

# SPHERLS

1.0

Generated by Doxygen 1.7.6.1

Fri Dec 7 2012 11:29:05



# Contents

<b>1</b>	<b>Using and Modifying SPHERLSanal</b>	<b>1</b>
1.1	Overview . . . . .	1
1.1.1	The Basics . . . . .	1
1.2	Compiling SPHERLS . . . . .	2
1.2.1	Requirements . . . . .	3
1.2.2	Installing PETSC Library . . . . .	3
1.2.3	Installing FFTW Library . . . . .	4
1.3	Using SPHERLS . . . . .	4
1.4	Modifying or Developing SPHERLS . . . . .	4
1.5	Message Passing . . . . .	6
<b>2</b>	<b>Boundary Conditions</b>	<b>7</b>
<b>3</b>	<b>Todo List</b>	<b>15</b>
<b>4</b>	<b>Directory Hierarchy</b>	<b>17</b>
4.1	Directories . . . . .	17
<b>5</b>	<b>Class Index</b>	<b>19</b>
5.1	Class List . . . . .	19
<b>6</b>	<b>File Index</b>	<b>21</b>
6.1	File List . . . . .	21
<b>7</b>	<b>Directory Documentation</b>	<b>23</b>
7.1	src/pythonextensions/hdfwrapper/ Directory Reference . . . . .	23
7.2	src/pythonextensions/ Directory Reference . . . . .	23
7.3	scripts/ Directory Reference . . . . .	23

7.4	<a href="#">src/SPHERLS/ Directory Reference</a>	24
7.5	<a href="#">src/SPHERLSanal/ Directory Reference</a>	25
7.6	<a href="#">src/SPHERLSgen/ Directory Reference</a>	25
7.7	<a href="#">src/ Directory Reference</a>	25
<b>8</b>	<b>Class Documentation</b>	<b>27</b>
8.1	<a href="#">ALLXMLClearTag Struct Reference</a>	27
8.2	<a href="#">plot_file.Axis Class Reference</a>	27
8.2.1	<a href="#">Detailed Description</a>	27
8.2.2	<a href="#">Constructor &amp; Destructor Documentation</a>	27
8.2.2.1	<a href="#">__init__</a>	27
8.2.3	<a href="#">Member Function Documentation</a>	28
8.2.3.1	<a href="#">load</a>	28
8.3	<a href="#">plot_profile.Axis Class Reference</a>	28
8.3.1	<a href="#">Detailed Description</a>	28
8.3.2	<a href="#">Constructor &amp; Destructor Documentation</a>	28
8.3.2.1	<a href="#">__init__</a>	28
8.3.3	<a href="#">Member Function Documentation</a>	28
8.3.3.1	<a href="#">load</a>	28
8.4	<a href="#">plot_file.Curve Class Reference</a>	29
8.4.1	<a href="#">Detailed Description</a>	29
8.4.2	<a href="#">Constructor &amp; Destructor Documentation</a>	29
8.4.2.1	<a href="#">__init__</a>	29
8.4.3	<a href="#">Member Function Documentation</a>	29
8.4.3.1	<a href="#">load</a>	29
8.5	<a href="#">plot_profile.Curve Class Reference</a>	30
8.5.1	<a href="#">Detailed Description</a>	30
8.5.2	<a href="#">Constructor &amp; Destructor Documentation</a>	30
8.5.2.1	<a href="#">__init__</a>	30
8.5.3	<a href="#">Member Function Documentation</a>	30
8.5.3.1	<a href="#">load</a>	30
8.6	<a href="#">datafile.DataFile Class Reference</a>	31
8.6.1	<a href="#">Detailed Description</a>	31
8.6.2	<a href="#">Member Function Documentation</a>	31

8.6.2.1	<a href="#">readFile</a>	31
8.6.2.2	<a href="#">readFileFixed</a>	31
8.6.2.3	<a href="#">readFileUnFixed</a>	31
8.7	<a href="#">plot_file.DataSet Class Reference</a>	32
8.7.1	<a href="#">Detailed Description</a>	32
8.7.2	<a href="#">Constructor &amp; Destructor Documentation</a>	32
8.7.2.1	<a href="#">__init__</a>	32
8.7.3	<a href="#">Member Function Documentation</a>	32
8.7.3.1	<a href="#">getCurve</a>	32
8.7.3.2	<a href="#">load</a>	32
8.8	<a href="#">plot_profile.DataSet Class Reference</a>	33
8.8.1	<a href="#">Detailed Description</a>	33
8.8.2	<a href="#">Constructor &amp; Destructor Documentation</a>	33
8.8.2.1	<a href="#">__init__</a>	33
8.8.3	<a href="#">Member Function Documentation</a>	33
8.8.3.1	<a href="#">getCurve</a>	33
8.8.3.2	<a href="#">load</a>	33
8.9	<a href="#">dump.dump Class Reference</a>	34
8.9.1	<a href="#">Constructor &amp; Destructor Documentation</a>	34
8.9.1.1	<a href="#">__init__</a>	34
8.9.2	<a href="#">Member Function Documentation</a>	34
8.9.2.1	<a href="#">printHeader</a>	34
8.9.2.2	<a href="#">printVar</a>	35
8.9.2.3	<a href="#">read</a>	35
8.9.2.4	<a href="#">readAsciiVar</a>	35
8.9.2.5	<a href="#">readBinaryVar</a>	35
8.9.2.6	<a href="#">readHeader</a>	35
8.9.2.7	<a href="#">readHeaderAscii</a>	36
8.9.2.8	<a href="#">readHeaderBinary</a>	36
8.9.2.9	<a href="#">setVarIDs</a>	36
8.10	<a href="#">eos Class Reference</a>	36
8.10.1	<a href="#">Detailed Description</a>	37
8.10.2	<a href="#">Constructor &amp; Destructor Documentation</a>	38
8.10.2.1	<a href="#">eos</a>	38

8.10.2.2	<a href="#">eos</a>	38
8.10.2.3	<a href="#">eos</a>	38
8.10.2.4	<a href="#">~eos</a>	38
8.10.3	<a href="#">Member Function Documentation</a>	38
8.10.3.1	<a href="#">dDRhoDP</a>	38
8.10.3.2	<a href="#">dGetEnergy</a>	39
8.10.3.3	<a href="#">dGetOpacity</a>	39
8.10.3.4	<a href="#">dGetPressure</a>	40
8.10.3.5	<a href="#">dSoundSpeed</a>	40
8.10.3.6	<a href="#">gamma1DelAdC_v</a>	40
8.10.3.7	<a href="#">getDlnPDlnTDlnPDlnPDEDT</a>	41
8.10.3.8	<a href="#">getEAndDTDE</a>	41
8.10.3.9	<a href="#">getEKappa</a>	41
8.10.3.10	<a href="#">getPAndDRhoDP</a>	42
8.10.3.11	<a href="#">getPEKappa</a>	42
8.10.3.12	<a href="#">getPEKappaGamma</a>	43
8.10.3.13	<a href="#">getPEKappaGammaCp</a>	43
8.10.3.14	<a href="#">getPKappaGamma</a>	43
8.10.3.15	<a href="#">operator=</a>	44
8.10.3.16	<a href="#">readAscii</a>	44
8.10.3.17	<a href="#">readBin</a>	44
8.10.3.18	<a href="#">readBobsAscii</a>	44
8.10.3.19	<a href="#">writeAscii</a>	45
8.10.3.20	<a href="#">writeBin</a>	45
8.10.4	<a href="#">Member Data Documentation</a>	45
8.10.4.1	<a href="#">dLogE</a>	45
8.10.4.2	<a href="#">dLogKappa</a>	45
8.10.4.3	<a href="#">dLogP</a>	46
8.10.4.4	<a href="#">dLogRhoDelta</a>	46
8.10.4.5	<a href="#">dLogRhoMin</a>	46
8.10.4.6	<a href="#">dLogTDelta</a>	46
8.10.4.7	<a href="#">dLogTMin</a>	46
8.10.4.8	<a href="#">dXMassFrac</a>	47
8.10.4.9	<a href="#">dYMassFrac</a>	47

8.10.4.10 nNumRho . . . . .	47
8.10.4.11 nNumT . . . . .	47
8.11 eos_interp.eosTable Class Reference . . . . .	47
8.11.1 Detailed Description . . . . .	48
8.11.2 Constructor & Destructor Documentation . . . . .	48
8.11.2.1 __init__ . . . . .	48
8.11.3 Member Function Documentation . . . . .	48
8.11.3.1 interpolate . . . . .	48
8.11.3.2 load . . . . .	48
8.11.3.3 plotLogE . . . . .	49
8.11.3.4 plotLogP . . . . .	49
8.11.3.5 write . . . . .	49
8.12 eos_interp.eosTableManager Class Reference . . . . .	50
8.12.1 Detailed Description . . . . .	50
8.12.2 Constructor & Destructor Documentation . . . . .	50
8.12.2.1 __init__ . . . . .	50
8.12.3 Member Function Documentation . . . . .	50
8.12.3.1 getTableFromComp . . . . .	50
8.12.3.2 interpComp . . . . .	50
8.12.3.3 load . . . . .	51
8.12.3.4 plotGrid . . . . .	51
8.13 exception2 Class Reference . . . . .	51
8.14 plot_2DSlices.File2DSlice Class Reference . . . . .	51
8.14.1 Member Function Documentation . . . . .	51
8.14.1.1 load . . . . .	52
8.15 make_hdf.fileSet Class Reference . . . . .	52
8.15.1 Constructor & Destructor Documentation . . . . .	52
8.15.1.1 __init__ . . . . .	52
8.15.2 Member Function Documentation . . . . .	52
8.15.2.1 convertDumpToHDF . . . . .	53
8.15.2.2 makeHDFFiles . . . . .	53
8.16 make_hdf2.fileSet Class Reference . . . . .	53
8.16.1 Constructor & Destructor Documentation . . . . .	53
8.16.1.1 __init__ . . . . .	53

8.16.2	Member Function Documentation	54
8.16.2.1	convertDumpToHDF	54
8.16.2.2	makeHDFFiles	54
8.17	Functions Class Reference	54
8.17.1	Detailed Description	55
8.17.2	Constructor & Destructor Documentation	55
8.17.2.1	Functions	55
8.17.3	Member Data Documentation	55
8.17.3.1	fpCalculateAveDensities	55
8.17.3.2	fpCalculateDeltat	55
8.17.3.3	fpCalculateNewAV	56
8.17.3.4	fpCalculateNewDensities	56
8.17.3.5	fpCalculateNewEddyVisc	56
8.17.3.6	fpCalculateNewEnergies	56
8.17.3.7	fpCalculateNewEOSVars	56
8.17.3.8	fpCalculateNewGridVelocities	56
8.17.3.9	fpCalculateNewRadii	56
8.17.3.10	fpCalculateNewVelocities	57
8.17.3.11	fpImplicitSolve	57
8.17.3.12	fpModelWrite	57
8.17.3.13	fpUpdateLocalBoundaryVelocitiesNewGrid	57
8.17.3.14	fpWriteWatchZones	57
8.18	Global Class Reference	57
8.18.1	Detailed Description	58
8.18.2	Constructor & Destructor Documentation	58
8.18.2.1	Global	58
8.18.3	Member Data Documentation	58
8.18.3.1	functions	58
8.18.3.2	grid	58
8.18.3.3	implicit	58
8.18.3.4	messPass	59
8.18.3.5	output	59
8.18.3.6	parameters	59
8.18.3.7	performance	59



8.18.3.8	procTop	59
8.18.3.9	time	59
8.19	Grid Class Reference	59
8.19.1	Detailed Description	61
8.19.2	Constructor & Destructor Documentation	63
8.19.2.1	Grid	63
8.19.3	Member Data Documentation	64
8.19.3.1	dLocalGridNew	64
8.19.3.2	dLocalGridOld	64
8.19.3.3	nCenIntOffset	65
8.19.3.4	nCotThetaJK	66
8.19.3.5	nCotThetaJp1halfK	66
8.19.3.6	nD	66
8.19.3.7	nDCosThetaJK	67
8.19.3.8	nDenAve	67
8.19.3.9	nDM	68
8.19.3.10	nDonorCellFrac	68
8.19.3.11	nDPhi	69
8.19.3.12	nDTheta	69
8.19.3.13	nE	69
8.19.3.14	nEddyVisc	70
8.19.3.15	nEndGhostUpdateExplicit	70
8.19.3.16	nEndGhostUpdateImplicit	71
8.19.3.17	nEndUpdateExplicit	71
8.19.3.18	nEndUpdateImplicit	72
8.19.3.19	nGamma	72
8.19.3.20	nGlobalGridDims	72
8.19.3.21	nGlobalGridPositionLocalGrid	72
8.19.3.22	nKappa	73
8.19.3.23	nLocalGridDims	73
8.19.3.24	nM	73
8.19.3.25	nNum1DZones	73
8.19.3.26	nNumDims	73
8.19.3.27	nNumGhostCells	74

8.19.3.28 nNumIntVars . . . . .	74
8.19.3.29 nNumVars . . . . .	74
8.19.3.30 nP . . . . .	75
8.19.3.31 nPhi . . . . .	75
8.19.3.32 nQ0 . . . . .	75
8.19.3.33 nQ1 . . . . .	76
8.19.3.34 nQ2 . . . . .	76
8.19.3.35 nR . . . . .	77
8.19.3.36 nSinThetalJK . . . . .	77
8.19.3.37 nSinThetalJp1halfK . . . . .	78
8.19.3.38 nStartGhostUpdateExplicit . . . . .	78
8.19.3.39 nStartGhostUpdateImplicit . . . . .	79
8.19.3.40 nStartUpdateExplicit . . . . .	79
8.19.3.41 nStartUpdateImplicit . . . . .	79
8.19.3.42 nT . . . . .	80
8.19.3.43 nTheta . . . . .	80
8.19.3.44 nU . . . . .	80
8.19.3.45 nU0 . . . . .	81
8.19.3.46 nV . . . . .	81
8.19.3.47 nVariables . . . . .	82
8.19.3.48 nW . . . . .	82
8.20 make_hdf.hdfFile Class Reference . . . . .	83
8.20.1 Member Function Documentation . . . . .	83
8.20.1.1 write . . . . .	83
8.21 Implicit Class Reference . . . . .	83
8.21.1 Detailed Description . . . . .	84
8.21.2 Constructor & Destructor Documentation . . . . .	84
8.21.2.1 Implicit . . . . .	84
8.21.3 Member Data Documentation . . . . .	84
8.21.3.1 dAverageRHS . . . . .	84
8.21.3.2 dCurrentRelTErrors . . . . .	84
8.21.3.3 dDerivativeStepFraction . . . . .	85
8.21.3.4 dMaxErrorInRHS . . . . .	85
8.21.3.5 dTolerance . . . . .	85

8.21.3.6	<a href="#">kspContext</a>	85
8.21.3.7	<a href="#">matCoeff</a>	85
8.21.3.8	<a href="#">nCurrentNumIterations</a>	85
8.21.3.9	<a href="#">nLocDer</a>	86
8.21.3.10	<a href="#">nLocFun</a>	86
8.21.3.11	<a href="#">nMaxNumIterations</a>	86
8.21.3.12	<a href="#">nMaxNumSolverIterations</a>	86
8.21.3.13	<a href="#">nNumDerPerRow</a>	86
8.21.3.14	<a href="#">nNumImplicitZones</a>	87
8.21.3.15	<a href="#">nNumRowsALocal</a>	87
8.21.3.16	<a href="#">nNumRowsALocalSB</a>	87
8.21.3.17	<a href="#">nTypeDer</a>	87
8.21.3.18	<a href="#">vecRHS</a>	87
8.21.3.19	<a href="#">vecscatTCorrections</a>	87
8.21.3.20	<a href="#">vecTCorrections</a>	88
8.21.3.21	<a href="#">vecTCorrectionsLocal</a>	88
8.22	<a href="#">eos_interp.interpTable Class Reference</a>	88
8.22.1	<a href="#">Detailed Description</a>	88
8.22.2	<a href="#">Constructor &amp; Destructor Documentation</a>	88
8.22.2.1	<a href="#">__init__</a>	88
8.22.3	<a href="#">Member Function Documentation</a>	89
8.22.3.1	<a href="#">interpolate</a>	89
8.22.3.2	<a href="#">plotLogE</a>	89
8.22.3.3	<a href="#">plotLogK</a>	89
8.22.3.4	<a href="#">plotLogP</a>	90
8.22.3.5	<a href="#">read</a>	90
8.23	<a href="#">eos_interp.interpTableManager Class Reference</a>	90
8.23.1	<a href="#">Constructor &amp; Destructor Documentation</a>	90
8.23.1.1	<a href="#">__init__</a>	90
8.23.2	<a href="#">Member Function Documentation</a>	91
8.23.2.1	<a href="#">createTables</a>	91
8.24	<a href="#">make_hdf.interpVar Class Reference</a>	91
8.25	<a href="#">light_curve.LightCurve Class Reference</a>	91
8.25.1	<a href="#">Member Function Documentation</a>	91

8.25.1.1	calculateCurve . . . . .	91
8.25.1.2	readBoloCorr . . . . .	92
8.25.1.3	readProfiles . . . . .	92
8.25.1.4	write . . . . .	92
8.26	MDeltaDelta Struct Reference . . . . .	92
8.27	MessPass Class Reference . . . . .	92
8.27.1	Detailed Description . . . . .	93
8.27.2	Constructor & Destructor Documentation . . . . .	93
8.27.2.1	MessPass . . . . .	93
8.27.3	Member Data Documentation . . . . .	93
8.27.3.1	requestRecv . . . . .	93
8.27.3.2	requestSend . . . . .	93
8.27.3.3	statusRecv . . . . .	93
8.27.3.4	statusSend . . . . .	94
8.27.3.5	typeRecvNewVar . . . . .	94
8.27.3.6	typeRecvOldGrid . . . . .	94
8.27.3.7	typeSendNewGrid . . . . .	94
8.27.3.8	typeSendNewVar . . . . .	94
8.28	NextToken Struct Reference . . . . .	94
8.29	eos_interp.opacityTable Class Reference . . . . .	95
8.29.1	Detailed Description . . . . .	95
8.29.2	Constructor & Destructor Documentation . . . . .	95
8.29.2.1	__init__ . . . . .	95
8.29.3	Member Function Documentation . . . . .	95
8.29.3.1	fillInDepNans . . . . .	95
8.29.3.2	interpolate . . . . .	95
8.29.3.3	load . . . . .	96
8.29.3.4	plotLogK . . . . .	96
8.30	eos_interp.opacityTableManager Class Reference . . . . .	96
8.30.1	Detailed Description . . . . .	97
8.30.2	Constructor & Destructor Documentation . . . . .	97
8.30.2.1	__init__ . . . . .	97
8.30.3	Member Function Documentation . . . . .	97
8.30.3.1	getTableFromComp . . . . .	97

8.30.3.2	interpComp	97
8.30.3.3	load	98
8.30.3.4	plotGrids	98
8.31	Output Class Reference	98
8.31.1	Detailed Description	99
8.31.2	Constructor & Destructor Documentation	99
8.31.2.1	Output	99
8.31.3	Member Data Documentation	99
8.31.3.1	bDump	99
8.31.3.2	bPrint	99
8.31.3.3	dDumpFrequencyTime	99
8.31.3.4	dPrintFrequencyTime	99
8.31.3.5	dTimeLastDump	99
8.31.3.6	dTimeLastPrint	100
8.31.3.7	nDumpFrequencyStep	100
8.31.3.8	nNumTimeStepsSinceLastPrint	100
8.31.3.9	nPrintFrequencyStep	100
8.31.3.10	nPrintMode	100
8.31.3.11	ofWatchZoneFiles	100
8.31.3.12	sBaseOutputFileName	101
8.31.3.13	watchzoneList	101
8.32	Parameters Class Reference	101
8.32.1	Detailed Description	102
8.32.2	Constructor & Destructor Documentation	102
8.32.2.1	Parameters	102
8.32.3	Member Data Documentation	102
8.32.3.1	bAdiabatic	102
8.32.3.2	bDEDMClamp	102
8.32.3.3	bEOSGammaLaw	103
8.32.3.4	dA	103
8.32.3.5	dAlpha	103
8.32.3.6	dAVThreshold	103
8.32.3.7	dDEDMClampMr	103
8.32.3.8	dDEDMClampValue	104

8.32.3.9	dDonorCellMin	104
8.32.3.10	dDonorCellMultiplier	104
8.32.3.11	dEddyViscosity	104
8.32.3.12	dEDMClampTemperature	105
8.32.3.13	dG	105
8.32.3.14	dGamma	105
8.32.3.15	dMaxConvectiveVelocity	105
8.32.3.16	dMaxConvectiveVelocity_c	105
8.32.3.17	dPi	106
8.32.3.18	dPrt	106
8.32.3.19	dSigma	106
8.32.3.20	dTolerance	106
8.32.3.21	eosTable	107
8.32.3.22	nMaxIterations	107
8.32.3.23	nTypeTurbulenceMod	107
8.32.3.24	sDebugProfileOutput	107
8.32.3.25	sEOSFileName	107
8.33	work_plot.PdVPlotSettings Class Reference	108
8.34	Performance Class Reference	108
8.34.1	Detailed Description	108
8.34.2	Constructor & Destructor Documentation	108
8.34.2.1	Performance	108
8.34.3	Member Data Documentation	108
8.34.3.1	dEndTimer	108
8.34.3.2	dStartTimer	109
8.35	plot_file.Plot Class Reference	109
8.35.1	Detailed Description	109
8.35.2	Constructor & Destructor Documentation	109
8.35.2.1	__init__	109
8.35.3	Member Function Documentation	109
8.35.3.1	load	109
8.36	plot_profile.Plot Class Reference	110
8.36.1	Detailed Description	110
8.36.2	Constructor & Destructor Documentation	110

8.36.2.1	<code>__init__</code>	110
8.36.3	Member Function Documentation	110
8.36.3.1	<code>load</code>	110
8.37	ProcTop Class Reference	110
8.37.1	Detailed Description	111
8.37.2	Constructor & Destructor Documentation	111
8.37.2.1	ProcTop	111
8.37.3	Member Data Documentation	111
8.37.3.1	<code>nCoords</code>	111
8.37.3.2	<code>nNeighborRanks</code>	111
8.37.3.3	<code>nNumNeighbors</code>	112
8.37.3.4	<code>nNumProcs</code>	112
8.37.3.5	<code>nNumRadialNeighbors</code>	112
8.37.3.6	<code>nPeriodic</code>	112
8.37.3.7	<code>nProcDims</code>	112
8.37.3.8	<code>nRadialNeighborNeighborIDs</code>	113
8.37.3.9	<code>nRadialNeighborRanks</code>	113
8.37.3.10	<code>nRank</code>	113
8.38	profileData Class Reference	113
8.38.1	Constructor & Destructor Documentation	114
8.38.1.1	profileData	114
8.38.2	Member Function Documentation	114
8.38.2.1	<code>clear</code>	114
8.38.2.2	<code>nMaxNumZones</code>	114
8.38.2.3	<code>set</code>	114
8.38.2.4	<code>set</code>	114
8.38.2.5	<code>setMax</code>	114
8.38.2.6	<code>setMax</code>	115
8.38.2.7	<code>setMaxAbs</code>	115
8.38.2.8	<code>setMaxAbs</code>	115
8.38.2.9	<code>setSum</code>	115
8.38.2.10	<code>setSum</code>	115
8.38.2.11	<code>test</code>	115
8.38.2.12	<code>toFile</code>	115

8.39	<a href="#">work_plot.Settings Class Reference</a>	116
8.39.1	<a href="#">Constructor &amp; Destructor Documentation</a>	116
8.39.1.1	<a href="#">__init__</a>	116
8.39.2	<a href="#">Member Function Documentation</a>	116
8.39.2.1	<a href="#">parseXML</a>	116
8.40	<a href="#">term Struct Reference</a>	116
8.40.1	<a href="#">Detailed Description</a>	117
8.40.2	<a href="#">Member Data Documentation</a>	117
8.40.2.1	<a href="#">dCoeff</a>	117
8.40.2.2	<a href="#">dPower</a>	117
8.41	<a href="#">plot_file.Text Class Reference</a>	117
8.41.1	<a href="#">Detailed Description</a>	117
8.41.2	<a href="#">Constructor &amp; Destructor Documentation</a>	117
8.41.2.1	<a href="#">__init__</a>	118
8.42	<a href="#">Time Class Reference</a>	118
8.42.1	<a href="#">Detailed Description</a>	118
8.42.2	<a href="#">Constructor &amp; Destructor Documentation</a>	119
8.42.2.1	<a href="#">Time</a>	119
8.42.3	<a href="#">Member Data Documentation</a>	119
8.42.3.1	<a href="#">bVariableTimeStep</a>	119
8.42.3.2	<a href="#">dConstTimeStep</a>	119
8.42.3.3	<a href="#">dDelE_t_E_max</a>	119
8.42.3.4	<a href="#">dDelRho_t_Rho_max</a>	119
8.42.3.5	<a href="#">dDelT_t_T_max</a>	119
8.42.3.6	<a href="#">dDeltat_n</a>	120
8.42.3.7	<a href="#">dDeltat_nm1half</a>	120
8.42.3.8	<a href="#">dDeltat_np1half</a>	120
8.42.3.9	<a href="#">dEndTime</a>	120
8.42.3.10	<a href="#">dPerChange</a>	120
8.42.3.11	<a href="#">dt</a>	121
8.42.3.12	<a href="#">dTimeStepFactor</a>	121
8.42.3.13	<a href="#">nEndTimeStep</a>	121
8.42.3.14	<a href="#">nTimeStepIndex</a>	121
8.43	<a href="#">make_hdf.variable Class Reference</a>	122



8.44	<a href="#">watchzone Struct Reference</a>	122
8.45	<a href="#">WatchZone Class Reference</a>	122
8.45.1	<a href="#">Detailed Description</a>	122
8.46	<a href="#">work_plot.WorkPlotSettings Class Reference</a>	122
8.47	<a href="#">XML Struct Reference</a>	122
8.48	<a href="#">XMLAttribute Struct Reference</a>	122
8.49	<a href="#">XMLCharacterEntity Struct Reference</a>	123
8.50	<a href="#">XMLClear Struct Reference</a>	123
8.51	<a href="#">XMLNode Struct Reference</a>	123
8.52	<a href="#">XMLNodeContents Struct Reference</a>	123
8.53	<a href="#">XMLParserBase64Tool Class Reference</a>	123
8.54	<a href="#">XMLResults Struct Reference</a>	123
<b>9</b>	<b><a href="#">File Documentation</a></b>	<b>125</b>
9.1	<a href="#">scripts/cp_files.py File Reference</a>	125
9.1.1	<a href="#">Detailed Description</a>	125
9.2	<a href="#">src/eos.cpp File Reference</a>	125
9.2.1	<a href="#">Detailed Description</a>	125
9.3	<a href="#">src/eos.h File Reference</a>	125
9.3.1	<a href="#">Detailed Description</a>	126
9.4	<a href="#">src/SPHERLS/dataManipulation.cpp File Reference</a>	126
9.4.1	<a href="#">Detailed Description</a>	127
9.4.2	<a href="#">Function Documentation</a>	127
9.4.2.1	<a href="#">average3DTo1DBoundariesNew</a>	127
9.4.2.2	<a href="#">average3DTo1DBoundariesOld</a>	127
9.4.2.3	<a href="#">fin</a>	128
9.4.2.4	<a href="#">init</a>	128
9.4.2.5	<a href="#">initImplicitCalculation</a>	129
9.4.2.6	<a href="#">initUpdateLocalBoundaries</a>	130
9.4.2.7	<a href="#">modelRead</a>	131
9.4.2.8	<a href="#">modelWrite_GL</a>	132
9.4.2.9	<a href="#">modelWrite_TEOS</a>	132
9.4.2.10	<a href="#">setDEDMClamp</a>	133
9.4.2.11	<a href="#">setupLocalGrid</a>	133

9.4.2.12	<a href="#">updateLocalBoundaries</a>	133
9.4.2.13	<a href="#">updateLocalBoundariesNewGrid</a>	134
9.4.2.14	<a href="#">updateLocalBoundaryVelocitiesNewGrid_R</a>	135
9.4.2.15	<a href="#">updateLocalBoundaryVelocitiesNewGrid_RT</a>	135
9.4.2.16	<a href="#">updateLocalBoundaryVelocitiesNewGrid_RTP</a>	135
9.4.2.17	<a href="#">updateNewGridWithOld</a>	135
9.4.2.18	<a href="#">updateOldGrid</a>	135
9.5	<a href="#">src/SPHERLS/dataManipulation.h File Reference</a>	136
9.5.1	<a href="#">Detailed Description</a>	137
9.5.2	<a href="#">Function Documentation</a>	137
9.5.2.1	<a href="#">average3DTo1DBoundariesNew</a>	137
9.5.2.2	<a href="#">average3DTo1DBoundariesOld</a>	137
9.5.2.3	<a href="#">fin</a>	138
9.5.2.4	<a href="#">init</a>	138
9.5.2.5	<a href="#">initImplicitCalculation</a>	139
9.5.2.6	<a href="#">initUpdateLocalBoundaries</a>	140
9.5.2.7	<a href="#">modelRead</a>	141
9.5.2.8	<a href="#">modelWrite_GL</a>	142
9.5.2.9	<a href="#">modelWrite_TEOS</a>	142
9.5.2.10	<a href="#">setDEDMClamp</a>	143
9.5.2.11	<a href="#">setupLocalGrid</a>	143
9.5.2.12	<a href="#">updateLocalBoundaries</a>	143
9.5.2.13	<a href="#">updateLocalBoundariesNewGrid</a>	144
9.5.2.14	<a href="#">updateLocalBoundaryVelocitiesNewGrid_R</a>	145
9.5.2.15	<a href="#">updateLocalBoundaryVelocitiesNewGrid_RT</a>	145
9.5.2.16	<a href="#">updateLocalBoundaryVelocitiesNewGrid_RTP</a>	145
9.5.2.17	<a href="#">updateNewGridWithOld</a>	145
9.5.2.18	<a href="#">updateOldGrid</a>	145
9.6	<a href="#">src/SPHERLS/dataMonitoring.cpp File Reference</a>	146
9.6.1	<a href="#">Detailed Description</a>	146
9.6.2	<a href="#">Function Documentation</a>	147
9.6.2.1	<a href="#">finWatchZones</a>	147
9.6.2.2	<a href="#">initWatchZones</a>	147
9.6.2.3	<a href="#">writeWatchZones_R_GL</a>	147

9.6.2.4	<a href="#">writeWatchZones_R_TEOS</a>	148
9.6.2.5	<a href="#">writeWatchZones_RT_GL</a>	148
9.6.2.6	<a href="#">writeWatchZones_RT_TEOS</a>	149
9.6.2.7	<a href="#">writeWatchZones_RTP_GL</a>	149
9.6.2.8	<a href="#">writeWatchZones_RTP_TEOS</a>	149
9.7	<a href="#">src/SPHERLS/dataMonitoring.h File Reference</a>	150
9.7.1	<a href="#">Detailed Description</a>	150
9.7.2	<a href="#">Function Documentation</a>	150
9.7.2.1	<a href="#">finWatchZones</a>	151
9.7.2.2	<a href="#">initWatchZones</a>	151
9.7.2.3	<a href="#">writeWatchZones_R_GL</a>	151
9.7.2.4	<a href="#">writeWatchZones_R_TEOS</a>	152
9.7.2.5	<a href="#">writeWatchZones_RT_GL</a>	152
9.7.2.6	<a href="#">writeWatchZones_RT_TEOS</a>	153
9.7.2.7	<a href="#">writeWatchZones_RTP_GL</a>	153
9.7.2.8	<a href="#">writeWatchZones_RTP_TEOS</a>	153
9.8	<a href="#">src/SPHERLS/global.cpp File Reference</a>	154
9.8.1	<a href="#">Detailed Description</a>	154
9.9	<a href="#">src/SPHERLS/global.h File Reference</a>	154
9.9.1	<a href="#">Detailed Description</a>	155
9.9.2	<a href="#">Define Documentation</a>	155
9.9.2.1	<a href="#">DEBUG_EQUATIONS</a>	155
9.9.2.2	<a href="#">DEDEM_CLAMP</a>	155
9.9.2.3	<a href="#">DUMP_VERSION</a>	155
9.9.2.4	<a href="#">SEDOV</a>	155
9.9.2.5	<a href="#">SIGNEGDEN</a>	156
9.9.2.6	<a href="#">SIGNEGENG</a>	156
9.9.2.7	<a href="#">SIGNEGTEMP</a>	156
9.9.2.8	<a href="#">TRACKMAXSOLVERERROR</a>	156
9.9.2.9	<a href="#">VISCOUS_ENERGY_EQ</a>	156
9.10	<a href="#">src/SPHERLS/main.cpp File Reference</a>	156
9.10.1	<a href="#">Detailed Description</a>	157
9.10.2	<a href="#">Function Documentation</a>	157
9.10.2.1	<a href="#">main</a>	157

9.10.2.2	signalHandler	158
9.11	src/SPHERLSanal/main.cpp File Reference	158
9.11.1	Detailed Description	159
9.11.2	Function Documentation	159
9.11.2.1	bFileExists	159
9.11.2.2	computeFourierTransFromList	159
9.11.2.3	convertBinToHDF4	159
9.11.2.4	dCalRhoAve2D	159
9.11.2.5	main	160
9.11.2.6	printHelp	161
9.12	src/SPHERLSgen/main.cpp File Reference	161
9.12.1	Detailed Description	162
9.12.2	Function Documentation	163
9.12.2.1	calculateFirstShell_SEDOV	163
9.12.2.2	calculateFirstShell_TEOS	163
9.12.2.3	dPartialDerivativeVar1	164
9.12.2.4	dPartialDerivativeVar2	164
9.12.2.5	EOS_GL	164
9.12.2.6	generateModel_GL	165
9.12.2.7	generateModel_SEDOV	165
9.12.2.8	generateModel_TEOS	165
9.12.2.9	main	165
9.12.2.10	readConfig	166
9.12.2.11	readUProfile	166
9.12.2.12	writeModel_Bin_R_GL	167
9.12.2.13	writeModel_Bin_R_TEOS	167
9.12.2.14	writeModel_Bin_RT_GL	167
9.12.2.15	writeModel_Bin_RT_TEOS	167
9.12.2.16	writeModel_Bin_RTP_GL	168
9.12.2.17	writeModel_Bin_RTP_TEOS	168
9.12.2.18	writeModel_R_GL	168
9.12.2.19	writeModel_R_TEOS	168
9.12.2.20	writeModel_RT_GL	169
9.12.2.21	writeModel_RT_TEOS	169

