTP_TEOS` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calDelt_CONST` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `implicitSolve_None` (`Grid` &grid, `Implicit` &implicit, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass, `Functions` &functions)
- void `implicitSolve_R` (`Grid` &grid, `Implicit` &implicit, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass, `Functions` &functions)
- void `implicitSolve_RT` (`Grid` &grid, `Implicit` &implicit, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass, `Functions` &functions)
- void `implicitSolve_RTP` (`Grid` &grid, `Implicit` &implicit, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass, `Functions` &functions)
- double `dImplicitEnergyFunction_None` (`Grid` &grid, `Parameters` &parameters, `Time` &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R` (`Grid` &grid, `Parameters` &parameters, `Time` &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R_SB` (`Grid` &grid, `Parameters` &parameters, `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`)
- double `dET4` (`Parameters` &`parameters`, double `dEddyVisc_ijk_np1half`, double `dRho_ijk_np1half`, double `dLengthScale4_ijk_np1half`)

### 6.11.1 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.

### 6.11.2 Function Documentation

**6.11.2.1** `void calDelt_CONST ( Grid & grid, Parameters & parameters, Time & time, ProcTop & procTop )`

This function is used when a constant tie step is desired.

References `Time::dConstTimeStep`, `Time::dDeltat_n`, `Time::dDeltat_nm1half`, `Time::dDeltat_np1half`, `Time::dt`, `ProcTop::nRank`, and `Time::nTimeStepIndex`.

Referenced by `setMainFunctions()`.

**6.11.2.2** `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

in	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in, out	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology. This function uses <a href="#">ProcTop::nRank</a> to pass messages.

References `Time::dDelE_t_E_max`, `Time::dDelRho_t_Rho_max`, `Time::dDeltat_n`, `Time::dDeltat_nm1half`, `Time::dDeltat_np1half`, `Parameters::dDonorCellMin`, `Parameters::dDonorCellMultiplier`, `Parameters::dGamma`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dMaxConvectiveVelocity`, `Time::dPerChange`, `Time::dt`, `Time::dTimeStepFactor`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDonorCellFrac`, `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()`.

### 6.11.2.3 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

in	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in, out	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology. This function uses <a href="#">ProcTop::nRank</a> to pass messages.

References `Time::dDelRho_t_Rho_max`, `Time::dDelT_t_T_max`, `Time::dDeltat_n`, `Time::dDeltat_nm1half`, `Time::dDeltat_np1half`, `Parameters::dDonorCellMin`, `Parameters::dDonorCellMultiplier`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dMaxConvectiveVelocity`, `Time::dPerChange`, `Time::dt`, `Time::dTimeStepFactor`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDonorCellFrac`, `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()`.

#### 6.11.2.4 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

in	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in, out	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology. This function uses <a href="#">ProcTop::nRank</a> to pass messages.

References Time::dDelE\_t\_E\_max, Time::dDelRho\_t\_Rho\_max, Time::dDeltat\_n, Time::dDeltat\_nm1half, Time::dDeltat\_np1half, Parameters::dDonorCellMin, Parameters::dDonorCellMultiplier, Parameters::dGamma, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, 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().

#### 6.11.2.5 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

in	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in, out	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology. This function uses <a href="#">ProcTop::nRank</a> to pass messages.

References Time::dDelRho\_t\_Rho\_max, Time::dDelT\_t\_T\_max, Time::dDeltat\_n, Time::dDeltat\_nm1half, Time::dDeltat\_np1half, Parameters::dDonorCellMin, Parameters::dDonorCellMultiplier, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, 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().

**6.11.2.6** 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

in	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in,out	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology. This function uses <a href="#">ProcTop::nRank</a> to pass messages.

References Time::dDelE\_t\_E\_max, Time::dDelRho\_t\_Rho\_max, Time::dDeltat\_n, Time::dDeltat\_nm1half, Time::dDeltat\_np1half, Parameters::dDonorCellMin, - Parameters::dDonorCellMultiplier, Parameters::dGamma, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinThetaJK, Grid::nStartUpdateExplicit, Time::nTimeStepIndex, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

**6.11.2.7** 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

in	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in,out	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology. This function uses <a href="#">ProcTop::nRank</a> to pass messages.

References Time::dDelRho\_t\_Rho\_max, Time::dDelT\_t\_T\_max, Time::dDeltat\_n, Time::dDeltat\_nm1half, Time::dDeltat\_np1half, Parameters::dDonorCellMin, - Parameters::dDonorCellMultiplier, Grid::dLocalGridNew, Grid::dLocalGridOld, - Parameters::dMaxConvectiveVelocity, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, Grid::nDPhi, Grid::

::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinTheta-IJK, Grid::nStartUpdateExplicit, Grid::nT, Time::nTimeStepIndex, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

**6.11.2.8** void calNewD\_R ( Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop* )

This function calculates new densities using terms in the radial direction only

#### Parameters

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology, uses <a href="#">ProcTop::nRank</a> 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::nDonorCellFrac, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGlobalGridPositionLocalGrid, Grid::nNumGhostCells, Grid::nR, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, - Grid::nU, and Grid::nU0.

Referenced by setMainFunctions().

**6.11.2.9** 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology, uses <a href="#">ProcTop::nRank</a> 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::nDonorCellFrac, Grid::nEnd-

GhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGlobalGridPositionLocalGrid, Grid::nNumGhostCells, Grid::nR, ProcTop::nRank, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().

**6.11.2.10** 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

in, out	grid	contains the local grid, and will hold the newly updated densities
in	parameters	various parameters needed for the calculation
in	time	contains time information, e.g. time step, current time etc.
in	procTop	contains information about the processor topology, uses <a href="#">ProcTop::nRank</a> 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::nDCosThetaJK, Grid::nDonorCellFrac, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGlobalGridPositionLocalGrid, Grid::nNumGhostCells, Grid::nR, ProcTop::nRank, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

**6.11.2.11** 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

in, out	grid	
---------	------	--

Referenced by setMainFunctions().

### 6.11.2.12 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

in,out	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().

### 6.11.2.13 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

in,out	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().

### 6.11.2.14 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

in,out	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().

**6.11.2.15** 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	

References Time::dDeltat\_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, - Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nDonorCellFrac, 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().

**6.11.2.16** 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	

**Boundary Conditions** Setting energy at surface equal to energy in last zone.

**Boundary Conditions** Upwind gradient in dA1 term should be zero as there is no flow into the star.

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

References Time::dDeltat\_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, - Parameters::dPi, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve,

Grid::nDM, Grid::nDonorCellFrac, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGlobalGridPositionLocalGrid, Grid::nKappa, Grid::nNumGhostCells, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, and Grid::nU0.

Referenced by setMainFunctions().

#### 6.11.2.17 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</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 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]`

**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.

References `Time::dDeltat_np1half`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `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`.

#### 6.11.2.18 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	

**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_np1half`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nDTheta`, `Grid::nE`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `ProcTop::nRank`, `Grid::nSinThetaJK`, `Grid::nSinThetaJp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `setMainFunctions()`.

**6.11.2.19** `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</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 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_np1half`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`,

Grid::nDM, Grid::nDonorCellFrac, Grid::nDTheta, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nKappa, Grid::nP, Grid::nQ0, Grid::nQ1, - Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().

#### 6.11.2.20 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	

**Boundary Conditions** Missing  $\Delta M_r$  outside model using Parameters::dAlpha times  $\Delta M_r$  in the last zone instead.

**Boundary Conditions** Setting energy at surface equal to energy in last zone.

**Boundary Conditions** missing density outside model, setting it to zero

**Boundary Conditions** missing eddy viscosity outside the model setting it to zero

**Boundary Conditions** Upwind gradient in dA1 term should be zero as there is no flow into the star.

**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.

References Parameters::dAlpha, Time::dDeltat\_np1half, dET4(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dPrt, Parameters::dSigma, Grid::nCentIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDonorCellFrac, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGlobalGridPositionLocalGrid, Grid::nKappa, Grid::nNumGhostCells, - Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nR, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().

6.11.2.21 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</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 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_np1half`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `-Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nE`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `ProcTop::nRank`, `Grid::nSinThetaJK`, `Grid::nSinThetaJp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `setMainFunctions()`.

6.11.2.22 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</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_np1half`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `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::nSinThetaJK`, `Grid::nSinThetaJp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `setMainFunctions()`.

**6.11.2.23** `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	

**Boundary Conditions** Missing  $\Delta M_r$  outside model using `Parameters::dAlpha` times  $\Delta M_r$  in the last zone instead.

**Boundary Conditions** Missing  $W$  at  $i+1$ , assuming the same as at  $i$

**Boundary Conditions** Setting energy at surface equal to energy in last zone.

**Boundary Conditions** missing density outside model, setting it to zero

**Boundary Conditions** missing eddy viscosity outside the model setting it to zero

**Boundary Conditions** Upwind gradient in dA1 term should be zero as there is no flow into the star.

**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.

References Parameters::dAlpha, Time::dDeltat\_np1half, dET4(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dPrt, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDonorCellFrac, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGlobalGridPositionLocalGrid, Grid::nKappa, Grid::nNumGhostCells, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinThetaJK, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

#### 6.11.2.24 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

in, out	<i>grid</i>	
in	<i>parameters</i>	

Referenced by setMainFunctions().

#### 6.11.2.25 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

in, out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	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.

#### 6.11.2.26 void calNewEddyVisc\_R\_SM( Grid & grid, Parameters & parameters )

This function calculates the eddy viscosity with only the radial terms.

##### Parameters

in, out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::nCenIntOffset, - Grid::nD, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and - Grid::nU0.

#### 6.11.2.27 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

in, out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	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().

#### 6.11.2.28 void calNewEddyVisc\_RT\_SM( Grid & *grid*, Parameters & *parameters* )

This function calculates the eddy viscosity with only the radial and theta terms.

##### Parameters

in, out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	contains parameters used in calculating the eddy viscosity.

**Boundary Conditions** assuming that theta velocity is constant across surface

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().

#### 6.11.2.29 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

in, out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	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::nSinThetaJK, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

Referenced by setMainFunctions().

#### 6.11.2.30 void calNewEddyVisc\_RTP\_SM( Grid & grid, Parameters & parameters )

This function calculates the eddy viscosity with only the radial, theta, and phi terms.

##### Parameters

in, out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	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::nCotThetaJK, Grid::nD, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nSinThetaJK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

#### 6.11.2.31 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

in, out	<i>grid</i>	supplies the input for calculating the pressure and also accepts the result of the pressure calculations.
in	<i>parameters</i>	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().

### 6.11.2.32 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

in,out	<i>grid</i>	supplies the input for calculating the pressure and also accepts the result of the pressure calculation
in	<i>parameters</i>	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().

### 6.11.2.33 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

in,out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation.
in	<i>parameters</i>	contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma, Grid::dLocalGridNew, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU.

Referenced by setMainFunctions().

### 6.11.2.34 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 equaiton of state for the calculation.



## Parameters

in,out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
in	<i>parameters</i>	contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridNew, 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().

6.11.2.35 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

in,out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculations.
in	<i>parameters</i>	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::nSinThetaJK, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nV.

Referenced by setMainFunctions().

6.11.2.36 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 component of the viscosity.

## Parameters

in,out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
in	<i>parameters</i>	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::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nV.

Referenced by setMainFunctions().

#### 6.11.2.37 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

in, out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculations.
in	<i>parameters</i>	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::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

#### 6.11.2.38 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

in, out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
in	<i>parameters</i>	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::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

6.11.2.39 void calNewR ( Grid & *grid*, Time & *time* )

This function calculates the radii, from the new radial grid velocities

## Parameters

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial velocities
in	<i>time</i>	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().

6.11.2.40 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

in, out	<i>grid</i>	supplies the input for calculating the pressure and also accepts the result of the pressure calculation
in	<i>parameters</i>	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().

6.11.2.41 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial grid velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology
in	<i>messPass</i>	

**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, ProcTop::nCoords, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, ProcTop::nNumRadialNeighbors, ProcTop::nRadialNeighborNeighborIDs, ProcTop::nRadialNeighborRanks, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, MessPass::typeRecvNewVar, and MessPass::typeSendNewVar.

Referenced by setMainFunctions().

**6.11.2.42 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial grid velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology
in, out	<i>messPass</i>	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::nDCosThetaJK, Grid::nDonorCellFrac, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nLocalGridDims, Grid::nNumGhostCells, 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().

**6.11.2.43 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial grid velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology
in, out	<i>messPass</i>	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::nDCosThetaJK`, `Grid::nDonorCellFrac`, `Grid::nDPhi`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nLocalGridDims`, `Grid::nNumGhostCells`, `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()`.

6.11.244 `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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 model, setting it to zero.

**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.

References `Parameters::dAlpha`, `Time::deltat_n`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDM`,

Grid::nDonorCellFrac, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGlobalGridPositionLocalGrid, Grid::nM, Grid::nP, Grid::nQ0, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

Referenced by calNewVelocities\_R().

**6.11.2.45 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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, Time::dDeltat\_n, Parameters::dG, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nDonorCellFrac, 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().

6.11.2.46 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology

**Boundary Conditions** Missing `grid.dLocalGridOld[grid.nD][nlCen+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][nlCen+1][0][0]` in calculation of  $\langle \rho \rangle_{i+1/2}$ , setting it to zero.

**Boundary Conditions** Missing `grid.dLocalGridOld[grid.nP][nlCen+1][j][k]` in calculation of  $S_1$ , setting it to  $-1.0 * dP\_ijk\_n$ .

**Boundary Conditions** Missing `grid.dLocalGridOld[grid.nDM][nlCen+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][nlCen+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][nlCen][0][0]` instead.

References `Parameters::dAlpha`, `Time::dDeltat_n`, `Parameters::dG`, `Grid::dLocalGrid-New`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `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()`.

6.11.2.47 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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  $V_i$  at  $i+1$  is equal to  $V_i$  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`, `Time::dDeltat_n`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaIJK`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nDTheta`, `Grid::nEddyVisc`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nGlobalGridPositionLocalGrid`, `Grid::nM`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIj1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `calNewVelocities_RT_LES()`.

6.11.2.48 `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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`, `Time::dDeltat_n`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nM`, `Grid::nP`, `Grid::nQ0`, `Grid::nR`, `Grid::nSinThetaJK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `calNewVelocities_RTP()`.

**6.11.2.49** `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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  $V_i$  at  $i+1$  is equal to  $V_i$  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`, `Time::dDeltat_n`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaJK`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nEddyVisc`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nGlobalGridPositionLocalGrid`, `Grid::nM`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `Grid::nSinThetaJK`, `Grid::nSinThetaJp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `calNewVelocities_RTP_LES()`.

6.11.250 `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated theta velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `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()`.

**6.11.251** `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated theta velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaJp1halfK`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nDTheta`, `Grid::nEddyVisc`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nGlobalGridPositionLocalGrid`, `Grid::nNumGhostCells`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaJp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `calNewVelocities_RT_LES()`.

### 6.11.2.52 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated theta velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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** Using centered gradient for upwind gradient outside star at surface.

References Time::deltat\_n, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaJp1halfK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDonorCellFrac, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdate-Explicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by calNewVelocities\_RTP().

### 6.11.2.53 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated theta velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaJp1halfK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDonorCellFrac, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGlobalGridPositionLocalGrid, - Grid::nNumGhostCells, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaJK, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by calNewVelocities\_RTP\_LES().

#### 6.11.254 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

in, out	<i>grid</i>	contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
in	<i>parameters</i>	contains parameters used in the calculation of the new velocities.
in	<i>time</i>	contains time step information, current time step, and current time
in	<i>procTop</i>	contains processor topology information

References calNewU\_R().

Referenced by setMainFunctions().