9.12.2.22 writeModel_RTP_GL . . . . .	169
9.12.2.23 writeModel_RTP_TEOS . . . . .	169
9.12.2.24 writeModelToScreen_GL . . . . .	170
9.12.2.25 writeModelToScreen_TEOS . . . . .	170
9.13 src/SPHERLS/main.h File Reference . . . . .	170
9.13.1 Detailed Description . . . . .	170
9.13.2 Function Documentation . . . . .	170
9.13.2.1 main . . . . .	170
9.13.2.2 signalHandler . . . . .	172
9.14 src/SPHERLSanal/main.h File Reference . . . . .	172
9.14.1 Detailed Description . . . . .	173
9.14.2 Function Documentation . . . . .	173
9.14.2.1 bFileExists . . . . .	173
9.14.2.2 computeFourierTransFromList . . . . .	174
9.14.2.3 convertBinToHDF4 . . . . .	174
9.14.2.4 dCalRhoAve2D . . . . .	174
9.14.2.5 printHelp . . . . .	174
9.14.3 Variable Documentation . . . . .	174
9.14.3.1 bExtraInfoInProfile . . . . .	174
9.14.3.2 bScientific . . . . .	174
9.14.3.3 dLSun . . . . .	175
9.14.3.4 dPi . . . . .	175
9.14.3.5 dSigma . . . . .	175
9.14.3.6 nC . . . . .	175
9.14.3.7 nD . . . . .	175
9.14.3.8 nDM . . . . .	175
9.14.3.9 nDumpFileVersion . . . . .	175
9.14.3.10 nE . . . . .	176
9.14.3.11 nF_con . . . . .	176
9.14.3.12 nGamma . . . . .	176
9.14.3.13 nKappa . . . . .	176
9.14.3.14 nKE . . . . .	176
9.14.3.15 nL_con . . . . .	176
9.14.3.16 nL_rad . . . . .	176

9.14.3.17 nM . . . . .	176
9.14.3.18 nP . . . . .	177
9.14.3.19 nPhi . . . . .	177
9.14.3.20 nPrecisionAscii . . . . .	177
9.14.3.21 nQ . . . . .	177
9.14.3.22 nR . . . . .	177
9.14.3.23 nT . . . . .	177
9.14.3.24 nTheta . . . . .	177
9.14.3.25 nU . . . . .	178
9.14.3.26 nU0 . . . . .	178
9.14.3.27 nV . . . . .	178
9.14.3.28 nW . . . . .	178
9.14.3.29 sEOSFile . . . . .	178
9.15 src/SPHERLSgen/main.h File Reference . . . . .	178
9.15.1 Detailed Description . . . . .	181
9.15.2 Function Documentation . . . . .	181
9.15.2.1 calculateFirstShell_SEDOV . . . . .	181
9.15.2.2 calculateFirstShell_TEOS . . . . .	181
9.15.2.3 dPartialDerivativeVar1 . . . . .	182
9.15.2.4 dPartialDerivativeVar2 . . . . .	182
9.15.2.5 EOS_GL . . . . .	182
9.15.2.6 generateModel_GL . . . . .	183
9.15.2.7 generateModel_SEDOV . . . . .	183
9.15.2.8 generateModel_TEOS . . . . .	183
9.15.2.9 main . . . . .	183
9.15.2.10 readConfig . . . . .	184
9.15.2.11 readUProfile . . . . .	184
9.15.2.12 writeModel_Bin_R_GL . . . . .	185
9.15.2.13 writeModel_Bin_R_TEOS . . . . .	185
9.15.2.14 writeModel_Bin_RT_GL . . . . .	185
9.15.2.15 writeModel_Bin_RT_TEOS . . . . .	186
9.15.2.16 writeModel_Bin_RTP_GL . . . . .	186
9.15.2.17 writeModel_Bin_RTP_TEOS . . . . .	186
9.15.2.18 writeModel_R_GL . . . . .	186

9.15.2.19	writeModel_R_TEOS	187
9.15.2.20	writeModel_RT_GL	187
9.15.2.21	writeModel_RT_TEOS	187
9.15.2.22	writeModel_RTP_GL	187
9.15.2.23	writeModel_RTP_TEOS	188
9.15.2.24	writeModelToScreen_GL	188
9.15.2.25	writeModelToScreen_TEOS	188
9.15.3	Variable Documentation	188
9.15.3.1	bAutoDeltaM	188
9.15.3.2	bBinaryOutput	188
9.15.3.3	bGammaLawEOS	188
9.15.3.4	blsSedov	189
9.15.3.5	bWriteToScreen	189
9.15.3.6	dAlpha	189
9.15.3.7	dDeltaPhi	189
9.15.3.8	dDeltaTheta	189
9.15.3.9	dEng	189
9.15.3.10	dEngCent	190
9.15.3.11	dEPro	190
9.15.3.12	dEProM	190
9.15.3.13	dG	190
9.15.3.14	dGamma	190
9.15.3.15	dL	190
9.15.3.16	dLSun	190
9.15.3.17	dMDelta	191
9.15.3.18	dMDeltaDelta	191
9.15.3.19	dMSun	191
9.15.3.20	dMTotal	191
9.15.3.21	dPi	191
9.15.3.22	dRDelta	191
9.15.3.23	dRho	191
9.15.3.24	dRMin	191
9.15.3.25	dRSun	192
9.15.3.26	dRSurf	192

9.15.3.27 dSigma . . . . .	192
9.15.3.28 dTeff . . . . .	192
9.15.3.29 dTimeStepFactor . . . . .	192
9.15.3.30 dTolerance . . . . .	192
9.15.3.31 dU . . . . .	193
9.15.3.32 dU0 . . . . .	193
9.15.3.33 dUPro . . . . .	193
9.15.3.34 dUProR . . . . .	193
9.15.3.35 dUSurf . . . . .	193
9.15.3.36 dV . . . . .	193
9.15.3.37 dW . . . . .	194
9.15.3.38 eosTable . . . . .	194
9.15.3.39 nNumCellsCent . . . . .	194
9.15.3.40 nNumDims . . . . .	194
9.15.3.41 nNumEProPoints . . . . .	194
9.15.3.42 nNumGhostCells . . . . .	194
9.15.3.43 nNumIters . . . . .	194
9.15.3.44 nNumPhi . . . . .	195
9.15.3.45 nNumR . . . . .	195
9.15.3.46 nNumTheta . . . . .	195
9.15.3.47 nNumUProPoints . . . . .	195
9.15.3.48 nNumZones1D . . . . .	195
9.15.3.49 nPeriodic . . . . .	196
9.15.3.50 nPrecision . . . . .	196
9.15.3.51 sEOSFile . . . . .	196
9.15.3.52 sOutPutfile . . . . .	196
9.15.3.53 sUDistType . . . . .	196
9.15.3.54 vecdE . . . . .	197
9.15.3.55 vecdKappa . . . . .	197
9.15.3.56 vecdM . . . . .	197
9.15.3.57 vecdMDel . . . . .	197
9.15.3.58 vecdP . . . . .	197
9.15.3.59 vecdR . . . . .	198
9.15.3.60 vecdRho . . . . .	198



9.15.3.61	vecdT	198
9.15.3.62	vectVelDist	198
9.16	src/SPHERLS/physEquations.cpp File Reference	199
9.16.1	Detailed Description	202
9.16.2	Function Documentation	203
9.16.2.1	calDelt_CONST	203
9.16.2.2	calDelt_R_GL	203
9.16.2.3	calDelt_R_TEOS	203
9.16.2.4	calDelt_RT_GL	204
9.16.2.5	calDelt_RT_TEOS	205
9.16.2.6	calDelt_RTP_GL	205
9.16.2.7	calDelt_RTP_TEOS	206
9.16.2.8	calNewD_R	206
9.16.2.9	calNewD_RT	207
9.16.2.10	calNewD_RTP	207
9.16.2.11	calNewDenave_None	208
9.16.2.12	calNewDenave_R	208
9.16.2.13	calNewDenave_RT	208
9.16.2.14	calNewDenave_RTP	209
9.16.2.15	calNewE_R_AD	209
9.16.2.16	calNewE_R_NA	210
9.16.2.17	calNewE_R_NA_LES	210
9.16.2.18	calNewE_RT_AD	211
9.16.2.19	calNewE_RT_NA	212
9.16.2.20	calNewE_RT_NA_LES	212
9.16.2.21	calNewE_RTP_AD	213
9.16.2.22	calNewE_RTP_NA	214
9.16.2.23	calNewE_RTP_NA_LES	214
9.16.2.24	calNewEddyVisc_None	215
9.16.2.25	calNewEddyVisc_R_CN	216
9.16.2.26	calNewEddyVisc_R_SM	216
9.16.2.27	calNewEddyVisc_RT_CN	216
9.16.2.28	calNewEddyVisc_RT_SM	217
9.16.2.29	calNewEddyVisc_RTP_CN	217

9.16.2.30 calNewEddyVisc_RTP_SM . . . . .	217
9.16.2.31 calNewP_GL . . . . .	218
9.16.2.32 calNewPEKappaGamma_TEOS . . . . .	218
9.16.2.33 calNewQ0_R_GL . . . . .	219
9.16.2.34 calNewQ0_R_TEOS . . . . .	219
9.16.2.35 calNewQ0Q1_RT_GL . . . . .	219
9.16.2.36 calNewQ0Q1_RT_TEOS . . . . .	220
9.16.2.37 calNewQ0Q1Q2_RTP_GL . . . . .	220
9.16.2.38 calNewQ0Q1Q2_RTP_TEOS . . . . .	221
9.16.2.39 calNewR . . . . .	221
9.16.2.40 calNewTPKappaGamma_TEOS . . . . .	222
9.16.2.41 calNewU0_R . . . . .	222
9.16.2.42 calNewU0_RT . . . . .	223
9.16.2.43 calNewU0_RTP . . . . .	223
9.16.2.44 calNewU_R . . . . .	224
9.16.2.45 calNewU_R_LES . . . . .	224
9.16.2.46 calNewU_RT . . . . .	225
9.16.2.47 calNewU_RT_LES . . . . .	226
9.16.2.48 calNewU_RTP . . . . .	227
9.16.2.49 calNewU_RTP_LES . . . . .	228
9.16.2.50 calNewV_RT . . . . .	229
9.16.2.51 calNewV_RT_LES . . . . .	230
9.16.2.52 calNewV_RTP . . . . .	230
9.16.2.53 calNewV_RTP_LES . . . . .	231
9.16.2.54 calNewVelocities_R . . . . .	232
9.16.2.55 calNewVelocities_R_LES . . . . .	232
9.16.2.56 calNewVelocities_RT . . . . .	232
9.16.2.57 calNewVelocities_RT_LES . . . . .	233
9.16.2.58 calNewVelocities_RTP . . . . .	233
9.16.2.59 calNewVelocities_RTP_LES . . . . .	234
9.16.2.60 calNewW_RTP . . . . .	234
9.16.2.61 calNewW_RTP_LES . . . . .	235
9.16.2.62 calOldDenave_None . . . . .	235
9.16.2.63 calOldDenave_R . . . . .	236

9.16.2.64 calOldDenave_RT . . . . .	236
9.16.2.65 calOldDenave_RTP . . . . .	236
9.16.2.66 calOldEddyVisc_R_CN . . . . .	237
9.16.2.67 calOldEddyVisc_R_SM . . . . .	237
9.16.2.68 calOldEddyVisc_RT_CN . . . . .	237
9.16.2.69 calOldEddyVisc_RT_SM . . . . .	238
9.16.2.70 calOldEddyVisc_RTP_CN . . . . .	238
9.16.2.71 calOldEddyVisc_RTP_SM . . . . .	239
9.16.2.72 calOldP_GL . . . . .	239
9.16.2.73 calOldPEKappaGamma_TEOS . . . . .	239
9.16.2.74 calOldQ0_R_GL . . . . .	240
9.16.2.75 calOldQ0_R_TEOS . . . . .	240
9.16.2.76 calOldQ0Q1_RT_GL . . . . .	241
9.16.2.77 calOldQ0Q1_RT_TEOS . . . . .	241
9.16.2.78 calOldQ0Q1Q2_RTP_GL . . . . .	242
9.16.2.79 calOldQ0Q1Q2_RTP_TEOS . . . . .	242
9.16.2.80 dEOS_GL . . . . .	242
9.16.2.81 dET4 . . . . .	243
9.16.2.82 dImplicitEnergyFunction_None . . . . .	243
9.16.2.83 dImplicitEnergyFunction_R . . . . .	243
9.16.2.84 dImplicitEnergyFunction_R_LES . . . . .	244
9.16.2.85 dImplicitEnergyFunction_R_LES_SB . . . . .	245
9.16.2.86 dImplicitEnergyFunction_R_SB . . . . .	245
9.16.2.87 dImplicitEnergyFunction_RT . . . . .	246
9.16.2.88 dImplicitEnergyFunction_RT_LES . . . . .	247
9.16.2.89 dImplicitEnergyFunction_RT_LES_SB . . . . .	248
9.16.2.90 dImplicitEnergyFunction_RT_SB . . . . .	249
9.16.2.91 dImplicitEnergyFunction_RTP . . . . .	250
9.16.2.92 dImplicitEnergyFunction_RTP_LES . . . . .	251
9.16.2.93 dImplicitEnergyFunction_RTP_LES_SB . . . . .	251
9.16.2.94 dImplicitEnergyFunction_RTP_SB . . . . .	253
9.16.2.95 implicitSolve_None . . . . .	254
9.16.2.96 implicitSolve_R . . . . .	254
9.16.2.97 implicitSolve_RT . . . . .	254

9.16.2.98	implicitSolve_RTP	255
9.16.2.99	initDonorFracAndMaxConVel_R_GL	255
9.16.2.100	initDonorFracAndMaxConVel_R_TEOS	256
9.16.2.101	initDonorFracAndMaxConVel_RT_GL	256
9.16.2.102	initDonorFracAndMaxConVel_RT_TEOS	256
9.16.2.103	initDonorFracAndMaxConVel_RTP_GL	257
9.16.2.104	initDonorFracAndMaxConVel_RTP_TEOS	257
9.16.2.105	nitInternalVars	257
9.16.2.106	setInternalVarInf	258
9.16.2.107	setMainFunctions	259
9.17	src/SPHERLS/physEquations.h File Reference	260
9.17.1	Detailed Description	264
9.17.2	Function Documentation	264
9.17.2.1	calDelt_CONST	264
9.17.2.2	calDelt_R_GL	264
9.17.2.3	calDelt_R_TEOS	264
9.17.2.4	calDelt_RT_GL	265
9.17.2.5	calDelt_RT_TEOS	266
9.17.2.6	calDelt_RTP_GL	266
9.17.2.7	calDelt_RTP_TEOS	267
9.17.2.8	calNewD_R	267
9.17.2.9	calNewD_RT	268
9.17.2.10	calNewD_RTP	268
9.17.2.11	calNewDenave_None	269
9.17.2.12	calNewDenave_R	269
9.17.2.13	calNewDenave_RT	269
9.17.2.14	calNewDenave_RTP	270
9.17.2.15	calNewE_R_AD	270
9.17.2.16	calNewE_R_NA	271
9.17.2.17	calNewE_R_NA_LES	271
9.17.2.18	calNewE_RT_AD	272
9.17.2.19	calNewE_RT_NA	273
9.17.2.20	calNewE_RT_NA_LES	273
9.17.2.21	calNewE_RTP_AD	274

9.17.2.22 calNewE_RTP_NA . . . . .	275
9.17.2.23 calNewE_RTP_NA_LES . . . . .	275
9.17.2.24 calNewEddyVisc_None . . . . .	276
9.17.2.25 calNewEddyVisc_R_CN . . . . .	277
9.17.2.26 calNewEddyVisc_R_SM . . . . .	277
9.17.2.27 calNewEddyVisc_RT_CN . . . . .	277
9.17.2.28 calNewEddyVisc_RT_SM . . . . .	278
9.17.2.29 calNewEddyVisc_RTP_CN . . . . .	278
9.17.2.30 calNewEddyVisc_RTP_SM . . . . .	278
9.17.2.31 calNewP_GL . . . . .	279
9.17.2.32 calNewPEKappaGamma_TEOS . . . . .	279
9.17.2.33 calNewQ0_R_GL . . . . .	280
9.17.2.34 calNewQ0_R_TEOS . . . . .	280
9.17.2.35 calNewQ0Q1_RT_GL . . . . .	280
9.17.2.36 calNewQ0Q1_RT_TEOS . . . . .	281
9.17.2.37 calNewQ0Q1Q2_RTP_GL . . . . .	281
9.17.2.38 calNewQ0Q1Q2_RTP_TEOS . . . . .	282
9.17.2.39 calNewR . . . . .	282
9.17.2.40 calNewTPKappaGamma_TEOS . . . . .	283
9.17.2.41 calNewU0_R . . . . .	283
9.17.2.42 calNewU0_RT . . . . .	284
9.17.2.43 calNewU0_RTP . . . . .	284
9.17.2.44 calNewU_R . . . . .	285
9.17.2.45 calNewU_R_LES . . . . .	285
9.17.2.46 calNewU_RT . . . . .	286
9.17.2.47 calNewU_RT_LES . . . . .	287
9.17.2.48 calNewU_RTP . . . . .	288
9.17.2.49 calNewU_RTP_LES . . . . .	289
9.17.2.50 calNewV_RT . . . . .	290
9.17.2.51 calNewV_RT_LES . . . . .	291
9.17.2.52 calNewV_RTP . . . . .	291
9.17.2.53 calNewV_RTP_LES . . . . .	292
9.17.2.54 calNewVelocities_R . . . . .	293
9.17.2.55 calNewVelocities_R_LES . . . . .	293

9.17.2.56 calNewVelocities_RT . . . . .	293
9.17.2.57 calNewVelocities_RT_LES . . . . .	294
9.17.2.58 calNewVelocities_RTP . . . . .	294
9.17.2.59 calNewVelocities_RTP_LES . . . . .	295
9.17.2.60 calNewW_RTP . . . . .	295
9.17.2.61 calNewW_RTP_LES . . . . .	296
9.17.2.62 calOldDenave_None . . . . .	296
9.17.2.63 calOldDenave_R . . . . .	297
9.17.2.64 calOldDenave_RT . . . . .	297
9.17.2.65 calOldDenave_RTP . . . . .	297
9.17.2.66 calOldEddyVisc_R_CN . . . . .	298
9.17.2.67 calOldEddyVisc_R_SM . . . . .	298
9.17.2.68 calOldEddyVisc_RT_CN . . . . .	298
9.17.2.69 calOldEddyVisc_RT_SM . . . . .	299
9.17.2.70 calOldEddyVisc_RTP_CN . . . . .	299
9.17.2.71 calOldEddyVisc_RTP_SM . . . . .	300
9.17.2.72 calOldP_GL . . . . .	300
9.17.2.73 calOldPEKappaGamma_TEOS . . . . .	300
9.17.2.74 calOldQ0_R_GL . . . . .	301
9.17.2.75 calOldQ0_R_TEOS . . . . .	301
9.17.2.76 calOldQ0Q1_RT_GL . . . . .	302
9.17.2.77 calOldQ0Q1_RT_TEOS . . . . .	302
9.17.2.78 calOldQ0Q1Q2_RTP_GL . . . . .	303
9.17.2.79 calOldQ0Q1Q2_RTP_TEOS . . . . .	303
9.17.2.80 dEOS_GL . . . . .	303
9.17.2.81 dET4 . . . . .	304
9.17.2.82 dImplicitEnergyFunction_None . . . . .	304
9.17.2.83 dImplicitEnergyFunction_R . . . . .	304
9.17.2.84 dImplicitEnergyFunction_R_LES . . . . .	305
9.17.2.85 dImplicitEnergyFunction_R_LES_SB . . . . .	306
9.17.2.86 dImplicitEnergyFunction_R_SB . . . . .	306
9.17.2.87 dImplicitEnergyFunction_RT . . . . .	307
9.17.2.88 dImplicitEnergyFunction_RT_LES . . . . .	308
9.17.2.89 dImplicitEnergyFunction_RT_LES_SB . . . . .	309

9.17.2.90 dImplicitEnergyFunction_RT_SB . . . . .	310
9.17.2.91 dImplicitEnergyFunction_RTP . . . . .	311
9.17.2.92 dImplicitEnergyFunction_RTP_LES . . . . .	312
9.17.2.93 dImplicitEnergyFunction_RTP_LES_SB . . . . .	312
9.17.2.94 dImplicitEnergyFunction_RTP_SB . . . . .	314
9.17.2.95 implicitSolve_None . . . . .	315
9.17.2.96 implicitSolve_R . . . . .	315
9.17.2.97 implicitSolve_RT . . . . .	315
9.17.2.98 implicitSolve_RTP . . . . .	316
9.17.2.99 initDonorFracAndMaxConVel_R_GL . . . . .	316
9.17.2.100initDonorFracAndMaxConVel_R_TEOS . . . . .	317
9.17.2.101initDonorFracAndMaxConVel_RT_GL . . . . .	317
9.17.2.102initDonorFracAndMaxConVel_RT_TEOS . . . . .	317
9.17.2.103initDonorFracAndMaxConVel_RTP_GL . . . . .	318
9.17.2.104initDonorFracAndMaxConVel_RTP_TEOS . . . . .	318
9.17.2.105nitInternalVars . . . . .	318
9.17.2.106setInternalVarInf . . . . .	319
9.17.2.107setMainFunctions . . . . .	320
9.18 src/SPHERLS/procTop.cpp File Reference . . . . .	321
9.18.1 Detailed Description . . . . .	321
9.19 src/SPHERLS/procTop.h File Reference . . . . .	321
9.19.1 Detailed Description . . . . .	321
9.20 src/SPHERLS/profileData.h File Reference . . . . .	321
9.20.1 Detailed Description . . . . .	322
9.21 src/SPHERLS/time.cpp File Reference . . . . .	322
9.21.1 Detailed Description . . . . .	322
9.22 src/SPHERLS/time.h File Reference . . . . .	322
9.22.1 Detailed Description . . . . .	322
9.23 src/SPHERLS/watchzone.cpp File Reference . . . . .	322
9.23.1 Detailed Description . . . . .	322
9.24 src/SPHERLS/watchzone.h File Reference . . . . .	322
9.24.1 Detailed Description . . . . .	323
9.25 src/SPHERLSanal/userguide.h File Reference . . . . .	323
9.25.1 Detailed Description . . . . .	323

9.26	<a href="#">src/SPHERLSgen/userguide.h File Reference</a>	323
9.26.1	<a href="#">Detailed Description</a>	323



# Chapter 1

## Using and Modifying SPHERLSanal

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

### 1.1 Overview

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

#### 1.1.1 The Basics

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

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

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

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

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

where  $\tau$  is the stress tensor for zero bulk viscosity,  $E$  is the specific internal energy,  $\mathbb{V}$  is the specific volume, and  $F$  is the radiative flux. In addition to these conservation equations an equation of state is needed, in this case the OPAL equation of state and opacities, and the Alaxander opacities at low temperatures are used. The equation of state tables are functions of density and temperature, and produce the energy, pressure, opacity, and adiabatic index of the gas for a given temperature and density.

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

- Explain what SPHERLS does
  - Equations
  - Roughly how it solves them
  - The order
- Different ways in which SPHERLS can be used, 1D,2D,3D, Adiabatic,Non-adiabatic, implicit, debugging options/test

## 1.2 Compiling SPHERLS

Once the correct libraries are installed, and their paths added to your

LD\_LIBRARY\_PATH

environment variable, it should just require typing make in the correct directories. SPHERLS is broken up into 3 main codes. SPHERLS it self, which is the main hydrodynamics code which integrates the initial static model, SPHERLSgen which creates the static model, and SPHERLSanal which is used for processing the output of SPHERLS and SPHERLSgen.

To Add

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

### 1.2.1 Requirements

- openMPI
- gcc
- PETSc library
- python for analysis scripts
- fftw library for analysis

### 1.2.2 Installing PETSC Library

- Download PETSc library, from the PETSc [website](#). Version petsc-lite-3.1-p8 has been tested to work with SPHERLS.
- The downloaded PETSc file (e.g. petsc-lite-3.1-p8.tar.gz) will need to be unzipped to do so type `gunzip petsc-lite-3.1-p8.tar.gz`
- Then untar it with `tar -xf petsc-lite-3.1-p8.tar`
- To install the library change into the directory made when you extracted the archive and type the following commands:

1. `./configure --prefix=<path-to-final-location-of-library> --with-c++-support --with-c-support --with-shared --download-f-blas-lapack=1`
2. `make all`
3. `make install`
4. `make PETSC_DIR=<path-to-final-location-of-library> test`

### 1.2.3 Installing FFTW Library

- Download the FFTW Library from the FFTS [website](#). Version fftw-3.2.2 has been tested to work with SPHERLS.
- The downloaded FFTW file (e.g. fftw-3.2.2.tar.gz) will need to be unzipped to do so type `gunzip fftw-3.2.2.tar.gz`
- Then untar it with `tar -xf fftw-3.2.2.tar.gz`
- To install the library change into the directory made when you extracted the archive and type the following commands:
  1. `./configure --prefix=<path-to-final-location-of-library>`
  2. `make`
  3. `make install`

## 1.3 Using SPHERLS

- Generating a starting model
  - Using SPHERLSgen
- The [XML](#) configuration file
- Starting a calculation and the "makeFile"
- analysis of results (using premade analysis scripts)
  - watchzones
  - peak KE tracking
  - model dumps

## 1.4 Modifying or Developing SPHERLS

- Basic layout/design of the code
  - model output
  - data monitoring
    - \* watch zones
    - \* peak KE tracking
  - internal/versus external variables
  - message passing

- grid layout
- ranges of grids
- boundary regions
- grid updating
- How to document SPHERLS
- How to modify SPHERLS
  - Common changes
    - \* How to add a new internal variable
      1. **Add to the internal variable count:** Decide in what cases the variable will be needed, 1D calculations, 2D calculations, when there is a gamma law gas or a tabulated equation of state, adiabatic or non-adiabatic etc. Then once decided it can be added to the total number of internal variables `Grid::nNumIntVars` by increasing the value by one in the function `modelRead` in the section below the comment "set number of internal variables ..." under the appropriate if block. If the specific if block for the situation you need isn't there, you can create your own, and add it there.
      2. **Create a new variable ID:** In the `grid.h` file under the `Grid` class are variable ID's. These ID's simply indicate the location of the variable in the array. One must add a new ID for the new variable as an integer. The value of the ID is set in the function `modelRead` in the same section as the number of internal variables. The value used should be the last integer after the last pre-existing variable ID. This should also be `Grid::nNumVars + Grid::nNumIntVars - 1`. The ID should also be initialized to -1, so that the code knows when it isn't being used. This is done in the grid class constructor, `Grid::Grid`. Simply add a line in the constructor setting your new ID = -1.
      3. **Set variable infos:** Decide what the dimensions of the new variable will be. It can be cell centered it can be or interface centered, it can also be only 1D, 2D, or 3D. Of course it will be only 1D if the entire calculation is 1D, or 2D if the calculation is 2D, but if the calculation is 3D it could also only be 2D, or 1D, and if 2D it could be only 1D. Also decide if the variable will change with time, dependent variables are only initialized and not updated during the calculations. This information is given to SPHERLS in the `setInternalVarInf` function in the `physEquations.cpp` file. The variable that is set is `Grid::nVariables`. It is a 2D array, the first index corresponds to the particular variable in question, the ID you made in the previous step can be used as the first index of this array. The second index refers to the three directions (0-2) and the time (3). If the variable is centered in the grid in direction 0 (r-direction) then this array element should have a value of 0. If the variable is interface centered in the grid in direction 0, then this array element should have a value of 1. If it isn't defined in direction 0, for example the theta independent variable isn't defined in the 0 direction then it should be -1. This is the same for the other 2 directions.

The last element (3) should be either 0 not updated every time step, or 1 if updated every timestep.

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

- \* How to add a new external variable
  - \* How to add a new physics functions
    - Function naming conventions
    - [Grid](#) variables
    - indecies and their ranges
- SPHERLS debugging tips

## 1.5 Message Passing

- Explain message passing in SPHERLS

## Chapter 2

# 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][n1Cen][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][n1Cen+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 density 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 3

### 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 `dDeltaTheta`**

At some point in the future it may be desirable to vary this value across theta.

**Member `dImplicitEnergyFunction_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.

**File `main.cpp`**

Want to make printing model to the screen an option in the configuration file, and the default should be not print the model.

It would also make sense to print to a binary model rather than an ascii model. It could however be an option with the default being to print to a binary file.

**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 `printHelp` ()**

need to updated, and revise help text to better describe the program. Some improvements could include: -better describing the "-x" appended to the file name base -mention that some times it expects a file name base, while others it wants the full file name -mention file extensions and naming of output files i.e. what the outputfile for the radially averaged profile will be called -perhaps mention some of the additional scripts used to extend the functionality of SPHERLSanal

**Member `readConfig` (`std::string sConfigFileName`, `std::string sStartNode`)**

need to check that T is increasing, and R is decreasing. This will get tricky if R and T types are mixed.

**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 4

# Directory Hierarchy

### 4.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

scripts . . . . .	23
src . . . . .	25
pythonextensions . . . . .	23
hdfwrapper . . . . .	23
SPHERLS . . . . .	24
SPHERLSanal . . . . .	25
SPHERLSgen . . . . .	25



## Chapter 5

# Class Index

### 5.1 Class List

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

<a href="#">ALLXMLClearTag</a>	27
<a href="#">plot_file.Axis</a>	27
<a href="#">plot_profile.Axis</a>	28
<a href="#">plot_file.Curve</a>	29
<a href="#">plot_profile.Curve</a>	30
<a href="#">datafile.DataFile</a>	31
<a href="#">plot_file.DataSet</a>	32
<a href="#">plot_profile.DataSet</a>	33
<a href="#">dump.dump</a>	34
<a href="#">eos</a>	36
<a href="#">eos_interp.eosTable</a>	47
<a href="#">eos_interp.eosTableManager</a>	50
<a href="#">exception2</a>	51
<a href="#">plot_2DSlices.File2DSlice</a>	51
<a href="#">make_hdf.fileSet</a>	52
<a href="#">make_hdf2.fileSet</a>	53
<a href="#">Functions</a>	54
<a href="#">Global</a>	57
<a href="#">Grid</a>	59
<a href="#">make_hdf.hdfFile</a>	83
<a href="#">Implicit</a>	83
<a href="#">eos_interp.interpTable</a>	88
<a href="#">eos_interp.interpTableManager</a>	90
<a href="#">make_hdf.interpVar</a>	91
<a href="#">light_curve.LightCurve</a>	91
<a href="#">MDeltaDelta</a>	92
<a href="#">MessPass</a>	92
<a href="#">NextToken</a>	94
<a href="#">eos_interp.opacityTable</a>	95

eos_interp.opacityTableManager	96
Output	98
Parameters	101
work_plot.PdVPlotSettings	108
Performance	108
plot_file.Plot	109
plot_profile.Plot	110
ProcTop	110
profileData	113
work_plot.Settings	116
term	116
plot_file.Text	117
Time	118
make_hdf.variable	122
watchzone	122
WatchZone	122
work_plot.WorkPlotSettings	122
XML	122
XMLAttribute	122
XMLCharacterEntity	123
XMLClear	123
XMLNode	123
XMLNodeContents	123
XMLParserBase64Tool	123
XMLResults	123

## Chapter 6

# File Index

### 6.1 File List

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

scripts/ <a href="#">cp_files.py</a>	125
src/ <a href="#">eos.cpp</a>	125
src/ <a href="#">eos.h</a>	125
src/ <b>exception2.h</b>	??
src/ <b>xmlFunctions.h</b>	??
src/ <b>xmlParser.h</b>	??
src/SPHERLS/ <a href="#">dataManipulation.cpp</a>	126
src/SPHERLS/ <a href="#">dataManipulation.h</a>	136
src/SPHERLS/ <a href="#">dataMonitoring.cpp</a>	146
src/SPHERLS/ <a href="#">dataMonitoring.h</a>	150
src/SPHERLS/ <b>fileExists.h</b>	??
src/SPHERLS/ <a href="#">global.cpp</a>	154
src/SPHERLS/ <a href="#">global.h</a>	154
src/SPHERLS/ <a href="#">main.cpp</a>	156
src/SPHERLS/ <a href="#">main.h</a>	170
src/SPHERLS/ <a href="#">physEquations.cpp</a>	199
src/SPHERLS/ <a href="#">physEquations.h</a>	260
src/SPHERLS/ <a href="#">procTop.cpp</a>	321
src/SPHERLS/ <a href="#">procTop.h</a>	321
src/SPHERLS/ <a href="#">profileData.h</a>	321
src/SPHERLS/ <a href="#">time.cpp</a>	322
src/SPHERLS/ <a href="#">time.h</a>	322
src/SPHERLS/ <a href="#">watchzone.cpp</a>	322
src/SPHERLS/ <a href="#">watchzone.h</a>	322
src/SPHERLSanal/ <a href="#">main.cpp</a>	158
src/SPHERLSanal/ <a href="#">main.h</a>	172
src/SPHERLSanal/ <a href="#">userguide.h</a>	323
src/SPHERLSgen/ <a href="#">main.cpp</a>	161
src/SPHERLSgen/ <a href="#">main.h</a>	178

src/SPHERLSgen/[userguide.h](#) . . . . . 323

## Chapter 7

# Directory Documentation

### 7.1 `src/pythonextensions/hdfwrapper/` Directory Reference

#### Files

- file `hdfmodule.cpp`
- file `setup.py`

### 7.2 `src/pythonextensions/` Directory Reference

#### Directories

- directory [hdfwrapper](#)

### 7.3 `scripts/` Directory Reference

#### Files

- file `average_PKE.py`
- file `combine_bins.py`
- file `combine_bins_persistent.py`
- file `compare_sedov_blasts.py`
- file [cp\\_files.py](#)
- file `datafile.py`
- file `diffDumps.py`
- file `disect_filename.py`
- file `dump.py`
- file `eos_interp.py`
- file `foureir_transform.py`

- file `light_curve.py`
- file `make_2DSlices.py`
- file `make_hdf.py`
- file `make_hdf2.py`
- file `make_profiles.py`
- file `mv_files.py`
- file `mywarnings.py`
- file `parse_formula.py`
- file `paths.py`
- file `period_from_PKE_ave.py`
- file `phase_light_curves.py`
- file `plot_2DSlices.py`
- file `plot_file.py`
- file `plot_light_curve.py`
- file `plot_Lum_diffs.py`
- file `plot_max_variance.py`
- file `plot_max_variance_ave.py`
- file `plot_max_variance_exploring.py`
- file `plot_profile.py`
- file `plot_reproducible.py`
- file `post_processing.py`
- file `ref_calcs.py`
- file `rm_files.py`
- file `SPHERLS_run.py`
- file `test_calculation.py`
- file `test_restart.py`
- file `work_plot.py`
- file `xmlParseFunctions.py`

## 7.4 src/SPHERLS/ Directory Reference

### Files

- file [dataManipulation.cpp](#)
- file [dataManipulation.h](#)
- file [dataMonitoring.cpp](#)
- file [dataMonitoring.h](#)
- file [fileExists.cpp](#)
- file [fileExists.h](#)
- file [global.cpp](#)
- file [global.h](#)
- file [main.cpp](#)
- file [main.h](#)
- file [physEquations.cpp](#)
- file [physEquations.h](#)



- file [procTop.cpp](#)
- file [procTop.h](#)
- file **profileData.cpp**
- file [profileData.h](#)
- file **testProfileData.cpp**
- file [time.cpp](#)
- file [time.h](#)
- file [watchzone.cpp](#)
- file [watchzone.h](#)

## 7.5 src/SPHERLSanal/ Directory Reference

### Files

- file [main.cpp](#)
- file [main.h](#)
- file [userguide.h](#)

## 7.6 src/SPHERLSgen/ Directory Reference

### Files

- file [main.cpp](#)
- file [main.h](#)
- file [userguide.h](#)

## 7.7 src/ Directory Reference

### Directories

- directory [pythonextensions](#)
- directory [SPHERLS](#)
- directory [SPHERLSanal](#)
- directory [SPHERLSgen](#)

### Files

- file [eos.cpp](#)
- file [eos.h](#)
- file **exception2.cpp**
- file **exception2.h**
- file **xmlFunctions.cpp**

- file **xmlFunctions.h**
- file **xmlParser.cpp**
- file **xmlParser.h**

## Chapter 8

# Class Documentation

### 8.1 ALLXMLClearTag Struct Reference

The documentation for this struct was generated from the following file:

- `src/xmlParser.h`

### 8.2 `plot_file.Axis` Class Reference

#### Public Member Functions

- `def \_\_init\_\_`
- `def load`

#### 8.2.1 Detailed Description

This class holds all the information needed for a particular x-axis.

#### 8.2.2 Constructor & Destructor Documentation

##### 8.2.2.1 `def plot_file.Axis.__init__( self, element, options )`

This function initializes the axis object.

References `plot_file.Plot.bMinorTics`, `plot_file.Axis.bMinorTics`, `plot_profile.Curve.bTime`, `plot_profile.Axis.bTime`, `plot_file.Plot.grid`, `plot_file.Axis.grid`, `Global.grid`, `plot_file.Plot.limits`, `plot_file.Axis.limits`, `plot_profile.Axis.period`, `plot_file.Axis.plotHeightWeights`, `plot_file.Axis.plots`, `plot_file.Plot.ticks`, `plot_file.Axis.ticks`, and `plot_file.Axis.xlabel`.

### 8.2.3 Member Function Documentation

#### 8.2.3.1 `def plot_file.Axis.load( self, files, options )`

This function loads the values needed for the x-axis data from the `fileData` argument

References `plot_file.Axis.plots`.

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

- `scripts/plot_file.py`

## 8.3 `plot_profile.Axis` Class Reference

### Public Member Functions

- `def __init__`
- `def load`

#### 8.3.1 Detailed Description

This class holds all the information needed for a particular x-axis. An axis can either be either of time, or of some column in the data files.

### 8.3.2 Constructor & Destructor Documentation

#### 8.3.2.1 `def plot_profile.Axis.__init__( self, element, options )`

This function initializes the axis object.

References `plot_file.Plot.bMinorTics`, `plot_profile.Plot.bMinorTics`, `plot_file.Axis.bMinorTics`, `plot_profile.Axis.bMinorTics`, `plot_profile.Curve.bTime`, `plot_profile.Axis.bTime`, `plot_profile.Curve.code`, `plot_profile.Axis.code`, `plot_profile.Axis.formula`, `make_hdf.variable.formula`, `make_hdf.interpVar.formula`, `plot_profile.Curve.formulaOrig`, `plot_profile.Axis.formulaOrig`, `plot_file.Plot.grid`, `plot_profile.Plot.grid`, `plot_file.Axis.grid`, `plot_profile.Axis.grid`, `Global.grid`, `plot_file.Plot.limits`, `plot_profile.Plot.limits`, `plot_file.Axis.limits`, `plot_profile.Axis.limits`, `plot_profile.Curve.nColumn`, `XMLResults.nColumn`, `plot_profile.Axis.nColumn`, `plot_profile.Axis.period`, `plot_profile.Axis.phase`, `plot_file.Axis.plots`, `plot_profile.Axis.plots`, `plot_file.Text.x`, `plot_file.Curve.x`, `plot_profile.Axis.x`, `plot_file.Axis.xlabel`, and `plot_profile.Axis.xlabel`.

### 8.3.3 Member Function Documentation

#### 8.3.3.1 `def plot_profile.Axis.load( self, fileData, options, dataSet, nFileCount )`

This function loads the values needed for the x-axis data from the `fileData` argument

References `plot_profile.Curve.bTime`, `plot_profile.Axis.bTime`, `plot_profile.Curve.code`, `plot_profile.Axis.code`, `plot_profile.Curve.formulaOrig`, `plot_profile.Axis.formulaOrig`, `plot_profile.Curve.nColumn`, `XMLResults.nColumn`, `plot_profile.Axis.nColumn`, `plot_profile.Axis.period`, `plot_file.Axis.plots`, `plot_profile.Axis.plots`, `plot_file.Text.x`, `plot_file.Curve.x`, `plot_profile.Axis.x`, `plot_file.Axis.xlabel`, and `plot_profile.Axis.xlabel`.

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

- `scripts/plot_profile.py`

## 8.4 plot\_file.Curve Class Reference

### Public Member Functions

- `def __init__`
- `def load`

#### 8.4.1 Detailed Description

This class holds all the information for a curve on a plot.

#### 8.4.2 Constructor & Destructor Documentation

##### 8.4.2.1 `def plot_file.Curve.__init__( self, element )`

This method initializes a curve object, the type parameter allows checking curve syntax against axis syntax to see if they match.

References `plot_file.Curve.capsize`, `plot_file.Curve.codeErr`, `plot_file.Curve.codeX`, `plot_file.Curve.codeY`, `plot_file.Curve.color`, `plot_file.Curve.ecolor`, `plot_file.Curve.elinewidth`, `plot_file.Curve.error`, `XMLResults.error`, `XML.error`, `plot_file.Curve.fileReference`, `plot_file.Curve.formulaErr`, `plot_file.Curve.formulaOrigErr`, `plot_file.Curve.formulaOrigX`, `plot_file.Curve.formulaOrigY`, `plot_file.Curve.formulaX`, `plot_file.Curve.formulaY`, `plot_2DSlices.File2DSlice.index`, `plot_file.Curve.index`, `plot_file.Curve.label`, `plot_file.Curve.linewidth`, `plot_file.Curve.marker`, `plot_file.Curve.markeredgcolor`, `plot_file.Curve.markerfacecolor`, `plot_file.Curve.markersize`, `plot_file.Curve.nColumnErr`, `plot_file.Curve.nColumnX`, `plot_file.Curve.nColumnY`, `plot_file.Curve.nRowShiftErr`, `plot_file.Curve.nRowShiftX`, `plot_file.Curve.nRowShiftY`, `plot_file.Curve.style`, `plot_file.Text.x`, `plot_file.Curve.x`, `plot_file.Text.y`, and `plot_file.Curve.y`.

#### 8.4.3 Member Function Documentation

##### 8.4.3.1 `def plot_file.Curve.load( self, files, options )`

This method adds a y value and index to the curve for the current fileData.

References `plot_file.Curve.codeErr`, `plot_file.Curve.codeX`, `plot_file.Curve.codeY`, `plot_file.Curve.fileReference`, `plot_file.Curve.nColumnErr`, `plot_file.Curve.nColumnX`, `plot_file.Curve.nColumnY`, `plot_file.Curve.nRowShiftErr`, `plot_file.Curve.nRowShiftX`, and `plot_file.Curve.nRowShiftY`.

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

- `scripts/plot_file.py`

## 8.5 `plot_profile.Curve` Class Reference

### Public Member Functions

- `def \_\_init\_\_`
- `def load`

#### 8.5.1 Detailed Description

This class holds all the information for a curve on a plot.

#### 8.5.2 Constructor & Destructor Documentation

##### 8.5.2.1 `def plot_profile.Curve.__init__( self, element, type, curveIndex )`

This method initializes a curve object, the `type` parameter allows checking curve syntax against axis syntax to see if they match.

References `plot_profile.Curve.bTime`, `plot_profile.Curve.code`, `plot_profile.Curve.color`, `plot_file.Curve.color`, `plot_profile.Axis.formula`, `make_hdf.variable.formula`, `make_hdf.interpVar.formula`, `plot_profile.Curve.formulaOrig`, `plot_profile.Curve.ID`, `plot_2DSlices.File2DSlice.index`, `plot_profile.Curve.index`, `plot_file.Curve.index`, `plot_profile.Curve.indexOfLastFileLoad`, `plot_profile.Curve.label`, `plot_file.Curve.label`, `plot_profile.Curve.linewidth`, `plot_file.Curve.linewidth`, `plot_profile.Curve.markersize`, `plot_file.Curve.markersize`, `plot_profile.Curve.nColumn`, `XMLResults.nColumn`, `plot_profile.Curve.nCurveIDForZoneRef`, `plot_profile.Curve.style`, `plot_file.Curve.style`, `plot_profile.Curve.testZoneAdjust`, `plot_file.Text.y`, `plot_profile.Curve.y`, `plot_file.Curve.y`, and `plot_profile.Curve.zone`.

#### 8.5.3 Member Function Documentation

##### 8.5.3.1 `def plot_profile.Curve.load( self, fileData, options, dataSet, nFileCount )`

This method adds a `y` value and index to the curve for the current `fileData`.

References `plot_profile.Curve.bTime`, `plot_profile.Curve.code`, `plot_profile.Curve.-formulaOrig`, `plot_profile.Curve.ID`, `plot_profile.Curve.indexOfLastFileLoad`, `plot_profile.Curve.nColumn`, `XMLResults.nColumn`, `plot_profile.Curve.testZoneAdjust`, and `plot_profile.Curve.zone`.

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

- `scripts/plot_profile.py`

## 8.6 datafile.DataFile Class Reference

### Public Member Functions

- def `readFile`
- def `readFileFixed`
- def `readFileUnFixed`

### 8.6.1 Detailed Description

A generic class for holding a file consisting of a header and columns of floats

### 8.6.2 Member Function Documentation

#### 8.6.2.1 def datafile.DataFile.readFile( self, sFileName )

a wrapper to determine which readFile function should be used

References `datafile.DataFile.fColumnValues`, `datafile.DataFile.readFileFixed()`, `datafile.DataFile.readFileUnFixed()`, and `datafile.DataFile.sFileName`.

#### 8.6.2.2 def datafile.DataFile.readFileFixed( self, sFileName )

Reads in a file when the size has already been set using `\ref setFileSize`, or by a previous file read using `\ref readFileUnFixed`.

References `datafile.DataFile.fColumnValues`, `datafile.DataFile.sColumnNames`, and `datafile.DataFile.sHeader`.

Referenced by `datafile.DataFile.readFile()`.

#### 8.6.2.3 def datafile.DataFile.readFileUnFixed( self, sFileName )

Reads in a file when the size is not fixed and needs to be determined from the input file being read in

References `datafile.DataFile.fColumnValues`, `datafile.DataFile.sColumnNames`, and `datafile.DataFile.sHeader`.

Referenced by `datafile.DataFile.readFile()`.

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

- `scripts/datafile.py`

## 8.7 `plot_file.DataSet` Class Reference

### Public Member Functions

- `def __init__`
- `def load`
- `def getCurve`

#### 8.7.1 Detailed Description

This class holds all the information for a single `dataSet`, which includes the `baseFileName` of the dataset, the range of the `dataSet` (start-end), the times and phases of the files within the range of the `dataSet`, and the plots made from the `dataSet`.

#### 8.7.2 Constructor & Destructor Documentation

##### 8.7.2.1 `def plot_file.DataSet.__init__( self, element, options )`

Initilizes the `dataSet` by setting `baseFileName`, `start`, `end`, and intilizing plots from an `xml` element

References `plot_file.DataSet.axes`, and `plot_file.DataSet.files`.

#### 8.7.3 Member Function Documentation

##### 8.7.3.1 `def plot_file.DataSet.getCurve( self, ID )`

Returns a curve object that has `ID`, `ID`

References `plot_file.DataSet.axes`, and `main()`.

##### 8.7.3.2 `def plot_file.DataSet.load( self, options )`

Loads the `dataSet`, this means that it sets, time, phases, and plots data

References `plot_file.DataSet.axes`, and `plot_file.DataSet.files`.

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



- `scripts/plot_file.py`

## 8.8 `plot_profile.DataSet` Class Reference

### Public Member Functions

- `def __init__`
- `def load`
- `def getCurve`

### 8.8.1 Detailed Description

This class holds all the information for a single `dataSet`, which includes the `baseFileName` of the dataset, the range of the `dataSet` (start-end), the times and phases of the files within the range of the `dataSet`, and the plots made from the `dataSet`.

### 8.8.2 Constructor & Destructor Documentation

#### 8.8.2.1 `def plot_profile.DataSet.__init__( self, element, options )`

Initilizes the `dataSet` by setting `baseFileName`, `start`, `end`, and intilizing plots from an `xml` element

References `plot_file.DataSet.axes`, `plot_profile.DataSet.axes`, `plot_profile.DataSet.baseFileName`, `plot_profile.DataSet.end`, `plot_2DSlices.File2DSlice.eosFile`, `light_curve.LightCurve.eosFile`, `plot_profile.DataSet.eosFile`, `plot_profile.DataSet.fileIndices`, `plot_profile.DataSet.hasNonTimeAxis`, `light_curve.LightCurve.nNumFiles`, `plot_profile.DataSet.nNumFiles`, and `plot_profile.DataSet.start`.

### 8.8.3 Member Function Documentation

#### 8.8.3.1 `def plot_profile.DataSet.getCurve( self, ID )`

Returns a curve object that has `ID`, `ID`

References `plot_file.DataSet.axes`, `plot_profile.DataSet.axes`, and `main()`.

#### 8.8.3.2 `def plot_profile.DataSet.load( self, options )`

Loads the `dataSet`, this means that it sets, time, phases, and plots data

References `plot_file.DataSet.axes`, `plot_profile.DataSet.axes`, `plot_profile.DataSet.baseFileName`, `plot_profile.DataSet.end`, `plot_2DSlices.File2DSlice.eosFile`, `light_curve.LightCurve.eosFile`, `plot_profile.DataSet.eosFile`, `light_curve.LightCurve.nNumFiles`, `plot_profile.DataSet.nNumFiles`, and `plot_profile.DataSet.start`.

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

- scripts/plot\_profile.py

## 8.9 dump.dump Class Reference

### Public Member Functions

- def [\\_\\_init\\_\\_](#)
- def [read](#)
- def [setVarIDs](#)
- def [readHeader](#)
- def [readHeaderBinary](#)
- def [readHeaderAscii](#)
- def [readBinaryVar](#)
- def [readAsciiVar](#)
- def [printHeader](#)
- def [printVar](#)

### 8.9.1 Constructor & Destructor Documentation

#### 8.9.1.1 def dump.dump.\_\_init\_\_( self, fileName )

Initilizes the dump by reading in a binary file.

References dump.dump.read().

### 8.9.2 Member Function Documentation

#### 8.9.2.1 def dump.dump.printHeader( self )

Prints the header of a binary dump file to the standard output.

References dump.dump.alpha, dump.dump.av, dump.dump.avthreshold, dump.dump.boundaryConditions, dump.dump.delta\_t\_nm1half, dump.dump.delta\_t\_np1half, dump.dump.eosString, dump.dump.eosStringLen, dump.dump.gamma, dump.dump.globalDims, dump.dump.num1DZones, dump.dump.numDims, dump.dump.numGhostCells, dump.dump.numVars, dump.dump.time, Global.time, dump.dump.timeStepIndex, dump.dump.type, XMLNodeContents.type, dump.dump.varInfo, dump.dump.varSize, and dump.dump.version.

Referenced by dump.dump.printVar().

### 8.9.2.2 def dump.dump.printVar( self, var )

Prints a variable to the standard output.

References dump.dump.boundaryConditions, main(), dump.dump.num1DZones, dump.dump.numGhostCells, dump.dump.numVars, dump.dump.printHeader(), dump.dump.printVar(), dump.dump.varInfo, dump.dump.vars, and dump.dump.varSize.

Referenced by dump.dump.printVar().

### 8.9.2.3 def dump.dump.read( self, fileName )

Reads in a binary dump file.

References dump.dump.f, dump.dump.fileName, dump.dump.numVars, dump.dump.readAsciiVar(), dump.dump.readBinaryVar(), dump.dump.readHeader(), dump.dump.setVarIds(), dump.dump.type, and XMLNodeContents.type.

Referenced by dump.dump.\_\_init\_\_().

### 8.9.2.4 def dump.dump.readAsciiVar( self, var )

Read in a variable from an ascii dump file. Must be called with var increasing from 0 to self.numVars.

References dump.dump.boundaryConditions, dump.dump.num1DZones, dump.dump.numGhostCells, dump.dump.varInfo, and dump.dump.varSize.

Referenced by dump.dump.read().

### 8.9.2.5 def dump.dump.readBinaryVar( self, var )

Read in a variable from a binary dump file. Must be called with var increasing from 0 to self.numVars.

References dump.dump.boundaryConditions, dump.dump.num1DZones, dump.dump.numGhostCells, dump.dump.varInfo, and dump.dump.varSize.

Referenced by dump.dump.read().

### 8.9.2.6 def dump.dump.readHeader( self )

Reads header information from binary dump file.

References dump.dump.readHeaderAscii(), dump.dump.readHeaderBinary(), dump.dump.type, and XMLNodeContents.type.

Referenced by dump.dump.read().

### 8.9.2.7 `def dump.dump.readHeaderAscii( self )`

Reads a header from a ascii file, after the type has been read in.

References `dump.dump.alpha`, `dump.dump.av`, `dump.dump.avthreshold`, `dump.dump.boundaryConditions`, `dump.dump.delta_t_nm1half`, `dump.dump.delta_t_np1half`, `dump.dump.eosString`, `dump.dump.eosStringLen`, `dump.dump.gamma`, `dump.dump.globalDims`, `dump.dump.num1DZones`, `dump.dump.numDims`, `dump.dump.numGhostCells`, `dump.dump.numVars`, `dump.dump.time`, `Global.time`, `dump.dump.timeStepIndex`, `dump.dump.varInfo`, `dump.dump.vars`, `dump.dump.varSize`, and `dump.dump.version`.

Referenced by `dump.dump.readHeader()`.

### 8.9.2.8 `def dump.dump.readHeaderBinary( self )`

Reads a header from a binary file, after the type has been read in.

References `dump.dump.alpha`, `dump.dump.av`, `dump.dump.avthreshold`, `dump.dump.boundaryConditions`, `dump.dump.delta_t_nm1half`, `dump.dump.delta_t_np1half`, `dump.dump.eosString`, `dump.dump.eosStringLen`, `dump.dump.gamma`, `dump.dump.globalDims`, `dump.dump.num1DZones`, `dump.dump.numDims`, `dump.dump.numGhostCells`, `dump.dump.numVars`, `dump.dump.time`, `Global.time`, `dump.dump.timeStepIndex`, `dump.dump.varInfo`, `dump.dump.vars`, `dump.dump.varSize`, and `dump.dump.version`.

Referenced by `dump.dump.readHeader()`.

### 8.9.2.9 `def dump.dump.setVarIDs( self )`

Sets names for the interger values of the grid variables

References `dump.dump.gamma`, `dump.dump.numDims`, `dump.dump.varIDs`, and `dump.dump.varNames`.

Referenced by `dump.dump.read()`.

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

- `scripts/dump.py`

## 8.10 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`

#### 8.10.1 Detailed Description

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

## 8.10.2 Constructor & Destructor Documentation

### 8.10.2.1 `eos::eos( )`

Constructor, doesn't really do anything

References `dLogE`, `dLogKappa`, `dLogP`, `nNumRho`, and `nNumT`.

### 8.10.2.2 `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

### 8.10.2.3 `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`.

### 8.10.2.4 `eos::~~eos( )`

Destructor, deletes dynamic arrays

References `dLogE`, `dLogKappa`, `dLogP`, and `nNumRho`.

## 8.10.3 Member Function Documentation

### 8.10.3.1 `double eos::dDRhoDP( double dT, double dRho )`

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

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

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

### 8.10.3.2 double eos::dGetEnergy( double *dT*, double *dRho* )

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

#### Parameters

in	<i>dT</i>	temperature to interpolate to.
in	<i>dRho</i>	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()*.

### 8.10.3.3 double eos::dGetOpacity( double *dT*, double *dRho* )

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

#### Parameters

in	<i>dT</i>	temperature to interpolate to.
in	<i>dRho</i>	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()*.

#### 8.10.3.4 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

in	<i>dT</i>	temperature to interpolate to.
in	<i>dRho</i>	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().

#### 8.10.3.5 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, nNumRho, and nNumT.

#### 8.10.3.6 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.

**8.10.3.7** 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	<i>dDlnPDlnRho</i>	derivative of ln(P) w.r.t. ln(Rho)
out	<i>dDEDT</i>	derivative of temperature w.r.t. energy at constant density

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

**8.10.3.8** 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	<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>dE</i>	energy at dT and dRho
out	<i>dDTDE</i>	derivative of temperature w.r.t. energy at constant density

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

Referenced by calNewTPKappaGamma\_TEOS().

**8.10.3.9** 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 dRho are not in log space.

## Parameters

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

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

Referenced by calculateFirstShell\_TEOS().

**8.10.3.10** 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.

Referenced by calculateFirstShell\_TEOS().

**8.10.3.11** 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.

**8.10.3.12** `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	<i>dT</i>	temperature to interpolate to.
in	<i>dRho</i>	density to interpolate to.
out	<i>dP</i>	pressure at <i>dT</i> and <i>dRho</i> .
out	<i>dE</i>	energy at <i>dT</i> and <i>dRho</i> .
out	<i>dKappa</i>	opacity at <i>dT</i> and <i>dRho</i> .
out	<i>dGamma</i>	adiabatic index at <i>dT</i> and <i>dRho</i> .

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

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

**8.10.3.13** `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	<i>dT</i>	temperature to interpolate to.
in	<i>dRho</i>	density to interpolate to.
out	<i>dP</i>	pressure at <i>dT</i> and <i>dRho</i> .
out	<i>dE</i>	energy at <i>dT</i> and <i>dRho</i> .
out	<i>dKappa</i>	opacity at <i>dT</i> and <i>dRho</i> .
out	<i>dGamma</i>	adiabatic index at <i>dT</i> and <i>dRho</i> .
out	<i>dCp</i>	specific heat at constant pressure at <i>dT</i> and <i>dRho</i> .

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

**8.10.3.14** `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, d-LogTMin, nNumRho, and nNumT.

Referenced by calNewTPKappaGamma\_TEOS().

#### 8.10.3.15 eos & eos::operator= ( const eos & eosRightSide )

Assignment operator, assigns one eos object to another.

References dLogE, dLogKappa, dLogP, dLogRhoDelta, dLogRhoMin, dLogTDelta, d-LogTMin, nNumRho, and nNumT.

#### 8.10.3.16 void eos::readAscii ( std::string sFileName )

This fuction 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, d-LogTMin, dXMassFrac, dYMassFrac, nNumRho, and nNumT.

#### 8.10.3.17 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 convertBinToHDF4(), init(), and readConfig().

#### 8.10.3.18 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.

8.10.3.19 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.

8.10.3.20 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.

## 8.10.4 Member Data Documentation

## 8.10.4.1 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(), getDlnPDlnPDlnPDlnPDEDT(), getEAndDTDE(), getEKappa(), getPEKappa(), getPEKappaGamma(), getPEKappaGammaCp(), getPKappaGamma(), operator=(), readAscii(), readBobsAscii(), writeAscii(), writeBin(), and ~eos().

## 8.10.4.2 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()`.

#### 8.10.4.3 `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()`.

#### 8.10.4.4 `double eos::dLogRhoDelta`

Increment of the density between table entries in log10.

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

#### 8.10.4.5 `double eos::dLogRhoMin`

Minimum density of the table in log10.

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

#### 8.10.4.6 `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()`.

#### 8.10.4.7 `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()`.

#### 8.10.4.8 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()`.

#### 8.10.4.9 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()`.

#### 8.10.4.10 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()`.

#### 8.10.4.11 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:

- [src/eos.h](#)
- [src/eos.cpp](#)

## 8.11 eos\_interp.eosTable Class Reference

### Public Member Functions

- [def load](#)

- def [write](#)
- def [plotLogE](#)
- def [plotLogP](#)
- def [interpolate](#)
- def [\\_\\_init\\_\\_](#)

### 8.11.1 Detailed Description

Holds equation of state data.

### 8.11.2 Constructor & Destructor Documentation

#### 8.11.2.1 def eos\_interp.eosTable.\_\_init\_\_( self, sFileName = None )

Returns a new instance of eosTable.

If sFileName is set it will use that to set the filename to load the data from.

References eos\_interp.eosTable.logD, eos\_interp.eosTable.logE, eos\_interp.eosTable.logP, eos\_interp.eosTable.logT, datafile.DataFile.sFileName, eos\_interp.eosTable.sFileName, eos\_interp.eosTable.status, light\_curve.LightCurve.temperature, eos\_interp.eosTable.X, and eos\_interp.eosTable.Z.

### 8.11.3 Member Function Documentation

#### 8.11.3.1 def eos\_interp.eosTable.interpolate( self, gridConfig, setExtrapolatedToNan = True )

Interpolate from self's table to the gridding specified by:

```
logDMin: first (smallest) logD value of grid
logDDel: spacing in logD
numLogD: number of logD grid points
logTMin: first (smallest) logT value of grid
logTDel: spacing in logT
numLogT: number of logT grid points
```

References eos\_interp.eosTable.\_\_fillDepNans(), eos\_interp.eosTable.logD, eos\_interp.eosTable.logE, eos\_interp.eosTable.logT, eos\_interp.eosTable.X, and eos\_interp.eosTable.Z.

#### 8.11.3.2 def eos\_interp.eosTable.load( self )

Reads in an OPAL equation of state file.

```
It puts the resulting file info into:
self.X: the hydrogen mass fraction
self.Z: the metal mass fraction
self.logD: numpy array of log density grid points [g/cm^3]
```



```
self.logT: numpy array of log tempeature grid points [K]
self.logE: numpy array of log energy [ergs/g]
self.logP: numpy array of log pressure [dynes/cm^2]
```

self.logD, self.logT, self.logE, and self.logP are all the same size numpy arrays, empty emelents have logE and logP as nans.

References eos\_interp.eosTable.\_\_fillInDepNans(), eos\_interp.eosTable.\_\_gmass(), eos\_interp.eosTable.logD, eos\_interp.eosTable.logE, eos\_interp.eosTable.logP, eos\_interp.eosTable.logT, datafile.DataFile.sFileName, eos\_interp.eosTable.sFileName, eos\_interp.eosTable.status, eos\_interp.eosTable.X, and eos\_interp.eosTable.Z.

**8.11.3.3** `def eos_interp.eosTable.plotLogE( self, otherTables =None, logDIndexList = None, wireFrame =True )`

Plots LogE

Keywords:

otherTables: a list of other eosTables to include in the plot

logDIndexList: a list of integers corresponding to which densities to plot the tables at

wireFrame: if set to true (the default) and logDIndexList is set to None it will plot a 3D wireframe of logE.

References eos\_interp.eosTable.logD, eos\_interp.eosTable.logE, and eos\_interp.eosTable.logT.

**8.11.3.4** `def eos_interp.eosTable.plotLogP( self, otherTables =None, logDIndexList = None, wireFrame =True )`

Plots LogP

Keywords:

otherTables: a list of other eosTables to include in the plot

logDIndexList: a list of integers corresponding to which densities to plot the tables at

wireFrame: if set to true (the default) and logDIndexList is set to None it will plot a 3D wireframe of logP.

References eos\_interp.eosTable.logD, eos\_interp.eosTable.logP, and eos\_interp.eosTable.logT.

**8.11.3.5** `def eos_interp.eosTable.write( self, args )`

Generic write function that calls either writeToScreen, or writeToFiel depending on if a file name is specified or not.

References eos\_interp.eosTable.\_\_writeToFile(), and eos\_interp.eosTable.\_\_writeToScreen().

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

- scripts/eos\_interp.py

## 8.12 eos\_interp.eosTableManager Class Reference

### Public Member Functions

- def [load](#)
- def [interpComp](#)
- def [plotGrid](#)
- def [getTableFromComp](#)
- def [\\_\\_init\\_\\_](#)

### 8.12.1 Detailed Description

Manages equation of state files, including how they are interpolated between.

### 8.12.2 Constructor & Destructor Documentation

#### 8.12.2.1 def eos\_interp.eosTableManager.\_\_init\_\_( self, eosFileName = None )

Returns a new instance of eosTableManager.

if eosFileName is set it will call `__initFromFile` to load settings from a file to initialize the new eosTableManager.

References `eos_interp.eosTableManager.__initFromFile()`, `eos_interp.eosTableManager.__quad()`, `eos_interp.eosTableManager.__quadInterpInZ()`, `eos_interp.eosTableManager.eosFileName`, `eos_interp.eosTableManager.eosTables`, `eos_interp.eosTable.X`, `eos_interp.opacityTable.X`, `eos_interp.opacityTableManager.X`, `eos_interp.eosTableManager.X`, `eos_interp.eosTable.Z`, `eos_interp.opacityTable.Z`, `eos_interp.opacityTableManager.Z`, and `eos_interp.eosTableManager.Z`.

### 8.12.3 Member Function Documentation

#### 8.12.3.1 def eos\_interp.eosTableManager.getTableFromComp( self, X, Z )

Returns a shallow copy of the eos table with matching composition. If none found it returns None.

References `eos_interp.eosTableManager.eosTables`.

#### 8.12.3.2 def eos\_interp.eosTableManager.interpComp( self, X, Z )

Interpolates a set of eos files and opacities to the desired X and Z, and returns an eosManager with this new set of files which can then be interpolated to the desired rho and T's.

References eos\_interp.opacityTableManager.\_\_cubicSplineInX(), eos\_interp.eosTableManager.\_\_cubicSplineInX(), eos\_interp.eosTableManager.\_\_quadInterpInZ(), eos\_interp.eosTable.X, eos\_interp.opacityTable.X, eos\_interp.opacityTableManager.X, eos\_interp.eosTableManager.X, eos\_interp.eosTable.Z, eos\_interp.opacityTable.Z, eos\_interp.opacityTableManager.Z, and eos\_interp.eosTableManager.Z.

#### 8.12.3.3 def eos\_interp.eosTableManager.load( self )

Loads eos files.

Sets the following:

self.Z: a list of Z (metal mass fraction) values of the equation of state files  
 self.X: a list of X (hydrogen mass fraction) values of the equation of state files

References eos\_interp.eosTableManager.eosTables, eos\_interp.eosTable.X, eos\_interp.opacityTable.X, eos\_interp.opacityTableManager.X, eos\_interp.eosTableManager.X, eos\_interp.eosTable.Z, eos\_interp.opacityTable.Z, eos\_interp.opacityTableManager.Z, and eos\_interp.eosTableManager.Z.

#### 8.12.3.4 def eos\_interp.eosTableManager.plotGrid( self, eosIndex )

Plot rho and T points that form the grid

References eos\_interp.eosTableManager.eosTables.