#### 6.11.255 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

in, out	<i>grid</i>	contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
in	<i>parameters</i>	contains parameters used in the calculation of the new velocities.
in	<i>time</i>	contains time step information, current time step, and current time
in	<i>procTop</i>	contains processor topology information

References calNewU\_R\_LES().

### 6.11.2.56 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

in, out	<i>grid</i>	contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
in	<i>parameters</i>	contains parameters used in the calculation of the new velocities.
in	<i>time</i>	contains time step information, current time step, and current time
in	<i>procTop</i>	contains processor topology information

References `calNewU_RT()`, and `calNewV_RT()`.

Referenced by `setMainFunctions()`.

### 6.11.2.57 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

in, out	<i>grid</i>	contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
in	<i>parameters</i>	contains parameters used in the calculation of the new velocities.
in	<i>time</i>	contains time step information, current time step, and current time
in	<i>procTop</i>	contains processor topology information

References `calNewU_RT_LES()`, and `calNewV_RT_LES()`.

Referenced by `setMainFunctions()`.

### 6.11.2.58 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

in, out	<i>grid</i>	contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
in	<i>parameters</i>	contains parameters used in the calculation of the new velocities.
in	<i>time</i>	contains time step information, current time step, and current time
in	<i>procTop</i>	contains processor topology information

References `calNewU_RTP()`, `calNewV_RTP()`, and `calNewW_RTP()`.

Referenced by `setMainFunctions()`.

**6.11.2.59** `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

in, out	<i>grid</i>	contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
in	<i>parameters</i>	contains parameters used in the calculation of the new velocities.
in	<i>time</i>	contains time step information, current time step, and current time
in	<i>procTop</i>	contains processor topology information

References `calNewU_RTP_LES()`, `calNewV_RTP_LES()`, and `calNewW_RTP_LES()`.

Referenced by `setMainFunctions()`.

**6.11.2.60** `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated theta velocities
in	<i>parameters</i>	various parameters needed for the calculation

in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaJK`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `Grid::nSinThetaJK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `calNewVelocities_RTP()`.

6.11.2.61 `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated theta velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaJK`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nEddyVisc`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nGlobalGridPositionLocalGrid`, `Grid::nNumGhostCells`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `Grid::nSinTheta`



IJK, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by calNewVelocities\_RTP\_LES().

#### 6.11.2.62 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.

#### 6.11.2.63 void calOldDenave\_R( Grid & grid )

This function does nothing as the averaged density is not needed in 1D calculations.

##### Parameters

<code>in,out</code>	<code>grid</code>	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().

#### 6.11.2.64 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

<code>in,out</code>	<code>grid</code>	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().

### 6.11.2.65 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

in, out	<i>grid</i>	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::nDCosThetaJK`, `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()`.

### 6.11.2.66 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.

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()`.

### 6.11.2.67 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

in, out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	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()`.

**6.11.2.68 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**

in,out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	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.

Referenced by initInternalVars().

**6.11.2.69 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**

in,out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	contains parameters used in calculating the eddy viscosity.

**Boundary Conditions** assuming that theta velocity is constant across surface

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().

**6.11.2.70 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**

in,out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	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::nSinThetaJK, Grid::nStartGhostUpdateExplicit, and - Grid::nStartUpdateExplicit.

Referenced by initInternalVars().

#### 6.11.2.71 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

in, out	grid	supplies the input for calculating the eddy viscosity.
in	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::nCotThetaJK, Grid::nD, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nNumGhostCells, Grid::nR, Grid::nSinThetaJK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by initInternalVars().

#### 6.11.2.72 void calOldP\_GL( Grid & grid, Parameters & parameters )

This function calculates the pressure using a gamma law gas, calculate by [dEOS\\_GL](#).

##### Parameters

in, out	grid	supplies the input for calculating the pressure and also accepts the results of the pressure calculations
in	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().

**6.11.2.73** 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

in,out	<i>grid</i>	supplies the input for calculating the pressure and also accepts the result of the pressure calculation
in	<i>parameters</i>	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()`.

**6.11.2.74** 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

in,out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
in	<i>parameters</i>	contains parameters used in calculating the artificial viscosity.

References `Parameters::dA`, `Parameters::dAVThreshold`, `Parameters::dGamma`, `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 `initInternalVars()`.

**6.11.2.75** 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

in, out	<i>grid</i>	supplies the input for calculating the pressure and also accepts the result of the pressure calculation
in	<i>parameters</i>	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::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU.

Referenced by initInternalVars().

#### 6.11.2.76 void calOldQ0Q1\_RT\_GL ( Grid & *grid*, Parameters & *parameters* )

This function calculates the artificial viscosity of a cell. It calculates it using the old values 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

in, out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
in	<i>parameters</i>	contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma, 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.

Referenced by initInternalVars().

#### 6.11.2.77 void calOldQ0Q1\_RT\_TEOS ( Grid & *grid*, Parameters & *parameters* )

This function calculates the artificial viscosity of a cell. It calculates it using the old values 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

in, out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
in	<i>parameters</i>	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::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nV.

Referenced by initInternalVars().

#### 6.11.2.78 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

in,out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
in	<i>parameters</i>	contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

Referenced by initInternalVars().

#### 6.11.2.79 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

in,out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
in	<i>parameters</i>	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::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

Referenced by `initInternalVars()`.

**6.11.2.80** `double dEOS_GL ( double dRho, double dE, Parameters parameters )`

Calculates the pressure from the energy and density using a  $\gamma$ -law gas.

#### Parameters

<code>in</code>	<code>dRho</code>	the density of a cell
<code>in</code>	<code>dE</code>	the energy of a cell
<code>in</code>	<code>parameters</code>	contains various parameters, including $\gamma$ needed to calculate the pressure.

#### Returns

the pressure

This version of `dEOS_GL` uses the same value of  $\gamma$  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()`.

**6.11.2.81** `double dET4 ( Parameters & parameters, double dEddyVisc_ijk_np1half, double dRho_ijk_np1half, double dLengthScale4_ijk_np1half ) [inline]`

This is an additional turbulence term to be added to the energy equation.

Referenced by `calNewE_RT_NA_LES()`, `calNewE_RTP_NA_LES()`, `dImplicitEnergyFunction_RT_LES()`, `dImplicitEnergyFunction_RT_LES_SB()`, `dImplicitEnergyFunction_RTP_LES()`, and `dImplicitEnergyFunction_RTP_LES_SB()`.

**6.11.2.82** `double dImplicitEnergyFunction_None ( Grid & grid, Parameters & parameters, Time & time, double dTempsf[], 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()`.

**6.11.2.83** `double dImplicitEnergyFunction_R ( Grid & grid, Parameters & parameters, Time & time, double dTempsf[], 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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	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 $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

References Parameters::bDEDMClamp, Parameters::dDEDMClampMr, Parameters::dDEDMClampValue, Time::dDeltat\_np1half, DEBUG\_EQUATIONS, 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::nDonorCellFrac, Grid::nE, Grid::nGlobalGridPositionLocalGrid, Grid::nM, Grid::nNumGhostCells, Grid::nQ0, Grid::nR, Grid::nT, Grid::nU, and Grid::nU0.

Referenced by setMainFunctions().

#### 6.11.2.84 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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	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 $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

**Todo** this function should probably be turffed, the LES terms aren't needed in 1D. keeping it for now though.

References Time::dDeltat\_np1half, 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::nDonorCellFrac, Grid::nE, Grid::nEddyVisc, Grid::nGlobalGridPositionLocalGrid, Grid::nNumGhostCells, Grid::nQ0, Grid::nR, Grid::nT, Grid::nU, and Grid::nU0.

**6.11.2.85** `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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	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 $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	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\_np1half, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nDonorCellFrac, Grid::nE, Grid::nEddyVisc, Grid::nQ0, Grid::nR, Grid::nT, Grid::nU, and Grid::nU0.

6.11.2.86 `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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	<code>dTemps[0]=dT_ijk_np1</code> is the temperature at radial position $(i, j, k)$ and time $n + 1$ and time $n + 1$ , <code>dTemps[1]=dT_im1jk_np1</code> is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

**Boundary Conditions** missing density outside model assuming it is zero

**Boundary Conditions** missing density outside model assuming it is zero

**Boundary Conditions** Assuming energy outside model is the same as the energy in the last zone inside the model.

**Boundary Conditions** A1 upwind set to zero as no material is flowing into the star

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

References `Parameters::bDEDMClamp`, `Parameters::dDEDMClampMr`, `Parameters::dDEDMClampValue`, `Time::dDeltat_np1half`, `DEBUG_EQUATIONS`, `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::nDonorCellFrac`, `Grid::nE`, `Grid::nGlobalGridPositionLocalGrid`, `Grid::nM`, `Grid::nNumGhostCells`, `Grid::nQ0`, `Grid::nR`, `Grid::nT`, `Grid::nU`, and `Grid::nU0`.

Referenced by `setMainFunctions()`.

6.11.2.87 `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 con-

tains 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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	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 $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

References Time::dDeltat\_np1half, 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::nDonorCellFrac, Grid::nDTheta, Grid::nE, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nSinThetaJK, Grid::nSinThetaJp1halfK, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().

#### 6.11.2.88 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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	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 $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

References Parameters::bDEDMClamp, Parameters::dDEDMClampMr, Parameters::dDEDMClampValue, Time::dDeltat\_np1half, DEBUG\_EQUATIONS, dET4(), 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::nDonorCellFrac, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nGlobalGridPositionLocalGrid, Grid::nM, Grid::nNumGhostCells, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nSinThetaJK, Grid::nSinThetaJp1halfK, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().

#### 6.11.2.89 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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	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 $n + 1$ .

in	$i$	is the radial index to evaluate the function at.
in	$j$	is the theta index to evaluate the function at.
in	$k$	is the phi index to evaluate the function at.

**Boundary Conditions** Missing  $\Delta M_r$  outside model using [Parameters::dAlpha](#) times  $\Delta M_r$  in the last zone instead.

**Boundary Conditions** missing density outside model assuming it is zero

**Boundary Conditions** missing desnity outside model assuming it is zero

**Boundary Conditions** assuming  $V$  at  $ip1$  half 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** A1 upwind set to zero as no material is flowing into the star

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

References [Parameters::bDEDMClamp](#), [Parameters::dAlpha](#), [Parameters::dDEDMClampMr](#), [Parameters::dDEDMClampValue](#), [Time::dDeltat\\_np1half](#), [DEBUG\\_EQUATIONS](#), [dET4\(\)](#), [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::nDonorCellFrac](#), [Grid::nDTheta](#), [Grid::nE](#), [Grid::nEddyVisc](#), [Grid::nGlobalGridPositionLocalGrid](#), [Grid::nM](#), [Grid::nNumGhostCells](#), [Grid::nQ0](#), [Grid::nQ1](#), [Grid::nR](#), [Grid::nSinThetaJK](#), [Grid::nSinThetaJp1halfK](#), [Grid::nT](#), [Grid::nU](#), [Grid::nU0](#), and [Grid::nV](#).

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

**6.11.2.90** `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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	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 $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	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\_np1half, 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::nDonorCellFrac, Grid::nDTheta, Grid::nE, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nSinThetaIJK, Grid::nSinThetaIp1halfK, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().

**6.11.2.91** `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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i> , <i>dTemps[0]=d-T_ijk_np1</i>	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 $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

References Time::dDeltat\_np1half, 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::nDonorCellFrac, 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().

#### 6.11.2.92 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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i> , <i>dTemps[0]=d-T<sub>ijk</sub>_np1</i>	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 $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

References Parameters::bDEDMClamp, Parameters::dDEDMClampMr, Parameters::dDEDMClampValue, Time::dDeltat\_np1half, DEBUG\_EQUATIONS, dET4(), 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::nDonorCellFrac, Grid::nDPhi, Grid::nDTheta, Grid::nE, Grid::nEddyVisc, Grid::nGlobalGridPositionLocalGrid, Grid::nM, Grid::nNumGhostCells, 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().

### 6.11.2.93 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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	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 $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

**Boundary Conditions** Missing  $\Delta M_r$  outside model using [Parameters::dAlpha](#) times  $\Delta M_r$  in the last zone instead.

**Boundary Conditions** missing density outside model assuming it is zero

**Boundary Conditions** missing desnity 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** A1 upwind set to zero as no material is flowing into the star

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

References [Parameters::bDEDMClamp](#), [Parameters::dAlpha](#), [Parameters::dDEDMClampMr](#), [Parameters::dDEDMClampValue](#), [Time::dDeltat\\_np1half](#), [DEBUG\\_EQUATIONS](#), [dET4\(\)](#), [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::nDonorCellFrac](#), [Grid::nDPhi](#), [Grid::nDTheta](#), [Grid::nE](#), [Grid::nEddyVisc](#),

Grid::nGlobalGridPositionLocalGrid, Grid::nM, Grid::nNumGhostCells, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaJk, Grid::nSinThetaJp1halfK, Grid::nT, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

#### 6.11.2.94 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

in	grid	
in	parameters	
in	time	
in	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 $n + 1$ .
in	i	is the radial index to evaluate the function at.
in	j	is the theta index to evaluate the function at.
in	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\_np1half, 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::nDonorCellFrac, Grid::nDPhi, Grid::nDTheta, Grid::nE, 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().

**6.11.2.95 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.

Referenced by setMainFunctions().

**6.11.2.96 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.

References calNewPEKappaGamma\_TEOS(), Implicit::dAverageRHS, Implicit::dCurrentRelTErrors, Implicit::dDerivativeStepFraction, Grid::dLocalGridNew, Implicit::dMaxErrorInRHS, Implicit::dTolerance, 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().

**6.11.2.97 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 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::dCurrentRelTErr`, `Implicit::dDerivativeStepFraction`, `Grid::dLocalGridNew`, `Implicit::dMaxErrorInRHS`, `Implicit::dTolerance`, `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()`.

**6.11.2.98** `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::dCurrentRelTErr`, `Implicit::dDerivativeStepFraction`, `Grid::dLocalGridNew`, `Implicit::dMaxErrorInRHS`, `Implicit::dTolerance`, `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()`.

**6.11.2.99** `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 function is for 1D, gamma law calculations.

References Parameters::dDonorCellMin, Parameters::dDonorCellMultiplier, Parameters::dGamma, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

Referenced by initInternalVars().

#### 6.11.2.100 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 function is for 1D, tabulated equation of state calculations.

References Parameters::dDonorCellMin, Parameters::dDonorCellMultiplier, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

Referenced by initInternalVars().

#### 6.11.2.101 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 function is for 2D, gamma law calculations.

References Parameters::dDonorCellMin, Parameters::dDonorCellMultiplier, Parameters::dGamma, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by initInternalVars().

#### 6.11.2.102 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::dDonorCellMin, Parameters::dDonorCellMultiplier, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, - Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by initInternalVars().

#### 6.11.2.103 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::dDonorCellMin, Parameters::dDonorCellMultiplier, Parameters::dGamma, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, 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().

#### 6.11.2.104 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::dDonorCellMin, Parameters::dDonorCellMultiplier, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, 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().

#### 6.11.2.105 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

in, out	<i>grid</i>	supplies information needed for initializing internal variables as well as storing the initialized internal variables
in	<i>procTop</i>	contains information about processor topology
in	<i>parameters</i>	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::nCotThetaJK, Grid::nCotThetaJp1halfK, Grid::nDCosThetaJK, Grid::nDPhi, Grid::nDTheta, Grid::nLocalGridDims, Grid::nNumDims, Grid::nNumGhostCells, Grid::nPhi, ProcTop::nRank, Grid::nSinThetaJK, Grid::nSinThetaJp1halfK, Grid::nTheta, and Parameters::nTypeTurbulenceMod.

Referenced by init().