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

- scripts/eos\_interp.py

## 8.13 exception2 Class Reference

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

- src/exception2.h
- src/exception2.cpp

## 8.14 plot\_2DSlices.File2DSlice Class Reference

### Public Member Functions

- def [load](#)

#### 8.14.1 Member Function Documentation

#### 8.14.1.1 `def plot_2DSlices.File2DSlice.load( self, fileName )`

```
sets:
fileName, file name of the 2D slice
planeType, type of the 2D slice ("rt", "rp", "tp")
eosFile, file name of the equation of state file, if using a gamma-law gas it is None
gamma, value of gamma for a gamma-law gas, if using an equation of state table it is None
coordinateNames, Names of the coordinates
coordinates, values of the coordinates
dataNames, names of the data columns
data, the data columns
```

References `plot_2DSlices.File2DSlice.coordinateNames`, `plot_2DSlices.File2DSlice.coordinates`, `plot_2DSlices.File2DSlice.data`, `make_hdf.hdfFile.data`, `make_hdf2.fileSet.data`, `plot_2DSlices.File2DSlice.dataNames`, `make_hdf2.fileSet.dataNames`, `plot_2DSlices.File2DSlice.eosFile`, `light_curve.LightCurve.eosFile`, `dump.dump.fileName`, `plot_2DSlices.File2DSlice.fileName`, `plot_2DSlices.File2DSlice.gamma`, `dump.dump.gamma`, `plot_2DSlices.File2DSlice.index`, `main()`, `plot_2DSlices.File2DSlice.planeType`, `plot_2DSlices.File2DSlice.time`, `light_curve.LightCurve.time`, `dump.dump.time`, and `Global.time`.

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

- `scripts/plot_2DSlices.py`

## 8.15 `make_hdf.fileSet` Class Reference

### Public Member Functions

- `def __init__`
- `def makeHDFFiles`
- `def convertDumpToHDF`

#### 8.15.1 Constructor & Destructor Documentation

##### 8.15.1.1 `def make_hdf.fileSet.__init__( self, element )`

Initialize an fileSet from an xml node

References `make_hdf.fileSet.__checkSupportedNodeAttributes()`, `make_hdf.fileSet.__setSupportedNodeAttributes()`, `plot_profile.DataSet.baseFileName`, `plot_profile.DataSet.end`, `make_hdf.fileSet.fileRange`, `make_hdf.hdfFile.interpVars`, `make_hdf.fileSet.interpVars`, `plot_profile.DataSet.start`, `make_hdf.fileSet.supportedNodeAttributes`, `make_hdf.fileSet.timeFile`, `make_hdf.hdfFile.variables`, and `make_hdf.fileSet.variables`.

#### 8.15.2 Member Function Documentation

8.15.2.1 `def make_hdf.fileSet.convertDumpToHDF( self, dump )`

Converts a dump ifle to an hdf file formated in the way sepcified in the xml configuration file

References `make_hdf.hdfFile.interpVars`, `make_hdf.fileSet.interpVars`, `main()`, `make_hdf.hdfFile.variables`, and `make_hdf.fileSet.variables`.

Referenced by `make_hdf2.fileSet.makeHDFFiles()`, and `make_hdf.fileSet.makeHDFFiles()`.

8.15.2.2 `def make_hdf.fileSet.makeHDFFiles( self, options )`

Makes HDF files specified by settings

References `plot_profile.DataSet.baseFileName`, `make_hdf.fileSet.convertDumpToHDF()`, `plot_profile.DataSet.end`, and `plot_profile.DataSet.start`.

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

- `scripts/make_hdf.py`

## 8.16 make\_hdf2.fileSet Class Reference

## Public Member Functions

- `def __init__`
- `def makeHDFFiles`
- `def convertDumpToHDF`

## 8.16.1 Constructor &amp; Destructor Documentation

8.16.1.1 `def make_hdf2.fileSet.__init__( self, element )`

Initialize an fileSet from an xml node

References `make_hdf2.fileSet.__checkSuppotedNodeAttributes()`, `make_hdf.fileSet.___checkSuppotedNodeAttributes()`, `make_hdf2.fileSet.__setSupportedNodeAttributes()`, `make_hdf.fileSet.__setSupportedNodeAttributes()`, `plot_profile.DataSet.baseFileName`, `plot_profile.DataSet.end`, `make_hdf2.fileSet.fileRange`, `make_hdf.fileSet.fileRange`, `make_hdf2.fileSet.frequency`, `light_curve.LightCurve.frequency`, `make_hdf2.fileSet.includeBoundaries`, `make_hdf2.fileSet.numRInterp`, `make_hdf2.fileSet.outputPath`, `make_hdf2.fileSet.radialCutZone`, `plot_profile.DataSet.start`, `make_hdf2.fileSet.supportedNodeAttributes`, `make_hdf.fileSet.supportedNodeAttributes`, `make_hdf2.fileSet.timeFile`, and `make_hdf.fileSet.timeFile`.

## 8.16.2 Member Function Documentation

### 8.16.2.1 `def make_hdf2.fileSet.convertDumpToHDF( self, dump )`

Converts a dump ifle to an hdf file formatted in the way sepcified in the xml configuration file

References `make_hdf.hdfFile.__interpolateLinearIn1DI()`, `make_hdf2.fileSet.__interpolateLinearIn1DI()`, `make_hdf.hdfFile.data`, `make_hdf2.fileSet.data`, `make_hdf2.fileSet.dataIDs`, `make_hdf.hdfFile.dataMax`, `make_hdf2.fileSet.dataMax`, `make_hdf.hdfFile.dataMin`, `make_hdf2.fileSet.dataMin`, `make_hdf2.fileSet.dataNames`, `make_hdf2.fileSet.dataShape`, `make_hdf2.fileSet.getDataFromDump()`, `make_hdf2.fileSet.includeBoundaries`, `main()`, `make_hdf2.fileSet.numRInterp`, `make_hdf2.fileSet.outputPath`, `make_hdf2.fileSet.radialCutZone`, and `make_hdf2.fileSet.setAdditionalVariables()`.

Referenced by `make_hdf2.fileSet.makeHDFFiles()`.

### 8.16.2.2 `def make_hdf2.fileSet.makeHDFFiles( self, options )`

Makes HDF files specified by settings

References `plot_profile.DataSet.baseFileName`, `make_hdf2.fileSet.convertDumpToHDF()`, `make_hdf.fileSet.convertDumpToHDF()`, `plot_profile.DataSet.end`, `make_hdf2.fileSet.frequency`, `light_curve.LightCurve.frequency`, `plot_profile.DataSet.start`, `make_hdf2.fileSet.timeFile`, and `make_hdf.fileSet.timeFile`.

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

- `scripts/make_hdf2.py`

## 8.17 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 &)`

### 8.17.1 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.

### 8.17.2 Constructor & Destructor Documentation

#### 8.17.2.1 Functions::Functions( )

Constructor for the class `Functions`.

References `fpCalculateAveDensities`, `fpCalculateDeltat`, `fpCalculateNewAV`, `fpCalculateNewDensities`, `fpCalculateNewEnergies`, `fpCalculateNewEOSVars`, `fpCalculateNewGridVelocities`, `fpCalculateNewRadii`, and `fpCalculateNewVelocities`.

### 8.17.3 Member Data Documentation

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

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

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

#### 8.17.3.2 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()`.

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

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

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

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

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

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

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

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

Referenced by main(), and setMainFunctions().

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

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

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

**8.17.3.7 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().

**8.17.3.8 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().

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

Function pointer to the function used to calculate new radii.

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



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

Function pointer to the function used to calculate new velocities.

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

**8.17.3.11** `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()`.

**8.17.3.12** `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()`.

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

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

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

**8.17.3.14** `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:

- [src/SPHERLS/global.h](#)
- [src/SPHERLS/global.cpp](#)

## 8.18 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

### 8.18.1 Detailed Description

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

### 8.18.2 Constructor & Destructor Documentation

#### 8.18.2.1 `Global::Global ( )`

Constructor for the class [Global](#).

### 8.18.3 Member Data Documentation

#### 8.18.3.1 Functions `Global::functions`

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

Referenced by `main()`.

#### 8.18.3.2 Grid `Global::grid`

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

Referenced by `plot_file.Plot::__init__()`, `plot_profile.Plot::__init__()`, `plot_file.Axis::__init__()`, `plot_profile.Axis::__init__()`, and `main()`.

#### 8.18.3.3 Implicit `Global::implicit`

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

Referenced by `main()`.

#### 8.18.3.4 MessPass Global::messPass

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

Referenced by `main()`.

#### 8.18.3.5 Output Global::output

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

Referenced by `main()`.

#### 8.18.3.6 Parameters Global::parameters

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

Referenced by `main()`.

#### 8.18.3.7 Performance Global::performance

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

Referenced by `main()`.

#### 8.18.3.8 ProcTop Global::procTop

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

Referenced by `main()`.

#### 8.18.3.9 Time Global::time

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

Referenced by `light_curve.LightCurve::calculateCurve()`, `plot_2DSlices.File2DSlice::load()`, `main()`, `dump.dump::printHeader()`, `dump.dump::readHeaderAscii()`, `dump.-dump::readHeaderBinary()`, and `light_curve.LightCurve::readProfiles()`.

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

- [src/SPHERLS/global.h](#)
- [src/SPHERLS/global.cpp](#)

## 8.19 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]

### 8.19.1 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.

## 8.19.2 Constructor & Destructor Documentation

### 8.19.2.1 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.

### 8.19.3 Member Data Documentation

#### 8.19.3.1 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\(\)](#).

#### 8.19.3.2 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()`.

### 8.19.3.3 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().

#### 8.19.3.4 int Grid::nCotThetaJK

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().

#### 8.19.3.5 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().

#### 8.19.3.6 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().

#### 8.19.3.7 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\(\)](#).

#### 8.19.3.8 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().

### 8.19.3.9 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().

### 8.19.3.10 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().

8.19.3.11 `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()`.

8.19.3.12 `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()`.

8.19.3.13 `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-`

#### 8.19.3.14 int Grid::nEddyVisc

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

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().

#### 8.19.3.16 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().

#### 8.19.3.17 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\_RT\_P(), 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(), setDEDM-Clamp(), and updateOldGrid().



### 8.19.3.18 `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()`.

### 8.19.3.19 `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()`.

### 8.19.3.20 `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()`.

### 8.19.3.21 `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()`.



8.19.3.22 `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()`.

8.19.3.23 `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()`.

8.19.3.24 `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()`.

8.19.3.25 `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()`.

8.19.3.26 `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()`.

#### 8.19.3.27 `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()`.

#### 8.19.3.28 `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()`.

#### 8.19.3.29 `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()`.

8.19.3.30 `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()`.

8.19.3.31 `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()`.

8.19.3.32 `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().

#### 8.19.3.33 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().

#### 8.19.3.34 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().

#### 8.19.3.35 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\(\)](#).

#### 8.19.3.36 int Grid::nSinThetaJK

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().

#### 8.19.3.37 int Grid::nSinThetaJp1halfK

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().

#### 8.19.3.38 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().

**8.19.3.39 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\(\)](#).

**8.19.3.40 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\(\)](#).

**8.19.3.41 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()`.

#### 8.19.3.42 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()`.

#### 8.19.3.43 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()`.

#### 8.19.3.44 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().

#### 8.19.3.45 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().

#### 8.19.3.46 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().

#### 8.19.3.47 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()`.

#### 8.19.3.48 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:

- [src/SPHERLS/global.h](#)
- [src/SPHERLS/global.cpp](#)

## 8.20 make\_hdf.hdfFile Class Reference

### Public Member Functions

- [def write](#)

### 8.20.1 Member Function Documentation

#### 8.20.1.1 [def make\\_hdf.hdfFile.write\( self \)](#)

this function writes the data specified in the configuration file to a new hdf file. It does this by interpolating where nessacary to get data at the right location

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

- [scripts/make\\_hdf.py](#)

## 8.21 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](#)

### 8.21.1 Detailed Description

This class holds data required for the implicit calculation.

### 8.21.2 Constructor & Destructor Documentation

#### 8.21.2.1 Implicit::Implicit ( )

constructor the the class [Implicit](#).

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

### 8.21.3 Member Data Documentation

#### 8.21.3.1 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\(\)](#).

#### 8.21.3.2 double Implicit::dCurrentRelTErr

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\(\)](#).

### 8.21.3.3 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().

### 8.21.3.4 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 coeffecient matrix, and the actual RHS. This value is the maximum from all values at each iteration of the solution, from each time step since the last model dump.

Referenced by fin(), Implicit(), implicitSolve\_R(), implicitSolve\_RT(), implicitSolve\_RTP(), and main().

### 8.21.3.5 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().

### 8.21.3.6 KSP Implicit::kspContext

PETSc solver context.

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

### 8.21.3.7 Mat Implicit::matCoeff

Parallel coeffecient matrix (spread across all processors)

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

### 8.21.3.8 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().

### 8.21.3.9 `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()`.

### 8.21.3.10 `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()`.

### 8.21.3.11 `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()`.

### 8.21.3.12 `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()`.

### 8.21.3.13 `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()`.

**8.21.3.14 int Implicit::nNumImplicitZones**

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

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

**8.21.3.15 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()`.

**8.21.3.16 int Implicit::nNumRowsALocalSB**

The number or 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()`.

**8.21.3.17 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()`.

**8.21.3.18 Vec Implicit::vecRHS**

RHS vector (spread across all processors)

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

**8.21.3.19 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()`.

### 8.21.3.20 Vec Implicit::vecTCorrections

Temperature corrections solution vector (spread across all processors)

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

### 8.21.3.21 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:

- [src/SPHERLS/global.h](#)
- [src/SPHERLS/global.cpp](#)

## 8.22 eos\_interp.interpTable Class Reference

### Public Member Functions

- def [interpolate](#)
- def [read](#)
- def [plotLogE](#)
- def [plotLogP](#)
- def [plotLogK](#)
- def [\\_\\_init\\_\\_](#)

### 8.22.1 Detailed Description

This class reads in and holds data for an equations of state and opacities from a file format in the same was as read to and written by the class defined in `eos.h`, and implemented in `eos`.

### 8.22.2 Constructor & Destructor Documentation

#### 8.22.2.1 def eos\_interp.interpTable.\_\_init\_\_( self, tableElement=None )

Reads in an interpolation table info from from the xml element `tableElement`.

References `Parameters.eosTable`, `eos_interp.interpTable.eosTable`, `eos_interp.interpTable.gridConfig`, `eos_interp.interpTable.opacityTable`, `eos_interp.interpTable.outputFile`, `eos_interp.interpTable.plot`, `eos_interp.interpTable.setNans`, `eos_interp.eosTable.X`, `eos_interp.opacityTable.X`, `eos_interp.opacityTableManager.X`, `eos_interp.eosTableManager.X`, `eos_interp.interpTable.X`, `eos_interp.eosTable.Z`, `eos_interp.opacityTable.Z`, `eos_interp.opacityTableManager.Z`, `eos_interp.eosTableManager.Z`, and `eos_interp.interpTable.Z`.



### 8.22.3 Member Function Documentation

**8.22.3.1** `def eos_interp.interpTable.interpolate( self, eosSet, opacitySet, withoutNans=None )`

creates the interpolated table and writes it out

References eos\_interp.interpTable.\_\_writeCompleteEOS(), eos\_interp.interpTable.eosAtNewComp, Parameters.eosTable, eos\_interp.interpTable.eosTable, eos\_interp.interpTable.gridConfig, eos\_interp.interpTable.opacityAtNewComp, eos\_interp.interpTable.opacityTable, eos\_interp.interpTable.outputFile, eos\_interp.interpTable.plot, eos\_interp.interpTable.setNans, eos\_interp.eosTable.X, eos\_interp.opacityTable.X, eos\_interp.opacityTableManager.X, eos\_interp.eosTableManager.X, eos\_interp.interpTable.X, eos\_interp.eosTable.Z, eos\_interp.opacityTable.Z, eos\_interp.opacityTableManager.Z, eos\_interp.eosTableManager.Z, and eos\_interp.interpTable.Z.

**8.22.3.2** `def eos_interp.interpTable.plotLogE( self, otherTables=None, logDIndexList=None, logDRangeList=None, wireFrame=True, rstride=1, cstride=1, outputfile=None )`

Plots LogE

Keywords:

otherTables: a list of other eosTables to include in the plot

logDIndexList: a list of integers corresponding to which densities to plot the tables at

wireFrame: if set to true (the default) and logDIndexList is set to None it will plot a 3D wireframe of logE.

References eos\_interp.eosTable.logD, eos\_interp.interpTable.logD, eos\_interp.eosTable.logE, eos\_interp.interpTable.logE, eos\_interp.eosTable.logT, eos\_interp.opacityTable.logT, eos\_interp.interpTable.logT, datafile.DataFile.sFileName, eos\_interp.eosTable.sFileName, eos\_interp.opacityTable.sFileName, and eos\_interp.interpTable.sFileName.

**8.22.3.3** `def eos_interp.interpTable.plotLogK( self, otherTables=None, logDIndexList=None, logDRangeList=None, wireFrame=True, outputfile=None )`

Plots opacity

Keywords:

otherTables: a list of opacity tables to also be plotted

logDIndex: a list of integers used to indicate a specific logR index to plot 2D line plots at.

References eos\_interp.eosTable.logD, eos\_interp.interpTable.logD, eos\_interp.opacityTable.logK, eos\_interp.interpTable.logK, eos\_interp.eosTable.logT, eos\_interp.opacityTable.logT, eos\_interp.interpTable.logT, datafile.DataFile.sFileName, eos\_interp.eosTable.sFileName, eos\_interp.opacityTable.sFileName, and eos\_interp.interpTable.sFileName.

**8.22.3.4** `def eos_interp.interpTable.plotLogP( self, otherTables = None, logDIndexList = None, logDRangeList = None, wireFrame = True, outputfile = None )`

Plots LogP

Keywords:

`otherTables`: a list of other eosTables to include in the plot  
`logDIndexList`: a list of integers corresponding to which densities to plot the tables at  
`wireFrame`: if set to true (the default) and `logDIndexList` is set to None it will plot a 3D wireframe of logP.

References `eos_interp.eosTable.logD`, `eos_interp.interpTable.logD`, `eos_interp.eosTable.logP`, `eos_interp.interpTable.logP`, `eos_interp.eosTable.logT`, `eos_interp.opacityTable.logT`, `eos_interp.interpTable.logT`, `datafile.DataFile.sFileName`, `eos_interp.eosTable.sFileName`, `eos_interp.opacityTable.sFileName`, and `eos_interp.interpTable.sFileName`.

**8.22.3.5** `def eos_interp.interpTable.read( self, sFilename )`

Reads in an interpolated table

References `eos_interp.interpTable.gridConfig`, `eos_interp.eosTable.logD`, `eos_interp.interpTable.logD`, `eos_interp.eosTable.logE`, `eos_interp.interpTable.logE`, `eos_interp.opacityTable.logK`, `eos_interp.interpTable.logK`, `eos_interp.eosTable.logP`, `eos_interp.interpTable.logP`, `eos_interp.eosTable.logT`, `eos_interp.opacityTable.logT`, `eos_interp.interpTable.logT`, `eos_interp.interpTable.numLogR`, `datafile.DataFile.sFileName`, `eos_interp.eosTable.sFileName`, `eos_interp.opacityTable.sFileName`, `eos_interp.interpTable.sFileName`, `eos_interp.eosTable.X`, `eos_interp.opacityTable.X`, `eos_interp.opacityTableManager.X`, `eos_interp.eosTableManager.X`, `eos_interp.interpTable.X`, `eos_interp.eosTable.Z`, `eos_interp.opacityTable.Z`, `eos_interp.opacityTableManager.Z`, `eos_interp.eosTableManager.Z`, and `eos_interp.interpTable.Z`.

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

- `scripts/eos_interp.py`

## 8.23 eos\_interp.interpTableManager Class Reference

### Public Member Functions

- `def createTables`
- `def __init__`

#### 8.23.1 Constructor & Destructor Documentation

**8.23.1.1** `def eos_interp.interpTableManager.__init__( self, configFile = None )`

Initializes `interpTableManager` from the given configuration file.

References eos\_interp.interpTableManager.\_\_readInterpTableConfigs(), eos\_interp.interpTableManager.configFile, eos\_interp.interpTableManager.eosSet, main(), eos\_interp.interpTableManager.opacitySet, and eos\_interp.interpTableManager.tables.

### 8.23.2 Member Function Documentation

**8.23.2.1** `def eos_interp.interpTableManager.createTables( self, withoutNans = None )`

Creates interpolated tables and write them out.

References eos\_interp.interpTableManager.eosSet, eos\_interp.interpTableManager.opacitySet, and eos\_interp.interpTableManager.tables.

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

- scripts/eos\_interp.py

## 8.24 make\_hdf.interpVar Class Reference

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

- scripts/make\_hdf.py

## 8.25 light\_curve.LightCurve Class Reference

### Public Member Functions

- `def readProfiles`
- `def readBoloCorr`
- `def calculateCurve`
- `def write`

### 8.25.1 Member Function Documentation

**8.25.1.1** `def light_curve.LightCurve.calculateCurve( self )`

Creates the light curve by converting luminosity to bolometric magnitude and then applying a bolometric correction and returns a 2D list of times and light curve magnitudes.

References light\_curve.LightCurve.BC, light\_curve.LightCurve.gridVelocity, light\_curve.LightCurve.interiorMass, light\_curve.LightCurve.loggDel, light\_curve.LightCurve.loggMin, light\_curve.LightCurve.luminosity, light\_curve.LightCurve.numLogg, light\_curve.LightCurve.numT, light\_curve.LightCurve.radius, light\_curve.LightCurve.TDel, light\_curve.LightCurve.temperature, light\_curve.LightCurve.time, dump.dump.time, Global.time, light\_curve.LightCurve.TMin, and light\_curve.LightCurve.withAcceleration.

### 8.25.1.2 `def light_curve.LightCurve.readBoloCorr( self )`

Reads in the bolometric correction table

References `light_curve.LightCurve.BC`, `light_curve.LightCurve.boloCorrFile`, `light_curve.LightCurve.columnBC`, `light_curve.LightCurve.loggDel`, `light_curve.LightCurve.loggMin`, `light_curve.LightCurve.numLogg`, `light_curve.LightCurve.numT`, `light_curve.LightCurve.TDel`, and `light_curve.LightCurve.TMin`.

### 8.25.1.3 `def light_curve.LightCurve.readProfiles( self, options )`

Reads the needed data to create the light curve from the radial profile files

References `plot_profile.DataSet.baseFileName`, `plot_profile.DataSet.end`, `light_curve.LightCurve.eosFile`, `light_curve.LightCurve.frequency`, `light_curve.LightCurve.gridVelocity`, `light_curve.LightCurve.interiorMass`, `light_curve.LightCurve.luminosity`, `light_curve.LightCurve.nNumFiles`, `light_curve.LightCurve.radius`, `plot_profile.DataSet.start`, `light_curve.LightCurve.temperature`, `light_curve.LightCurve.time`, `dump.dump.time`, `Global.time`, and `light_curve.LightCurve.zonesFromSurf`.

### 8.25.1.4 `def light_curve.LightCurve.write( self, curve )`

Writes out the light curve to the specified output file.

References `main()`, `light_curve.LightCurve.outputFile`, and `eos_interp.interpTable.outputFile`.

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

- `scripts/light_curve.py`

## 8.26 MDeltaDelta Struct Reference

The documentation for this struct was generated from the following file:

- `src/SPHERLSgen/main.h`

## 8.27 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](#)

### 8.27.1 Detailed Description

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

### 8.27.2 Constructor & Destructor Documentation

#### 8.27.2.1 `MessPass::MessPass( )`

Constructor for class [MessPass](#).

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

### 8.27.3 Member Data Documentation

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

Message handles.

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

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

Message handles.

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

#### 8.27.3.3 `MPI::Status* MessPass::statusRecv`

Message status.

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

#### 8.27.3.4 `MPI::Status*` `MessPass::statusSend`

Message status.

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

#### 8.27.3.5 `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()`.

#### 8.27.3.6 `MPI::Datatype*` `MessPass::typeRecvOldGrid`

Recv data types for entire grid. It is of size `ProcTop::nNumNeighbors`.

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

#### 8.27.3.7 `MPI::Datatype*` `MessPass::typeSendNewGrid`

Send data types for entire grid. It is of size `ProcTop::nNumNeighbors`.

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

#### 8.27.3.8 `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:

- `src/SPHERLS/global.h`
- `src/SPHERLS/global.cpp`

## 8.28 NextToken Struct Reference

The documentation for this struct was generated from the following file:

- `src/xmlParser.cpp`

## 8.29 eos\_interp.opacityTable Class Reference

### Public Member Functions

- def [load](#)
- def [plotLogK](#)
- def [interpolate](#)
- def [\\_\\_init\\_\\_](#)
- def [fillInDepNans](#)

### 8.29.1 Detailed Description

Holds opacity table data.

Initialize with a composition (X,Z), file name and weather the file name contains multiple.

### 8.29.2 Constructor & Destructor Documentation

**8.29.2.1** `def eos_interp.opacityTable.__init__( self, X=None, Z=None, sFileName=None, multitableFile=None )`

Initializes the opacity object.

```
sets:
self.X: the hydrogen mass fraction
self.Z: the metal mass fraction
self.sFileName: the file name to load the table from
self.multitableFile: weather or not the file has more than one table in it
```

References eos\_interp.opacityTable.logK, eos\_interp.opacityTable.logR, eos\_interp.eosTable.logT, eos\_interp.opacityTable.logT, eos\_interp.opacityTable.multitableFile, datafile.DataFile.sFileName, eos\_interp.eosTable.sFileName, eos\_interp.opacityTable.sFileName, eos\_interp.eosTable.X, eos\_interp.opacityTable.X, eos\_interp.eosTable.Z, and eos\_interp.opacityTable.Z.

### 8.29.3 Member Function Documentation

**8.29.3.1** `def eos_interp.opacityTable.fillInDepNans( self )`

Fills in logR and logT values to make a rectangular grid

References eos\_interp.opacityTable.logR, eos\_interp.eosTable.logT, and eos\_interp.opacityTable.logT.

**8.29.3.2** `def eos_interp.opacityTable.interpolate( self, gridConfig, setExtrapolatedToNan=True )`

Interpolate from self's table to the griding specified by:

parameters:  
 logDMin: first (smallest) logD value of grid  
 logDDel: spacing in logD  
 numLogD: number of logD grid points  
 logTMin: first (smallest) logT value of grid  
 logTDel: spacing in logT  
 numLogT: number of logT grid points

keyword:  
 setExtrapolatedToNan: controls whether extrapolated points are set to nans (default is True)

returns:  
 an opacity table interpolated to the specified grid. In addition to the regular members of an opacity table logD is also included.

References eos\_interp.eosTable.\_\_fillDepNans(), eos\_interp.opacityTable.\_\_fillDepNans(), eos\_interp.opacityTable.logK, eos\_interp.opacityTable.logR, eos\_interp.eosTable.logT, eos\_interp.opacityTable.logT, eos\_interp.eosTable.X, eos\_interp.opacityTable.X, eos\_interp.eosTable.Z, and eos\_interp.opacityTable.Z.

### 8.29.3.3 def eos\_interp.opacityTable.load( self )

Load from a file an opacity table for composition of the current opacity object. It does this by advancing a file until the composition is matched and then calls \_\_loadTableFromFile to load the logR, logT, and logK values.

References eos\_interp.opacityTable.\_\_loadTableFromFile(), eos\_interp.opacityTable.\_multitableFile, datafile.DataFile.sFileName, eos\_interp.eosTable.sFileName, eos\_interp.opacityTable.sFileName, eos\_interp.eosTable.X, eos\_interp.opacityTable.X, eos\_interp.eosTable.Z, and eos\_interp.opacityTable.Z.

### 8.29.3.4 def eos\_interp.opacityTable.plotLogK( self, otherTables = None, logRIndex = None, wireFrame = True )

Plots opacity

Keywords:  
 otherTables: a list of opacity tables to also be plotted  
 logRIndex: a list of integers used to indicate a specific logR index to plot 2D line plots at

References eos\_interp.opacityTable.logK, eos\_interp.opacityTable.logR, eos\_interp.eosTable.logT, and eos\_interp.opacityTable.logT.

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

- scripts/eos\_interp.py

## 8.30 eos\_interp.opacityTableManager Class Reference



## Public Member Functions

- def [load](#)
- def [interpComp](#)
- def [plotGrids](#)
- def [getTableFromComp](#)
- def [\\_\\_init\\_\\_](#)

### 8.30.1 Detailed Description

Manages opacity files, including how they are interpolated between in composition.

### 8.30.2 Constructor & Destructor Documentation

**8.30.2.1** `def eos_interp.opacityTableManager.__init__( self, opacityConfigFile = None )`

Creates a new instance of opacityTableManager.

If opacityConfigFile is set it will try to parse it for xml settings to get all the file names of the opacity files to include in the opacityTableManager.

References eos\_interp.opacityTableManager.\_\_getCompositions(), eos\_interp.opacityTableManager.\_\_merge2files(), eos\_interp.opacityTableManager.opacityConfigFileName, eos\_interp.opacityTableManager.opacityFileNames, eos\_interp.opacityTableManager.opacityTables, eos\_interp.eosTable.X, eos\_interp.opacityTable.X, eos\_interp.opacityTableManager.X, eos\_interp.eosTable.Z, eos\_interp.opacityTable.Z, and eos\_interp.opacityTableManager.Z.

### 8.30.3 Member Function Documentation

**8.30.3.1** `def eos_interp.opacityTableManager.getTableFromComp( self, X, Z )`

Returns a shallow copy of the opacity table with matching composition.

References eos\_interp.opacityTableManager.opacityTables.

**8.30.3.2** `def eos_interp.opacityTableManager.interpComp( self, X, Z )`

Interpolates a set of opacity files to the desired X and Z, and returns an the interpolated opacityTable.

Parameters:  
X: hydrogen mass fraction  
Z: metal mass fraction

References eos\_interp.opacityTableManager.\_\_bicubicSplineInXZ(), eos\_interp.eosTable.X, eos\_interp.opacityTable.X, eos\_interp.opacityTableManager.X, eos\_interp.eosTable.Z, eos\_interp.opacityTable.Z, and eos\_interp.opacityTableManager.Z.

### 8.30.3.3 `def eos_interp.opacityTableManager.load( self )`

Loads opacity files and merge files at duplicate compositions (i.e. merges low and high temperature opacity tables).

Sets the following:

`self.X`: list of hydrogen mass fractions covered by opacity tables  
`self.Z`: list of metal mass fractions covered by opacity tables

References `eos_interp.opacityTableManager.__merge()`, `eos_interp.opacityTableManager.__setCompLists()`, and `eos_interp.opacityTableManager.opacityTables`.

### 8.30.3.4 `def eos_interp.opacityTableManager.plotGrids( self, opacityIndex )`

Plot LogR and LogT points that form the opacity grid.

Parameters:

`opacityIndex`: a list of integers used to select which opacity tables will be plotted

References `eos_interp.opacityTableManager.opacityTables`.

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

- `scripts/eos_interp.py`

## 8.31 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](#)

### 8.31.1 Detailed Description

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

### 8.31.2 Constructor & Destructor Documentation

#### 8.31.2.1 `Output::Output ( )`

Constructor for this class.

References `bDump`, `nDumpFrequencyStep`, `nNumTimeStepsSinceLastPrint`, `ofWatchZoneFiles`, and `sBaseOutputFileName`.

### 8.31.3 Member Data Documentation

#### 8.31.3.1 `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()`.

#### 8.31.3.2 `bool Output::bPrint`

Should status updates be printed to the screen.

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

#### 8.31.3.3 `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()`.

#### 8.31.3.4 `double Output::dPrintFrequencyTime`

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

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

#### 8.31.3.5 `double Output::dTimeLastDump`

The simulation time at which the last dump was made using the `Output::dDumpFrequencyTime` criterion.

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

#### 8.31.3.6 `double Output::dTimeLastPrint`

Simulation time when last status was printed.

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

#### 8.31.3.7 `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()`.

#### 8.31.3.8 `int Output::nNumTimeStepsSinceLastPrint`

The number of time steps since the last model dump.

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

#### 8.31.3.9 `int Output::nPrintFrequencyStep`

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

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

#### 8.31.3.10 `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()`.

#### 8.31.3.11 `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()`.

8.31.3.12 `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()`.

8.31.3.13 `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:

- [src/SPHERLS/global.h](#)
- [src/SPHERLS/global.cpp](#)

## 8.32 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 [dDEDMClampValue](#)
- double [dDEDMClampMr](#)
- double [dEDMClampTemperature](#)
- bool [bDEDMClamp](#)
- std::string [sDebugProfileOutput](#)

### 8.32.1 Detailed Description

This class holds parameters and constants used for calculation.

### 8.32.2 Constructor & Destructor Documentation

#### 8.32.2.1 `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](#).

### 8.32.3 Member Data Documentation

#### 8.32.3.1 `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\(\)](#).

#### 8.32.3.2 `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\(\)](#).

**8.32.3.3 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()`.

**8.32.3.4 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()`.

**8.32.3.5 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()`.

**8.32.3.6 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()`.

**8.32.3.7 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()`.

### 8.32.3.8 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()`.

### 8.32.3.9 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()`, `initDonorFracAndMaxConVel_RT_TEOS()`, `initDonorFracAndMaxConVel_RTP_GL()`, `initDonorFracAndMaxConVel_RTP_TEOS()`, `Parameters()`, and `setDEDMClamp()`.

### 8.32.3.10 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()`.

### 8.32.3.11 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()`.



**8.32.3.12 double Parameters::dEDMClampTemperature**

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

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

**8.32.3.13 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()`.

**8.32.3.14 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()`.

**8.32.3.15 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()`.

**8.32.3.16 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()`.

**8.32.3.17 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().

**8.32.3.18 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 LUVD11 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().

**8.32.3.19 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().

**8.32.3.20 double Parameters::dTolerance**

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

Referenced by calNewTPKappaGamma\_TEOS(), and init().

**8.32.3.21 eos Parameters::eosTable**

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

Referenced by `eos_interp.interpTable::__init__()`, `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()`, `init()`, and `eos_interp.interpTable::interpolate()`.

**8.32.3.22 int Parameters::nMaxIterations**

The maximum number of iterations to try to get the the relative error in the temperture below [parameters::dTolerance](#).

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

**8.32.3.23 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()`.

**8.32.3.24 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()`.

**8.32.3.25 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:

- [src/SPHERLS/global.h](#)
- [src/SPHERLS/global.cpp](#)

### 8.33 work\_plot.PdVPlotSettings Class Reference

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

- scripts/work\_plot.py

### 8.34 Performance Class Reference

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

#### Public Member Functions

- [Performance](#) ()

#### Public Attributes

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

#### 8.34.1 Detailed Description

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

#### 8.34.2 Constructor & Destructor Documentation

##### 8.34.2.1 Performance::Performance ( )

Constructor for the class [Performance](#).

References [dEndTimer](#), and [dStartTimer](#).

#### 8.34.3 Member Data Documentation

##### 8.34.3.1 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\(\)](#).

## 8.34.3.2 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:

- [src/SPHERLS/global.h](#)
- [src/SPHERLS/global.cpp](#)

## 8.35 plot\_file.Plot Class Reference

## Public Member Functions

- [def \\_\\_init\\_\\_](#)
- [def load](#)

## 8.35.1 Detailed Description

This class holds all the information for a single plot, namely the list of curves for that plot.

## 8.35.2 Constructor &amp; Destructor Documentation

## 8.35.2.1 def plot\_file.Plot.\_\_init\_\_( self, element )

This method initlizes the plot object

References `plot_file.Plot.bMinorTics`, `plot_file.Plot.curves`, `plot_file.Plot.grid`, `Global.-grid`, `plot_file.Plot.legendloc`, `plot_file.Plot.limits`, `plot_file.Plot.numpoints`, `plot_file.Plot.-texts`, `plot_file.Plot.ticks`, `plot_file.Plot.weightHeight`, and `plot_file.Plot.ylabel`.

## 8.35.3 Member Function Documentation

## 8.35.3.1 def plot\_file.Plot.load( self, files, options )

loads the data for a plot, y-data is stored in the curves, and sets the ylabel from the first file read in

References `plot_file.Plot.curves`.

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

- [scripts/plot\\_file.py](#)

## 8.36 plot\_profile.Plot Class Reference

### Public Member Functions

- [def \\_\\_init\\_\\_](#)
- [def load](#)

#### 8.36.1 Detailed Description

This class holds all the information for a single plot, namely the list of curves for that pl

#### 8.36.2 Constructor & Destructor Documentation

##### 8.36.2.1 `def plot_profile.Plot.__init__( self, element, type )`

This method initlizes the plot object

References `plot_file.Plot.bMinorTics`, `plot_profile.Plot.bMinorTics`, `plot_file.Axis.bMinorTics`, `plot_file.Plot.curves`, `plot_profile.Plot.curves`, `plot_file.Plot.grid`, `plot_profile.Plot.grid`, `plot_file.Axis.grid`, `Global.grid`, `plot_file.Plot.legendloc`, `plot_profile.Plot.legendloc`, `plot_file.Plot.limits`, `plot_profile.Plot.limits`, `plot_file.Axis.limits`, `plot_file.Plot.ylabel`, and `plot_profile.Plot.ylabel`.

#### 8.36.3 Member Function Documentation

##### 8.36.3.1 `def plot_profile.Plot.load( self, fileData, options, dataSet, nFileCount )`

loads the data for a plot, y-data is stored in the curves, and sets the ylabel from the first file read in

References `plot_file.Plot.curves`, `plot_profile.Plot.curves`, `plot_file.Plot.ylabel`, and `plot_profile.Plot.ylabel`.

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

- `scripts/plot_profile.py`

## 8.37 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](#)

### 8.37.1 Detailed Description

This class manages information which pertains to the processor topology.

### 8.37.2 Constructor & Destructor Documentation

#### 8.37.2.1 ProcTop::ProcTop( )

Constructor for class [ProcTop](#).

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

### 8.37.3 Member Data Documentation

#### 8.37.3.1 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\(\)](#).

#### 8.37.3.2 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\(\)](#).

### 8.37.3.3 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()`.

### 8.37.3.4 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()`.

### 8.37.3.5 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()`.

### 8.37.3.6 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()`.

### 8.37.3.7 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()`.



**8.37.3.8 int\* ProcTop::nRadialNeighborNeighborIDs**

Holds the ID of a radialial neighbor, to be used to obtain their [ProcTop::nRank](#) from [ProcTop::nNeighborRanks](#)

Referenced by [calNewU0\\_R\(\)](#), [calNewU0\\_RT\(\)](#), [calNewU0\\_RTP\(\)](#), [initUpdateLocalBoundaries\(\)](#), and [ProcTop\(\)](#).

**8.37.3.9 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\(\)](#).

**8.37.3.10 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:

- [src/SPHERLS/procTop.h](#)
- [src/SPHERLS/procTop.cpp](#)

**8.38 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](#) ()

### 8.38.1 Constructor & Destructor Documentation

#### 8.38.1.1 [profileData::profileData](#) ( )

Constructor for class

### 8.38.2 Member Function Documentation

#### 8.38.2.1 void [profileData::clear](#) ( )

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

Referenced by [main](#)().

#### 8.38.2.2 unsigned int [profileData::nMaxNumZones](#) ( )

Returns the maximum number of zones found under a key

Referenced by [toFile](#)().

#### 8.38.2.3 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.

#### 8.38.2.4 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.

#### 8.38.2.5 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](#)().

**8.38.2.6** 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.

**8.38.2.7** 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.

**8.38.2.8** 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.

**8.38.2.9** 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().

**8.38.2.10** void profileData::setSum ( std::string *sName*, unsigned int *nZone*, int *nValue* )

If the value is already set it will add to it

**8.38.2.11** 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().

Referenced by main().

**8.38.2.12** 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 bFileExists(), Time::dt, ProcTop::nCoords, nMaxNumZones(), ProcTop::nNumProcs, ProcTop::nNumRadialNeighbors, ProcTop::nRadialNeighborRanks, and - ProcTop::nRank.

Referenced by main().

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

- [src/SPHERLS/profileData.h](#)
- [src/SPHERLS/profileData.cpp](#)

## 8.39 work\_plot.Settings Class Reference

### Public Member Functions

- [def \\_\\_init\\_\\_](#)
- [def parseXML](#)

### 8.39.1 Constructor & Destructor Documentation

#### 8.39.1.1 `def work_plot.Settings.__init__( self, oldColumns = False )`

Initialize settings

References `work_plot.Settings.AV`, `work_plot.Settings.deltaMColumn`, `work_plot.Settings.deltaMColumnHeader`, `work_plot.WorkPlotSettings.outputFile`, `light_curve.LightCurve.outputFile`, `work_plot.PdVPlotSettings.outputFile`, `work_plot.Settings.outputFile`, `eos_interp.interpTable.outputFile`, `work_plot.Settings.pColumn`, `work_plot.Settings.pColumnHeader`, `work_plot.WorkPlotSettings.plotPdVCurves`, `work_plot.Settings.plotPdVCurves`, `work_plot.Settings.QColumn`, `work_plot.Settings.QColumnHeader`, `work_plot.Settings.rhoColumn`, `work_plot.Settings.rhoColumnHeader`, `work_plot.Settings.tColumn`, and `work_plot.Settings.tColumnHeader`.

### 8.39.2 Member Function Documentation

#### 8.39.2.1 `def work_plot.Settings.parseXML( self, fileName )`

Get user settings from XML file

References `work_plot.Settings.AV`, `work_plot.Settings.files`, `plot_file.DataSet.files`, `main()`, `work_plot.Settings.PdVPlotSettings`, `work_plot.WorkPlotSettings.plotPdVCurves`, `work_plot.Settings.plotPdVCurves`, and `work_plot.Settings.workPlotSettings`.

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

- `scripts/work_plot.py`

## 8.40 term Struct Reference

```
#include <main.h>
```

## Public Attributes

- double `dCoeff`
- double `dPower`

### 8.40.1 Detailed Description

Structured variable holding a coefficient and a power. Multiple terms combined together constructing a polynomial used for approximating the initial velocity profile.

See also

[vectVelDist](#), [sUDistType](#)

### 8.40.2 Member Data Documentation

#### 8.40.2.1 `double term::dCoeff`

Coefficient of the term in the polynomial.

Referenced by `readConfig()`.

#### 8.40.2.2 `double term::dPower`

Power of the term in the polynomial.

Referenced by `readConfig()`.

The documentation for this struct was generated from the following file:

- `src/SPHERLSgen/main.h`

## 8.41 `plot_file.Text` Class Reference

### Public Member Functions

- `def __init__`

### 8.41.1 Detailed Description

This class holds information for a text object on a plot.

### 8.41.2 Constructor & Destructor Documentation

#### 8.41.2.1 `def plot_file.Text.__init__( self, element )`

This method initialize a text object from an xml element

References `plot_file.Text.text`, `XMLNodeContents.text`, `plot_file.Text.x`, and `plot_file.Text.y`.

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

- `scripts/plot_file.py`

## 8.42 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](#)

#### 8.42.1 Detailed Description

This class manages information which pertains to time variables.

## 8.42.2 Constructor & Destructor Documentation

### 8.42.2.1 `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 `n-TimeStepIndex`.

## 8.42.3 Member Data Documentation

### 8.42.3.1 `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()`.

### 8.42.3.2 `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()`.

### 8.42.3.3 `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()`.

### 8.42.3.4 `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()`.

### 8.42.3.5 `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()`.

### 8.42.3.6 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\\_RTP\\_LES\(\)](#), [calNewV\\_RT\(\)](#), [calNewV\\_RT\\_LES\(\)](#), [calNewV\\_RTP\(\)](#), [calNewV\\_RTP\\_LES\(\)](#), [calNewW\\_RTP\(\)](#), [calNewW\\_RTP\\_LES\(\)](#), [modelRead\(\)](#), and [Time\(\)](#).

### 8.42.3.7 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\(\)](#).

### 8.42.3.8 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\(\)](#).

### 8.42.3.9 double Time::dEndTime

The end time of the current calculation in seconds.

Referenced by [init\(\)](#), [main\(\)](#), and [Time\(\)](#).

### 8.42.3.10 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()`.

#### 8.42.3.11 `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()`.

#### 8.42.3.12 `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()`.

#### 8.42.3.13 `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()`.

#### 8.42.3.14 `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:

- `src/SPHERLS/time.h`
- `src/SPHERLS/time.cpp`

### 8.43 `make_hdf.variable` Class Reference

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

- `scripts/make_hdf.py`

### 8.44 `watchzone` Struct Reference

The documentation for this struct was generated from the following file:

- `src/SPHERLSanal/main.h`

### 8.45 `WatchZone` Class Reference

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

#### 8.45.1 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:

- `src/SPHERLS/watchzone.h`
- `src/SPHERLS/watchzone.cpp`

### 8.46 `work_plot.WorkPlotSettings` Class Reference

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

- `scripts/work_plot.py`

### 8.47 `XML` Struct Reference

The documentation for this struct was generated from the following file:

- `src/xmlParser.cpp`

### 8.48 `XMLAttribute` Struct Reference

The documentation for this struct was generated from the following file:

- `src/xmlParser.h`

## 8.49 XMLCharacterEntity Struct Reference

The documentation for this struct was generated from the following file:

- src/xmlParser.cpp

## 8.50 XMLClear Struct Reference

The documentation for this struct was generated from the following file:

- src/xmlParser.h

## 8.51 XMLNode Struct Reference

### Classes

- struct **XMLNodeDataTag**

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

- src/xmlParser.h
- src/xmlParser.cpp

## 8.52 XMLNodeContents Struct Reference

The documentation for this struct was generated from the following file:

- src/xmlParser.h

## 8.53 XMLParserBase64Tool Class Reference

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

- src/xmlParser.h
- src/xmlParser.cpp

## 8.54 XMLResults Struct Reference

The documentation for this struct was generated from the following file:

- src/xmlParser.h



## Chapter 9

# File Documentation

### 9.1 scripts/cp\_files.py File Reference

#### Functions

- def `cp_files.main`
- def `cp_files.cp_files`

#### 9.1.1 Detailed Description

### 9.2 src/eos.cpp File Reference

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

#### 9.2.1 Detailed Description

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

### 9.3 src/eos.h File Reference

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

#### Classes

- class [eos](#)

### 9.3.1 Detailed Description

Header file for [eos.cpp](#)

## 9.4 src/SPHERLS/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)

### 9.4.1 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.

### 9.4.2 Function Documentation

#### 9.4.2.1 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](#)().

#### 9.4.2.2 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().

**9.4.2.3** 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>bWriteCurrentStateToFile</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::dCurrentReITError, Time::dDeIRho\_t\_Rho\_max, Time::dDeIT\_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().

**9.4.2.4** 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 calulation timer is started, [Performance::dStartTimer](#)



- It also reads in the equation of state table if using a tabulated equation of state (`Parameters::bEOSGammaLaw = false`) by calling `eos::readBin`
- Initializes the watchZones, i.e. figure out which processors have which watch zones, opens the files and prints headers.

## Parameters

out	<i>procTop</i>	all parts of this stucture are set, and do not change throughout the rest of the calculation.
out	<i>grid</i>	through the function <code>modelRead</code> the function <code>setupLocalGrid</code> 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`, `bFileExists()`, `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()`.

#### 9.4.2.5 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().

#### 9.4.2.6 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::nEndGhostUpdate-

Implicit, 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().

**9.4.2.7 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::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().

#### 9.4.2.8 void modelWrite\_GL ( std::string *sFileName*, ProcTop & *procTop*, Grid & *grid*, Time & *time*, Parameters & *parameters* )

Writes out a model in distributed model format, meaning that each processor writes its own local grid to a file in binary format. They can be combined, and or converted to ascii format using SPHERLSanal. This is for a gamma-law gas model.

##### Parameters

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().

#### 9.4.2.9 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().

#### 9.4.2.10 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().

#### 9.4.2.11 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

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

#### 9.4.2.12 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()`.

**9.4.2.13** `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()`.

**9.4.2.14** 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()`.

**9.4.2.15** 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()`.

**9.4.2.16** 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()`.

**9.4.2.17** 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()`.

**9.4.2.18** 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()`.

## 9.5 src/SPHERLS/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)

### 9.5.1 Detailed Description

Header file for [dataManipulation.cpp](#)

### 9.5.2 Function Documentation

#### 9.5.2.1 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](#)().

#### 9.5.2.2 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()`.

**9.5.2.3** `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>bWriteCurrentStateToFile</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::dCurrentReITError`, `Time::dDeIRho_t_Rho_max`, `Time::dDeIT_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()`.

**9.5.2.4** `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 calulation timer is started, [Performance::dStartTimer](#)

- It also reads in the equation of state table if using a tabulated equation of state (`Parameters::bEOSGammaLaw = false`) by calling `eos::readBin`
- Initializes the watchZones, i.e. figure out which processors have which watch zones, opens the files and prints headers.

## Parameters

out	<i>procTop</i>	all parts of this stucture are set, and do not change throughout the rest of the calculation.
out	<i>grid</i>	through the function <code>modelRead</code> the function <code>setupLocalGrid</code> 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`, `bFileExists()`, `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()`.

#### 9.5.2.5 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().

#### 9.5.2.6 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::nEndGhostUpdate-

Implicit, 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().

#### 9.5.2.7 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().

### 9.5.2.8 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().

### 9.5.2.9 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().

#### 9.5.2.10 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().

#### 9.5.2.11 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().

#### 9.5.2.12 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()`.

**9.5.2.13** `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()`.



**9.5.2.14** 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()`.

**9.5.2.15** 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()`.

**9.5.2.16** 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()`.

**9.5.2.17** 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()`.

**9.5.2.18** 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()`.

## 9.6 src/SPHERLS/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)

### 9.6.1 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.

## 9.6.2 Function Documentation

### 9.6.2.1 void finWatchZones ( Output & *output* )

Closes the files opened for writing out the watchzones

#### Parameters

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

References Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by fin().

### 9.6.2.2 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

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, bFileExists(), Parameters::dGamma, - ProcTop::nCoords, Grid::nD, Grid::nGlobalGridDims, Grid::nLocalGridDims, Grid::nNum1DZones, Grid::nNumDims, Grid::nNumGhostCells, ProcTop::nNumProcs, ProcTop::nProcDims, ProcTop::nRank, Time::nTimeStepIndex, Output::ofWatchZoneFiles, Output::sBaseOutputFileName, and Output::watchzoneList.

Referenced by init().

### 9.6.2.3 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

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().

#### 9.6.2.4 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

in, out	output	
in	grid	
in	parameters	
in	time	
in	procTop	

References Grid::dLocalGridOld, Parameters::dPi, Time::dt, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nP, Grid::nQ0, Grid::nR, Grid::nT, Time::nTimeStepIndex, Grid::nU, Grid::nU0, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

#### 9.6.2.5 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().

### 9.6.2.6 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	<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::nR, Grid::nT, Time::nTimeStepIndex, Parameters::nTypeTurbulenceMod, - Grid::nU, Grid::nU0, Grid::nV, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

### 9.6.2.7 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().

### 9.6.2.8 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()`.

## 9.7 src/SPHERLS/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)

### 9.7.1 Detailed Description

Header file for `dataMonitoring.cpp`

### 9.7.2 Function Documentation

9.7.2.1 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().

9.7.2.2 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, bFileExists(), Parameters::dGamma, - ProcTop::nCoords, Grid::nD, Grid::nGlobalGridDims, Grid::nLocalGridDims, Grid::nNum1DZones, Grid::nNumDims, Grid::nNumGhostCells, ProcTop::nNumProcs, ProcTop::nProcDims, ProcTop::nRank, Time::nTimeStepIndex, Output::ofWatchZoneFiles, Output::sBaseOutputFileName, and Output::watchzoneList.

Referenced by init().

9.7.2.3 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().

#### 9.7.2.4 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	output	
in	grid	
in	parameters	
in	time	
in	procTop	

References Grid::dLocalGridOld, Parameters::dPi, Time::dt, Grid::nCenIntOffset, Grid::nD, Grid::nDM, Grid::nE, Grid::nP, Grid::nQ0, Grid::nR, Grid::nT, Time::nTimeStepIndex, Grid::nU, Grid::nU0, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

#### 9.7.2.5 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().



### 9.7.2.6 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	<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::nR, Grid::nT, Time::nTimeStepIndex, Parameters::nTypeTurbulenceMod, - Grid::nU, Grid::nU0, Grid::nV, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

### 9.7.2.7 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().

### 9.7.2.8 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::nType-TurbulenceMod, Grid::nU, Grid::nU0, Grid::nV, Grid::nW, Output::ofWatchZoneFiles, and Output::watchzoneList.

Referenced by setMainFunctions().

## 9.8 src/SPHERLS/global.cpp File Reference

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

### 9.8.1 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.

## 9.9 src/SPHERLS/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 SIGNEG DEN 0`
- `#define SIGNEG ENG 0`
- `#define SIGNEG TEMP 0`
- `#define TRACKMAX SOLVER ERROR 0`
- `#define SEDOV 0`
- `#define VISCOUS\_ENERGY\_EQ 1`
- `#define DUMP\_VERSION 1`
- `#define DEBUG\_EQUATIONS 0`
- `#define DEDEM\_CLAMP 1`

### 9.9.1 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.

### 9.9.2 Define Documentation

#### 9.9.2.1 `#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()`.

#### 9.9.2.2 `#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.

#### 9.9.2.3 `#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()`.

#### 9.9.2.4 `#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.

**9.9.2.5 #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.

**9.9.2.6 #define SIGNEG ENG 0**

Raise signal on calculation of negative energy if set to 1, else don't rais a signal. - Otherwise it will be handled through the normal exception method. This is useful when debugging, it will stop the debugger at the location of the calculation of the negative energy. If not 1, it will speed up calculation slightly and generate more useful output upon detection of negative energy. If 1 and not being run in the debugger, it likely won't generate any usefull output upon negative energies, and wil simply abort the program.

**9.9.2.7 #define SIGNEG TEMP 0**

Raise signal on calculation of negative temperature if set to 1, else don't rais a signal. Otherwise it will be handled through the normal exception method. This is useful when debugging, it will stop the debugger at the location of the calculation of the negative energy. If not 1, it will speed up calculation slightly and generate more useful output upon detection of negative energy. If 1 and not being run in the debugger, it likely won't generate any usefull output upon negative energies, and wil simply abort the program.

**9.9.2.8 #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 weather the solver is working accurately this is very handy to turn on.

**9.9.2.9 #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

**9.10 src/SPHERLS/main.cpp File Reference**

```
#include <mpi.h> #include <sstream> #include <string> ×
#include <fstream> #include <cmath> #include <vector>
#include <algorithm> #include <iomanip> #include <csignal> ×
#include <fenv.h> #include "main.h" #include "global.h" ×
#include "watchzone.h" #include "exception2.h" #include
```

```
"xmlParser.h" #include "xmlFunctions.h" #include "data-
Manipulation.h" #include "dataMonitoring.h" #include "phys-
Equations.h"
```

## Functions

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

### 9.10.1 Detailed Description

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

### 9.10.2 Function Documentation

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

Main driving function of SPHERLS.

#### Parameters

in	argc	number of arguments passed from the command line
in	argv	array of character strings of size argc containing the arguments from the command line.

The flow of this function is as follows:

- Initialize program by calling [init\(\)](#)
- Set function pointers by calling [setMainFunctions\(\)](#)
- Update new grid with old grid by calling [updateNewGridWithOld\(\)](#)
- Update boundaries of local grids
- Calculate the first time step by calling [Functions::fpCalculateDeltat\(\)](#)
- Enter while loop until end time ([Time::dEndTime](#)) is reached, and for each iteration of the loop:
  - Test to see if a model dump is needed (by checking [Output::bDump](#) and [Output::nDumpFrequency](#)), if so dump one by calling [modelWrite\(\)](#)
  - Write out information for any watchzones present by calling [writeWatchZones\(\)](#)
  - Write out information for peak kinetic energy per period by calling [writePeakKE\(\)](#)
  - calculate time step by calling function pointed to by [Functions::fpCalculateDeltat](#)

- Calculate new velocities by calling the function pointed to by `Functions::fpCalculateNewVelocities()`
- Update velocities on new grid boundaries between processors by calling `updateLocalBoundariesNewGrid()` three times indicating the  $r$ -velocity (U),  $\theta$ -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

Referenced by `eos_interp.interpTableManager::__init__()`, `make_hdf2.fileSet::convertDumpToHDF()`, `make_hdf.fileSet::convertDumpToHDF()`, `plot_file.DataSet::getCurve()`, `plot_profile.DataSet::getCurve()`, `plot_2DSlices.File2DSlice::load()`, `work_plot.Settings::parseXML()`, `dump.dump::printVar()`, and `light_curve.LightCurve::write()`.

#### 9.10.2.2 void signalHandler ( int nSig )

Used for catching signals.

Referenced by `main()`.

## 9.11 src/SPHERLSanal/main.cpp File Reference

```
#include "main.h" #include <math.h> #include <iomanip> ×
#include "eos.h" #include "mf hdf.h"
```

### Functions

- int `main` (int argc, char \*argv[])
- void `printHelp` ()
- bool `bFileExists` (std::string strFilename)
- double `dCalRhoAve2D` (double \*\*\*\*dGrid, int nI, int nStartY, int nEndY, int nStartZ, int nEndZ)
- void `computeFourierTransFromList` (std::string sInFileName, std::string sOutFileName)
- void `convertBinToHDF4` (std::string sFileName)

### 9.11.1 Detailed Description

This code is used to manipulate the outputfiles generated by SHPERLS.

### 9.11.2 Function Documentation

#### 9.11.2.1 `bool bFileExists ( std::string sFilename )`

Tests if the file exists by attempting to open the file for reading, if it fails it returns false, if it succeeds it returns true. This does not take into consideration permissions but that is ok for this project.

##### Parameters

<code>in</code>	<code>sFilename</code>	file name of the file to check if it exists or not
-----------------	------------------------	--

##### Returns

returns true or false depending on whether the file exists

Referenced by `init()`, `initWatchZones()`, and `profileData::toFile()`.

#### 9.11.2.2 `void computeFourierTransFromList ( std::string sInFileName, std::string sOutFileName )`

Calculates a volume weighted average density given the grid variables, `dGrid` and the radial index, `nl`, the start and stop indices in the Y and Z direction. For the 2D case.

Referenced by `main()`.

#### 9.11.2.3 `void convertBinToHDF4 ( std::string sFileName )`

converts a collected binary file to an hdf file

References `dAlpha`, `dGamma`, `eosTable`, `nC`, `nD`, `nDM`, `nE`, `nGamma`, `nKappa`, `nKE`, `nL_con`, `nL_rad`, `nM`, `nNumGhostCells`, `nP`, `nPhi`, `nQ`, `nR`, `nT`, `nTheta`, `nU`, `nU0`, `nV`, `nW`, and `eos::readBin()`.

Referenced by `main()`.

#### 9.11.2.4 `double dCalRhoAve2D ( double **** dGrid, int nl, int nStartY, int nEndY, int nStartZ, int nEndZ )`

Calculates a volume weighted average density given the grid variables, `dGrid` and the radial index, `nl`, the start and stop indices in the Y and Z direction. For the 3D case.

References `nD`, `nR`, and `nTheta`.

### 9.11.2.5 `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, bExtraInfoInProfile, Output::bPrint, bScientific, computeFourierTransFromList(), convertBinToHDF4(), 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, nPrecisionAscii, Output::nPrintFrequencyStep, Output::nPrintMode, Grid::nR, ProcTop::nRank, Grid::nT, Time::nTimeStepIndex, Grid::nU0, Global::output, Global::parameters, Global::performance, printHelp(), Global::procTop, Output::sBaseOutputFileName, Parameters::sDebugProfileOutput, sEOSFile, setMainFunctions(), signalHandler(), Global::time, updateLocalBoundaries(), updateLocalBoundariesNewGrid(), and updateNewGridWithOld().

#### 9.11.2.6 void printHelp ( )

**Todo** need to updated, and revise help text to better describe the program. Some improvements could include: -better describing the "-x" appended to the file name base -mention that some times it expects a file name base, while others it wants the full file name -mention file extensions and naming of output files i.e. what the outputfile for the radially averaged profile will be called -perhaps mention some of the additional scripts used to extend the functionality of SPHERLSanal

Referenced by main().

## 9.12 src/SPHERLSgen/main.cpp File Reference

```
#include "exception2.h" #include "xmlParser.h" #include
"xmlFunctions.h" #include "eos.h" #include "main.h" #include
<iostream> #include <sstream> #include <fstream> #include
<cmath> #include <cmath> #include <vector> #include
<limits> #include <string> #include <algorithm> #include
<iomanip>
```

## Functions

- int `main` ()
- void `readConfig` (std::string sConfigFileName, std::string sStartNode)
- void `readUProfile` (std::string sProfileFileName)
- void `generateModel_SEDOV` ()
- void `generateModel_TEOS` ()
- void `generateModel_GL` ()
- void `calculateFirstShell_SEDOV` ()
- void `calculateFirstShell_TEOS` ()
- void `writeModel_R_TEOS` ()
- void `writeModel_R_GL` ()
- void `writeModel_RT_TEOS` ()
- void `writeModel_RT_GL` ()
- void `writeModel_RTP_TEOS` ()
- void `writeModel_RTP_GL` ()
- void `writeModel_Bin_R_TEOS` ()
- void `writeModel_Bin_R_GL` ()
- void `writeModel_Bin_RT_TEOS` ()
- void `writeModel_Bin_RT_GL` ()
- void `writeModel_Bin_RTP_TEOS` ()
- void `writeModel_Bin_RTP_GL` ()
- void `writeModelToScreen_TEOS` ()
- void `writeModelToScreen_GL` ()
- double `dPartialDerivativeVar1` (double dVar1, double dVar2, int nShell, double(\*d-Function)(double dVar1, double dVar2, int nShell), double dH, double &dError)
- double `dPartialDerivativeVar2` (double dVar1, double dVar2, int nShell, double(\*d-Function)(double dVar1, double dVar2, int nShell), double dH, double &dError)
- double `EOS_GL` (double dP, double dE)

### 9.12.1 Detailed Description

This code is used to generate the starting model for SPHERLS

**Todo** Want to make printing model to the screen an option in the configuration file, and the default should be not print the model.

**Todo** It would also make sense to print to a binary model rather than an ascii model. It could however be an option with the default being to print to a binary file.

## 9.12.2 Function Documentation

### 9.12.2.1 void calculateFirstShell\_SEDOV( )

Calculates the first shell of a spherical blast wave model, or in other words a sedov test model.

References dEng, dGamma, dPi, dRDelta, dRho, dRMin, dV, nNumR, vecdE, vecdM, vecdMdel, vecdP, vecdR, and vecdRho.

Referenced by generateModel\_SEDOV().

### 9.12.2.2 void calculateFirstShell\_TEOS( )

Calculates the quantities required in the first shell of the model. This is done by

1. setting the mass at the surface equal to `dMTotal`
2. setting the temperature at the surface equal to the surface temperature given by  $T_s = \sqrt[4]{2T_{eff}}$  where  $T_{eff}$  is `dTeff`.
3. Calculating the outer radius from  $R_s = \sqrt{\frac{L}{4\pi\sigma T_s^4}}$  where  $L$  is `dL*dLSun`,  $\pi$  is `dPi`,  $\sigma$  is `dSigma`.
4. Calculate the mass step. In the first shell this is simply  $-1.0 * \text{dMDelta} * \text{dMSun}$
5. Calculate the pressure at zone center by solving the static momentum conservation equation in 1D for the pressure at the center of the first zone,  

$$P = \frac{-GM_r}{8\pi r^4} (0.5 + \alpha) \Delta M$$
 where  $\alpha$  is `dAlpha`,  $\Delta M$  is `vecdMdel[0]`,  $r$  is `vecdR[0]`,  $M_r$  is `vecdM[0]` and  $G$  is `dG`. The pressure was directly solved for by applying the boundary condition that  $P = 0$  at the outer most interface. This lead to the pressure at `vecdM[0] - dAlpha * dMDelta` equal to the negative of the pressure at `vecdM[0] + 0.5 * vecdMdel[0]`, allowing for direct solution of the pressure. Note that `vecdMdel[0]` is negative to indicate moving into the star.
6. Find the density such that the calculated pressure is recovered at the surface temperature  $T_s$ . This is done by calculating the derivative  $\frac{\partial \rho}{\partial P}$  from the equation of state and using it to calculate a linear correction to the density. This correction is repeatedly applied to the density until the correction is smaller than `dTolerance`.
7. Once the density is found, the energy (`vecdE[0]`) and the opacity (`vecdKappa[0]`) are easily found from the equation of state table.
8. Finally the mass at the inner interface, and the radius at the inner interface are calculated. The radius is simply that required to produce a volume to give the density of the zone at the mass of the shell.

References dAlpha, dG, dL, eos::dLogRhoDelta, eos::dLogRhoMin, dLSun, dMDelta, dMSun, dMTotal, dPi, dRho, dRSurf, dSigma, dTeff, dTolerance, eosTable, eos::getEKappa(), eos::getPAndDRhoDP(), nNumIters, vecdE, vecdKappa, vecdM, vecdMdel, vecdP, vecdR, vecdRho, and vecdT.

Referenced by generateModel\_TEOS().

**9.12.2.3** `double dPartialDerivativeVar1 ( double dVar1, double dVar2, int nShell,  
double(*) (double dVar1, double dVar2, int nShell) dFunction, double dH, double &  
dError )`

Computes the partial derivative of a function of two variables with respect to the first variable.

#### Parameters

in	<i>dVar1</i>	location at which derivative should be evaluated, and also the variable that the partial derivative is take with respect to.
in	<i>dVar2</i>	location at which derivative should be evaluated.
in	<i>dFunction</i>	pointer to the double precision function of two variables, <i>dVar1</i> and <i>dVar2</i> .

#### Returns

returns the partial derivative with respect to *dVar1* of the given function

**9.12.2.4** `double dPartialDerivativeVar2 ( double dVar1, double dVar2, int nShell,  
double(*) (double dVar1, double dVar2, int nShell) dFunction, double dH, double &  
dError )`

Computes the partial derivative of a function of two variables with respect to the second variable.

#### Parameters

in	<i>dVar1</i>	location at which derivative should be evaluated, and also the variable that the partial derivative is take with respect to.
in	<i>dVar2</i>	location at which derivative should be evaluated.
in	<i>dFunction</i>	pointer to the double precision function of two variables, <i>dVar1</i> and <i>dVar2</i> .

#### Returns

returns the partial derivative with respect to *dVar1* of the given function

**9.12.2.5** `double EOS_GL ( double dP, double dE )`

,

References dGamma.

## 9.12.2.6 void generateModel\_GL( )

This function drives model generation the spherical static stellar model. It does this by calling `calculateFirstShell` to calculate the first shell of the model, then calling `calculateShell` repeatedly to generate the rest of the radial shells until the desired depth is reached (indicated by `dRStop`). Finally a radial velocity profile is generated by calling `makeVelocityDist`.

See also

[vecdM](#), [vecdMdel](#), [vecdP](#), [vecdE](#), [vecdRho](#), [vecdR](#), [vecdT](#), [vecdKappa](#), [vecdU](#), [vecdV](#), [vecdW](#) for a description of the quantities calculated in the model.

References `dRSun`, `vecdR`, and `vecdT`.

Referenced by `readConfig()`.

## 9.12.2.7 void generateModel\_SEDOV( )

This function drives model generation for a sedov spherical blast wave model. It does this by calling [calculateFirstShell\\_SEDOV](#) to calculate the first shell of the model, then calling `calculateShell_SEDOV` repeatedly until all the zones are set.

References `calculateFirstShell_SEDOV()`, and `nNumR`.

Referenced by `readConfig()`.