### 6.11.2.106 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

in, out	<i>grid</i>	supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.
in	<i>parameters</i>	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::nCotThetaJK, Grid::nCotThetaJp1halfK, Grid::nDCosThetaJK, Grid::nDenAve, Grid::nDonorCellFrac, 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().

**6.11.2.107** 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

out	<i>functions</i>	is of class <a href="#">Functions</a> and is used to specify the functions called to calculate the evolution of the input model.
in	<i>procTop</i>	is of type <a href="#">ProcTop</a> . <a href="#">ProcTop::nRank</a> is used to set different functions based on processor rank. For instance processor rank 1 requires 1D versions of the equations.
in	<i>parameters</i>	is of class <a href="#">Parameters</a> . It holds various constants and run-time parameters.
in	<i>grid</i>	of type <a href="#">Grid</a> . This function requires the number of dimensions, specified by <a href="#">Grid::nNumDims</a> .
in	<i>time</i>	of type <a href="#">Time</a> . This function requires knowledge of the type of time setp being used, specified by <a href="#">Time::bVariableTimeStep</a> .
in	<i>implicit</i>	of type <a href="#">Implicit</a> . This function needs to know if there is an implicit region, specified when <a href="#">Implicit::nNumImplicitZones</a> >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::fp](#)

CalculateNewAV, Functions::fpCalculateNewDensities, Functions::fpCalculateNewEddyVisc, Functions::fpCalculateNewEnergies, Functions::fpCalculateNewEOSVars, Functions::fpCalculateNewGridVelocities, Functions::fpCalculateNewRadii, Functions::fpCalculateNewVelocities, 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().

## 6.12 physEquations.h File Reference

```
#include "global.h"
```

### Functions

- void [setMainFunctions](#) ([Functions](#) &functions, [ProcTop](#) &procTop, [Parameters](#) &parameters, [Grid](#) &grid, [Time](#) &time, [Implicit](#) &implicit)
- void [setInternalVarInf](#) ([Grid](#) &grid, [Parameters](#) &parameters)
- void [initInternalVars](#) ([Grid](#) &grid, [ProcTop](#) &procTop, [Parameters](#) &parameters)
- void [calNewVelocities\\_R](#) ([Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities\\_R\\_LES](#) ([Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities\\_RT](#) ([Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities\\_RT\\_LES](#) ([Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities\\_RTP](#) ([Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewVelocities\\_RTP\\_LES](#) ([Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU\\_R](#) ([Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU\\_R\\_LES](#) ([Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU\\_RT](#) ([Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU\\_RT\\_LES](#) ([Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calNewU\\_RTP](#) ([Grid](#) &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)

- void `calNewU_RTP_LES` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewV_RT` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewV_RT_LES` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewV_RTP` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewV_RTP_LES` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewW_RTP` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewW_RTP_LES` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewU0_R` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewU0_RT` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewU0_RTP` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass)
- void `calNewR` (`Grid` &grid, `Time` &time)
- void `calNewD_R` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewD_RT` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewD_RTP` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_R_AD` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_R_NA` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_R_NA_LES` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RT_AD` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RT_NA` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RT_NA_LES` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RTP_AD` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RTP_NA` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `calNewE_RTP_NA_LES` (`Grid` &grid, `Parameters` &parameters, `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](#) &parameters)
- void [calNewTPKappaGamma\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewPEKappaGamma\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewQ0\\_R\\_GL](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewQ0\\_R\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewQ0Q1\\_RT\\_GL](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewQ0Q1\\_RT\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewQ0Q1Q2\\_RTP\\_GL](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewQ0Q1Q2\\_RTP\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewEddyVisc\\_None](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewEddyVisc\\_R\\_CN](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewEddyVisc\\_RT\\_CN](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewEddyVisc\\_RTP\\_CN](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewEddyVisc\\_R\\_SM](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewEddyVisc\\_RT\\_SM](#) (Grid &grid, [Parameters](#) &parameters)
- void [calNewEddyVisc\\_RTP\\_SM](#) (Grid &grid, [Parameters](#) &parameters)
- 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](#) &parameters)
- void [calOldPEKappaGamma\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldQ0\\_R\\_GL](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldQ0\\_R\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldQ0Q1\\_RT\\_GL](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldQ0Q1\\_RT\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldQ0Q1Q2\\_RTP\\_GL](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldQ0Q1Q2\\_RTP\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldEddyVisc\\_R\\_CN](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldEddyVisc\\_RT\\_CN](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldEddyVisc\\_RTP\\_CN](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldEddyVisc\\_R\\_SM](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldEddyVisc\\_RT\\_SM](#) (Grid &grid, [Parameters](#) &parameters)
- void [calOldEddyVisc\\_RTP\\_SM](#) (Grid &grid, [Parameters](#) &parameters)
- void [calDelt\\_R\\_GL](#) (Grid &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calDelt\\_R\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calDelt\\_RT\\_GL](#) (Grid &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calDelt\\_RT\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calDelt\\_RTP\\_GL](#) (Grid &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)
- void [calDelt\\_RTP\\_TEOS](#) (Grid &grid, [Parameters](#) &parameters, [Time](#) &time, [ProcTop](#) &procTop)

- void `calDelt_CONST` (`Grid` &grid, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop)
- void `implicitSolve_None` (`Grid` &grid, `Implicit` &implicit, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass, `Functions` &functions)
- void `implicitSolve_R` (`Grid` &grid, `Implicit` &implicit, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass, `Functions` &functions)
- void `implicitSolve_RT` (`Grid` &grid, `Implicit` &implicit, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass, `Functions` &functions)
- void `implicitSolve_RTP` (`Grid` &grid, `Implicit` &implicit, `Parameters` &parameters, `Time` &time, `ProcTop` &procTop, `MessPass` &messPass, `Functions` &functions)
- double `dImplicitEnergyFunction_None` (`Grid` &grid, `Parameters` &parameters, `Time` &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R` (`Grid` &grid, `Parameters` &parameters, `Time` &time, double dTemps[], int i, int j, int k)
- double `dImplicitEnergyFunction_R_SB` (`Grid` &grid, `Parameters` &parameters, `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)
- double `dET4` (`Parameters` &parameters, double dEddyVisc\_ijk\_np1half, double dRho\_ijk\_np1half, double dLengthScale4\_ijk\_np1half)

### 6.12.1 Detailed Description

Header file for [physEquations.cpp](#)

### 6.12.2 Function Documentation

**6.12.2.1 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, ProcTop::nRank, and Time::nTimeStepIndex.

Referenced by setMainFunctions().

**6.12.2.2 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

in	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in, out	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology. This function uses <a href="#">ProcTop::nRank</a> to pass messages.

References Time::dDelE\_t\_E\_max, Time::dDelRho\_t\_Rho\_max, Time::dDeltat\_n, Time::dDeltat\_nm1half, Time::dDeltat\_np1half, Parameters::dDonorCellMin, Parameters::dDonorCellMultiplier, Parameters::dGamma, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, 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().

**6.12.2.3 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

in	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in,out	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology. This function uses <a href="#">ProcTop::nRank</a> to pass messages.

References `Time::dDelRho_t_Rho_max`, `Time::dDelT_t_T_max`, `Time::dDeltat_n`, `Time::dDeltat_nm1half`, `Time::dDeltat_np1half`, `Parameters::dDonorCellMin`, `Parameters::dDonorCellMultiplier`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dMaxConvectiveVelocity`, `Time::dPerChange`, `Time::dt`, `Time::dTimeStepFactor`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDonorCellFrac`, `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()`.

#### 6.12.2.4 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

in	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in,out	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology. This function uses <a href="#">ProcTop::nRank</a> to pass messages.

References `Time::dDelE_t_E_max`, `Time::dDelRho_t_Rho_max`, `Time::dDeltat_n`, `Time::dDeltat_nm1half`, `Time::dDeltat_np1half`, `Parameters::dDonorCellMin`, `Parameters::dDonorCellMultiplier`, `Parameters::dGamma`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dMaxConvectiveVelocity`, `Time::dPerChange`, `Time::dt`, `Time::dTimeStepFactor`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDonorCellFrac`, `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()`.

### 6.12.2.5 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

in	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in, out	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology. This function uses <a href="#">ProcTop::nRank</a> to pass messages.

References Time::dDelRho\_t\_Rho\_max, Time::dDeIT\_t\_T\_max, Time::dDeltat\_n, Time::dDeltat\_nm1half, Time::dDeltat\_np1half, Parameters::dDonorCellMin, - Parameters::dDonorCellMultiplier, Grid::dLocalGridNew, Grid::dLocalGridOld, - Parameters::dMaxConvectiveVelocity, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, 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().

### 6.12.2.6 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

in	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in, out	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology. This function uses <a href="#">ProcTop::nRank</a> to pass messages.

References Time::dDeIE\_t\_E\_max, Time::dDelRho\_t\_Rho\_max, Time::dDeltat\_n, Time::dDeltat\_nm1half, Time::dDeltat\_np1half, Parameters::dDonorCellMin, - Parameters::dDonorCellMultiplier, Parameters::dGamma, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, 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().

**6.12.2.7 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

in	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in, out	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology. This function uses <a href="#">ProcTop::nRank</a> to pass messages.

References Time::dDelRho\_t\_Rho\_max, Time::dDelT\_t\_T\_max, Time::dDeltat\_n, Time::dDeltat\_nm1half, Time::dDeltat\_np1half, Parameters::dDonorCellMin, - Parameters::dDonorCellMultiplier, Grid::dLocalGridNew, Grid::dLocalGridOld, - Parameters::dMaxConvectiveVelocity, Time::dPerChange, Time::dt, Time::dTimeStepFactor, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, ProcTop::nRank, Grid::nSinTheta\_IJK, Grid::nStartUpdateExplicit, Grid::nT, Time::nTimeStepIndex, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

**6.12.2.8 void calNewD\_R( Grid & *grid*, Parameters & *parameters*, Time & *time*, ProcTop & *procTop* )**

This function calculates new densities using terms in the radial direction only

#### Parameters

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology, uses <a href="#">ProcTop::nRank</a> 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::nDonorCellFrac, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGlobalGridPositionLocalGrid, Grid::nNumGhostCells, Grid::nR, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, -

Grid::nU, and Grid::nU0.

Referenced by setMainFunctions().

**6.12.2.9 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology, uses <a href="#">ProcTop::nRank</a> when reporting negative densities

**Boundary Conditions** doesn't allow mass flux through outter interface

References Time::deltat\_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDCosThetaIJK, Grid::nDonorCellFrac, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGlobalGridPositionLocalGrid, Grid::nNumGhostCells, Grid::nR, ProcTop::nRank, Grid::nSinThetaIp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().

**6.12.2.10 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology, uses <a href="#">ProcTop::nRank</a> when reporting negative densities

**Boundary Conditions** doesn't allow mass flux through outter interface

References Time::deltat\_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDCosThetaIJK, Grid::nDonorCellFrac, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGlobalGridPositionLocalGrid, Grid::nNumGhostCells, Grid::nR, ProcTop::nRank, Grid::nSin-

ThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

#### 6.12.2.11 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

in, out	<i>grid</i>	
---------	-------------	--

Referenced by setMainFunctions().

#### 6.12.2.12 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

in, out	<i>grid</i>	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().

#### 6.12.2.13 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

in, out	<i>grid</i>	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().

#### 6.12.2.14 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

in, out	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().

#### 6.12.2.15 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

in, out	grid	contains the local grid, and will hold the newly updated densities
in	parameters	various parameters needed for the calculation
in	time	contains time information, e.g. time step, current time etc.
in	procTop	

References Time::dDeltat\_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, - Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nDonorCellFrac, 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().

6.12.2.16 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	

**Boundary Conditions** Setting energy at surface equal to energy in last zone.

**Boundary Conditions** Upwind gradient in dA1 term should be zero as there is no flow into the star.

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

References Time::dDeltat\_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, - Parameters::dPi, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDonorCellFrac, Grid::nE, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGlobalGridPositionLocalGrid, Grid::nKappa, Grid::nNumGhostCells, Grid::nP, Grid::nQ0, Grid::nR, ProcTop::nRank, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nT, Grid::nU, and Grid::nU0.

Referenced by setMainFunctions().

6.12.2.17 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</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 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]`

**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.

References `Time::dDeltat_np1half`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `-Parameters::dPi`, `Parameters::dSigma`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `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`.

**6.12.2.18** `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	

**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_np1half`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `-Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nDTheta`, `Grid::nE`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `ProcTop::nRank`, `Grid::nSinThetaJK`, `Grid::nSinThetaJp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `setMainFunctions()`.

6.12.2.19 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</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 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_np1half`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, - `Parameters::dPi`, `Parameters::dSigma`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `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()`.

6.12.2.20 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	

**Boundary Conditions** Missing  $\Delta M_r$  outside model using `Parameters::dAlpha` times  $\Delta M_r$  in the last zone instead.

**Boundary Conditions** Setting energy at surface equal to energy in last zone.

**Boundary Conditions** missing density outside model, setting it to zero

**Boundary Conditions** missing eddy viscosity outside the model setting it to zero

**Boundary Conditions** Upwind gradient in dA1 term should be zero as there is no flow into the star.

**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.

References `Parameters::dAlpha`, `Time::dDeltat_np1half`, `dET4()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dPrt`, `Parameters::dSigma`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nDTheta`, `Grid::nE`, `Grid::nEddyVisc`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nGlobalGridPositionLocalGrid`, `Grid::nKappa`, `Grid::nNumGhostCells`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `ProcTop::nRank`, `Grid::nSinThetaIJK`, `Grid::nSinThetaJp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `setMainFunctions()`.

**6.12.2.21** `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</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 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\_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, - Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDonorCellFrac, 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().

**6.12.2.22 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</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\_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, - Parameters::dPi, Parameters::dSigma, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDonorCellFrac, 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().

**6.12.2.23 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated densities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	

**Boundary Conditions** Missing  $\Delta M_r$  outside model using `Parameters::dAlpha` times  $\Delta M_r$  in the last zone instead.

**Boundary Conditions** Missing W at i+1, assuming the same as at i

**Boundary Conditions** Setting energy at surface equal to energy in last zone.

**Boundary Conditions** missing density outside model, setting it to zero

**Boundary Conditions** missing eddy viscosity outside the model setting it to zero

**Boundary Conditions** Upwind gradient in dA1 term should be zero as there is no flow into the star.

**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.

References `Parameters::dAlpha`, `Time::dDeltat_np1half`, `dET4()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dPrt`, `Parameters::dSigma`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nE`, `Grid::nEddyVisc`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nGlobalGridPositionLocalGrid`, `Grid::nKappa`, `Grid::nNumGhostCells`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `ProcTop::nRank`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `setMainFunctions()`.

#### 6.12.2.24 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

in,out	<i>grid</i>	
in	<i>parameters</i>	

Referenced by setMainFunctions().

#### 6.12.2.25 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

in,out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	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.

#### 6.12.2.26 void calNewEddyVisc\_R\_SM( Grid & *grid*, Parameters & *parameters* )

This function calculates the eddy viscosity with only the radial terms.

##### Parameters

in,out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	contains parameters used in calculating the eddy viscosity.

References Parameters::dEddyViscosity, Grid::dLocalGridNew, Grid::nCenIntOffset, - Grid::nD, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and - Grid::nU0.

#### 6.12.2.27 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

in,out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	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().

#### 6.12.2.28 void calNewEddyVisc\_RT\_SM ( Grid & *grid*, Parameters & *parameters* )

This function calculates the eddy viscosity with only the radial and theta terms.

##### Parameters

in, out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	contains parameters used in calculating the eddy viscosity.

**Boundary Conditions** assuming that theta velocity is constant across surface

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().