## 9.12.2.8 void generateModel\_TEOS( )

This function drives model generation for a spherical static stellar model. It does this by calling [calculateFirstShell\\_TEOS](#) to calculate the first shell of the model, then calling `calculateShell_TEOS` repeatedly to generate the rest of the radial shells until the desired depth is reached (indicated by `dRStop`). Finally a radial velocity profile is generated by calling `makeVelocityDist`.

See also

[vecdM](#), [vecdMdel](#), [vecdP](#), [vecdE](#), [vecdRho](#), [vecdR](#), [vecdT](#), [vecdKappa](#), [vecdU](#), [vecdV](#), [vecdW](#) for a description of the quantities calculated in the model.

References `calculateFirstShell_TEOS()`, `dRSun`, `vecdR`, and `vecdT`.

Referenced by `readConfig()`.

## 9.12.2.9 int main( )

Main driving function for SPHERLSgen

References `readConfig()`.

### 9.12.2.10 void readConfig ( std::string sConfigFileName, std::string sStartNode )

Reads in an xml configuration file and sets the values of many global variables.

#### See also

main.h for variables that can be set in the configuration file and what the tag names specify those variables.

#### Parameters

in	<i>sConfigFileName</i>	filename and path to configuration file. Often "SPHERL-Sgen.xml"
in	<i>sStartNode</i>	name of the starting node in the configuration file. Often "data"

**Todo** need to check that T is increasing, and R is decreasing. This will get tricky if R and T types are mixed.

References bAutoDeltaM, bBinaryOutput, bGammaLawEOS, bWriteToScreen, dAlpha, term::dCoeff, dDeltaPhi, dDeltaTheta, dEng, dEngCent, dG, dGamma, dL, dLSun, dMDelta, dMSun, dMTot, term::dPower, dRDelta, dRho, dRMin, dRSun, dRSurf, dSigma, dTeff, dTimeStepFactor, dTolerance, dUSurf, eosTable, generateModel\_GL(), generateModel\_SEDOV(), generateModel\_TEOS(), nNumCellsCent, nNumDims, nNumGhostCells, nNumIters, nNumPhi, nNumR, nNumTheta, nNumZones1D, n-Periodic, nPrecision, eos::readBin(), readUProfile(), sEOSFile, sOutPutfile, sUDistType, vecdE, vecdKappa, vecdM, vecdMdel, vecdP, vecdR, vecdRho, vecdT, vectVelDist, writeModel\_Bin\_R\_GL(), writeModel\_Bin\_R\_TEOS(), writeModel\_Bin\_RT\_GL(), writeModel\_Bin\_RT\_TEOS(), writeModel\_Bin\_RTP\_GL(), writeModel\_Bin\_RTP\_TEOS(), writeModel\_R\_GL(), writeModel\_R\_TEOS(), writeModel\_RT\_GL(), writeModel\_RT\_TEOS(), writeModel\_RTP\_GL(), writeModel\_RTP\_TEOS(), writeModelToScreen\_GL(), and writeModelToScreen\_TEOS().

Referenced by main().

### 9.12.2.11 void readUProfile ( std::string sProfileFileName )

This function reads in the radial velocity profile. The radial velocities are stored in [dUPro](#), the radii of those points are stored in [dUProR](#), and the size of these arrays is given by [nNumUProPoints](#). These radial velocities are used to interpolate a radial velocity profile for the output model.

#### Parameters

in	<i>sProfileFileName</i>	the name of the file containing the radial velocity profile.
----	-------------------------	--

References dUPro, dUProR, and nNumUProPoints.

Referenced by readConfig().

**9.12.2.12 void writeModel\_Bin\_R\_GL( )**

Writes out the model generated using a gamma law gas in 1D in radius to a binary file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dGamma, dTimeStepFactor, dU, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, sOutPutfile, vecdE, vecdMdel, vecdP, vecdR, and vecdRho.

Referenced by readConfig().

**9.12.2.13 void writeModel\_Bin\_R\_TEOS( )**

Writes out the model generated using a tabulated equation of state in 1D to a binary file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dTimeStepFactor, dU, dU0, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, sEOSFile, sOutPutfile, vecdMdel, vecdP, vecdR, vecdRho, and vecdT.

Referenced by readConfig().

**9.12.2.14 void writeModel\_Bin\_RT\_GL( )**

Writes out the model generated using a gamma law gas in 2D in radius and theta to a binary file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dDeltaTheta, dGamma, dPi, dTimeStepFactor, dU, dU0, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, sOutPutfile, vecdE, vecdMdel, vecdP, vecdR, and vecdRho.

Referenced by readConfig().

**9.12.2.15 void writeModel\_Bin\_RT\_TEOS( )**

Writes out the model generated using a tabulated equation of state in 2D in radius and theta to a binary file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dDeltaTheta, dPi, dTimeStepFactor, dU, dU0, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, sEOSFile, sOutPutfile, vecdMdel, vecdP, vecdR, vecdRho, and vecdT.

Referenced by readConfig().

### 9.12.2.16 void writeModel\_Bin\_RTP\_GL ( )

Writes out the model generated using a gamma law gas in 3D in radius, theta and phi to a binary file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dDeltaPhi, dDeltaTheta, dGamma, dPi, dTimeStepFactor, dU, dU0, dW, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, sOutPutfile, vecdE, vecdMdel, vecdP, vecdR, and vecdRho.

Referenced by readConfig().

### 9.12.2.17 void writeModel\_Bin\_RTP\_TEOS ( )

Writes out the model generated using a tabulated equation of state in 3D in radius theta and phi to a binary file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dDeltaPhi, dDeltaTheta, dPi, dTimeStepFactor, dU, dU0, dW, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, sEOSFile, sOutPutfile, vecdMdel, vecdP, vecdR, vecdRho, and vecdT.

Referenced by readConfig().

### 9.12.2.18 void writeModel\_R\_GL ( )

Writes out the model generated using a gamma law gas in 1D in radius to an ascii file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dGamma, dTimeStepFactor, dU, dU0, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, nPrecision, sOutPutfile, vecdE, vecdMdel, vecdP, vecdR, and vecdRho.

Referenced by readConfig().

### 9.12.2.19 void writeModel\_R\_TEOS ( )

Writes out the model generated using a tabulated equation of state in 1D to an ascii file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dTimeStepFactor, dU, dU0, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, nPrecision, sEOSFile, sOutPutfile, vecdMdel, vecdP, vecdR, vecdRho, and vecdT.

Referenced by readConfig().



**9.12.2.20 void writeModel\_RT\_GL ( )**

Writes out the model generated using a gamma law gas in 2D in radius and theta to an ascii file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dDeltaTheta, dGamma, dPi, dTimeStepFactor, dU, dU0, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, nPrecision, sOutPutfile, vecdE, vecdMdel, vecdP, vecdR, and vecdRho.

Referenced by readConfig().

**9.12.2.21 void writeModel\_RT\_TEOS ( )**

Writes out the model generated using a tabulated equation of state in 2D in radius and theta to an ascii file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dDeltaTheta, dPi, dTimeStepFactor, dU, dU0, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, nPrecision, sEOSFile, sOutPutfile, vecdMdel, vecdP, vecdR, vecdRho, and vecdT.

Referenced by readConfig().

**9.12.2.22 void writeModel\_RTP\_GL ( )**

Writes out the model generated using a gamma law gas in 3D in radius, theta and phi to an ascii file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dDeltaPhi, dDeltaTheta, dGamma, dPi, dTimeStepFactor, dU, dU0, dW, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, nPrecision, sOutPutfile, vecdE, vecdMdel, vecdP, vecdR, and vecdRho.

Referenced by readConfig().

**9.12.2.23 void writeModel\_RTP\_TEOS ( )**

Writes out the model generated using a tabulated equation of state in 3D in radius theta and phi to an ascii file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dDeltaPhi, dDeltaTheta, dPi, dTimeStepFactor, dU, dU0, dW, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, nPrecision, sEOSFile, sOutPutfile, vecdMdel, vecdP, vecdR, vecdRho, and vecdT.

Referenced by `readConfig()`.

#### 9.12.2.24 `void writeModelToScreen_GL( )`

writes the calculated model to standard output

References `dU`, `nPrecision`, `vecdE`, `vecdMDeI`, `vecdP`, `vecdR`, and `vecdRho`.

Referenced by `readConfig()`.

#### 9.12.2.25 `void writeModelToScreen_TEOS( )`

writes the calculated model to standard output

References `dU`, `nPrecision`, `vecdE`, `vecdMDeI`, `vecdP`, `vecdR`, `vecdRho`, and `vecdT`.

Referenced by `readConfig()`.

## 9.13 `src/SPHERLS/main.h` File Reference

### Functions

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

### 9.13.1 Detailed Description

Header file for [main.cpp](#)

### 9.13.2 Function Documentation

#### 9.13.2.1 `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),  $\theta$ -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](#), [bExtraInfoInProfile](#), [Output::bPrint](#), [bScientific](#), [computeFourierTransFromList\(\)](#), [convertBinToHDF4\(\)](#), [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](#), [nPrecisionAscii](#), [Output::nPrintFrequencyStep](#), [Output::nPrintMode](#), [Grid::nR](#),

ProcTop::nRank, Grid::nT, Time::nTimeStepIndex, Grid::nU0, Global::output, Global::parameters, Global::performance, printHelp(), Global::procTop, Output::sBaseOutputFileName, Parameters::sDebugProfileOutput, sEOSFile, setMainFunctions(), signalHandler(), Global::time, updateLocalBoundaries(), updateLocalBoundariesNewGrid(), and updateNewGridWithOld().

#### 9.13.2.2 void signalHandler( int nSig )

Used for catching signals.

Referenced by main().

### 9.14 src/SPHERLSanal/main.h File Reference

```
#include "../..//config.h"    #include <fftw3.h>    #include
"mfhdf.h" #include <cstdlib> #include <iostream> #include
<fstream> #include <sstream> #include <string> #include
<exception>    #include <sys/stat.h>    #include <cmath> ×
#include "exception2.h" #include <csignal> #include <fenv.-
h> #include <limits> #include <vector>
```

#### Classes

- struct [watchzone](#)

#### Functions

- void [printHelp](#) ()
- bool [bFileExists](#) (std::string strFilename)
- double [dCalRhoAve2D](#) (double \*\*\*\*dGrid, int nI, int nStartY, int nEndY, int nStartZ, int nEndZ)
- void [computeFourierTransFromList](#) (std::string sInFileName, std::string sOutFileName)
- void [convertBinToHDF4](#) (std::string sFileName)

#### Variables

- int [nM](#)
- int [nTheta](#)
- int [nPhi](#)
- int [nDM](#)
- int [nR](#)
- int [nD](#)
- int [nU](#)

- int [nU0](#)
- int [nV](#)
- int [nW](#)
- int [nE](#)
- int [nT](#)
- int [nP](#)
- int [nQ](#)
- int [nKappa](#)
- int [nGamma](#)
- int [nL\\_rad](#)
- int [nL\\_con](#)
- int [nKE](#)
- int [nC](#)
- int [nF\\_con](#)
- int [nPrecisionAscii](#) = 16
- bool [bScientific](#) = true
- const double [dPi](#) = 3.1415926535897932384626433832795
- const double [dSigma](#) = 5.67040040E-5
- const double [dLSun](#) = 3.839e33
- const int [nDumpFileVersion](#) = 1
- bool [bExtraInfoInProfile](#) = false
- std::string [sEOSFile](#) = ""

### 9.14.1 Detailed Description

Header file for [main.cpp](#).

### 9.14.2 Function Documentation

#### 9.14.2.1 bool [bFileExists](#) ( std::string *strFilename* )

Tests if the file exists by attempting to open the file for reading, if it fails it returns false, if it succeeds it returns true. This does not take into consideration permissions but that is ok for this project.

#### Parameters

<a href="#">in</a>	<a href="#">sFilename</a>	file name of the file to check if it exists or not
--------------------	---------------------------	--

#### Returns

returns true or false depending on whether the file exists

Referenced by [init\(\)](#), [initWatchZones\(\)](#), and [profileData::toFile\(\)](#).

#### 9.14.2.2 void computeFourierTransFromList( std::string *sInFileName*, std::string *sOutFileName* )

Calculates a volume weighted average density given the grid variables, dGrid and the radial index, nl, the start and stop indices in the Y and Z direction. For the 2D case.

Referenced by main().

#### 9.14.2.3 void convertBinToHDF4( std::string *sFileName* )

converts a collected binary file to an hdf file

References dAlpha, dGamma, eosTable, nC, nD, nDM, nE, nGamma, nKappa, nKE, nL\_con, nL\_rad, nM, nNumGhostCells, nP, nPhi, nQ, nR, nT, nTheta, nU, nU0, nV, nW, and eos::readBin().

Referenced by main().

#### 9.14.2.4 double dCalRhoAve2D( double \*\*\*\* *dGrid*, int *nl*, int *nStartY*, int *nEndY*, int *nStartZ*, int *nEndZ* )

Calculates a volume weighted average density given the grid variables, dGrid and the radial index, nl, the start and stop indices in the Y and Z direction. For the 3D case.

References nD, nR, and nTheta.

#### 9.14.2.5 void printHelp( )

**Todo** need to updated, and revise help text to better describe the program. Some improvements could include: -better describing the "-x" appended to the file name base -mention that some times it expects a file name base, while others it wants the full file name -mention file extensions and naming of output files i.e. what the outputfile for the radially averaged profile will be called -perhaps mention some of the additional scripts used to extend the functionality of SPHERLSanal

Referenced by main().

### 9.14.3 Variable Documentation

#### 9.14.3.1 bool bExtraInfoInProfile = false

If true include extra information in radial profile about equation of state and opacity derivatives.

Referenced by main().

#### 9.14.3.2 bool bScientific = true

**Output** ascii files in scientific format

Referenced by `main()`.

**9.14.3.3** `const double dLSun = 3.839e33`

Luminosity of the sun in erg/s

Referenced by `calculateFirstShell_TEOS()`, and `readConfig()`.

**9.14.3.4** `const double dPi = 3.1415926535897932384626433832795`

Pi

Referenced by `calculateFirstShell_SEDOV()`, `calculateFirstShell_TEOS()`, `writeModel-_Bin_RT_GL()`, `writeModel_Bin_RT_TEOS()`, `writeModel_Bin_RTP_GL()`, `writeModel-_Bin_RTP_TEOS()`, `writeModel_RT_GL()`, `writeModel_RT_TEOS()`, `writeModel_RTP-_GL()`, and `writeModel_RTP_TEOS()`.

**9.14.3.5** `const double dSigma = 5.67040040E-5`

Boltzman constant

Referenced by `calculateFirstShell_TEOS()`, and `readConfig()`.

**9.14.3.6** `int nC`

Index of the sound speed.

Referenced by `convertBinToHDF4()`.

**9.14.3.7** `int nD`

Index of  $\rho$ , density, in grids, This should be the same as that used in SPHERLS defined in [global.h](#)

Referenced by `convertBinToHDF4()`, and `dCalRhoAve2D()`.

**9.14.3.8** `int nDM`

Index of  $\delta M$  in grids, This should be the same as that used in SPHERLS defined in [global.h](#)

Referenced by `convertBinToHDF4()`.

**9.14.3.9** `const int nDumpFileVersion = 1`

Version of the dump file supported

**9.14.3.10 int nE**

Index of  $E$ , internal energy, in grids, This should be the same as that used in SPHERLS defined in [global.h](#)

Referenced by `convertBinToHDF4()`.

**9.14.3.11 int nF\_con**

Index of the convective luminosity.

**9.14.3.12 int nGamma**

Index of the adiabatic gamma.

Referenced by `convertBinToHDF4()`.

**9.14.3.13 int nKappa**

Index of the opacity in grids.

Referenced by `convertBinToHDF4()`.

**9.14.3.14 int nKE**

Index of the Kinetic energy.

Referenced by `convertBinToHDF4()`.

**9.14.3.15 int nL\_con**

Index of the Convective Luminosity.

Referenced by `convertBinToHDF4()`.

**9.14.3.16 int nL\_rad**

Index of the Radiative Luminosity.

Referenced by `convertBinToHDF4()`.

**9.14.3.17 int nM**

Index of  $M_r$  in grids, This should be the same as that used in SPHERLS defined in [global.h](#)

Referenced by `convertBinToHDF4()`.



**9.14.3.18 int nP**

Index of  $P$ , pressure

Referenced by `convertBinToHDF4()`.

**9.14.3.19 int nPhi**

Index of  $\phi$  in grids, This should be the same as that used in SPHERLS defined in [global.h](#)

Referenced by `convertBinToHDF4()`.

**9.14.3.20 int nPrecisionAscii = 16**

Set presicision of ascii output

Referenced by `main()`.

**9.14.3.21 int nQ**

Index of the artificial viscosity in grids.

Referenced by `convertBinToHDF4()`.

**9.14.3.22 int nR**

Index of  $R$ , radius, in grids, This should be the same as that used in SPHERLS defined in [global.h](#)

Referenced by `convertBinToHDF4()`, and `dCalRhoAve2D()`.

**9.14.3.23 int nT**

Index of  $T$ , temperature, in grids, This should be the same as that used in SPHERLS defined in [global.h](#)

Referenced by `convertBinToHDF4()`.

**9.14.3.24 int nTheta**

Index of  $\theta$  in grids, This should be the same as that used in SPHERLS defined in [global.h](#)

Referenced by `convertBinToHDF4()`, and `dCalRhoAve2D()`.

**9.14.3.25 int nU**

Index of  $u$ , radial velocity, in grids, This should be the same as that used in SPHERLS defined in [global.h](#)

Referenced by `convertBinToHDF4()`.

**9.14.3.26 int nU0**

Index of  $u_0$ , radial grid velocity, in grids, This should be the same as that used in SPHERLS defined in [global.h](#)

Referenced by `convertBinToHDF4()`.

**9.14.3.27 int nV**

Index of  $v$ , theta velocity in grids, This should be the same as that used in SPHERLS defined in [global.h](#)

Referenced by `convertBinToHDF4()`.

**9.14.3.28 int nW**

Index of  $w$ , phi velocity, in grids, This should be the same as that used in SPHERLS defined in [global.h](#)

Referenced by `convertBinToHDF4()`.

**9.14.3.29 std::string sEOSFile = ""**

path to an equation of state file, used for overriding the path/eos file in the model files.

Referenced by `main()`, `readConfig()`, `writeModel_Bin_R_TEOS()`, `writeModel_Bin_RT_TEOS()`, `writeModel_Bin_RTP_TEOS()`, `writeModel_R_TEOS()`, `writeModel_RT_TEOS()`, and `writeModel_RTP_TEOS()`.

**9.15 src/SPHERLSgen/main.h File Reference**

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

**Classes**

- struct [term](#)
- struct [MDeltaDelta](#)

## Functions

- void [readConfig](#) (std::string sConfigFileName, std::string sStartNode)
- void [readUProfile](#) (std::string sProfileFileName)
- void [generateModel\\_SEDOV](#) ()
- void [generateModel\\_TEOS](#) ()
- void [generateModel\\_GL](#) ()
- void [calculateFirstShell\\_SEDOV](#) ()
- void [calculateFirstShell\\_TEOS](#) ()
- double [dPartialDerivativeVar1](#) (double dVar1, double dVar2, int nShell, double(\*d-Function)(double dVar1, double dVar2, int nShell), double dH, double &dError)
- double [dPartialDerivativeVar2](#) (double dVar1, double dVar2, int nShell, double(\*d-Function)(double dVar1, double dVar2, int nShell), double dH, double &dError)
- void [writeModel\\_R\\_TEOS](#) ()
- void [writeModel\\_Bin\\_R\\_TEOS](#) ()
- void [writeModel\\_RT\\_TEOS](#) ()
- void [writeModel\\_Bin\\_RT\\_TEOS](#) ()
- void [writeModel\\_RTP\\_TEOS](#) ()
- void [writeModel\\_Bin\\_RTP\\_TEOS](#) ()
- void [writeModel\\_R\\_GL](#) ()
- void [writeModel\\_Bin\\_R\\_GL](#) ()
- void [writeModel\\_RT\\_GL](#) ()
- void [writeModel\\_Bin\\_RT\\_GL](#) ()
- void [writeModel\\_RTP\\_GL](#) ()
- void [writeModel\\_Bin\\_RTP\\_GL](#) ()
- double [EOS\\_GL](#) (double dP, double dE)
- void [writeModelToScreen\\_GL](#) ()
- void [writeModelToScreen\\_TEOS](#) ()
- int [main](#) ()

## Variables

- unsigned int [nNumR](#)
- unsigned int [nNumTheta](#)
- unsigned int [nNumPhi](#)
- unsigned int [nNumDims](#)
- double [dMSun](#)
- double [dRSun](#)
- double [dLSun](#)
- double [dSigma](#)
- double [dRSurf](#)
- double [dG](#)
- double [dPi](#) = 3.1415926535897932384626433832795
- double [dL](#)
- double [dTeff](#)
- double [dMTotal](#)

- double [dMDelta](#)
- double [dRDelta](#)
- double [dRMin](#)
- double [dMDeltaDelta](#)
- double [dDeltaTheta](#)
- double [dDeltaPhi](#)
- int [nNumCellsCent](#)
- double [dEngCent](#)
- double [dEng](#)
- double [dRho](#)
- double [dTolerance](#)
- int [nNumIters](#)
- std::string [sUDistType](#) = "POLY"
- unsigned int [nNumUProPoints](#) = 0
- double \* [dUPro](#)
- double \* [dUProR](#)
- double [dUSurf](#) = 1.0
- std::string [sOutPutfile](#)
- unsigned int [nNumZones1D](#)
- unsigned int [nNumGhostCells](#)
- unsigned int [nPrecision](#) = 14
- unsigned int [nPeriodic](#) [3] = {0,0,0}
- double [dTimeStepFactor](#) = 1.0
- double [dAlpha](#) = 0.2
- std::vector< double > [vecdM](#)
- std::vector< double > [vecdMdel](#)
- std::vector< double > [vecdP](#)
- std::vector< double > [vecdE](#)
- std::vector< double > [vecdRho](#)
- std::vector< double > [vecdR](#)
- std::vector< double > [vecdT](#)
- std::vector< double > [vecdKappa](#)
- double \*\*\* [dU](#)
- double \* [dU0](#)
- double \*\*\* [dV](#)
- double \*\*\* [dW](#)
- std::vector< [term](#) > [vectVelDist](#)
- [eos](#) [eosTable](#)
- bool [bGammaLawEOS](#) = true
- double [dGamma](#)
- int [nNumEProPoints](#) = 0
- double \* [dEPro](#)
- double \* [dEProM](#)
- std::string [sEOSFile](#)
- bool [bBinaryOutput](#)
- bool [bWriteToScreen](#)
- bool [bIsSedov](#)
- bool [bAutoDeltaM](#)

### 9.15.1 Detailed Description

Header file for [main.cpp](#).

### 9.15.2 Function Documentation

#### 9.15.2.1 void calculateFirstShell\_SEDOV( )

Calculates the first shell of a spherical blast wave model, or in other words a sedov test model.

References dEng, dGamma, dPi, dRDelta, dRho, dRMin, dV, nNumR, vecdE, vecdM, vecdMDel, vecdP, vecdR, and vecdRho.

Referenced by generateModel\_SEDOV().

#### 9.15.2.2 void calculateFirstShell\_TEOS( )

Calculates the quantities required in the first shell of the model. This is done by

1. setting the mass at the surface equal to [dMTotal](#)
2. setting the temperature at the surface equal to the surface temperature given by  $T_s = \sqrt[4]{2}T_{eff}$  where  $T_{eff}$  is [dTeff](#).
3. Calculating the outer radius from  $R_s = \sqrt{\frac{L}{4\pi\sigma T_s^4}}$  where  $L$  is [dL\\*dLSun](#),  $\pi$  is [dPi](#),  $\sigma$  is [dSigma](#).
4. Calculate the mass step. In the first shell this is simply  $-1.0*dMDelta*dMSun$
5. Calculate the pressure at zone center by solving the static momentum conservation equation in 1D for the pressure at the center of the first zone,
 
$$P = \frac{-GM_r}{8\pi r^4} (0.5 + \alpha) \Delta M$$
 where  $\alpha$  is [dAlpha](#),  $\Delta M$  is [vecdMDel\[0\]](#),  $r$  is [vecdR\[0\]](#),  $M_r$  is [vecdM\[0\]](#) and  $G$  is [dG](#). The pressure was directly solved for by applying the boundary condition that  $P = 0$  at the outer most interface. This leads to the pressure at [vecdM\[0\] - dAlpha \\* dMDelta](#) equal to the negative of the pressure at [vecdM\[0\] + 0.5\\*vecdMDel\[0\]](#), allowing for direct solution of the pressure. Note that [vecdMDel\[0\]](#) is negative to indicate moving into the star.
6. Find the density such that the calculated pressure is recovered at the surface temperature  $T_s$ . This is done by calculating the derivative  $\frac{\partial \rho}{\partial P}$  from the equation of state and using it to calculate a linear correction to the density. This correction is repeatedly applied to the density until the correction is smaller than [dTolerance](#).
7. Once the density is found, the energy ([vecdE\[0\]](#)) and the opacity ([vecdKappa\[0\]](#)) are easily found from the equation of state table.
8. Finally the mass at the inner interface, and the radius at the inner interface are calculated. The radius is simply that required to produce a volume to give the density of the zone at the mass of the shell.

References dAlpha, dG, dL, eos::dLogRhoDelta, eos::dLogRhoMin, dLSun, dMDelta, dMSun, dMTotal, dPi, dRho, dRSurf, dSigma, dTeff, dTolerance, eosTable, eos::getEKappa(), eos::getPAndDRhoDP(), nNumIters, vecdE, vecdKappa, vecdM, vecdMdel, vecdP, vecdR, vecdRho, and vecdT.

Referenced by generateModel\_TEOS().

**9.15.2.3** `double dPartialDerivativeVar1 ( double dVar1, double dVar2, int nShell, double (*)(double dVar1, double dVar2, int nShell) dFunction, double dH, double & dError )`

Computes the partial derivative of a function of two variables with respect to the first variable.

#### Parameters

in	<i>dVar1</i>	location at which derivative should be evaluated, and also the variable that the partial derivative is take with respect to.
in	<i>dVar2</i>	location at which derivative should be evaluated.
in	<i>dFunction</i>	pointer to the double precision function of two variables, dVar1 and dVar2.

#### Returns

returns the partial derivative with respect to dVar1 of the given function

**9.15.2.4** `double dPartialDerivativeVar2 ( double dVar1, double dVar2, int nShell, double (*)(double dVar1, double dVar2, int nShell) dFunction, double dH, double & dError )`

Computes the partial derivative of a function of two variables with respect to the second variable.

#### Parameters

in	<i>dVar1</i>	location at which derivative should be evaluated, and also the variable that the partial derivative is take with respect to.
in	<i>dVar2</i>	location at which derivative should be evaluated.
in	<i>dFunction</i>	pointer to the double precision function of two variables, dVar1 and dVar2.

#### Returns

returns the partial derivative with respect to dVar1 of the given function

**9.15.2.5** `double EOS_GL ( double dP, double dE )`

,

References dGamma.

#### 9.15.2.6 void generateModel\_GL( )

This function drives model generation the spherical static stellar model. It does this by calling `calculateFirstShell` to calculate the first shell of the model, then calling `calculateShell` repeatedly to generate the rest of the radial shells until the desired depth is reached (indicated by `dRStop`). Finally a radial velocity profile is generated by calling `makeVelocityDist`.

See also

[vecdM](#), [vecdMdel](#), [vecdP](#), [vecdE](#), [vecdRho](#), [vecdR](#), [vecdT](#), [vecdKappa](#), [vecdU](#), [vecdV](#), [vecdW](#) for a discription of the quantities calculated in the model.

References dRSun, [vecdR](#), and [vecdT](#).

Referenced by `readConfig()`.

#### 9.15.2.7 void generateModel\_SEDOV( )

This fuction drives model generation for a sedov spherical blast wave model. It does this by calling [calculateFirstShell\\_SEDOV](#) to calculate the first shell of the model, then calling `calculateShell_SEDOV` repeatedly until all the zones are set.

References `calculateFirstShell_SEDOV()`, and `nNumR`.

Referenced by `readConfig()`.

#### 9.15.2.8 void generateModel\_TEOS( )

This function drives model generation for a spherical static stellar model. It does this by calling [calculateFirstShell\\_TEOS](#) to calculate the first shell of the model, then calling `calculateShell_TEOS` repeatedly to generate the rest of the radial shells until the desired depth is reached (indicated by `dRStop`). Finally a radial velocity profile is generated by calling `makeVelocityDist`.

See also

[vecdM](#), [vecdMdel](#), [vecdP](#), [vecdE](#), [vecdRho](#), [vecdR](#), [vecdT](#), [vecdKappa](#), [vecdU](#), [vecdV](#), [vecdW](#) for a discription of the quantities calculated in the model.

References `calculateFirstShell_TEOS()`, `dRSun`, [vecdR](#), and [vecdT](#).

Referenced by `readConfig()`.

#### 9.15.2.9 int main( )

Main driving functon for SPHERLSgen

Documentation for a function

[more details](#)

References `profileData::clear()`, `readConfig()`, `profileData::test()`, and `profileData::toFile()`.

#### 9.15.2.10 void readConfig ( std::string sConfigFileName, std::string sStartNode )

Reads in an xml configuration file and sets the values of many global variables.

See also

`main.h` for variables that can be set in the configuration file and what the tag names specify those variables.

#### Parameters

in	<i>sConfigFileName</i>	filename and path to configuration file. Often "SPHERL-Sgen.xml"
in	<i>sStartNode</i>	name of the starting node in the configuration file. Often "data"

**Todo** need to check that T is increasing, and R is decreasing. This will get tricky if R and T types are mixed.

References `bAutoDeltaM`, `bBinaryOutput`, `bGammaLawEOS`, `bWriteToScreen`, `dAlpha`, `term::dCoeff`, `dDeltaPhi`, `dDeltaTheta`, `dEng`, `dEngCent`, `dG`, `dGamma`, `dL`, `dLSun`, `dMDelta`, `dMSun`, `dMTotal`, `term::dPower`, `dRDelta`, `dRho`, `dRMin`, `dRSun`, `dRSurf`, `dSigma`, `dTeff`, `dTimeStepFactor`, `dTolerance`, `dUSurf`, `eosTable`, `generateModel_GL()`, `generateModel_SEDOV()`, `generateModel_TEOS()`, `nNumCellsCent`, `nNumDims`, `nNumGhostCells`, `nNumIters`, `nNumPhi`, `nNumR`, `nNumTheta`, `nNumZones1D`, `nPeriodic`, `nPrecision`, `eos::readBin()`, `readUPProfile()`, `sEOSFile`, `sOutPutfile`, `sUDistType`, `vecdE`, `vecdKappa`, `vecdM`, `vecdMDeI`, `vecdP`, `vecdR`, `vecdRho`, `vecdT`, `vectVelDist`, `writeModel_Bin_R_GL()`, `writeModel_Bin_R_TEOS()`, `writeModel_Bin_RT_GL()`, `writeModel_Bin_RT_TEOS()`, `writeModel_Bin_RTP_GL()`, `writeModel_Bin_RTP_TEOS()`, `writeModel_R_GL()`, `writeModel_R_TEOS()`, `writeModel_RT_GL()`, `writeModel_RT_TEOS()`, `writeModel_RTP_GL()`, `writeModel_RTP_TEOS()`, `writeModelToScreen_GL()`, and `writeModelToScreen_TEOS()`.

Referenced by `main()`.

#### 9.15.2.11 void readUPProfile ( std::string sProfileFileName )

This function reads in the radial velocity profile. The radial velocities are stored in `dUPro`, the radii of those points are stored in `dUProR`, and the size of these arrays is given by `nNumUProPoints`. These radial velocities are used to interpolate a radial velocity profile for the output model.



## Parameters

in	<i>sProfileFileName</i>	the name of the file containing the radial velocity profile.
----	-------------------------	--

References dUPro, dUProR, and nNumUProPoints.

Referenced by readConfig().

## 9.15.2.12 void writeModel\_Bin\_R\_GL ( )

Writes out the model generated using a gamma law gas in 1D in radius to a binary file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dGamma, dTimeStepFactor, dU, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, sOutPutfile, vecdE, vecdMdel, vecdP, vecdR, and vecdRho.

Referenced by readConfig().

## 9.15.2.13 void writeModel\_Bin\_R\_TEOS ( )

Writes out the model generated using a tabulated equation of state in 1D to a binary file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dTimeStepFactor, dU, dU0, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, sEOSFile, sOutPutfile, vecdMdel, vecdP, vecdR, vecdRho, and vecdT.

Referenced by readConfig().

## 9.15.2.14 void writeModel\_Bin\_RT\_GL ( )

Writes out the model generated using a gamma law gas in 2D in radius and theta to a binary file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dDeltaTheta, dGamma, dPi, dTimeStepFactor, dU, dU0, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, sOutPutfile, vecdE, vecdMdel, vecdP, vecdR, and vecdRho.

Referenced by readConfig().

### 9.15.2.15 void writeModel\_Bin\_RT\_TEOS( )

Writes out the model generated using a tabulated equation of state in 2D in radius and theta to a binary file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dDeltaTheta, dPi, dTimeStepFactor, dU, dU0, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, sEOSFile, sOutPutfile, vecdMDeI, vecdP, vecdR, vecdRho, and vecdT.

Referenced by readConfig().

### 9.15.2.16 void writeModel\_Bin\_RTP\_GL( )

Writes out the model generated using a gamma law gas in 3D in radius, theta and phi to a binary file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dDeltaPhi, dDeltaTheta, dGamma, dPi, dTimeStepFactor, dU, dU0, dW, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, sOutPutfile, vecdE, vecdMDeI, vecdP, vecdR, and vecdRho.

Referenced by readConfig().