#### 6.12.2.29 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

in, out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	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::nSinThetaJK, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

Referenced by setMainFunctions().

#### 6.12.2.30 void calNewEddyVisc\_RTP\_SM ( Grid & *grid*, Parameters & *parameters* )

This function calculates the eddy viscosity with only the radial, theta, and phi terms.

##### Parameters

in, out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	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::nCotThetaJK, Grid::nD, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nSinThetaJK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

#### 6.12.2.31 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

in,out	<i>grid</i>	supplies the input for calculating the pressure and also accepts the result of the pressure calculations.
in	<i>parameters</i>	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().

#### 6.12.2.32 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

in,out	<i>grid</i>	supplies the input for calculating the pressure and also accepts the result of the pressure calculation
in	<i>parameters</i>	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().

### 6.12.2.33 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

in, out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation.
in	<i>parameters</i>	contains parameters used when calculating the artificial viscosity, namely the adiabatic gamma.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma, Grid::dLocalGridNew, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU.

Referenced by setMainFunctions().

### 6.12.2.34 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 equation of state for the calculation.

#### Parameters

in, out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
in	<i>parameters</i>	contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Grid::dLocalGridNew, 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().

### 6.12.2.35 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 components of the viscosity. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

## Parameters

in, out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculations.
in	<i>parameters</i>	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::nSinThetaJK, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nV.

Referenced by setMainFunctions().

6.12.2.36 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

in, out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
in	<i>parameters</i>	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::nSinThetaJK, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nV.

Referenced by setMainFunctions().

6.12.2.37 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 components of the viscosity. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

## Parameters

in, out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculations.
in	<i>parameters</i>	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::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

#### 6.12.2.38 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

in, out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
in	<i>parameters</i>	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::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

#### 6.12.2.39 void calNewR ( Grid & *grid*, Time & *time* )

This function calculates the radii, from the new radial grid velocities

##### Parameters

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial velocities
in	<i>time</i>	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().



6.12.2.40 `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

in,out	<i>grid</i>	supplies the input for calculating the pressure and also accepts the result of the pressure calculation
in	<i>parameters</i>	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()`.

6.12.2.41 `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

in,out	<i>grid</i>	contains the local grid, and will hold the newly updated radial grid velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology
in	<i>messPass</i>	

**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.

References `Grid::dLocalGridNew`, `ProcTop::nCoords`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `ProcTop::nNumRadialNeighbors`, `ProcTop::nRadialNeighborNeighborIDs`, `ProcTop::nRadialNeighborRanks`, `ProcTop::nRank`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, `MessPass::typeRecvNewVar`, and `MessPass::typeSendNewVar`.

Referenced by `setMainFunctions()`.

#### 6.12.2.42 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial grid velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology
in, out	<i>messPass</i>	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::nDonorCellFrac, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nLocalGridDims, Grid::nNumGhostCells, 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().

#### 6.12.2.43 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial grid velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology
in, out	<i>messPass</i>	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::nDCosThetalJK, Grid::nDonorCellFrac, Grid::nDPhi, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nLocalGridDims, - Grid::nNumGhostCells, 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().

**6.12.2.44** 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

in,out	<i>grid</i>	contains the local grid, and will hold the newly updated radial velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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 model, setting it to zero.

**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.

References Parameters::dAlpha, Time::dDeltat\_n, Parameters::dG, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nDonorCellFrac, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGlobalGridPositionLocalGrid, Grid::nM, Grid::nP, Grid::nQ0, Grid::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

Referenced by calNewVelocities\_R().

**6.12.2.45** 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	contains information about the processor topology

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

**Boundary Conditions** Missing `grid.dLocalGridOld[grid.nDM][nlCen+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][nlCen+1][0][0]` in calculation of upwind gradient, when moving inward. Using centered gradient instead.

References `Parameters::dAlpha`, `Time::dDeltat_n`, `Parameters::dG`, `Grid::dLocalGrid-New`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDM`, `Grid::nDonorCellFrac`, `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()`.

**6.12.2.46** `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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`, `Time::dDeltat_n`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `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()`.

**6.12.2.47** `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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  $V_i$  at  $i+1$  is equal to  $V_i$  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`, `Time::dDeltat_n`, `Parameters::dG`, `Grid::dLocalGrid-New`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaIJK`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nDTheta`, `Grid::nEddyVisc`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nGlobalGridPositionLocalGrid`, `Grid::nM`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `calNewVelocities_RT_LES()`.

**6.12.2.48** `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

<code>in, out</code>	<code>grid</code>	contains the local grid, and will hold the newly updated radial velocities
<code>in</code>	<code>parameters</code>	various parameters needed for the calculation
<code>in</code>	<code>time</code>	contains time information, e.g. time step, current time etc.
<code>in</code>	<code>procTop</code>	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`, `Time::dDeltat_n`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nM`, `Grid::nP`, `Grid::nQ0`, `Grid::nR`, `Grid::nSinThetaJK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `calNewVelocities_RTP()`.

**6.12.2.49** `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated radial velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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  $v_i$  at  $i+1$  is equal to  $v_i$  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`, `Time::dDeltat_n`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaJK`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nEddyVisc`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nGlobalGridPositionLocalGrid`, `Grid::nM`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `Grid::nSinThetaJK`, `Grid::nSinThetaJp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `calNewVelocities_RTP_LES()`.

6.12.2.50 `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated theta velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `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()`.

**6.12.2.51** `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

<code>in, out</code>	<code>grid</code>	contains the local grid, and will hold the newly updated theta velocities
<code>in</code>	<code>parameters</code>	various parameters needed for the calculation
<code>in</code>	<code>time</code>	contains time information, e.g. time step, current time etc.
<code>in</code>	<code>procTop</code>	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`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaJp1halfK`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nDTheta`, `Grid::nEddyVisc`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nGlobalGridPositionLocalGrid`, `Grid::nNumGhostCells`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaJp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `calNewVelocities_RT_LES()`.

**6.12.2.52** `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated theta velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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.

References Time::dDeltat\_n, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaJp1halfK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDonorCellFrac, Grid::nDPhi, Grid::nDTheta, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by calNewVelocities\_RTP().

6.12.2.53 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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated theta velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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, Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Grid::nCenIntOffset, Grid::nCotThetaJp1halfK, Grid::nD, Grid::nDenAve, Grid::nDM, Grid::nDonorCellFrac, Grid::nDPhi, Grid::nDTheta, Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGlobalGridPositionLocalGrid, - Grid::nNumGhostCells, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaJK, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by `calNewVelocities_RTP_LES()`.

**6.12.2.54** `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

in,out	<i>grid</i>	contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
in	<i>parameters</i>	contains parameters used in the calculation of the new velocities.
in	<i>time</i>	contains time step information, current time step, and current time
in	<i>procTop</i>	contains processor topology information

References `calNewU_R()`.

Referenced by `setMainFunctions()`.

**6.12.2.55** `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

in,out	<i>grid</i>	contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
in	<i>parameters</i>	contains parameters used in the calculation of the new velocities.
in	<i>time</i>	contains time step information, current time step, and current time
in	<i>procTop</i>	contains processor topology information

References `calNewU_R_LES()`.

**6.12.2.56** `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

in,out	<i>grid</i>	contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
in	<i>parameters</i>	contains parameters used in the calculation of the new velocities.
in	<i>time</i>	contains time step information, current time step, and current time
in	<i>procTop</i>	contains processor topology information

References `calNewU_RT()`, and `calNewV_RT()`.

Referenced by `setMainFunctions()`.

**6.12.2.57 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

in,out	<i>grid</i>	contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
in	<i>parameters</i>	contains parameters used in the calculation of the new velocities.
in	<i>time</i>	contains time step information, current time step, and current time
in	<i>procTop</i>	contains processor topology information

References `calNewU_RT_LES()`, and `calNewV_RT_LES()`.

Referenced by `setMainFunctions()`.

**6.12.2.58 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

in,out	<i>grid</i>	contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
in	<i>parameters</i>	contains parameters used in the calculation of the new velocities.

in	<i>time</i>	contains time step information, current time step, and current time
in	<i>procTop</i>	contains processor topology information

References `calNewU_RTP()`, `calNewV_RTP()`, and `calNewW_RTP()`.

Referenced by `setMainFunctions()`.

**6.12.2.59** `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

in, out	<i>grid</i>	contains the local grid data and supplies the needed data to calculate the new velocities as well as holding the new velocities.
in	<i>parameters</i>	contains parameters used in the calculation of the new velocities.
in	<i>time</i>	contains time step information, current time step, and current time
in	<i>procTop</i>	contains processor topology information

References `calNewU_RTP_LES()`, `calNewV_RTP_LES()`, and `calNewW_RTP_LES()`.

Referenced by `setMainFunctions()`.

**6.12.2.60** `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated theta velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaIJK`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `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()`.

**6.12.2.61** `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

in, out	<i>grid</i>	contains the local grid, and will hold the newly updated theta velocities
in	<i>parameters</i>	various parameters needed for the calculation
in	<i>time</i>	contains time information, e.g. time step, current time etc.
in	<i>procTop</i>	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`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaIJK`, `Grid::nD`, `Grid::nDenAve`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nEddyVisc`, `Grid::nEndGhostUpdateExplicit`, `Grid::nEndUpdateExplicit`, `Grid::nGlobalGridPositionLocalGrid`, `Grid::nNumGhostCells`, `Grid::nP`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaJp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `calNewVelocities_RTP_LES()`.

**6.12.2.62** `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.

#### 6.12.2.63 void calOldDenave\_R ( Grid & grid )

This function does nothing as the averaged density is not needed in 1D calculations.

##### Parameters

in, out	<i>grid</i>	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()`.

#### 6.12.2.64 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

in, out	<i>grid</i>	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()`.

#### 6.12.2.65 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

in, out	<i>grid</i>	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::nDCosThetaJK, 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().

#### 6.12.2.66 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().

#### 6.12.2.67 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

in, out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	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().

#### 6.12.2.68 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

in,out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	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.

Referenced by initInternalVars().

#### 6.12.2.69 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

in,out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	contains parameters used in calculating the eddy viscosity.

**Boundary Conditions** assuming that theta velocity is constant across surface

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().

#### 6.12.2.70 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

in,out	<i>grid</i>	supplies the input for calculating the eddy viscosity.
in	<i>parameters</i>	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().

### 6.12.2.71 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

<i>in, out</i>	<i>grid</i>	supplies the input for calculating the eddy viscosity.
<i>in</i>	<i>parameters</i>	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().

### 6.12.2.72 void calOldP\_GL ( Grid & *grid*, Parameters & *parameters* )

This function calculates the pressure using a gamma law gas, calculate by [dEOS\\_GL](#).

#### Parameters

<i>in, out</i>	<i>grid</i>	supplies the input for calculating the pressure and also accepts the results of the pressure calculations
<i>in</i>	<i>parameters</i>	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().

### 6.12.2.73 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 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

in,out	<i>grid</i>	supplies the input for calculating the pressure and also accepts the result of the pressure calculation
in	<i>parameters</i>	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().

6.12.2.74 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

in,out	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
in	<i>parameters</i>	contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma, 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 initInternalVars().

6.12.2.75 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

in,out	<i>grid</i>	supplies the input for calculating the pressure and also accepts the result of the pressure calculation
in	<i>parameters</i>	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::nR, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, and Grid::nU.

Referenced by `initInternalVars()`.

#### 6.12.2.76 void `calOldQ0Q1_RT_GL` ( Grid & *grid*, Parameters & *parameters* )

This function calculates the artificial viscosity of a cell. It calculates it using the old values 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

<code>in, out</code>	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
<code>in</code>	<i>parameters</i>	contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma, 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.

Referenced by `initInternalVars()`.

#### 6.12.2.77 void `calOldQ0Q1_RT_TEOS` ( Grid & *grid*, Parameters & *parameters* )

This function calculates the artificial viscosity of a cell. It calculates it using the old values 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

<code>in, out</code>	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
<code>in</code>	<i>parameters</i>	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::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nV.

Referenced by `initInternalVars()`.

**6.12.2.78 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**

<i>in, out</i>	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
<i>in</i>	<i>parameters</i>	contains parameters used in calculating the artificial viscosity.

References Parameters::dA, Parameters::dAVThreshold, Parameters::dGamma, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetalJK, Grid::nSinThetalJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

Referenced by initInternalVars().

**6.12.2.79 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**

<i>in, out</i>	<i>grid</i>	supplies the input for calculating the artificial viscosity and also accepts the result of the artificial viscosity calculation
<i>in</i>	<i>parameters</i>	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::nSinThetalJK, Grid::nSinThetalJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

Referenced by initInternalVars().

**6.12.2.80 double dEOS\_GL ( double *dRho*, double *dE*, Parameters *parameters* )**

Calculates the pressure from the energy and density using a  $\gamma$ -law gas.

## Parameters

in	<i>dRho</i>	the density of a cell
in	<i>dE</i>	the energy of a cell
in	<i>parameters</i>	contians various parameters, including $\gamma$ needed to calculate the pressure.

## Returns

the pressure

This version of [dEOS\\_GL](#) uses the same value of  $\gamma$  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().

**6.12.2.81** `double dET4 ( Parameters & parameters, double dEddyVisc_ijk_np1half, double dRho_ijk_np1half, double dLengthScale4_ijk_np1half ) [inline]`

This is an additional turbulence term to be added to the energy equation.

Referenced by calNewE\_RT\_NA\_LES(), calNewE\_RTP\_NA\_LES(), dImplicitEnergyFunction\_RT\_LES(), dImplicitEnergyFunction\_RT\_LES\_SB(), dImplicitEnergyFunction\_RTP\_LES(), and dImplicitEnergyFunction\_RTP\_LES\_SB().

**6.12.2.82** `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().

**6.12.2.83** `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 funciton 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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	

in	<i>dTemps</i>	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 $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

References Parameters::bDEDMClamp, Parameters::dDEDMClampMr, Parameters::dDEDMClampValue, Time::dDeltat\_np1half, DEBUG\_EQUATIONS, 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::nDonorCellFrac, Grid::nE, Grid::nGlobalGridPositionLocalGrid, Grid::nM, Grid::nNumGhostCells, Grid::nQ0, Grid::nR, Grid::nT, Grid::nU, and Grid::nU0.

Referenced by setMainFunctions().

#### 6.12.2.84 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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	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 $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

**Todo** this function should probably be turffed, the LES terms aren't needed in 1D. keeping it for now though.

References Time::dDeltat\_np1half, eos::dGetEnergy(), eos::dGetOpacity(), eos::dGetPressure(), Grid::dLocalGridNew, Grid::dLocalGridOld, Parameters::dPi, Parameters::dSigma, Parameters::eosTable, Grid::nCenIntOffset, Grid::nD, Grid::nDenAve, Grid::n-

DM, Grid::nDonorCellFrac, Grid::nE, Grid::nEddyVisc, Grid::nGlobalGridPositionLocalGrid, Grid::nNumGhostCells, Grid::nQ0, Grid::nR, Grid::nT, Grid::nU, and Grid::nU0.

#### 6.12.2.85 `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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	$dTemps[0]=dT_{ijk\_np1}$ 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 $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	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_np1half`, `eos::dGetEnergy()`, `eos::dGetOpacity()`, `eos::dGetPressure()`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Parameters::dSigma`, `Parameters::eosTable`, `Grid::nCenIntOffset`, `Grid::nD`, `Grid::nDM`, `Grid::nDonorCellFrac`, `Grid::nE`, `Grid::nEddyVisc`, `Grid::nQ0`, `Grid::nR`, `Grid::nT`, `Grid::nU`, and `Grid::nU0`.

#### 6.12.2.86 `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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	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 $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

**Boundary Conditions** missing density outside model assuming it is zero

**Boundary Conditions** missing density outside model assuming it is zero

**Boundary Conditions** Assuming energy outside model is the same as the energy in the last zone inside the model.

**Boundary Conditions** A1 upwind set to zero as no material is flowing into the star

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

References Parameters::bDEDMClamp, Parameters::dDEDMClampMr, Parameters::dDEDMClampValue, Time::dDeltat\_np1half, DEBUG\_EQUATIONS, 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::nDonorCellFrac, Grid::nE, Grid::nGlobalGridPositionLocalGrid, Grid::nM, Grid::nNumGhostCells, Grid::nQ0, Grid::nR, Grid::nT, Grid::nU, and Grid::nU0.

Referenced by setMainFunctions().

**6.12.2.87** `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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	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 $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

References Time::dDeltat\_np1half, 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::nDonorCellFrac, Grid::nDTheta, Grid::nE, Grid::nQ0, Grid::nQ1, Grid::nR, Grid::nSinThetaJk, Grid::nSinThetaJp1halfK, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().

#### 6.12.2.88 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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	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 $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

References `Parameters::bDEDMClamp`, `Parameters::dDEDMClampMr`, `Parameters::dDEDMClampValue`, `Time::dDeltat_np1half`, `DEBUG_EQUATIONS`, `dET4()`, `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::nDonorCellFrac`, `Grid::nDTheta`, `Grid::nE`, `Grid::nEddyVisc`, `Grid::nGlobalGridPositionLocalGrid`, `Grid::nM`, `Grid::nNumGhostCells`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaJK`, `Grid::nSinThetaJp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `setMainFunctions()`.

**6.12.2.89** `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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	<code>dTemps[0]=dT_ijk_np1</code> is the temperature at radial position $(i, j, k)$ and time $n + 1$ , <code>dTemps[1]=dT_ip1jk_np1</code> is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$ , <code>dTemps[2]=dT_im1jk_np1</code> is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$ , <code>dTemps[3]=dT_ijp1k_np1</code> is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$ , <code>dTemps[4]=dT_ijm1k_np1</code> is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

**Boundary Conditions** Missing  $\Delta M_r$  outside model using `Parameters::dAlpha` times  $\Delta M_r$  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 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** A1 upwind set to zero as no material is flowing into the star

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

References `Parameters::bDEDMClamp`, `Parameters::dAlpha`, `Parameters::dDEDMClampMr`, `Parameters::dDEDMClampValue`, `Time::dDeltat_np1half`, `DEBUG_EQUATIONS`, `dET4()`, `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::nDonorCellFrac`, `Grid::nDTheta`, `Grid::nE`, `Grid::nEddyVisc`, `Grid::nGlobalGridPositionLocalGrid`, `Grid::nM`, `Grid::nNumGhostCells`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaJK`, `Grid::nSinThetaJp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `setMainFunctions()`.

**6.12.2.90** `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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	<code>dTemps[0]=dT_ijk_np1</code> is the temperature at radial position $(i, j, k)$ and time $n + 1$ , <code>dTemps[1]=dT_ip1jk_np1</code> is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$ , <code>dTemps[2]=dT_im1jk_np1</code> is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$ , <code>dTemps[3]=dT_ijp1k_np1</code> is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$ , <code>dTemps[4]=dT_ijm1k_np1</code> is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	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_np1half`, `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::nDonorCellFrac`, `Grid::nDTheta`, `Grid::nE`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIj1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `setMainFunctions()`.

**6.12.2.91** `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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i> , <i>dTemps[0]=dT_ijk_np1</i>	is the temperature at radial position $(i, j, k)$ and time $n + 1$ , <code>dTemps[1]=dT_ip1jk_np1</code> is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$ , <code>dTemps[2]=dT_im1jk_np1</code> is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$ , <code>dTemps[3]=dT_ijp1k_np1</code> is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$ , <code>dTemps[4]=dT_ijm1k_np1</code> is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$ , <code>dTemps[5]=dT_ijkp1_np1</code> is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$ , <code>dTemps[6]=dT_ijkm1_np1</code> is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

References `Time::dDeltat_np1half`, `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::nDonorCellFrac`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nE`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `Grid::nSinThetaIJK`, `Grid::nSinThetaIj1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `setMainFunctions()`.

### 6.12.2.92 `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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i> , <i>dTemps[0]=dT<sub>ijk_np1</sub></i>	is the temperature at radial position $(i, j, k)$ and time $n + 1$ , <i>dTemps[1]=dT<sub>ip1jk_np1</sub></i> is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$ , <i>dTemps[2]=dT<sub>im1jk_np1</sub></i> is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$ , <i>dTemps[3]=dT<sub>ijp1k_np1</sub></i> is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$ , <i>dTemps[4]=dT<sub>ijm1k_np1</sub></i> is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$ , <i>dTemps[5]=dT<sub>ijkp1_np1</sub></i> is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$ , <i>dTemps[6]=dT<sub>ijkm1_np1</sub></i> is the temperature at radial position $(i, j, k - 1)$ and time $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

References `Parameters::bDEDMClamp`, `Parameters::dDEDMClampMr`, `Parameters::dDEDMClampValue`, `Time::dDeltat_np1half`, `DEBUG_EQUATIONS`, `dET4()`, `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::nDonorCellFrac`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nE`, `Grid::nEddyVisc`, `Grid::nGlobalGridPositionLocalGrid`, `Grid::nM`, `Grid::nNumGhostCells`, `Grid::nQ0`, `Grid::nQ1`, `Grid::nQ2`, `Grid::nR`, `Grid::nSinThetaJK`, `Grid::nSinThetaJp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `setMainFunctions()`.

### 6.12.2.93 `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

in	<i>grid</i>	
in	<i>parameters</i>	
in	<i>time</i>	
in	<i>dTemps</i>	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 $n + 1$ .
in	<i>i</i>	is the radial index to evaluate the function at.
in	<i>j</i>	is the theta index to evaluate the function at.
in	<i>k</i>	is the phi index to evaluate the function at.

**Boundary Conditions** Missing  $\Delta M_r$  outside model using `Parameters::dAlpha` times  $\Delta M_r$  in the last zone instead.

**Boundary Conditions** missing density outside model assuming it is zero

**Boundary Conditions** missing desnity 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** A1 upwind set to zero as no material is flowing into the star

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

References `Parameters::bDEDMClamp`, `Parameters::dAlpha`, `Parameters::dDEDMClampMr`, `Parameters::dDEDMClampValue`, `Time::dDeltat_np1half`, `DEBUG_EQUATIONS`, `dET4()`, `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::nDonorCellFrac`, `Grid::nDPhi`, `Grid::nDTheta`, `Grid::nE`, `Grid::nEddyVisc`,

Grid::nGlobalGridPositionLocalGrid, Grid::nM, Grid::nNumGhostCells, Grid::nQ0, Grid::nQ1, Grid::nQ2, Grid::nR, Grid::nSinThetaJK, Grid::nSinThetaJp1halfK, Grid::nT, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

#### 6.12.2.94 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

in	grid	
in	parameters	
in	time	
in	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 $n + 1$ .
in	i	is the radial index to evaluate the function at.
in	j	is the theta index to evaluate the function at.
in	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\_np1half, 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::nDonorCellFrac, 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().

**6.12.2.95** `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.

Referenced by setMainFunctions().

**6.12.2.96** `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::dCurrentRelTErrors, Implicit::dDerivativeStepFraction, Grid::dLocalGridNew, Implicit::dMaxErrorlnRHS, Implicit::dTolerance, 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().

**6.12.2.97** `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 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 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`, `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()`.

**6.12.2.98** `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`, `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()`.

**6.12.2.99** `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 function is for 1D, gamma law calculations.

References Parameters::dDonorCellMin, Parameters::dDonorCellMultiplier, Parameters::dGamma, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

Referenced by initInternalVars().

#### 6.12.2.100 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 function is for 1D, tabulated equation of state calculations.

References Parameters::dDonorCellMin, Parameters::dDonorCellMultiplier, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nStartUpdateExplicit, Grid::nU, and Grid::nU0.

Referenced by initInternalVars().

#### 6.12.2.101 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 function is for 2D, gamma law calculations.

References Parameters::dDonorCellMin, Parameters::dDonorCellMultiplier, Parameters::dGamma, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by initInternalVars().

#### 6.12.2.102 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::dDonorCellMin, Parameters::dDonorCellMultiplier, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, - Grid::nGamma, Grid::nP, Grid::nQ0, Grid::nQ1, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by initInternalVars().

#### 6.12.2.103 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::dDonorCellMin, Parameters::dDonorCellMultiplier, Parameters::dGamma, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, 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().

#### 6.12.2.104 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::dDonorCellMin, Parameters::dDonorCellMultiplier, Grid::dLocalGridOld, Parameters::dMaxConvectiveVelocity, Grid::nCenIntOffset, Grid::nD, Grid::nDonorCellFrac, 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().

#### 6.12.2.105 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

in,out	<i>grid</i>	supplies information needed for initializing internal variables as well as storing the initialized internal variables
in	<i>procTop</i>	contains information about processor topology
in	<i>parameters</i>	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::nCotThetaJp1halfK, Grid::nDCosThetaIJK, Grid::nDPhi, Grid::nDTheta, Grid::nLocalGridDims, Grid::nNumDims, Grid::nNumGhostCells, Grid::nPhi, ProcTop::nRank, Grid::nSinThetaIJK, Grid::nSinThetaJp1halfK, Grid::nTheta, and Parameters::nTypeTurbulenceMod.

Referenced by init().

6.12.2.106 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

in,out	<i>grid</i>	supplies the information needed to calculate the horizontal density average, it also stores the calculated horizontally averaged density.
in	<i>parameters</i>	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::nCotThetaJp1halfK, Grid::nDCosThetaIJK, Grid::nDenAve, Grid::nDonorCellFrac, 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::nSinThetaJK, Grid::nSinThetaJp1halfK, Parameters::nTypeTurbulenceMod, and Grid::nVariables.

Referenced by modelRead().

**6.12.2.107 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

out	<i>functions</i>	is of class <a href="#">Functions</a> and is used to specify the functions called to calculate the evolution of the input model.
in	<i>procTop</i>	is of type <a href="#">ProcTop</a> . <a href="#">ProcTop::nRank</a> is used to set different functions based on processor rank. For instance processor rank 1 requires 1D versions of the equations.
in	<i>parameters</i>	is of class <a href="#">Parameters</a> . It holds various constants and run-time parameters.
in	<i>grid</i>	of type <a href="#">Grid</a> . This function requires the number of dimensions, specified by <a href="#">Grid::nNumDims</a> .
in	<i>time</i>	of type <a href="#">Time</a> . This function requires knowledge of the type of time setp being used, specified by <a href="#">Time::bVariableTimeStep</a> .
in	<i>implicit</i>	of type <a href="#">Implicit</a> . This function needs to know if there is an implicit region, specified when <a href="#">Implicit::nNumImplicitZones</a> >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\\_TEO\\_S\(\)](#), [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::fp](#)

CalculateNewAV, Functions::fpCalculateNewDensities, Functions::fpCalculateNewEddyVisc, Functions::fpCalculateNewEnergies, Functions::fpCalculateNewEOSVars, Functions::fpCalculateNewGridVelocities, Functions::fpCalculateNewRadii, Functions::fpCalculateNewVelocities, 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().

## 6.13 procTop.cpp File Reference

```
#include "procTop.h" #include <cstring>
```

### 6.13.1 Detailed Description

Implementation file for the [ProcTop](#) class

## 6.14 procTop.h File Reference

### Classes

- class [ProcTop](#)

### 6.14.1 Detailed Description

Header file for the [ProcTop](#) class

## 6.15 profileData.h File Reference

```
#include <string>    #include <vector>    #include <map> ×
#include <limits>    #include "time.h"    #include "procTop.h"
#include <fstream>
```

### Classes

- class [profileData](#)

### 6.15.1 Detailed Description

Header file for keepMax::cpp

## 6.16 time.cpp File Reference

```
#include "time.h" #include <limits>
```

### 6.16.1 Detailed Description

Implementation file for the [Time](#) class

## 6.17 time.h File Reference

### Classes

- class [Time](#)

### 6.17.1 Detailed Description

Header file for the [ProcTop](#) class

## 6.18 watchzone.cpp File Reference

```
#include "watchzone.h" #include "exception2.h" #include <sstream>
```

### 6.18.1 Detailed Description

This file holds the implementation of the watchzone class.

## 6.19 watchzone.h File Reference

```
#include <string> #include <fstream>
```

### Classes

- class [WatchZone](#)



### **6.19.1 Detailed Description**

This file holds the definition of the watchzone class.

# Index

- ~eos
  - eos, [17](#)
- /home/cgeroux/Documents/WORK/SPH-ERLS\_git/src/eos.cpp, [83](#)
- /home/cgeroux/Documents/WORK/SPH-ERLS\_git/src/eos.h, [83](#)
- DEBUG\_EQUATIONS
  - global.h, [113](#)
- DEDEM\_CLAMP
  - global.h, [113](#)
- DUMP\_VERSION
  - global.h, [113](#)
- Functions, [26](#)
  - Functions, [27](#)
  - fpCalculateAveDensities, [27](#)
  - fpCalculateDeltat, [27](#)
  - fpCalculateNewAV, [27](#)
  - fpCalculateNewDensities, [28](#)
  - fpCalculateNewEOSVars, [28](#)
  - fpCalculateNewEddyVisc, [28](#)
  - fpCalculateNewEnergies, [28](#)
  - fpCalculateNewGridVelocities, [28](#)
  - fpCalculateNewRadii, [28](#)
  - fpCalculateNewVelocities, [28](#)
  - fpImplicitSolve, [29](#)
  - fpModelWrite, [29](#)
  - fpUpdateLocalBoundaryVelocities-NewGrid, [29](#)
  - fpWriteWatchZones, [29](#)
  - Functions, [27](#)
- Global, [29](#)
  - Global, [30](#)
  - functions, [30](#)
  - Global, [30](#)
  - grid, [30](#)
  - implicit, [30](#)
  - messPass, [30](#)
  - output, [31](#)
  - parameters, [31](#)
  - performance, [31](#)
  - procTop, [31](#)
  - time, [31](#)
- Grid, [31](#)
  - Grid, [35](#)
  - dLocalGridNew, [36](#)
  - dLocalGridOld, [36](#)
  - Grid, [35](#)
  - nCenIntOffset, [37](#)
  - nCotThetaIJK, [38](#)
  - nCotThetaIjp1halfK, [38](#)
  - nD, [38](#)
  - nDCosThetaIJK, [39](#)
  - nDM, [40](#)
  - nDPhi, [40](#)
  - nDTheta, [41](#)
  - nDenAve, [39](#)
  - nDonorCellFrac, [40](#)
  - nE, [41](#)
  - nEddyVisc, [42](#)
  - nEndGhostUpdateExplicit, [42](#)
  - nEndGhostUpdateImplicit, [43](#)
  - nEndUpdateExplicit, [43](#)
  - nEndUpdateImplicit, [43](#)
  - nGamma, [44](#)
  - nGlobalGridDims, [44](#)
  - nGlobalGridPositionLocalGrid, [44](#)
  - nKappa, [44](#)
  - nLocalGridDims, [45](#)
  - nM, [45](#)
  - nNum1DZones, [45](#)
  - nNumDims, [45](#)
  - nNumGhostCells, [46](#)
  - nNumIntVars, [46](#)
  - nNumVars, [46](#)
  - nP, [46](#)
  - nPhi, [47](#)
  - nQ0, [47](#)
  - nQ1, [48](#)
  - nQ2, [48](#)
  - nR, [49](#)
  - nSinThetaIJK, [49](#)
  - nSinThetaIjp1halfK, [50](#)