### 9.15.2.17 void writeModel\_Bin\_RTP\_TEOS( )

Writes out the model generated using a tabulated equation of state in 3D in radius theta and phi to a binary file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dDeltaPhi, dDeltaTheta, dPi, dTimeStepFactor, dU, dU0, dW, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, sEOSFile, sOutPutfile, vecdMDeI, vecdP, vecdR, vecdRho, and vecdT.

Referenced by readConfig().

### 9.15.2.18 void writeModel\_R\_GL( )

Writes out the model generated using a gamma law gas in 1D in radius to an ascii file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dGamma, dTimeStepFactor, dU, dU0, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, nPrecision, sOutPutfile, vecdE, vecdMDeI, vecdP, vecdR, and vecdRho.

Referenced by readConfig().

#### 9.15.2.19 void writeModel\_R\_TEOS ( )

Writes out the model generated using a tabulated equation of state in 1D to an ascii file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dTimeStepFactor, dU, dU0, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, nPrecision, sEOSFile, sOutPutfile, vecdMDeI, vecdP, vecdR, vecdRho, and vecdT.

Referenced by readConfig().

#### 9.15.2.20 void writeModel\_RT\_GL ( )

Writes out the model generated using a gamma law gas in 2D in radius and theta to an ascii file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dDeltaTheta, dGamma, dPi, dTimeStepFactor, dU, dU0, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, nPrecision, sOutPutfile, vecdE, vecdMDeI, vecdP, vecdR, and vecdRho.

Referenced by readConfig().

#### 9.15.2.21 void writeModel\_RT\_TEOS ( )

Writes out the model generated using a tabulated equation of state in 2D in radius and theta to an ascii file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dDeltaTheta, dPi, dTimeStepFactor, dU, dU0, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, nPrecision, sEOSFile, sOutPutfile, vecdMDeI, vecdP, vecdR, vecdRho, and vecdT.

Referenced by readConfig().

#### 9.15.2.22 void writeModel\_RTP\_GL ( )

Writes out the model generated using a gamma law gas in 3D in radius, theta and phi to an ascii file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dDeltaPhi, dDeltaTheta, dGamma, dPi, dTimeStepFactor, dU, dU0, dW, nNumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, nPrecision, sOutPutfile, vecdE, vecdMDeI, vecdP, vecdR, and vecdRho.

Referenced by readConfig().

**9.15.2.23 void writeModel\_RTP\_TEOS( )**

Writes out the model generated using a tabulated equation of state in 3D in radius theta and phi to an ascii file.

It writes the model out in the format accepted by SPHERLS. It writes the model the file [sOutPutfile](#)

References dAlpha, dDeltaPhi, dDeltaTheta, dPi, dTimeStepFactor, dU, dU0, dW, n-NumGhostCells, nNumPhi, nNumTheta, nNumZones1D, nPeriodic, nPrecision, sEOS-File, sOutPutfile, vecdMdel, vecdP, vecdR, vecdRho, and vecdT.

Referenced by readConfig().

**9.15.2.24 void writeModelToScreen\_GL( )**

writes the calculated model to standard output

References dU, nPrecision, vecdE, vecdMdel, vecdP, vecdR, and vecdRho.

Referenced by readConfig().

**9.15.2.25 void writeModelToScreen\_TEOS( )**

writes the calculated model to standard output

References dU, nPrecision, vecdE, vecdMdel, vecdP, vecdR, vecdRho, and vecdT.

Referenced by readConfig().

**9.15.3 Variable Documentation****9.15.3.1 bool bAutoDeltaM**

If true it will use an algorithm to choose the mass spacing.

Referenced by readConfig().

**9.15.3.2 bool bBinaryOutput**

If true the output file is in binary format. If false it outputs an ascii file.

Referenced by readConfig().

**9.15.3.3 bool bGammaLawEOS = true**

if true will cause model to be generated using a gamma law gas

Referenced by readConfig().

**9.15.3.4 bool blsSedov**

If true it will generate a starting model to preform a Sedov (blast wave) test of the code.

**9.15.3.5 bool bWriteToScreen**

If true the model is written to the screen, else model is only printed to a file.

Referenced by readConfig().

**9.15.3.6 double dAlpha = 0.2**

Parameter used to indicate mass above outter most zone. It multiplies [dMDelta](#) to calculate the amount of mass lying above the outter interface. Appropriate values for this paramter are between 0 and 0.5. This value is set in the configuration file with the "<alpha>" tag.

Referenced by calculateFirstShell\_TEOS(), convertBinToHDF4(), readConfig(), writeModel\_Bin\_R\_GL(), writeModel\_Bin\_R\_TEOS(), writeModel\_Bin\_RT\_GL(), writeModel\_Bin\_RT\_TEOS(), writeModel\_Bin\_RTP\_GL(), writeModel\_Bin\_RTP\_TEOS(), writeModel\_R\_GL(), writeModel\_R\_TEOS(), writeModel\_RT\_GL(), writeModel\_RT\_TEOS(), writeModel\_RTP\_GL(), and writeModel\_RTP\_TEOS().

**9.15.3.7 double dDeltaPhi**

The spacing to use between phi's. It is set in the configuration file under the "<dimensions>" node with the "<delta-phi>" tag.

Referenced by readConfig(), writeModel\_Bin\_RTP\_GL(), writeModel\_Bin\_RTP\_TEOS(), writeModel\_RT\_GL(), and writeModel\_RTP\_TEOS().

**9.15.3.8 double dDeltaTheta**

The spacing to use between theta's. It is set in the configuration file under the "<dimensions>" node with the "<delta-theta>" tag.

**Todo** At some point in the future it may be desirable to vary this value across theta.

Referenced by readConfig(), writeModel\_Bin\_RT\_GL(), writeModel\_Bin\_RT\_TEOS(), writeModel\_Bin\_RTP\_GL(), writeModel\_Bin\_RTP\_TEOS(), writeModel\_RT\_GL(), writeModel\_RT\_TEOS(), writeModel\_RTP\_GL(), and writeModel\_RTP\_TEOS().

**9.15.3.9 double dEng**

The energy in the remained of a sedov model exluding the central region, which is calculated from [dEngCent](#).

Referenced by calculateFirstShell\_SEDOV(), and readConfig().

**9.15.3.10 double dEngCent**

The central energy of a sedov model, This value is units of [ergs] and will be divided by the mass in the central region to produce the specific energy of the central region.

Referenced by readConfig().

**9.15.3.11 double\* dEPro**

energy profile for interpolation [erg]

**9.15.3.12 double\* dEProM**

mass points along energy profile [g]

**9.15.3.13 double dG**

Gravitational constant. Set in the configuration file with the "<G>" tag.

Referenced by calculateFirstShell\_TEOS(), calNewEddyVisc\_RTP\_SM(), calOldEddyVisc\_RTP\_SM(), and readConfig().

**9.15.3.14 double dGamma**

adiabatic gamma,  $5/3=1.66$  is stable to convection, will want to vary with depth in future versions

Referenced by calculateFirstShell\_SEDOV(), convertBinToHDF4(), EOS\_GL(), readConfig(), writeModel\_Bin\_R\_GL(), writeModel\_Bin\_RT\_GL(), writeModel\_Bin\_RTP\_GL(), writeModel\_R\_GL(), writeModel\_RT\_GL(), and writeModel\_RTP\_GL().

**9.15.3.15 double dL**

Luminosity of the model to be computed [ $L_{\text{sun}}$ ]. Set in the configuration file with the "<L>" tag.

Referenced by calculateFirstShell\_TEOS(), and readConfig().

**9.15.3.16 double dLSun**

Value to use for a solar luminosity [ $\text{ergs s}^{-1}$ ]. Set in the configuration file with the "<L-sun>" tag.

**9.15.3.17 double dMDelta**

Initial mass step size [solar masses]. Set in the configuration file with the "<M-delta>" tag.

Referenced by calculateFirstShell\_TEOS(), and readConfig().

**9.15.3.18 double dMDeltaDelta**

Fraction to increase [dMDelta](#) by each shell. Set in the configuration file with the "<M-delta-delta>" tag.

**9.15.3.19 double dMSun**

Value to use for a solar mass [g]. Set in the configuration file with the "<M-sun>" tag.

Referenced by calculateFirstShell\_TEOS(), and readConfig().

**9.15.3.20 double dMTotal**

Total mass of the model to be computed [solar masses]. Set in the configuration file with the "<M-total>" tag.

Referenced by calculateFirstShell\_TEOS(), and readConfig().

**9.15.3.21 double dPi = 3.1415926535897932384626433832795**

Pi to a ridiculous number of decimals

**9.15.3.22 double dRDelta**

Radial spacing when generating a sedov test model, otherwise it is undefined.

Referenced by calculateFirstShell\_SEDOV(), and readConfig().

**9.15.3.23 double dRho**

The density of the sedov model, set to be uniform throughout.

Referenced by calculateFirstShell\_SEDOV(), calculateFirstShell\_TEOS(), and readConfig().

**9.15.3.24 double dRMin**

Minimum radius when generating a sedov test model, otherwise it is undefined.

Referenced by calculateFirstShell\_SEDOV(), and readConfig().

**9.15.3.25 double dRSun**

Value to use for a solar radius [cm]. Set in the configuration file with the "<R-sun>" tag. Referenced by generateModel\_GL(), generateModel\_TEOS(), and readConfig().

**9.15.3.26 double dRSurf**

surface radius of the model to be computed [solar radii]. Referenced by calculateFirstShell\_TEOS(), and readConfig().

**9.15.3.27 double dSigma**

Stefan-Boltzmann constant [ $\text{ergs s}^{-1} \text{cm}^{-2} \text{K}^{-4}$ ]. Set in the configuration file with the "<sigma>" tag.

**9.15.3.28 double dTeff**

Effective temperature of the model to be computed [K]. Set in the configuration file with the "<T-eff>" tag. Referenced by calculateFirstShell\_TEOS(), and readConfig().

**9.15.3.29 double dTimeStepFactor = 1.0**

Fraction to multiply the courant timestep by when calculating the time step. This value should be the same at that used when following the hydrodynamics of the static model with SPHERLS. The time step is included in the initial static model to make the model format compatible with that required for resuming calculations with SPHERLS. It is set in the configuration file with the "<timeStepFactor>" tag.

Referenced by readConfig(), writeModel\_Bin\_R\_GL(), writeModel\_Bin\_R\_TEOS(), writeModel\_Bin\_RT\_GL(), writeModel\_Bin\_RT\_TEOS(), writeModel\_Bin\_RTP\_GL(), writeModel\_Bin\_RTP\_TEOS(), writeModel\_R\_GL(), writeModel\_R\_TEOS(), writeModel\_RT\_GL(), writeModel\_RT\_TEOS(), writeModel\_RTP\_GL(), and writeModel\_RTP\_TEOS().

**9.15.3.30 double dTolerance**

Allowed tolerance when converging a model. This value is compared to the relative error/change in a quantity while converging it e.x.  $\delta p / p$  and when this error is smaller than dTolerance iterations cease. A good value is usually around  $5e-15$  (about machine precision).

Referenced by calculateFirstShell\_TEOS(), and readConfig().



**9.15.3.31 double\*\*\* dU**

Holds the radial velocity. They are interface centered.

Referenced by `writeModel_Bin_R_GL()`, `writeModel_Bin_R_TEOS()`, `writeModel_Bin_RT_GL()`, `writeModel_Bin_RT_TEOS()`, `writeModel_Bin_RTP_GL()`, `writeModel_Bin_RTP_TEOS()`, `writeModel_R_GL()`, `writeModel_R_TEOS()`, `writeModel_RT_GL()`, `writeModel_RT_TEOS()`, `writeModel_RTP_GL()`, `writeModel_RTP_TEOS()`, `writeModelToScreen_GL()`, and `writeModelToScreen_TEOS()`.

**9.15.3.32 double\* dU0**

Holds the radial grid velocity. It is the same as the unperturbed radial velocity [dU](#).

Referenced by `writeModel_Bin_R_TEOS()`, `writeModel_Bin_RT_GL()`, `writeModel_Bin_RT_TEOS()`, `writeModel_Bin_RTP_GL()`, `writeModel_Bin_RTP_TEOS()`, `writeModel_R_GL()`, `writeModel_R_TEOS()`, `writeModel_RT_GL()`, `writeModel_RT_TEOS()`, `writeModel_RTP_GL()`, and `writeModel_RTP_TEOS()`.

**9.15.3.33 double\* dUPro**

Radial velocity profile, velocity array of size [nNumUProPoints](#). It is used when generating a velocity profile if `sUDistType="PRO"`. The array values are set in [readUProfile](#) by reading them from the velocity profile file.

Referenced by `readUProfile()`.

**9.15.3.34 double\* dUProR**

Radial vleocity profile, radii array of size [nNumUProPoints](#). It is used when generating a velocity profile if `sUDistType="PRO"`. The array values are set in [readUProfile](#) by reading them from the velocity profile file.

Referenced by `readUProfile()`.

**9.15.3.35 double dUSurf = 1.0**

Scaling for radial velocity profile when generating a velocity profile of type `sUDistTypes="PRO"`. This will be the value of the radial velocity at the surface of the model, and sets the scaling of the rest of the velocities. The value of the variable is set in the configuration file under the "`<velocityDist>`" node with the "`<uSurf>`" tag.

Referenced by `readConfig()`.

**9.15.3.36 double\*\*\* dV**

Holds the theta velocity, assumed to be zero and are centered on theta interfaces.

Referenced by `calculateFirstShell_SEDOV()`.

**9.15.3.37 double\*\*\* dW**

Holds the phi velocity, assumed to be zero and are centered on phi interfaces.

Referenced by `writeModel_Bin_RTP_GL()`, `writeModel_Bin_RTP_TEOS()`, `writeModel_RTP_GL()`, and `writeModel_RTP_TEOS()`.

**9.15.3.38 eos eosTable**

It is of type `eos` and holds the equation of state and opacity information and functions used to provide a tabulated equation of state.

See also

SPHERLS reference manual for a more in depth description of the [eos](#) class.

Referenced by `calculateFirstShell_TEOS()`, `convertBinToHDF4()`, and `readConfig()`.

**9.15.3.39 int nNumCellsCent**

The number of radial cells to include in the higher energy center of a sedov model.

Referenced by `readConfig()`.

**9.15.3.40 unsigned int nNumDims**

number of dimensions for the output model.

Referenced by `readConfig()`.

**9.15.3.41 int nNumEProPoints = 0**

number of points in the energy profile

**9.15.3.42 unsigned int nNumGhostCells**

Number of ghost cells to use while outputting model. It is set in the configuration file with the "<num-ghost-cells>" tag.

Referenced by `convertBinToHDF4()`, `readConfig()`, `writeModel_Bin_R_GL()`, `writeModel_Bin_R_TEOS()`, `writeModel_Bin_RT_GL()`, `writeModel_Bin_RT_TEOS()`, `writeModel_Bin_RTP_GL()`, `writeModel_Bin_RTP_TEOS()`, `writeModel_RT_GL()`, `writeModel_RT_TEOS()`, `writeModel_RTP_GL()`, and `writeModel_RTP_TEOS()`.

**9.15.3.43 int nNumIters**

Allowed number of iterations before stop trying to converge temperature.

Referenced by `calculateFirstShell_TEOS()`, and `readConfig()`.

**9.15.3.44 unsigned int nNumPhi**

Number of phi zones. Set in the configuration file under the "<dimensions>" node using the "<num-phi>" tag.

Referenced by readConfig(), writeModel\_Bin\_R\_GL(), writeModel\_Bin\_R\_TEOS(), writeModel\_Bin\_RT\_GL(), writeModel\_Bin\_RT\_TEOS(), writeModel\_Bin\_RTP\_GL(), writeModel\_Bin\_RTP\_TEOS(), writeModel\_R\_GL(), writeModel\_R\_TEOS(), writeModel\_RT\_GL(), writeModel\_RT\_TEOS(), writeModel\_RTP\_GL(), and writeModel\_RTP\_TEOS().

**9.15.3.45 unsigned int nNumR**

Number of radial zones. When generating a stellar model this value is dependent on the depth to which the model is integrated to (dRStop), the initial mass spacing (dMDelta) and the rate at which the mass spacing is changed (dMDeltaDelta). If generating a sedov model for testing this value is explicitly set by the user.

Referenced by calculateFirstShell\_SEDOV(), generateModel\_SEDOV(), and readConfig().

**9.15.3.46 unsigned int nNumTheta**

Number of theta zones. Set in the configuration file under the "<dimensions>" node using the "<num-theta>" tag.

Referenced by readConfig(), writeModel\_Bin\_R\_GL(), writeModel\_Bin\_R\_TEOS(), writeModel\_Bin\_RT\_GL(), writeModel\_Bin\_RT\_TEOS(), writeModel\_Bin\_RTP\_GL(), writeModel\_Bin\_RTP\_TEOS(), writeModel\_R\_GL(), writeModel\_R\_TEOS(), writeModel\_RT\_GL(), writeModel\_RT\_TEOS(), writeModel\_RTP\_GL(), and writeModel\_RTP\_TEOS().

**9.15.3.47 unsigned int nNumUProPoints = 0**

Number of points in the velocity profile arrays dUPro and dUProR. This value is set in readUProfile by reading it from the velocity profile file.

Referenced by readUProfile().

**9.15.3.48 unsigned int nNumZones1D**

The number of zones to be included in the 1D region. If it is -1 all zones will be output in 1D. It is set in the configuration file with under the "<dimensions>" node with the "<num-1d>" tag.

Referenced by readConfig(), writeModel\_Bin\_R\_GL(), writeModel\_Bin\_R\_TEOS(), writeModel\_Bin\_RT\_GL(), writeModel\_Bin\_RT\_TEOS(), writeModel\_Bin\_RTP\_GL(), writeModel\_Bin\_RTP\_TEOS(), writeModel\_R\_GL(), writeModel\_R\_TEOS(), write-

Model\_RT\_GL(), writeModel\_RT\_TEOS(), writeModel\_RTP\_GL(), and writeModel\_RTP\_TEOS().

#### 9.15.3.49 unsigned int nPeriodic[3] = {0,0,0}

Indicates which directions should have periodic boundary conditions in the evolved model. The three array values are set under the "<periodic>" node with tags "<x0>", "<x1>" and "<x2>" for the three directions, where x0 is the radial direction, x1 is the theta direction, and x2 is the phi direction.

Referenced by readConfig(), writeModel\_Bin\_R\_GL(), writeModel\_Bin\_R\_TEOS(), writeModel\_Bin\_RT\_GL(), writeModel\_Bin\_RT\_TEOS(), writeModel\_Bin\_RTP\_GL(), writeModel\_Bin\_RTP\_TEOS(), writeModel\_R\_GL(), writeModel\_R\_TEOS(), writeModel\_RT\_GL(), writeModel\_RT\_TEOS(), writeModel\_RTP\_GL(), and writeModel\_RTP\_TEOS().

#### 9.15.3.50 unsigned int nPrecision = 14

\ Number of decimal places to include in the output model. It is set in the configuration file with the "<precision>" tag.

Referenced by readConfig(), writeModel\_R\_GL(), writeModel\_R\_TEOS(), writeModel\_RT\_GL(), writeModel\_RT\_TEOS(), writeModel\_RTP\_GL(), writeModel\_RTP\_TEOS(), writeModelToScreen\_GL(), and writeModelToScreen\_TEOS().

#### 9.15.3.51 std::string sEOSFile

filename for the equation of state table file

#### 9.15.3.52 std::string sOutPutfile

Name of the file to write the calculated model to. It is set in the configuration file with the "<fileName>" tag.

Referenced by readConfig(), writeModel\_Bin\_R\_GL(), writeModel\_Bin\_R\_TEOS(), writeModel\_Bin\_RT\_GL(), writeModel\_Bin\_RT\_TEOS(), writeModel\_Bin\_RTP\_GL(), writeModel\_Bin\_RTP\_TEOS(), writeModel\_R\_GL(), writeModel\_R\_TEOS(), writeModel\_RT\_GL(), writeModel\_RT\_TEOS(), writeModel\_RTP\_GL(), and writeModel\_RTP\_TEOS().

#### 9.15.3.53 std::string sUDistType = "POLY"

Specifies the method used to generate the radial velocity profile. It is set in the configuration file as the "type" attribute in the "<velocityDist>" tag. Accepted values are "POLY" for a polynomial profile and "PRO" to use a tabulated velocity profile.

See also

[term](#) for more details about "PLOY" velocity profiles, and [dUPro](#), [dUProR](#), [nNumUProPoints](#), and [dUSurf](#) for more details about "PRO" velocity profiles.

Referenced by `readConfig()`.

#### 9.15.3.54 `std::vector<double> vecdE`

Holds the internal energy of the radial shells. They are zone centered.

Referenced by `calculateFirstShell_SEDOV()`, `calculateFirstShell_TEOS()`, `readConfig()`, `writeModel_Bin_R_GL()`, `writeModel_Bin_RT_GL()`, `writeModel_Bin_RTP_GL()`, `writeModel_R_GL()`, `writeModel_RT_GL()`, `writeModel_RTP_GL()`, `writeModelToScreen_GL()`, and `writeModelToScreen_TEOS()`.

#### 9.15.3.55 `std::vector<double> vecdKappa`

Holds the opacity of the radial shells. They are zone centered.

Referenced by `calculateFirstShell_TEOS()`, and `readConfig()`.

#### 9.15.3.56 `std::vector<double> vecdM`

Holds independent variable, interior mass. They are interface centered.

Referenced by `calculateFirstShell_SEDOV()`, `calculateFirstShell_TEOS()`, and `readConfig()`.

#### 9.15.3.57 `std::vector<double> vecdMdel`

Holds the mass of the shells. They are zone centered

Referenced by `calculateFirstShell_SEDOV()`, `calculateFirstShell_TEOS()`, `readConfig()`, `writeModel_Bin_R_GL()`, `writeModel_Bin_R_TEOS()`, `writeModel_Bin_RT_GL()`, `writeModel_Bin_RT_TEOS()`, `writeModel_Bin_RTP_GL()`, `writeModel_Bin_RTP_TEOS()`, `writeModel_R_GL()`, `writeModel_R_TEOS()`, `writeModel_RT_GL()`, `writeModel_RT_TEOS()`, `writeModel_RTP_GL()`, `writeModel_RTP_TEOS()`, `writeModelToScreen_GL()`, and `writeModelToScreen_TEOS()`.

#### 9.15.3.58 `std::vector<double> vecdP`

Holds the pressures of the radial shells. They are zone centered.

Referenced by `calculateFirstShell_SEDOV()`, `calculateFirstShell_TEOS()`, `readConfig()`, `writeModel_Bin_R_GL()`, `writeModel_Bin_R_TEOS()`, `writeModel_Bin_RT_GL()`, `writeModel_Bin_RT_TEOS()`, `writeModel_Bin_RTP_GL()`, `writeModel_Bin_RTP_TEOS()`, `writeModel_R_GL()`, `writeModel_R_TEOS()`, `writeModel_RT_GL()`,

`writeModel_RT_TEOS()`, `writeModel_RTP_GL()`, `writeModel_RTP_TEOS()`, `writeModelToScreen_GL()`, and `writeModelToScreen_TEOS()`.

#### 9.15.3.59 `std::vector<double> vecdR`

Holds the radius of the radial shells. They are interface centered.

Referenced by `calculateFirstShell_SEDOV()`, `calculateFirstShell_TEOS()`, `generateModel_GL()`, `generateModel_TEOS()`, `readConfig()`, `writeModel_Bin_R_GL()`, `writeModel_Bin_R_TEOS()`, `writeModel_Bin_RT_GL()`, `writeModel_Bin_RT_TEOS()`, `writeModel_Bin_RTP_GL()`, `writeModel_Bin_RTP_TEOS()`, `writeModel_R_GL()`, `writeModel_R_TEOS()`, `writeModel_RT_GL()`, `writeModel_RT_TEOS()`, `writeModel_RTP_GL()`, `writeModel_RTP_TEOS()`, `writeModelToScreen_GL()`, and `writeModelToScreen_TEOS()`.

#### 9.15.3.60 `std::vector<double> vecdRho`

Holds the density of the radial shells. They are zone centered.

Referenced by `calculateFirstShell_SEDOV()`, `calculateFirstShell_TEOS()`, `readConfig()`, `writeModel_Bin_R_GL()`, `writeModel_Bin_R_TEOS()`, `writeModel_Bin_RT_GL()`, `writeModel_Bin_RT_TEOS()`, `writeModel_Bin_RTP_GL()`, `writeModel_Bin_RTP_TEOS()`, `writeModel_R_GL()`, `writeModel_R_TEOS()`, `writeModel_RT_GL()`, `writeModel_RT_TEOS()`, `writeModel_RTP_GL()`, `writeModel_RTP_TEOS()`, `writeModelToScreen_GL()`, and `writeModelToScreen_TEOS()`.

#### 9.15.3.61 `std::vector<double> vecdT`

Holds the temperature of the radial shells. They are zone centered.

Referenced by `calculateFirstShell_TEOS()`, `generateModel_GL()`, `generateModel_TEOS()`, `readConfig()`, `writeModel_Bin_R_TEOS()`, `writeModel_Bin_RT_TEOS()`, `writeModel_Bin_RTP_TEOS()`, `writeModel_R_TEOS()`, `writeModel_RT_TEOS()`, `writeModel_RTP_TEOS()`, and `writeModelToScreen_TEOS()`.

#### 9.15.3.62 `std::vector<term> vectVelDist`

Holds multiple [term](#)'s used to make up a polynomial to calculate the initial radial velocity profile of the model.

See also

[term](#), [sUDistType](#)

Referenced by `readConfig()`.

## 9.16 src/SPHERLS/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)

### 9.16.1 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.

### 9.16.2 Function Documentation

#### 9.16.2.1 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().

#### 9.16.2.2 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().

#### 9.16.2.3 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().

#### 9.16.2.4 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().

### 9.16.2.5 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().

### 9.16.2.6 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 gass 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::nSinThetalJK, Grid::nStartUpdateExplicit, Time::nTimeStepIndex, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

**9.16.2.7 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().

**9.16.2.8 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().

**9.16.2.9** 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 outer interface

References Time::dDeltat\_np1half, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nD, Grid::nDCosThetaJK, Grid::nDonorCellFrac, Grid::nEndGhostUpdateExplicit, 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().

**9.16.2.10** 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 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::nSin-

ThetaUp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

#### 9.16.2.11 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().

#### 9.16.2.12 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().

#### 9.16.2.13 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::nDCosThetaJK, Grid::nDenAve, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

Referenced by setMainFunctions().

#### 9.16.2.14 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	<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::nDCosThetaJK, Grid::nDenAve, Grid::nDPhi, Grid::nEndGhostUpdateExplicit, - Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

Referenced by setMainFunctions().

#### 9.16.2.15 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().

### 9.16.2.16 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().

### 9.16.2.17 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`.

9.16.2.18 `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()`.

### 9.16.2.19 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()`.

### 9.16.2.20 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()`.

**9.16.2.21** `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().

#### 9.16.2.22 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	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	

**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().

#### 9.16.2.23 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()`.

#### 9.16.2.24 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().

#### 9.16.2.25 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.

#### 9.16.2.26 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.

#### 9.16.2.27 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().

#### 9.16.2.28 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().

#### 9.16.2.29 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().

#### 9.16.2.30 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, dG, 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().

#### 9.16.2.31 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().

#### 9.16.2.32 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().

**9.16.2.33 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().

**9.16.2.34 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().

**9.16.2.35 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().

#### 9.16.2.36 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().

#### 9.16.2.37 void calNewQ0Q1Q2\_RTP\_GL ( Grid & *grid*, Parameters & *parameters* )

This function calculates the artificial viscosity of a cell. It calculates it using the new values of quantities and places the result in the new grid. It does this for the radial, theta, and phi componenets of the viscosity. It uses the sound speed derived from the adiabatic gamma given for the gamma law gas equation of state.

## Parameters

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::nSinThetaJK, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

#### 9.16.2.38 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::nSinThetaJK, Grid::nSinThetaJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

#### 9.16.2.39 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().

#### 9.16.2.40 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().

#### 9.16.2.41 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 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().

9.16.2.42 `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()`.

9.16.2.43 `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::nDCosThetaIJK`, `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()`.

#### 9.16.2.44 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][nCen][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()`.

#### 9.16.2.45 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()`.

**9.16.2.46** `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()`.

9.16.2.47 `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

<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.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::deltat_n`, `Parameters::dG`, `Grid::dLocalGridNew`, `Grid::dLocalGridOld`, `Parameters::dPi`, `Grid::nCenIntOffset`, `Grid::nCotThetaJK`, `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::nSinThetaJK`, `Grid::nSinThetaJp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `calNewVelocities_RT_LES()`.

**9.16.2.48** `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::nSinThetaIJK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `calNewVelocities_RTP()`.

**9.16.2.49** `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()`.

**9.16.2.50** `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()`.

9.16.2.51 `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::nSinThetaJK`, `Grid::nSinThetaJp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `calNewVelocities_RT_LES()`.

9.16.2.52 `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().

9.16.2.53 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()`.

**9.16.2.54** `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()`.

**9.16.2.55** `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()`.

**9.16.2.56** `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()`.

**9.16.2.57** `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()`.

**9.16.2.58** `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()`.

#### 9.16.2.59 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()`.

#### 9.16.2.60 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 outter 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()`.

**9.16.2.61** `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()`.

**9.16.2.62** `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.

#### 9.16.2.63 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\(\)](#).

#### 9.16.2.64 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\(\)](#).

#### 9.16.2.65 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().

#### 9.16.2.66 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().

#### 9.16.2.67 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().

#### 9.16.2.68 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().

#### 9.16.2.69 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().

#### 9.16.2.70 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().

9.16.2.71 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	<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, dG, 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().

9.16.2.72 void calOldP\_GL ( Grid & *grid*, Parameters & *parameters* )

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

## Parameters

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

9.16.2.73 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().

9.16.2.74 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().

9.16.2.75 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()`.

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

Referenced by `initInternalVars()`.

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

Referenced by `initInternalVars()`.

### 9.16.2.78 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

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::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

Referenced by initInternalVars().

### 9.16.2.79 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

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::nSinThetaIjp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

Referenced by initInternalVars().

### 9.16.2.80 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  $p(\gamma - 1)E$ .

References Parameters::dGamma.

Referenced by calNewP\_GL(), and calOldP\_GL().

**9.16.2.81** `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().

**9.16.2.82** `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().

**9.16.2.83** `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().

#### 9.16.2.84 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.

**9.16.2.85** `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`.

**9.16.2.86** `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().

**9.16.2.87** `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::nSinThetaIJK, Grid::nSinThetaIp1halfK, Grid::nT, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().

#### 9.16.2.88 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().

**9.16.2.89** `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>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()`.

**9.16.2.90** `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::nSinThetaJk`, `Grid::nSinThetaJp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `setMainFunctions()`.

#### 9.16.291 `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<sub>ijk_np1</sub></i>	is the temperature at radial position $(i, j, k)$ and time $n + 1$ , <code>dTemps[1]=dT<sub>ip1jk_np1</sub></code> is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$ , <code>dTemps[2]=dT<sub>im1jk_np1</sub></code> is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$ , <code>dTemps[3]=dT<sub>ijp1k_np1</sub></code> is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$ , <code>dTemps[4]=dT<sub>ijm1k_np1</sub></code> is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$ , <code>dTemps[5]=dT<sub>ijkp1_np1</sub></code> is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$ , <code>dTemps[6]=dT<sub>ijkm1_np1</sub></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::nSinThetaJk`, `Grid::nSinThetaJp1halfK`, `Grid::nT`, `Grid::nU`, `Grid::nU0`, `Grid::nV`, and `Grid::nW`.

Referenced by `setMainFunctions()`.

9.16.2.92 `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</i> [0]= <i>dT_ijk_np1</i>	is the temperature at radial position $(i, j, k)$ and time $n + 1$ , <i>dTemps</i> [1]= <i>dT_ip1jk_np1</i> is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$ , <i>dTemps</i> [2]= <i>dT_im1jk_np1</i> is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$ , <i>dTemps</i> [3]= <i>dT_ijp1k_np1</i> is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$ , <i>dTemps</i> [4]= <i>dT_ijm1k_np1</i> is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$ , <i>dTemps</i> [5]= <i>dT_ijkp1_np1</i> is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$ , <i>dTemps</i> [6]= <i>dT_ijkm1_np1</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().

9.16.2.93 `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().

**9.16.2.94** `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	<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** 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().

**9.16.2.95** 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().

**9.16.2.96** 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().

**9.16.2.97** 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()`.

**9.16.2.98** `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()`.

**9.16.2.99** `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().

#### 9.16.2.100 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().

#### 9.16.2.101 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().

#### 9.16.2.102 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().

#### 9.16.2.103 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().

#### 9.16.2.104 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().

#### 9.16.2.105 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().

### 9.16.2.106 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().

9.16.2.107 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().

## 9.17 src/SPHERLS/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)

### 9.17.1 Detailed Description

Header file for [physEquations.cpp](#)

### 9.17.2 Function Documentation

#### 9.17.2.1 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().

#### 9.17.2.2 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().

#### 9.17.2.3 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()`.

#### 9.17.2.4 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()`.

### 9.17.2.5 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().

### 9.17.2.6 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().

**9.17.2.7 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().

**9.17.2.8 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().

**9.17.2.9 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().

**9.17.2.10 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().

#### 9.17.2.11 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().

#### 9.17.2.12 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().

#### 9.17.2.13 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::nDCosThetaJK, Grid::nDenAve, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

Referenced by setMainFunctions().

#### 9.17.2.14 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::nDCosThetaJK, Grid::nDenAve, Grid::nDPhi, Grid::nEndGhostUpdateExplicit, - Grid::nEndUpdateExplicit, Grid::nR, Grid::nStartGhostUpdateExplicit, and Grid::nStartUpdateExplicit.

Referenced by setMainFunctions().

#### 9.17.2.15 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().

9.17.2.16 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().