- nStartGhostUpdateExplicit, [50](#)
- nStartGhostUpdateImplicit, [50](#)
- nStartUpdateExplicit, [51](#)
- nStartUpdateImplicit, [51](#)
- nT, [52](#)
- nTheta, [52](#)
- nU, [52](#)
- nU0, [53](#)
- nV, [53](#)
- nVariables, [54](#)
- nW, [54](#)
- Implicit, [55](#)
  - Implicit, [56](#)
  - dAverageRHS, [56](#)
  - dCurrentRelError, [56](#)
  - dDerivativeStepFraction, [56](#)
  - dMaxErrorInRHS, [56](#)
  - dTolerance, [56](#)
  - Implicit, [56](#)
  - kspContext, [57](#)
  - matCoeff, [57](#)
  - nCurrentNumIterations, [57](#)
  - nLocDer, [57](#)
  - nLocFun, [57](#)
  - nMaxNumIterations, [57](#)
  - nMaxNumSolverIterations, [58](#)
  - nNumDerPerRow, [58](#)
  - nNumImplicitZones, [58](#)
  - nNumRowsALocal, [58](#)
  - nNumRowsALocalSB, [58](#)
  - nTypeDer, [58](#)
  - vecRHS, [59](#)
  - vecTCorrections, [59](#)
  - vecTCorrectionsLocal, [59](#)
  - vecscatTCorrections, [59](#)
- MessPass, [59](#)
  - MessPass, [60](#)
  - MessPass, [60](#)
  - requestRecv, [60](#)
  - requestSend, [60](#)
  - statusRecv, [60](#)
  - statusSend, [61](#)
  - typeRecvNewVar, [61](#)
  - typeRecvOldGrid, [61](#)
  - typeSendNewGrid, [61](#)
  - typeSendNewVar, [61](#)
- Output, [62](#)
  - Output, [62](#)
  - bDump, [62](#)
  - bPrint, [63](#)
  - dDumpFrequencyTime, [63](#)
  - dPrintFrequencyTime, [63](#)
  - dTimeLastDump, [63](#)
  - dTimeLastPrint, [63](#)
  - nDumpFrequencyStep, [63](#)
  - nNumTimeStepsSinceLastPrint, [63](#)
  - nPrintFrequencyStep, [64](#)
  - nPrintMode, [64](#)
  - ofWatchZoneFiles, [64](#)
  - Output, [62](#)
  - sBaseOutputFileName, [64](#)
  - watchzoneList, [64](#)
- Parameters, [65](#)
  - Parameters, [66](#)
  - bAdiabatic, [66](#)
  - bDEDMClamp, [66](#)
  - bEOSGammaLaw, [66](#)
  - dA, [66](#)
  - dAVThreshold, [67](#)
  - dAlpha, [66](#)
  - dDEDMClampMr, [67](#)
  - dDEDMClampValue, [67](#)
  - dDonorCellMin, [67](#)
  - dDonorCellMultiplier, [68](#)
  - dEDMClampTemperature, [68](#)
  - dEddyViscosity, [68](#)
  - dG, [68](#)
  - dGamma, [68](#)
  - dMaxConvectiveVelocity, [69](#)
  - dMaxConvectiveVelocity\_c, [69](#)
  - dPi, [69](#)
  - dPrt, [69](#)
  - dSigma, [70](#)
  - dTolerance, [70](#)
  - eosTable, [70](#)
  - nMaxIterations, [70](#)
  - nTypeTurbulenceMod, [70](#)
  - Parameters, [66](#)
  - sDebugProfileOutput, [71](#)
  - sEOSFileName, [71](#)
- Performance, [71](#)
  - Performance, [72](#)
  - dEndTimer, [72](#)
  - dStartTimer, [72](#)
  - Performance, [72](#)
- ProcTop, [72](#)
  - ProcTop, [73](#)
  - nCoords, [73](#)
  - nNeighborRanks, [73](#)
  - nNumNeighbors, [73](#)

- nNumProcs, [73](#)
- nNumRadialNeighbors, [74](#)
- nPeriodic, [74](#)
- nProcDims, [74](#)
- nRadialNeighborNeighborIDs, [74](#)
- nRadialNeighborRanks, [74](#)
- nRank, [75](#)
- ProcTop, [73](#)
- SED OV
  - global.h, [113](#)
- SIGNEG DEN
  - global.h, [113](#)
- SIGNEG ENG
  - global.h, [113](#)
- SIGNEG TEMP
  - global.h, [113](#)
- TRACKMAXSOLVERERROR
  - global.h, [114](#)
- Time, [77](#)
  - Time, [78](#)
  - bVariableTimeStep, [78](#)
  - dConstTimeStep, [78](#)
  - dDelE\_t\_E\_max, [78](#)
  - dDelRho\_t\_Rho\_max, [79](#)
  - dDelT\_t\_T\_max, [79](#)
  - dDeltat\_n, [79](#)
  - dDeltat\_nm1half, [79](#)
  - dDeltat\_np1half, [79](#)
  - dEndTime, [80](#)
  - dPerChange, [80](#)
  - dTimeStepFactor, [80](#)
  - dt, [80](#)
  - nEndTimeStep, [81](#)
  - nTimeStepIndex, [81](#)
  - Time, [78](#)
- VISCOUS\_ENERGY\_EQ
  - global.h, [114](#)
- WatchZone, [81](#)
- average3DTo1DBoundariesNew
  - dataManipulation.cpp, [85](#)
  - dataManipulation.h, [95](#)
- average3DTo1DBoundariesOld
  - dataManipulation.cpp, [85](#)
  - dataManipulation.h, [95](#)
- bAdiabatic
  - Parameters, [66](#)
- bDEDMClamp
  - Parameters, [66](#)
- bDump
  - Output, [62](#)
- bEOSGammaLaw
  - Parameters, [66](#)
- bPrint
  - Output, [63](#)
- bVariableTimeStep
  - Time, [78](#)
- calDelt\_CONST
  - physEquations.cpp, [122](#)
  - physEquations.h, [184](#)
- calDelt\_R\_GL
  - physEquations.cpp, [122](#)
  - physEquations.h, [184](#)
- calDelt\_R\_TEOS
  - physEquations.cpp, [123](#)
  - physEquations.h, [184](#)
- calDelt\_RT\_GL
  - physEquations.cpp, [123](#)
  - physEquations.h, [185](#)
- calDelt\_RT\_TEOS
  - physEquations.cpp, [124](#)
  - physEquations.h, [185](#)
- calDelt\_RTP\_GL
  - physEquations.cpp, [125](#)
  - physEquations.h, [186](#)
- calDelt\_RTP\_TEOS
  - physEquations.cpp, [125](#)
  - physEquations.h, [187](#)
- calNewD\_R
  - physEquations.cpp, [126](#)
  - physEquations.h, [187](#)
- calNewD\_RT
  - physEquations.cpp, [126](#)
  - physEquations.h, [188](#)
- calNewD\_RTP
  - physEquations.cpp, [127](#)
  - physEquations.h, [188](#)
- calNewDenave\_None
  - physEquations.cpp, [127](#)
  - physEquations.h, [189](#)
- calNewDenave\_R
  - physEquations.cpp, [127](#)
  - physEquations.h, [189](#)
- calNewDenave\_RT
  - physEquations.cpp, [128](#)
  - physEquations.h, [189](#)
- calNewDenave\_RTP
  - physEquations.cpp, [128](#)

- physEquations.h, 190
- calNewE\_R\_AD
  - physEquations.cpp, 129
  - physEquations.h, 190
- calNewE\_R\_NA
  - physEquations.cpp, 129
  - physEquations.h, 190
- calNewE\_R\_NA\_LES
  - physEquations.cpp, 130
  - physEquations.h, 191
- calNewE\_RT\_AD
  - physEquations.cpp, 130
  - physEquations.h, 192
- calNewE\_RT\_NA
  - physEquations.cpp, 131
  - physEquations.h, 192
- calNewE\_RT\_NA\_LES
  - physEquations.cpp, 132
  - physEquations.h, 193
- calNewE\_RTP\_AD
  - physEquations.cpp, 132
  - physEquations.h, 194
- calNewE\_RTP\_NA
  - physEquations.cpp, 133
  - physEquations.h, 195
- calNewE\_RTP\_NA\_LES
  - physEquations.cpp, 134
  - physEquations.h, 195
- calNewEddyVisc\_None
  - physEquations.cpp, 135
  - physEquations.h, 196
- calNewEddyVisc\_R\_CN
  - physEquations.cpp, 135
  - physEquations.h, 197
- calNewEddyVisc\_R\_SM
  - physEquations.cpp, 135
  - physEquations.h, 197
- calNewEddyVisc\_RT\_CN
  - physEquations.cpp, 136
  - physEquations.h, 197
- calNewEddyVisc\_RT\_SM
  - physEquations.cpp, 136
  - physEquations.h, 198
- calNewEddyVisc\_RTP\_CN
  - physEquations.cpp, 136
  - physEquations.h, 198
- calNewEddyVisc\_RTP\_SM
  - physEquations.cpp, 137
  - physEquations.h, 198
- calNewP\_GL
  - physEquations.cpp, 137
  - physEquations.h, 199
- calNewPEKappaGamma\_TEOS
  - physEquations.cpp, 137
  - physEquations.h, 199
- calNewQ0\_R\_GL
  - physEquations.cpp, 138
  - physEquations.h, 199
- calNewQ0\_R\_TEOS
  - physEquations.cpp, 138
  - physEquations.h, 200
- calNewQ0Q1\_RT\_GL
  - physEquations.cpp, 139
  - physEquations.h, 200
- calNewQ0Q1\_RT\_TEOS
  - physEquations.cpp, 139
  - physEquations.h, 201
- calNewQ0Q1Q2\_RTP\_GL
  - physEquations.cpp, 140
  - physEquations.h, 201
- calNewQ0Q1Q2\_RTP\_TEOS
  - physEquations.cpp, 140
  - physEquations.h, 202
- calNewR
  - physEquations.cpp, 140
  - physEquations.h, 202
- calNewTPKappaGamma\_TEOS
  - physEquations.cpp, 141
  - physEquations.h, 202
- calNewU0\_R
  - physEquations.cpp, 141
  - physEquations.h, 203
- calNewU0\_RT
  - physEquations.cpp, 142
  - physEquations.h, 203
- calNewU0\_RTP
  - physEquations.cpp, 142
  - physEquations.h, 204
- calNewU\_R
  - physEquations.cpp, 143
  - physEquations.h, 205
- calNewU\_R\_LES
  - physEquations.cpp, 144
  - physEquations.h, 205
- calNewU\_RT
  - physEquations.cpp, 144
  - physEquations.h, 206
- calNewU\_RT\_LES
  - physEquations.cpp, 145
  - physEquations.h, 207

- calNewU\_RTP
  - physEquations.cpp, [146](#)
  - physEquations.h, [208](#)
- calNewU\_RTP\_LES
  - physEquations.cpp, [147](#)
  - physEquations.h, [209](#)
- calNewV\_RT
  - physEquations.cpp, [148](#)
  - physEquations.h, [210](#)
- calNewV\_RT\_LES
  - physEquations.cpp, [149](#)
  - physEquations.h, [211](#)
- calNewV\_RTP
  - physEquations.cpp, [149](#)
  - physEquations.h, [211](#)
- calNewV\_RTP\_LES
  - physEquations.cpp, [150](#)
  - physEquations.h, [212](#)
- calNewVelocities\_R
  - physEquations.cpp, [151](#)
  - physEquations.h, [213](#)
- calNewVelocities\_R\_LES
  - physEquations.cpp, [151](#)
  - physEquations.h, [213](#)
- calNewVelocities\_RT
  - physEquations.cpp, [151](#)
  - physEquations.h, [213](#)
- calNewVelocities\_RT\_LES
  - physEquations.cpp, [152](#)
  - physEquations.h, [214](#)
- calNewVelocities\_RTP
  - physEquations.cpp, [152](#)
  - physEquations.h, [214](#)
- calNewVelocities\_RTP\_LES
  - physEquations.cpp, [153](#)
  - physEquations.h, [215](#)
- calNewW\_RTP
  - physEquations.cpp, [153](#)
  - physEquations.h, [215](#)
- calNewW\_RTP\_LES
  - physEquations.cpp, [154](#)
  - physEquations.h, [216](#)
- calOldDenave\_None
  - physEquations.cpp, [155](#)
  - physEquations.h, [216](#)
- calOldDenave\_R
  - physEquations.cpp, [155](#)
  - physEquations.h, [217](#)
- calOldDenave\_RT
  - physEquations.cpp, [155](#)
  - physEquations.h, [217](#)
- calOldDenave\_RTP
  - physEquations.cpp, [155](#)
  - physEquations.h, [217](#)
- calOldEddyVisc\_R\_CN
  - physEquations.cpp, [156](#)
  - physEquations.h, [218](#)
- calOldEddyVisc\_R\_SM
  - physEquations.cpp, [156](#)
  - physEquations.h, [218](#)
- calOldEddyVisc\_RT\_CN
  - physEquations.cpp, [156](#)
  - physEquations.h, [218](#)
- calOldEddyVisc\_RT\_SM
  - physEquations.cpp, [157](#)
  - physEquations.h, [219](#)
- calOldEddyVisc\_RTP\_CN
  - physEquations.cpp, [157](#)
  - physEquations.h, [219](#)
- calOldEddyVisc\_RTP\_SM
  - physEquations.cpp, [158](#)
  - physEquations.h, [219](#)
- calOldP\_GL
  - physEquations.cpp, [158](#)
  - physEquations.h, [220](#)
- calOldPEKappaGamma\_TEOS
  - physEquations.cpp, [158](#)
  - physEquations.h, [220](#)
- calOldQ0\_R\_GL
  - physEquations.cpp, [159](#)
  - physEquations.h, [221](#)
- calOldQ0\_R\_TEOS
  - physEquations.cpp, [159](#)
  - physEquations.h, [221](#)
- calOldQ0Q1\_RT\_GL
  - physEquations.cpp, [160](#)
  - physEquations.h, [222](#)
- calOldQ0Q1\_RT\_TEOS
  - physEquations.cpp, [160](#)
  - physEquations.h, [222](#)
- calOldQ0Q1Q2\_RTP\_GL
  - physEquations.cpp, [161](#)
  - physEquations.h, [222](#)
- calOldQ0Q1Q2\_RTP\_TEOS
  - physEquations.cpp, [161](#)
  - physEquations.h, [223](#)
- clear
  - profileData, [76](#)
- dA

- Parameters, [66](#)
- dAVThreshold
  - Parameters, [67](#)
- dAlpha
  - Parameters, [66](#)
- dAverageRHS
  - Implicit, [56](#)
- dConstTimeStep
  - Time, [78](#)
- dCurrentRelTError
  - Implicit, [56](#)
- dDEDMClampMr
  - Parameters, [67](#)
- dDEDMClampValue
  - Parameters, [67](#)
- dDRhoDP
  - eos, [17](#)
- dDelE\_t\_E\_max
  - Time, [78](#)
- dDelRho\_t\_Rho\_max
  - Time, [79](#)
- dDelT\_t\_T\_max
  - Time, [79](#)
- dDeltat\_n
  - Time, [79](#)
- dDeltat\_nm1half
  - Time, [79](#)
- dDeltat\_np1half
  - Time, [79](#)
- dDerivativeStepFraction
  - Implicit, [56](#)
- dDonorCellMin
  - Parameters, [67](#)
- dDonorCellMultiplier
  - Parameters, [68](#)
- dDumpFrequencyTime
  - Output, [63](#)
- dEDMClampTemperature
  - Parameters, [68](#)
- dEOS\_GL
  - physEquations.cpp, [162](#)
  - physEquations.h, [223](#)
- dET4
  - physEquations.cpp, [162](#)
  - physEquations.h, [224](#)
- dEddyViscosity
  - Parameters, [68](#)
- dEndTime
  - Time, [80](#)
- dEndTimer
  - Performance, [72](#)
- dG
  - Parameters, [68](#)
- dGamma
  - Parameters, [68](#)
- dGetEnergy
  - eos, [17](#)
- dGetOpacity
  - eos, [18](#)
- dGetPressure
  - eos, [18](#)
- dImplicitEnergyFunction\_None
  - physEquations.cpp, [162](#)
  - physEquations.h, [224](#)
- dImplicitEnergyFunction\_R
  - physEquations.cpp, [162](#)
  - physEquations.h, [224](#)
- dImplicitEnergyFunction\_R\_LES
  - physEquations.cpp, [163](#)
  - physEquations.h, [225](#)
- dImplicitEnergyFunction\_R\_LES\_SB
  - physEquations.cpp, [164](#)
  - physEquations.h, [226](#)
- dImplicitEnergyFunction\_R\_SB
  - physEquations.cpp, [164](#)
  - physEquations.h, [226](#)
- dImplicitEnergyFunction\_RT
  - physEquations.cpp, [165](#)
  - physEquations.h, [227](#)
- dImplicitEnergyFunction\_RT\_LES
  - physEquations.cpp, [166](#)
  - physEquations.h, [228](#)
- dImplicitEnergyFunction\_RT\_LES\_SB
  - physEquations.cpp, [167](#)
  - physEquations.h, [229](#)
- dImplicitEnergyFunction\_RT\_SB
  - physEquations.cpp, [168](#)
  - physEquations.h, [230](#)
- dImplicitEnergyFunction\_RTP
  - physEquations.cpp, [169](#)
  - physEquations.h, [231](#)
- dImplicitEnergyFunction\_RTP\_LES
  - physEquations.cpp, [170](#)
  - physEquations.h, [231](#)
- dImplicitEnergyFunction\_RTP\_LES\_SB
  - physEquations.cpp, [171](#)
  - physEquations.h, [232](#)
- dImplicitEnergyFunction\_RTP\_SB
  - physEquations.cpp, [173](#)
  - physEquations.h, [234](#)

- dLocalGridNew
  - Grid, [36](#)
- dLocalGridOld
  - Grid, [36](#)
- dLogE
  - eos, [24](#)
- dLogKappa
  - eos, [24](#)
- dLogP
  - eos, [24](#)
- dLogRhoDelta
  - eos, [24](#)
- dLogRhoMin
  - eos, [24](#)
- dLogTDelta
  - eos, [25](#)
- dLogTMin
  - eos, [25](#)
- dMaxConvectiveVelocity
  - Parameters, [69](#)
- dMaxConvectiveVelocity\_c
  - Parameters, [69](#)
- dMaxErrorInRHS
  - Implicit, [56](#)
- dPerChange
  - Time, [80](#)
- dPi
  - Parameters, [69](#)
- dPrintFrequencyTime
  - Output, [63](#)
- dPrt
  - Parameters, [69](#)
- dSigma
  - Parameters, [70](#)
- dSoundSpeed
  - eos, [19](#)
- dStartTimer
  - Performance, [72](#)
- dTimeLastDump
  - Output, [63](#)
- dTimeLastPrint
  - Output, [63](#)
- dTimeStepFactor
  - Time, [80](#)
- dTolerance
  - Implicit, [56](#)
  - Parameters, [70](#)
- dXMassFrac
  - eos, [25](#)
- dYMassFrac
  - eos, [25](#)
- dataManipulation.cpp, [84](#)
  - average3DTo1DBoundariesNew, [85](#)
  - average3DTo1DBoundariesOld, [85](#)
  - fin, [85](#)
  - init, [86](#)
  - initImplicitCalculation, [87](#)
  - initUpdateLocalBoundaries, [88](#)
  - modelRead, [89](#)
  - modelWrite\_GL, [89](#)
  - modelWrite\_TEOS, [90](#)
  - setDEDMClamp, [90](#)
  - setupLocalGrid, [91](#)
  - updateLocalBoundaries, [91](#)
  - updateLocalBoundariesNewGrid, [92](#)
  - updateLocalBoundaryVelocitiesNew-Grid\_R, [92](#)
  - updateLocalBoundaryVelocitiesNew-Grid\_RT, [92](#)
  - updateLocalBoundaryVelocitiesNew-Grid\_RTP, [93](#)
  - updateNewGridWithOld, [93](#)
  - updateOldGrid, [93](#)
- dataManipulation.h, [94](#)
  - average3DTo1DBoundariesNew, [95](#)
  - average3DTo1DBoundariesOld, [95](#)
  - fin, [95](#)
  - init, [96](#)
  - initImplicitCalculation, [97](#)
  - initUpdateLocalBoundaries, [98](#)
  - modelRead, [99](#)
  - modelWrite\_GL, [99](#)
  - modelWrite\_TEOS, [100](#)
  - setDEDMClamp, [100](#)
  - setupLocalGrid, [100](#)
  - updateLocalBoundaries, [101](#)
  - updateLocalBoundariesNewGrid, [101](#)
  - updateLocalBoundaryVelocitiesNew-Grid\_R, [102](#)
  - updateLocalBoundaryVelocitiesNew-Grid\_RT, [102](#)
  - updateLocalBoundaryVelocitiesNew-Grid\_RTP, [102](#)
  - updateNewGridWithOld, [103](#)
  - updateOldGrid, [103](#)
- dataMonitoring.cpp, [103](#)
  - finWatchZones, [104](#)
  - initWatchZones, [104](#)
  - writeWatchZones\_R\_GL, [105](#)



- writeWatchZones\_R\_TEOS, [105](#)
- writeWatchZones\_RT\_GL, [106](#)
- writeWatchZones\_RT\_TEOS, [106](#)
- writeWatchZones\_RTP\_GL, [106](#)
- writeWatchZones\_RTP\_TEOS, [107](#)
- dataMonitoring.h, [107](#)
  - finWatchZones, [108](#)
  - initWatchZones, [108](#)
  - writeWatchZones\_R\_GL, [109](#)
  - writeWatchZones\_R\_TEOS, [109](#)
  - writeWatchZones\_RT\_GL, [110](#)
  - writeWatchZones\_RT\_TEOS, [110](#)
  - writeWatchZones\_RTP\_GL, [110](#)
  - writeWatchZones\_RTP\_TEOS, [111](#)
- dt
  - Time, [80](#)
- eos, [15](#)
  - ~eos, [17](#)
  - dDRhoDP, [17](#)
  - dGetEnergy, [17](#)
  - dGetOpacity, [18](#)
  - dGetPressure, [18](#)
  - dLogE, [24](#)
  - dLogKappa, [24](#)
  - dLogP, [24](#)
  - dLogRhoDelta, [24](#)
  - dLogRhoMin, [24](#)
  - dLogTDelta, [25](#)
  - dLogTMin, [25](#)
  - dSoundSpeed, [19](#)
  - dXMassFrac, [25](#)
  - dYMassFrac, [25](#)
  - eos, [16](#)
  - gamma1DelAdC\_v, [19](#)
  - getDlnPDlnTDlnPDlnPDEDT, [19](#)
  - getEAndDTDE, [20](#)
  - getEKappa, [20](#)
  - getPAndDRhoDP, [20](#)
  - getPEKappa, [21](#)
  - getPEKappaGamma, [21](#)
  - getPEKappaGammaCp, [21](#)
  - getPKappaGamma, [22](#)
  - nNumRho, [25](#)
  - nNumT, [26](#)
  - operator=, [22](#)
  - readAscii, [22](#)
  - readBin, [23](#)
  - readBobsAscii, [23](#)
  - writeAscii, [23](#)
  - writeBin, [23](#)
- eosTable
  - Parameters, [70](#)
- exception2, [26](#)
- fin
  - dataManipulation.cpp, [85](#)
  - dataManipulation.h, [95](#)
- finWatchZones
  - dataMonitoring.cpp, [104](#)
  - dataMonitoring.h, [108](#)
- fpCalculateAveDensities
  - Functions, [27](#)
- fpCalculateDeltat
  - Functions, [27](#)
- fpCalculateNewAV
  - Functions, [27](#)
- fpCalculateNewDensities
  - Functions, [28](#)
- fpCalculateNewEOSVars
  - Functions, [28](#)
- fpCalculateNewEddyVisc
  - Functions, [28](#)
- fpCalculateNewEnergies
  - Functions, [28](#)
- fpCalculateNewGridVelocities
  - Functions, [28](#)
- fpCalculateNewRadii
  - Functions, [28](#)
- fpCalculateNewVelocities
  - Functions, [28](#)
- fpImplicitSolve
  - Functions, [29](#)
- fpModelWrite
  - Functions, [29](#)
- fpUpdateLocalBoundaryVelocitiesNew-Grid
  - Functions, [29](#)
- fpWriteWatchZones
  - Functions, [29](#)
- functions
  - Global, [30](#)
- gamma1DelAdC\_v
  - eos, [19](#)
- getDlnPDlnTDlnPDlnPDEDT
  - eos, [19](#)
- getEAndDTDE
  - eos, [20](#)
- getEKappa

- eos, 20
- getPAndDRhoDP
  - eos, 20
- getPEKappa
  - eos, 21
- getPEKappaGamma
  - eos, 21
- getPEKappaGammaCp
  - eos, 21
- getPKappaGamma
  - eos, 22
- global.cpp, 111
- global.h, 112
  - DEBUG\_EQUATIONS, 113
  - DEDEM\_CLAMP, 113
  - DUMP\_VERSION, 113
  - SEDOV, 113
  - SIGNEGDEN, 113
  - SIGNEGENG, 113
  - SIGNEGTEMP, 113
  - TRACKMAXSOLVERERROR, 114
  - VISCOUS\_ENERGY\_EQ, 114
- grid
  - Global, 30
- implicit
  - Global, 30
- implicitSolve\_None
  - physEquations.cpp, 174
  - physEquations.h, 235
- implicitSolve\_R
  - physEquations.cpp, 174
  - physEquations.h, 235
- implicitSolve\_RT
  - physEquations.cpp, 174
  - physEquations.h, 235
- implicitSolve\_RTP
  - physEquations.cpp, 175
  - physEquations.h, 236
- init
  - dataManipulation.cpp, 86
  - dataManipulation.h, 96
- initDonorFracAndMaxConVel\_R\_GL
  - physEquations.cpp, 175
  - physEquations.h, 236
- initDonorFracAndMaxConVel\_R\_TEOS
  - physEquations.cpp, 176
  - physEquations.h, 237
- initDonorFracAndMaxConVel\_RT\_GL
  - physEquations.cpp, 176
- physEquations.h, 237
- initDonorFracAndMaxConVel\_RT\_TEOS
  - physEquations.cpp, 176
  - physEquations.h, 237
- initDonorFracAndMaxConVel\_RTP\_GL
  - physEquations.cpp, 177
  - physEquations.h, 238
- initDonorFracAndMaxConVel\_RTP\_TEO-S
  - physEquations.cpp, 177
  - physEquations.h, 238
- initImplicitCalculation
  - dataManipulation.cpp, 87
  - dataManipulation.h, 97
- initInternalVars
  - physEquations.cpp, 177
  - physEquations.h, 238
- initUpdateLocalBoundaries
  - dataManipulation.cpp, 88
  - dataManipulation.h, 98
- initWatchZones
  - dataMonitoring.cpp, 104
  - dataMonitoring.h, 108
- kspContext
  - Implicit, 57
- main
  - main.cpp, 114
  - main.h, 116
- main.cpp, 114
  - main, 114
  - signalHandler, 116
- main.h, 116
  - main, 116
  - signalHandler, 118
- matCoeff
  - Implicit, 57
- messPass
  - Global, 30
- modelRead
  - dataManipulation.cpp, 89
  - dataManipulation.h, 99
- modelWrite\_GL
  - dataManipulation.cpp, 89
  - dataManipulation.h, 99
- modelWrite\_TEOS
  - dataManipulation.cpp, 90
  - dataManipulation.h, 100
- nCenIntOffset

- Grid, [37](#)
- nCoords
  - ProcTop, [73](#)
- nCotThetaJK
  - Grid, [38](#)
- nCotThetaJp1halfK
  - Grid, [38](#)
- nCurrentNumIterations
  - Implicit, [57](#)
- nD
  - Grid, [38](#)
- nDCosThetaJK
  - Grid, [39](#)
- nDM
  - Grid, [40](#)
- nDPhi
  - Grid, [40](#)
- nDTheta
  - Grid, [41](#)
- nDenAve
  - Grid, [39](#)
- nDonorCellFrac
  - Grid, [40](#)
- nDumpFrequencyStep
  - Output, [63](#)
- nE
  - Grid, [41](#)
- nEddyVisc
  - Grid, [42](#)
- nEndGhostUpdateExplicit
  - Grid, [42](#)
- nEndGhostUpdateImplicit
  - Grid, [43](#)
- nEndTimeStep
  - Time, [81](#)
- nEndUpdateExplicit
  - Grid, [43](#)
- nEndUpdateImplicit
  - Grid, [43](#)
- nGamma
  - Grid, [44](#)
- nGlobalGridDims
  - Grid, [44](#)
- nGlobalGridPositionLocalGrid
  - Grid, [44](#)
- nKappa
  - Grid, [44](#)
- nLocDer
  - Implicit, [57](#)
- nLocFun
  - Implicit, [57](#)
- nLocalGridDims
  - Grid, [45](#)
- nM
  - Grid, [45](#)
- nMaxIterations
  - Parameters, [70](#)
- nMaxNumIterations
  - Implicit, [57](#)
- nMaxNumSolverIterations
  - Implicit, [58](#)
- nMaxNumZones
  - profileData, [76](#)
- nNeighborRanks
  - ProcTop, [73](#)
- nNum1DZones
  - Grid, [45](#)
- nNumDerPerRow
  - Implicit, [58](#)
- nNumDims
  - Grid, [45](#)
- nNumGhostCells
  - Grid, [46](#)
- nNumImplicitZones
  - Implicit, [58](#)
- nNumIntVars
  - Grid, [46](#)
- nNumNeighbors
  - ProcTop, [73](#)
- nNumProcs
  - ProcTop, [73](#)
- nNumRadialNeighbors
  - ProcTop, [74](#)
- nNumRho
  - eos, [25](#)
- nNumRowsALocal
  - Implicit, [58](#)
- nNumRowsALocalSB
  - Implicit, [58](#)
- nNumT
  - eos, [26](#)
- nNumTimeStepsSinceLastPrint
  - Output, [63](#)
- nNumVars
  - Grid, [46](#)
- nP
  - Grid, [46](#)
- nPeriodic
  - ProcTop, [74](#)
- nPhi

- Grid, [47](#)
- nPrintFrequencyStep
  - Output, [64](#)
- nPrintMode
  - Output, [64](#)
- nProcDims
  - ProcTop, [74](#)
- nQ0
  - Grid, [47](#)
- nQ1
  - Grid, [48](#)
- nQ2
  - Grid, [48](#)
- nR
  - Grid, [49](#)
- nRadialNeighborNeighborIDs
  - ProcTop, [74](#)
- nRadialNeighborRanks
  - ProcTop, [74](#)
- nRank
  - ProcTop, [75](#)
- nSinThetaIJK
  - Grid, [49](#)
- nSinThetaIp1halfK
  - Grid, [50](#)
- nStartGhostUpdateExplicit
  - Grid, [50](#)
- nStartGhostUpdateImplicit
  - Grid, [50](#)
- nStartUpdateExplicit
  - Grid, [51](#)
- nStartUpdateImplicit
  - Grid, [51](#)
- nT
  - Grid, [52](#)
- nTheta
  - Grid, [52](#)
- nTimeStepIndex
  - Time, [81](#)
- nTypeDer
  - Implicit, [58](#)
- nTypeTurbulenceMod
  - Parameters, [70](#)
- nU
  - Grid, [52](#)
- nU0
  - Grid, [53](#)
- nV
  - Grid, [53](#)
- nVariables
  - Grid, [54](#)
- nW
  - Grid, [54](#)
- ofWatchZoneFiles
  - Output, [64](#)
- operator=
  - eos, [22](#)
- output
  - Global, [31](#)
- parameters
  - Global, [31](#)
- performance
  - Global, [31](#)
- physEquations.cpp, [118](#)
  - calDelt\_CONST, [122](#)
  - calDelt\_R\_GL, [122](#)
  - calDelt\_R\_TEOS, [123](#)
  - calDelt\_RT\_GL, [123](#)
  - calDelt\_RT\_TEOS, [124](#)
  - calDelt\_RTP\_GL, [125](#)
  - calDelt\_RTP\_TEOS, [125](#)
  - calNewD\_R, [126](#)
  - calNewD\_RT, [126](#)
  - calNewD\_RTP, [127](#)
  - calNewDenave\_None, [127](#)
  - calNewDenave\_R, [127](#)
  - calNewDenave\_RT, [128](#)
  - calNewDenave\_RTP, [128](#)
  - calNewE\_R\_AD, [129](#)
  - calNewE\_R\_NA, [129](#)
  - calNewE\_R\_NA\_LES, [130](#)
  - calNewE\_RT\_AD, [130](#)
  - calNewE\_RT\_NA, [131](#)
  - calNewE\_RT\_NA\_LES, [132](#)
  - calNewE\_RTP\_AD, [132](#)
  - calNewE\_RTP\_NA, [133](#)
  - calNewE\_RTP\_NA\_LES, [134](#)
  - calNewEddyVisc\_None, [135](#)
  - calNewEddyVisc\_R\_CN, [135](#)
  - calNewEddyVisc\_R\_SM, [135](#)
  - calNewEddyVisc\_RT\_CN, [136](#)
  - calNewEddyVisc\_RT\_SM, [136](#)
  - calNewEddyVisc\_RTP\_CN, [136](#)
  - calNewEddyVisc\_RTP\_SM, [137](#)
  - calNewP\_GL, [137](#)
  - calNewPEKappaGamma\_TEOS, [137](#)
  - calNewQ0\_R\_GL, [138](#)

- calNewQ0\_R\_TEOS, [138](#)
- calNewQ0Q1\_RT\_GL, [139](#)
- calNewQ0Q1\_RT\_TEOS, [139](#)
- calNewQ0Q1Q2\_RTP\_GL, [140](#)
- calNewQ0Q1Q2\_RTP\_TEOS, [140](#)
- calNewR, [140](#)
- calNewTPKappaGamma\_TEOS, [141](#)
- calNewU0\_R, [141](#)
- calNewU0\_RT, [142](#)
- calNewU0\_RTP, [142](#)
- calNewU\_R, [143](#)
- calNewU\_R\_LES, [144](#)
- calNewU\_RT, [144](#)
- calNewU\_RT\_LES, [145](#)
- calNewU\_RTP, [146](#)
- calNewU\_RTP\_LES, [147](#)
- calNewV\_RT, [148](#)
- calNewV\_RT\_LES, [149](#)
- calNewV\_RTP, [149](#)
- calNewV\_RTP\_LES, [150](#)
- calNewVelocities\_R, [151](#)
- calNewVelocities\_R\_LES, [151](#)
- calNewVelocities\_RT, [151](#)
- calNewVelocities\_RT\_LES, [152](#)
- calNewVelocities\_RTP, [152](#)
- calNewVelocities\_RTP\_LES, [153](#)
- calNewW\_RTP, [153](#)
- calNewW\_RTP\_LES, [154](#)
- calOldDenave\_None, [155](#)
- calOldDenave\_R, [155](#)
- calOldDenave\_RT, [155](#)
- calOldDenave\_RTP, [155](#)
- calOldEddyVisc\_R\_CN, [156](#)
- calOldEddyVisc\_R\_SM, [156](#)
- calOldEddyVisc\_RT\_CN, [156](#)
- calOldEddyVisc\_RT\_SM, [157](#)
- calOldEddyVisc\_RTP\_CN, [157](#)
- calOldEddyVisc\_RTP\_SM, [158](#)
- calOldP\_GL, [158](#)
- calOldPEKappaGamma\_TEOS, [158](#)
- calOldQ0\_R\_GL, [159](#)
- calOldQ0\_R\_TEOS, [159](#)
- calOldQ0Q1\_RT\_GL, [160](#)
- calOldQ0Q1\_RT\_TEOS, [160](#)
- calOldQ0Q1Q2\_RTP\_GL, [161](#)
- calOldQ0Q1Q2\_RTP\_TEOS, [161](#)
- dEOS\_GL, [162](#)
- dET4, [162](#)
- dImplicitEnergyFunction\_None, [162](#)
- dImplicitEnergyFunction\_R, [162](#)
- dImplicitEnergyFunction\_R\_LES, [163](#)
- dImplicitEnergyFunction\_R\_LES\_SB, [164](#)
- dImplicitEnergyFunction\_R\_SB, [164](#)
- dImplicitEnergyFunction\_RT, [165](#)
- dImplicitEnergyFunction\_RT\_LES, [166](#)
- dImplicitEnergyFunction\_RT\_LES\_SB, [167](#)
- dImplicitEnergyFunction\_RT\_SB, [168](#)
- dImplicitEnergyFunction\_RTP, [169](#)
- dImplicitEnergyFunction\_RTP\_LES, [170](#)
- dImplicitEnergyFunction\_RTP\_LES\_SB, [171](#)
- dImplicitEnergyFunction\_RTP\_SB, [173](#)
- implicitSolve\_None, [174](#)
- implicitSolve\_R, [174](#)
- implicitSolve\_RT, [174](#)
- implicitSolve\_RTP, [175](#)
- initDonorFracAndMaxConVel\_R\_GL, [175](#)
- initDonorFracAndMaxConVel\_R\_TEOS, [176](#)
- initDonorFracAndMaxConVel\_RT\_GL, [176](#)
- initDonorFracAndMaxConVel\_RT\_TEOS, [176](#)
- initDonorFracAndMaxConVel\_RTP\_GL, [177](#)
- initDonorFracAndMaxConVel\_RTP\_TEOS, [177](#)
- initInternalVars, [177](#)
- setInternalVarInf, [178](#)
- setMainFunctions, [179](#)
- physEquations.h, [180](#)
- calDelt\_CONST, [184](#)
- calDelt\_R\_GL, [184](#)
- calDelt\_R\_TEOS, [184](#)
- calDelt\_RT\_GL, [185](#)
- calDelt\_RT\_TEOS, [185](#)
- calDelt\_RTP\_GL, [186](#)
- calDelt\_RTP\_TEOS, [187](#)
- calNewD\_R, [187](#)
- calNewD\_RT, [188](#)
- calNewD\_RTP, [188](#)

- calNewDenave\_None, 189
- calNewDenave\_R, 189
- calNewDenave\_RT, 189
- calNewDenave\_RTP, 190
- calNewE\_R\_AD, 190
- calNewE\_R\_NA, 190
- calNewE\_R\_NA\_LES, 191
- calNewE\_RT\_AD, 192
- calNewE\_RT\_NA, 192
- calNewE\_RT\_NA\_LES, 193
- calNewE\_RTP\_AD, 194
- calNewE\_RTP\_NA, 195
- calNewE\_RTP\_NA\_LES, 195
- calNewEddyVisc\_None, 196
- calNewEddyVisc\_R\_CN, 197
- calNewEddyVisc\_R\_SM, 197
- calNewEddyVisc\_RT\_CN, 197
- calNewEddyVisc\_RT\_SM, 198
- calNewEddyVisc\_RTP\_CN, 198
- calNewEddyVisc\_RTP\_SM, 198
- calNewP\_GL, 199
- calNewPEKappaGamma\_TEOS, 199
- calNewQ0\_R\_GL, 199
- calNewQ0\_R\_TEOS, 200
- calNewQ0Q1\_RT\_GL, 200
- calNewQ0Q1\_RT\_TEOS, 201
- calNewQ0Q1Q2\_RTP\_GL, 201
- calNewQ0Q1Q2\_RTP\_TEOS, 202
- calNewR, 202
- calNewTPKappaGamma\_TEOS, 202
- calNewU0\_R, 203
- calNewU0\_RT, 203
- calNewU0\_RTP, 204
- calNewU\_R, 205
- calNewU\_R\_LES, 205
- calNewU\_RT, 206
- calNewU\_RT\_LES, 207
- calNewU\_RTP, 208
- calNewU\_RTP\_LES, 209
- calNewV\_RT, 210
- calNewV\_RT\_LES, 211
- calNewV\_RTP, 211
- calNewV\_RTP\_LES, 212
- calNewVelocities\_R, 213
- calNewVelocities\_R\_LES, 213
- calNewVelocities\_RT, 213
- calNewVelocities\_RT\_LES, 214
- calNewVelocities\_RTP, 214
- calNewVelocities\_RTP\_LES, 215
- calNewW\_RTP, 215
- calNewW\_RTP\_LES, 216
- calOldDenave\_None, 216
- calOldDenave\_R, 217
- calOldDenave\_RT, 217
- calOldDenave\_RTP, 217
- calOldEddyVisc\_R\_CN, 218
- calOldEddyVisc\_R\_SM, 218
- calOldEddyVisc\_RT\_CN, 218
- calOldEddyVisc\_RT\_SM, 219
- calOldEddyVisc\_RTP\_CN, 219
- calOldEddyVisc\_RTP\_SM, 219
- calOldP\_GL, 220
- calOldPEKappaGamma\_TEOS, 220
- calOldQ0\_R\_GL, 221
- calOldQ0\_R\_TEOS, 221
- calOldQ0Q1\_RT\_GL, 222
- calOldQ0Q1\_RT\_TEOS, 222
- calOldQ0Q1Q2\_RTP\_GL, 222
- calOldQ0Q1Q2\_RTP\_TEOS, 223
- dEOS\_GL, 223
- dET4, 224
- dImplicitEnergyFunction\_None, 224
- dImplicitEnergyFunction\_R, 224
- dImplicitEnergyFunction\_R\_LES, 225
- dImplicitEnergyFunction\_R\_LES\_SB, 226
- dImplicitEnergyFunction\_R\_SB, 226
- dImplicitEnergyFunction\_RT, 227
- dImplicitEnergyFunction\_RT\_LES, 228
- dImplicitEnergyFunction\_RT\_LES\_SB, 229
- dImplicitEnergyFunction\_RT\_SB, 230
- dImplicitEnergyFunction\_RTP, 231
- dImplicitEnergyFunction\_RTP\_LES, 231
- dImplicitEnergyFunction\_RTP\_LES\_SB, 232
- dImplicitEnergyFunction\_RTP\_SB, 234
- implicitSolve\_None, 235
- implicitSolve\_R, 235
- implicitSolve\_RT, 235
- implicitSolve\_RTP, 236
- initDonorFracAndMaxConVel\_R\_GL, 236

- initDonorFracAndMaxConVel\_R\_TE-OS, [237](#)
- initDonorFracAndMaxConVel\_RT\_GL, [237](#)
- initDonorFracAndMaxConVel\_RT\_TEOS, [237](#)
- initDonorFracAndMaxConVel\_RTP\_GL, [238](#)
- initDonorFracAndMaxConVel\_RTP\_TEOS, [238](#)
- initInternalVars, [238](#)
- setInternalVarInf, [239](#)
- setMainFunctions, [240](#)
- procTop
  - Global, [31](#)
- procTop.cpp, [241](#)
- procTop.h, [241](#)
- profileData, [75](#)
  - clear, [76](#)
  - nMaxNumZones, [76](#)
  - profileData, [76](#)
  - profileData, [76](#)
  - set, [76](#)
  - setMax, [76](#)
  - setMaxAbs, [76](#)
  - setSum, [77](#)
  - test, [77](#)
  - toFile, [77](#)
- profileData.h, [241](#)
- readAscii
  - eos, [22](#)
- readBin
  - eos, [23](#)
- readBobsAscii
  - eos, [23](#)
- requestRecv
  - MessPass, [60](#)
- requestSend
  - MessPass, [60](#)
- sBaseOutputFileName
  - Output, [64](#)
- sDebugProfileOutput
  - Parameters, [71](#)
- sEOSFileName
  - Parameters, [71](#)
- set
  - profileData, [76](#)
- setDEDMClamp
  - dataManipulation.cpp, [90](#)
  - dataManipulation.h, [100](#)
- setInternalVarInf
  - physEquations.cpp, [178](#)
  - physEquations.h, [239](#)
- setMainFunctions
  - physEquations.cpp, [179](#)
  - physEquations.h, [240](#)
- setMax
  - profileData, [76](#)
- setMaxAbs
  - profileData, [76](#)
- setSum
  - profileData, [77](#)
- setupLocalGrid
  - dataManipulation.cpp, [91](#)
  - dataManipulation.h, [100](#)
- signalHandler
  - main.cpp, [116](#)
  - main.h, [118](#)
- statusRecv
  - MessPass, [60](#)
- statusSend
  - MessPass, [61](#)
- test
  - profileData, [77](#)
- time
  - Global, [31](#)
- time.cpp, [242](#)
- time.h, [242](#)
- toFile
  - profileData, [77](#)
- typeRecvNewVar
  - MessPass, [61](#)
- typeRecvOldGrid
  - MessPass, [61](#)
- typeSendNewGrid
  - MessPass, [61](#)
- typeSendNewVar
  - MessPass, [61](#)
- updateLocalBoundaries
  - dataManipulation.cpp, [91](#)
  - dataManipulation.h, [101](#)
- updateLocalBoundariesNewGrid
  - dataManipulation.cpp, [92](#)
  - dataManipulation.h, [101](#)
- updateLocalBoundaryVelocitiesNewGrid\_R

dataManipulation.cpp, [92](#)  
dataManipulation.h, [102](#)  
updateLocalBoundaryVelocitiesNewGrid-  
\_RT  
dataManipulation.cpp, [92](#)  
dataManipulation.h, [102](#)  
updateLocalBoundaryVelocitiesNewGrid-  
\_RTP  
dataManipulation.cpp, [93](#)  
dataManipulation.h, [102](#)  
updateNewGridWithOld  
dataManipulation.cpp, [93](#)  
dataManipulation.h, [103](#)  
updateOldGrid  
dataManipulation.cpp, [93](#)  
dataManipulation.h, [103](#)  
  
vecRHS  
Implicit, [59](#)  
vecTCorrections  
Implicit, [59](#)  
vecTCorrectionsLocal  
Implicit, [59](#)  
vecscatTCorrections  
Implicit, [59](#)  
  
watchzone.cpp, [242](#)  
watchzone.h, [242](#)  
watchzoneList  
Output, [64](#)  
writeAscii  
eos, [23](#)  
writeBin  
eos, [23](#)  
writeWatchZones\_R\_GL  
dataMonitoring.cpp, [105](#)  
dataMonitoring.h, [109](#)  
writeWatchZones\_R\_TEOS  
dataMonitoring.cpp, [105](#)  
dataMonitoring.h, [109](#)  
writeWatchZones\_RT\_GL  
dataMonitoring.cpp, [106](#)  
dataMonitoring.h, [110](#)  
writeWatchZones\_RT\_TEOS  
dataMonitoring.cpp, [106](#)  
dataMonitoring.h, [110](#)  
writeWatchZones\_RTP\_GL  
dataMonitoring.cpp, [106](#)  
dataMonitoring.h, [110](#)  
writeWatchZones\_RTP\_TEOS