9.17.2.17 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.

9.17.2.18 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::nSinThetaIJK, Grid::nSinThetaIp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, and Grid::nV.

Referenced by setMainFunctions().



9.17.2.19 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()`.

9.17.2.20 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()`.

**9.17.2.21** `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().

**9.17.2.22 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().

**9.17.2.23 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()`.

#### 9.17.2.24 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().

#### 9.17.2.25 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.

#### 9.17.2.26 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.

#### 9.17.2.27 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().

#### 9.17.2.28 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().

#### 9.17.2.29 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().

#### 9.17.2.30 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, dG, Grid::dLocalGridNew, Grid::dLocalGridOld, Grid::nCenIntOffset, Grid::nCotThetaIJK, Grid::nD, Grid::nDPhi, Grid::nDTheta, - Grid::nEddyVisc, Grid::nEndGhostUpdateExplicit, Grid::nEndUpdateExplicit, Grid::nR, Grid::nSinThetaIJK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nU0, Grid::nV, and Grid::nW.

Referenced by setMainFunctions().

#### 9.17.2.31 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().

#### 9.17.2.32 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().

### 9.17.2.33 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().

### 9.17.2.34 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().

### 9.17.2.35 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().

9.17.2.36 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().

9.17.2.37 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().

#### 9.17.2.38 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().

#### 9.17.2.39 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().

9.17.2.40 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()`.

9.17.2.41 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()`.

### 9.17.2.42 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().

### 9.17.2.43 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().

**9.17.2.44** 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][nCen][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().

**9.17.2.45** 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::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()`.

9.17.2.46 `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  $\langle p \rangle_{i+1/2}$ , 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()`.

**9.17.2.47** `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][nCen+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::nSinThetaIp1halfK`, `Grid::nStartGhostUpdateExplicit`, `Grid::nStartUpdateExplicit`, `Grid::nU`, `Grid::nU0`, and `Grid::nV`.

Referenced by `calNewVelocities_RT_LES()`.

9.17.2.48 `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()`.

**9.17.2.49** `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  $\$V\$$  at  $\$i+1\$$  is equal to  $\$v\$$  at  $\$i\$$ .

**Boundary Conditions** Missing pressure outside surface setting it equal to negative pressure in the center of the first cell so that it will be zero at surface.

**Boundary Conditions** assume viscosity is zero outside the star.

**Boundary Conditions** Missing mass outside model, setting it to zero.

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

References `Parameters::dAlpha`, `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()`.

9.17.2.50 `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().

**9.17.2.51** 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

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().

**9.17.2.52** 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().

9.17.2.53 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()`.

**9.17.2.54** `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()`.

**9.17.2.55** `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()`.

**9.17.2.56** `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()`.

**9.17.2.57 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()`.

**9.17.2.58 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()`.

**9.17.2.59** `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()`.

**9.17.2.60** `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()`.

**9.17.2.61** `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

<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** 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()`.

**9.17.2.62** `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.

#### 9.17.2.63 void calOldDenave\_R ( Grid & *grid* )

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

##### Parameters

<i>in, out</i>	<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()`.

#### 9.17.2.64 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

<i>in, out</i>	<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()`.

#### 9.17.2.65 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().

#### 9.17.2.66 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().

#### 9.17.2.67 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().

#### 9.17.2.68 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().

#### 9.17.2.69 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().

#### 9.17.2.70 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().

### 9.17.2.71 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, dG, 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().

### 9.17.2.72 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().

### 9.17.2.73 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().

9.17.2.74 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().

9.17.2.75 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()`.

#### 9.17.2.76 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()`.

#### 9.17.2.77 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()`.

9.17.2.78 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

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::nSinThetalJK, Grid::nSinThetalJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

Referenced by initInternalVars().

9.17.2.79 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

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::nSinThetalJK, Grid::nSinThetalJp1halfK, Grid::nStartGhostUpdateExplicit, Grid::nStartUpdateExplicit, Grid::nU, Grid::nV, and Grid::nW.

Referenced by initInternalVars().

9.17.2.80 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().

**9.17.2.81** `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().

**9.17.2.82** `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().

**9.17.2.83** `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().

#### 9.17.2.84 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.

**9.17.2.85** `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`.

**9.17.2.86** `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().

**9.17.2.87** `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().

#### 9.17.2.88 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()`.

**9.17.2.89** `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()`.

**9.17.2.90** `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()`.

**9.17.2.91** `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()`.

**9.17.2.92** `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</i> [0]= <i>dT_ijk_np1</i>	is the temperature at radial position $(i, j, k)$ and time $n + 1$ , <i>dTemps</i> [1]= <i>dT_ip1jk_np1</i> is the temperature at radial position $(i + 1, j, k)$ and time $n + 1$ , <i>dTemps</i> [2]= <i>dT_im1jk_np1</i> is the temperature at radial position $(i - 1, j, k)$ and time $n + 1$ , <i>dTemps</i> [3]= <i>dT_ijp1k_np1</i> is the temperature at radial position $(i, j + 1, k)$ and time $n + 1$ , <i>dTemps</i> [4]= <i>dT_ijm1k_np1</i> is the temperature at radial position $(i, j - 1, k)$ and time $n + 1$ , <i>dTemps</i> [5]= <i>dT_ijkp1_np1</i> is the temperature at radial position $(i, j, k + 1)$ and time $n + 1$ , <i>dTemps</i> [6]= <i>dT_ijkm1_np1</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()`.

**9.17.2.93** `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().

#### 9.17.2.94 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().

**9.17.2.95** `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().

**9.17.2.96** `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::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().

**9.17.2.97** `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::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()`.

**9.17.2.98** `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()`.

**9.17.2.99** `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().

#### 9.17.2.100 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().

#### 9.17.2.101 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().

#### 9.17.2.102 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().

#### 9.17.2.103 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().

#### 9.17.2.104 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().

#### 9.17.2.105 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().

9.17.2.106 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().

**9.17.2.107 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().

## 9.18 src/SPHERLS/procTop.cpp File Reference

```
#include "procTop.h" #include <cstring>
```

### 9.18.1 Detailed Description

Implementation file for the [ProcTop](#) class

## 9.19 src/SPHERLS/procTop.h File Reference

### Classes

- class [ProcTop](#)

### 9.19.1 Detailed Description

Header file for the [ProcTop](#) class

## 9.20 src/SPHERLS/profileData.h File Reference

```
#include <string>    #include <vector>    #include <map> ×
#include <limits>    #include "time.h"    #include "procTop.h"
#include <fstream>
```

### Classes

- class [profileData](#)

### 9.20.1 Detailed Description

Header file for keepMax::cpp

## 9.21 src/SPHERLS/time.cpp File Reference

```
#include "time.h" #include <limits>
```

### 9.21.1 Detailed Description

Implementation file for the [Time](#) class

## 9.22 src/SPHERLS/time.h File Reference

### Classes

- class [Time](#)

### 9.22.1 Detailed Description

Header file for the [ProcTop](#) class

## 9.23 src/SPHERLS/watchzone.cpp File Reference

```
#include "watchzone.h" #include "exception2.h" #include  
<sstream>
```

### 9.23.1 Detailed Description

This file holds the implementation of the watchzone class.

## 9.24 src/SPHERLS/watchzone.h File Reference

```
#include <string> #include <fstream>
```

### Classes

- class [WatchZone](#)

**9.24.1 Detailed Description**

This file holds the definition of the watchzone class.

**9.25 src/SPHERLSanal/userguide.h File Reference****9.25.1 Detailed Description**

contains the text for the userguide

**9.26 src/SPHERLSgen/userguide.h File Reference****9.26.1 Detailed Description**

contains the text for the userguide

# Index

- ~eos
  - eos, [38](#)
- ALLXMLClearTag, [27](#)
- DEBUG\_EQUATIONS
  - global.h, [155](#)
- DEDEM\_CLAMP
  - global.h, [155](#)
- DUMP\_VERSION
  - global.h, [155](#)
- EOS\_GL
  - en/main.cpp, [164](#)
  - en/main.h, [182](#)
- Functions, [54](#)
  - Functions, [55](#)
  - fpCalculateAveDensities, [55](#)
  - fpCalculateDeltat, [55](#)
  - fpCalculateNewAV, [55](#)
  - fpCalculateNewDensities, [56](#)
  - fpCalculateNewEOSVars, [56](#)
  - fpCalculateNewEddyVisc, [56](#)
  - fpCalculateNewEnergies, [56](#)
  - fpCalculateNewGridVelocities, [56](#)
  - fpCalculateNewRadii, [56](#)
  - fpCalculateNewVelocities, [56](#)
  - fpImplicitSolve, [57](#)
  - fpModelWrite, [57](#)
  - fpUpdateLocalBoundaryVelocities-NewGrid, [57](#)
  - fpWriteWatchZones, [57](#)
  - Functions, [55](#)
- Global, [57](#)
  - Global, [58](#)
  - functions, [58](#)
  - Global, [58](#)
  - grid, [58](#)
  - implicit, [58](#)
  - messPass, [58](#)
  - output, [59](#)
  - parameters, [59](#)
  - performance, [59](#)
  - procTop, [59](#)
  - time, [59](#)
- Grid, [59](#)
  - Grid, [63](#)
  - dLocalGridNew, [64](#)
  - dLocalGridOld, [64](#)
  - Grid, [63](#)
  - nCenIntOffset, [65](#)
  - nCotThetaIJK, [66](#)
  - nCotThetaIjp1halfK, [66](#)
  - nD, [66](#)
  - nDCosThetaIJK, [67](#)
  - nDM, [68](#)
  - nDPhi, [68](#)
  - nDTheta, [69](#)
  - nDenAve, [67](#)
  - nDonorCellFrac, [68](#)
  - nE, [69](#)
  - nEddyVisc, [70](#)
  - nEndGhostUpdateExplicit, [70](#)
  - nEndGhostUpdateImplicit, [71](#)
  - nEndUpdateExplicit, [71](#)
  - nEndUpdateImplicit, [71](#)
  - nGamma, [72](#)
  - nGlobalGridDims, [72](#)
  - nGlobalGridPositionLocalGrid, [72](#)
  - nKappa, [72](#)
  - nLocalGridDims, [73](#)
  - nM, [73](#)
  - nNum1DZones, [73](#)
  - nNumDims, [73](#)
  - nNumGhostCells, [74](#)
  - nNumIntVars, [74](#)
  - nNumVars, [74](#)
  - nP, [74](#)
  - nPhi, [75](#)
  - nQ0, [75](#)
  - nQ1, [76](#)
  - nQ2, [76](#)
  - nR, [77](#)
  - nSinThetaIJK, [77](#)
  - nSinThetaIjp1halfK, [78](#)

- nStartGhostUpdateExplicit, [78](#)
- nStartGhostUpdateImplicit, [78](#)
- nStartUpdateExplicit, [79](#)
- nStartUpdateImplicit, [79](#)
- nT, [80](#)
- nTheta, [80](#)
- nU, [80](#)
- nU0, [81](#)
- nV, [81](#)
- nVariables, [82](#)
- nW, [82](#)
- Implicit, [83](#)
  - Implicit, [84](#)
  - dAverageRHS, [84](#)
  - dCurrentRelTErr, [84](#)
  - dDerivativeStepFraction, [84](#)
  - dMaxErrorInRHS, [85](#)
  - dTolerance, [85](#)
  - Implicit, [84](#)
  - kspContext, [85](#)
  - matCoeff, [85](#)
  - nCurrentNumIterations, [85](#)
  - nLocDer, [85](#)
  - nLocFun, [86](#)
  - nMaxNumIterations, [86](#)
  - nMaxNumSolverIterations, [86](#)
  - nNumDerPerRow, [86](#)
  - nNumImplicitZones, [86](#)
  - nNumRowsALocal, [87](#)
  - nNumRowsALocalSB, [87](#)
  - nTypeDer, [87](#)
  - vecRHS, [87](#)
  - vecTCorrections, [87](#)
  - vecTCorrectionsLocal, [88](#)
  - vecscatTCorrections, [87](#)
- MDDeltaDelta, [92](#)
- MessPass, [92](#)
  - MessPass, [93](#)
  - MessPass, [93](#)
  - requestRecv, [93](#)
  - requestSend, [93](#)
  - statusRecv, [93](#)
  - statusSend, [93](#)
  - typeRecvNewVar, [94](#)
  - typeRecvOldGrid, [94](#)
  - typeSendNewGrid, [94](#)
  - typeSendNewVar, [94](#)
- NextToken, [94](#)
- Output, [98](#)
  - Output, [99](#)
- bDump, [99](#)
- bPrint, [99](#)
- dDumpFrequencyTime, [99](#)
- dPrintFrequencyTime, [99](#)
- dTimeLastDump, [99](#)
- dTimeLastPrint, [100](#)
- nDumpFrequencyStep, [100](#)
- nNumTimeStepsSinceLastPrint, [100](#)
- nPrintFrequencyStep, [100](#)
- nPrintMode, [100](#)
- ofWatchZoneFiles, [100](#)
- Output, [99](#)
- sBaseOutputFileName, [100](#)
- watchzoneList, [101](#)
- Parameters, [101](#)
  - Parameters, [102](#)
  - bAdiabatic, [102](#)
  - bDEDMClamp, [102](#)
  - bEOSGammaLaw, [102](#)
  - dA, [103](#)
  - dAVThreshold, [103](#)
  - dAlpha, [103](#)
  - dDEDMClampMr, [103](#)
  - dDEDMClampValue, [103](#)
  - dDonorCellMin, [104](#)
  - dDonorCellMultiplier, [104](#)
  - dEDMClampTemperature, [104](#)
  - dEddyViscosity, [104](#)
  - dG, [105](#)
  - dGamma, [105](#)
  - dMaxConvectiveVelocity, [105](#)
  - dMaxConvectiveVelocity\_c, [105](#)
  - dPi, [105](#)
  - dPrt, [106](#)
  - dSigma, [106](#)
  - dTolerance, [106](#)
  - eosTable, [106](#)
  - nMaxIterations, [107](#)
  - nTypeTurbulenceMod, [107](#)
  - Parameters, [102](#)
  - sDebugProfileOutput, [107](#)
  - sEOSFileName, [107](#)
- Performance, [108](#)
  - Performance, [108](#)
  - dEndTimer, [108](#)
  - dStartTimer, [108](#)
  - Performance, [108](#)
- ProcTop, [110](#)
  - ProcTop, [111](#)
  - nCoords, [111](#)

- nNeighborRanks, [111](#)
- nNumNeighbors, [111](#)
- nNumProcs, [112](#)
- nNumRadialNeighbors, [112](#)
- nPeriodic, [112](#)
- nProcDims, [112](#)
- nRadialNeighborNeighborIDs, [112](#)
- nRadialNeighborRanks, [113](#)
- nRank, [113](#)
- ProcTop, [111](#)
- SEDV
  - global.h, [155](#)
- SIGNEGDN
  - global.h, [155](#)
- SIGNEGNG
  - global.h, [156](#)
- SIGNEGTEMP
  - global.h, [156](#)
- TRACKMAXSOLVERERROR
  - global.h, [156](#)
- Time, [118](#)
  - Time, [119](#)
  - bVariableTimeStep, [119](#)
  - dConstTimeStep, [119](#)
  - dDeIE\_t\_E\_max, [119](#)
  - dDeIRho\_t\_Rho\_max, [119](#)
  - dDeIT\_t\_T\_max, [119](#)
  - dDeltat\_n, [119](#)
  - dDeltat\_nm1half, [120](#)
  - dDeltat\_np1half, [120](#)
  - dEndTime, [120](#)
  - dPerChange, [120](#)
  - dTimeStepFactor, [121](#)
  - dt, [121](#)
  - nEndTimeStep, [121](#)
  - nTimeStepIndex, [121](#)
  - Time, [119](#)
- VISCOUS\_ENERGY\_EQ
  - global.h, [156](#)
- WatchZone, [122](#)
- XML, [122](#)
- XMLAttribute, [122](#)
- XMLCharacterEntity, [123](#)
- XMLClear, [123](#)
- XMLNode, [123](#)
- XMLNodeContents, [123](#)
- XMLParserBase64Tool, [123](#)
- XMLResults, [123](#)
- \_\_init\_\_
  - dump::dump, [34](#)
- eos\_interp::eosTable, [48](#)
- eos\_interp::eosTableManager, [50](#)
- eos\_interp::interpTable, [88](#)
- eos\_interp::interpTableManager, [90](#)
- eos\_interp::opacityTable, [95](#)
- eos\_interp::opacityTableManager, [97](#)
- make\_hdf2::fileSet, [53](#)
- make\_hdf::fileSet, [52](#)
- plot\_file::Axis, [27](#)
- plot\_file::Curve, [29](#)
- plot\_file::DataSet, [32](#)
- plot\_file::Plot, [109](#)
- plot\_file::Text, [117](#)
- plot\_profile::Axis, [28](#)
- plot\_profile::Curve, [30](#)
- plot\_profile::DataSet, [33](#)
- plot\_profile::Plot, [110](#)
- work\_plot::Settings, [116](#)
- average3DTo1DBoundariesNew
  - dataManipulation.cpp, [127](#)
  - dataManipulation.h, [137](#)
- average3DTo1DBoundariesOld
  - dataManipulation.cpp, [127](#)
  - dataManipulation.h, [137](#)
- bAdiabatic
  - Parameters, [102](#)
- bAutoDeltaM
  - en/main.h, [188](#)
- bBinaryOutput
  - en/main.h, [188](#)
- bDEDMClamp
  - Parameters, [102](#)
- bDump
  - Output, [99](#)
- bEOSGammaLaw
  - Parameters, [102](#)
- bExtraInfoInProfile
  - nal/main.h, [174](#)
- bFileExists
  - nal/main.cpp, [159](#)
  - nal/main.h, [173](#)
- bGammaLawEOS
  - en/main.h, [188](#)
- bIsSedov
  - en/main.h, [188](#)
- bPrint
  - Output, [99](#)
- bScientific

- nal/main.h, [174](#)
- bVariableTimeStep
  - Time, [119](#)
- bWriteToScreen
  - en/main.h, [189](#)
- calDelt\_CONST
  - physEquations.cpp, [203](#)
  - physEquations.h, [264](#)
- calDelt\_R\_GL
  - physEquations.cpp, [203](#)
  - physEquations.h, [264](#)
- calDelt\_R\_TEOS
  - physEquations.cpp, [203](#)
  - physEquations.h, [264](#)
- calDelt\_RT\_GL
  - physEquations.cpp, [204](#)
  - physEquations.h, [265](#)
- calDelt\_RT\_TEOS
  - physEquations.cpp, [204](#)
  - physEquations.h, [265](#)
- calDelt\_RTP\_GL
  - physEquations.cpp, [205](#)
  - physEquations.h, [266](#)
- calDelt\_RTP\_TEOS
  - physEquations.cpp, [206](#)
  - physEquations.h, [267](#)
- calNewD\_R
  - physEquations.cpp, [206](#)
  - physEquations.h, [267](#)
- calNewD\_RT
  - physEquations.cpp, [207](#)
  - physEquations.h, [268](#)
- calNewD\_RTP
  - physEquations.cpp, [207](#)
  - physEquations.h, [268](#)
- calNewDenave\_None
  - physEquations.cpp, [208](#)
  - physEquations.h, [269](#)
- calNewDenave\_R
  - physEquations.cpp, [208](#)
  - physEquations.h, [269](#)
- calNewDenave\_RT
  - physEquations.cpp, [208](#)
  - physEquations.h, [269](#)
- calNewDenave\_RTP
  - physEquations.cpp, [209](#)
  - physEquations.h, [270](#)
- calNewE\_R\_AD
  - physEquations.cpp, [209](#)
  - physEquations.h, [270](#)
- calNewE\_R\_NA
  - physEquations.cpp, [209](#)
  - physEquations.h, [270](#)
- calNewE\_R\_NA\_LES
  - physEquations.cpp, [210](#)
  - physEquations.h, [271](#)
- calNewE\_RT\_AD
  - physEquations.cpp, [211](#)
  - physEquations.h, [272](#)
- calNewE\_RT\_NA
  - physEquations.cpp, [211](#)
  - physEquations.h, [272](#)
- calNewE\_RT\_NA\_LES
  - physEquations.cpp, [212](#)
  - physEquations.h, [273](#)
- calNewE\_RTP\_AD
  - physEquations.cpp, [213](#)
  - physEquations.h, [274](#)
- calNewE\_RTP\_NA
  - physEquations.cpp, [214](#)
  - physEquations.h, [275](#)
- calNewE\_RTP\_NA\_LES
  - physEquations.cpp, [214](#)
  - physEquations.h, [275](#)
- calNewEddyVisc\_None
  - physEquations.cpp, [215](#)
  - physEquations.h, [276](#)
- calNewEddyVisc\_R\_CN
  - physEquations.cpp, [216](#)
  - physEquations.h, [277](#)
- calNewEddyVisc\_R\_SM
  - physEquations.cpp, [216](#)
  - physEquations.h, [277](#)
- calNewEddyVisc\_RT\_CN
  - physEquations.cpp, [216](#)
  - physEquations.h, [277](#)
- calNewEddyVisc\_RT\_SM
  - physEquations.cpp, [217](#)
  - physEquations.h, [278](#)
- calNewEddyVisc\_RTP\_CN
  - physEquations.cpp, [217](#)
  - physEquations.h, [278](#)
- calNewEddyVisc\_RTP\_SM
  - physEquations.cpp, [217](#)
  - physEquations.h, [278](#)
- calNewP\_GL
  - physEquations.cpp, [218](#)
  - physEquations.h, [279](#)
- calNewPEKappaGamma\_TEOS
  - physEquations.h, [270](#)

- physEquations.cpp, 218
- physEquations.h, 279
- calNewQ0\_R\_GL
  - physEquations.cpp, 218
  - physEquations.h, 279
- calNewQ0\_R\_TEOS
  - physEquations.cpp, 219
  - physEquations.h, 280
- calNewQ0Q1\_RT\_GL
  - physEquations.cpp, 219
  - physEquations.h, 280
- calNewQ0Q1\_RT\_TEOS
  - physEquations.cpp, 220
  - physEquations.h, 281
- calNewQ0Q1Q2\_RTP\_GL
  - physEquations.cpp, 220
  - physEquations.h, 281
- calNewQ0Q1Q2\_RTP\_TEOS
  - physEquations.cpp, 221
  - physEquations.h, 282
- calNewR
  - physEquations.cpp, 221
  - physEquations.h, 282
- calNewTPKappaGamma\_TEOS
  - physEquations.cpp, 221
  - physEquations.h, 282
- calNewU0\_R
  - physEquations.cpp, 222
  - physEquations.h, 283
- calNewU0\_RT
  - physEquations.cpp, 222
  - physEquations.h, 283
- calNewU0\_RTP
  - physEquations.cpp, 223
  - physEquations.h, 284
- calNewU\_R
  - physEquations.cpp, 224
  - physEquations.h, 285
- calNewU\_R\_LES
  - physEquations.cpp, 224
  - physEquations.h, 285
- calNewU\_RT
  - physEquations.cpp, 225
  - physEquations.h, 286
- calNewU\_RT\_LES
  - physEquations.cpp, 226
  - physEquations.h, 287
- calNewU\_RTP
  - physEquations.cpp, 227
  - physEquations.h, 288
- calNewU\_RTP\_LES
  - physEquations.cpp, 228
  - physEquations.h, 289
- calNewV\_RT
  - physEquations.cpp, 229
  - physEquations.h, 290
- calNewV\_RT\_LES
  - physEquations.cpp, 230
  - physEquations.h, 291
- calNewV\_RTP
  - physEquations.cpp, 230
  - physEquations.h, 291
- calNewV\_RTP\_LES
  - physEquations.cpp, 231
  - physEquations.h, 292
- calNewVelocities\_R
  - physEquations.cpp, 232
  - physEquations.h, 293
- calNewVelocities\_R\_LES
  - physEquations.cpp, 232
  - physEquations.h, 293
- calNewVelocities\_RT
  - physEquations.cpp, 232
  - physEquations.h, 293
- calNewVelocities\_RT\_LES
  - physEquations.cpp, 233
  - physEquations.h, 294
- calNewVelocities\_RTP
  - physEquations.cpp, 233
  - physEquations.h, 294
- calNewVelocities\_RTP\_LES
  - physEquations.cpp, 234
  - physEquations.h, 295
- calNewW\_RTP
  - physEquations.cpp, 234
  - physEquations.h, 295
- calNewW\_RTP\_LES
  - physEquations.cpp, 235
  - physEquations.h, 296
- calOldDenave\_None
  - physEquations.cpp, 235
  - physEquations.h, 296
- calOldDenave\_R
  - physEquations.cpp, 236
  - physEquations.h, 297
- calOldDenave\_RT
  - physEquations.cpp, 236
  - physEquations.h, 297
- calOldDenave\_RTP
  - physEquations.cpp, 236



- physEquations.h, [297](#)
- calOldEddyVisc\_R\_CN
  - physEquations.cpp, [237](#)
  - physEquations.h, [298](#)
- calOldEddyVisc\_R\_SM
  - physEquations.cpp, [237](#)
  - physEquations.h, [298](#)
- calOldEddyVisc\_RT\_CN
  - physEquations.cpp, [237](#)
  - physEquations.h, [298](#)
- calOldEddyVisc\_RT\_SM
  - physEquations.cpp, [238](#)
  - physEquations.h, [299](#)
- calOldEddyVisc\_RTP\_CN
  - physEquations.cpp, [238](#)
  - physEquations.h, [299](#)
- calOldEddyVisc\_RTP\_SM
  - physEquations.cpp, [238](#)
  - physEquations.h, [299](#)
- calOldP\_GL
  - physEquations.cpp, [239](#)
  - physEquations.h, [300](#)
- calOldPEKappaGamma\_TEOS
  - physEquations.cpp, [239](#)
  - physEquations.h, [300](#)
- calOldQ0\_R\_GL
  - physEquations.cpp, [240](#)
  - physEquations.h, [301](#)
- calOldQ0\_R\_TEOS
  - physEquations.cpp, [240](#)
  - physEquations.h, [301](#)
- calOldQ0Q1\_RT\_GL
  - physEquations.cpp, [241](#)
  - physEquations.h, [302](#)
- calOldQ0Q1\_RT\_TEOS
  - physEquations.cpp, [241](#)
  - physEquations.h, [302](#)
- calOldQ0Q1Q2\_RTP\_GL
  - physEquations.cpp, [241](#)
  - physEquations.h, [302](#)
- calOldQ0Q1Q2\_RTP\_TEOS
  - physEquations.cpp, [242](#)
  - physEquations.h, [303](#)
- calculateCurve
  - light\_curve::LightCurve, [91](#)
- calculateFirstShell\_SEDOV
  - en/main.cpp, [163](#)
  - en/main.h, [181](#)
- calculateFirstShell\_TEOS
  - en/main.cpp, [163](#)
- en/main.h, [181](#)
- clear
  - profileData, [114](#)
- computeFourierTransFromList
  - nal/main.cpp, [159](#)
  - nal/main.h, [173](#)
- convertBinToHDF4
  - nal/main.cpp, [159](#)
  - nal/main.h, [174](#)
- convertDumpToHDF
  - make\_hdf2::fileSet, [54](#)
  - make\_hdf::fileSet, [52](#)
- createTables
  - eos\_interp::interpTableManager, [91](#)
- dA
  - Parameters, [103](#)
- dAVThreshold
  - Parameters, [103](#)
- dAlpha
  - en/main.h, [189](#)
  - Parameters, [103](#)
- dAverageRHS
  - Implicit, [84](#)
- dCalRhoAve2D
  - nal/main.cpp, [159](#)
  - nal/main.h, [174](#)
- dCoeff
  - term, [117](#)
- dConstTimeStep
  - Time, [119](#)
- dCurrentRelTErrors
  - Implicit, [84](#)
- dDEDMClampMr
  - Parameters, [103](#)
- dDEDMClampValue
  - Parameters, [103](#)
- dDRhoDP
  - eos, [38](#)
- dDelE\_t\_E\_max
  - Time, [119](#)
- dDelRho\_t\_Rho\_max
  - Time, [119](#)
- dDelT\_t\_T\_max
  - Time, [119](#)
- dDeltaPhi
  - en/main.h, [189](#)
- dDeltaTheta
  - en/main.h, [189](#)
- dDeltat\_n

- Time, 119
- dDeltat\_nm1half
  - Time, 120
- dDeltat\_np1half
  - Time, 120
- dDerivativeStepFraction
  - Implicit, 84
- dDonorCellMin
  - Parameters, 104
- dDonorCellMultiplier
  - Parameters, 104
- dDumpFrequencyTime
  - Output, 99
- dEDMClampTemperature
  - Parameters, 104
- dEOS\_GL
  - physEquations.cpp, 242
  - physEquations.h, 303
- dEPro
  - en/main.h, 190
- dEProM
  - en/main.h, 190
- dET4
  - physEquations.cpp, 243
  - physEquations.h, 304
- dEddyViscosity
  - Parameters, 104
- dEndTime
  - Time, 120
- dEndTimer
  - Performance, 108
- dEng
  - en/main.h, 189
- dEngCent
  - en/main.h, 189
- dG
  - en/main.h, 190
  - Parameters, 105
- dGamma
  - en/main.h, 190
  - Parameters, 105
- dGetEnergy
  - eos, 38
- dGetOpacity
  - eos, 39
- dGetPressure
  - eos, 39
- dImplicitEnergyFunction\_None
  - physEquations.cpp, 243
  - physEquations.h, 304
- dImplicitEnergyFunction\_R
  - physEquations.cpp, 243
  - physEquations.h, 304
- dImplicitEnergyFunction\_R\_LES
  - physEquations.cpp, 244
  - physEquations.h, 305
- dImplicitEnergyFunction\_R\_LES\_SB
  - physEquations.cpp, 245
  - physEquations.h, 306
- dImplicitEnergyFunction\_R\_SB
  - physEquations.cpp, 245
  - physEquations.h, 306
- dImplicitEnergyFunction\_RT
  - physEquations.cpp, 246
  - physEquations.h, 307
- dImplicitEnergyFunction\_RT\_LES
  - physEquations.cpp, 247
  - physEquations.h, 308
- dImplicitEnergyFunction\_RT\_LES\_SB
  - physEquations.cpp, 248
  - physEquations.h, 309
- dImplicitEnergyFunction\_RT\_SB
  - physEquations.cpp, 249
  - physEquations.h, 310
- dImplicitEnergyFunction\_RTP
  - physEquations.cpp, 250
  - physEquations.h, 311
- dImplicitEnergyFunction\_RTP\_LES
  - physEquations.cpp, 250
  - physEquations.h, 311
- dImplicitEnergyFunction\_RTP\_LES\_SB
  - physEquations.cpp, 251
  - physEquations.h, 312
- dImplicitEnergyFunction\_RTP\_SB
  - physEquations.cpp, 253
  - physEquations.h, 314
- dL
  - en/main.h, 190
- dLSun
  - en/main.h, 190
  - nal/main.h, 175
- dLocalGridNew
  - Grid, 64
- dLocalGridOld
  - Grid, 64
- dLogE
  - eos, 45
- dLogKappa
  - eos, 45
- dLogP

- eos, [46](#)
- dLogRhoDelta
  - eos, [46](#)
- dLogRhoMin
  - eos, [46](#)
- dLogTDelta
  - eos, [46](#)
- dLogTMin
  - eos, [46](#)
- dMDelta
  - en/main.h, [190](#)
- dMDeltaDelta
  - en/main.h, [191](#)
- dMSun
  - en/main.h, [191](#)
- dMTotal
  - en/main.h, [191](#)
- dMaxConvectiveVelocity
  - Parameters, [105](#)
- dMaxConvectiveVelocity\_c
  - Parameters, [105](#)
- dMaxErrorInRHS
  - Implicit, [85](#)
- dPartialDerivativeVar1
  - en/main.cpp, [164](#)
  - en/main.h, [182](#)
- dPartialDerivativeVar2
  - en/main.cpp, [164](#)
  - en/main.h, [182](#)
- dPerChange
  - Time, [120](#)
- dPi
  - en/main.h, [191](#)
  - nal/main.h, [175](#)
  - Parameters, [105](#)
- dPower
  - term, [117](#)
- dPrintFrequencyTime
  - Output, [99](#)
- dPrt
  - Parameters, [106](#)
- dRDelta
  - en/main.h, [191](#)
- dRMin
  - en/main.h, [191](#)
- dRSun
  - en/main.h, [191](#)
- dRSurf
  - en/main.h, [192](#)
- dRho
  - en/main.h, [191](#)
- dSigma
  - en/main.h, [192](#)
  - nal/main.h, [175](#)
  - Parameters, [106](#)
- dSoundSpeed
  - eos, [40](#)
- dStartTimer
  - Performance, [108](#)
- dTeff
  - en/main.h, [192](#)
- dTimeLastDump
  - Output, [99](#)
- dTimeLastPrint
  - Output, [100](#)
- dTimeStepFactor
  - en/main.h, [192](#)
  - Time, [121](#)
- dTolerance
  - en/main.h, [192](#)
  - Implicit, [85](#)
  - Parameters, [106](#)
- dU
  - en/main.h, [192](#)
- dU0
  - en/main.h, [193](#)
- dUPro
  - en/main.h, [193](#)
- dUProR
  - en/main.h, [193](#)
- dUSurf
  - en/main.h, [193](#)
- dV
  - en/main.h, [193](#)
- dW
  - en/main.h, [193](#)
- dXMassFrac
  - eos, [47](#)
- dYMassFrac
  - eos, [47](#)
- dataManipulation.cpp
  - average3DTo1DBoundariesNew, [127](#)
  - average3DTo1DBoundariesOld, [127](#)
  - fin, [128](#)
  - init, [128](#)
  - initImplicitCalculation, [129](#)
  - initUpdateLocalBoundaries, [130](#)
  - modelRead, [131](#)
  - modelWrite\_GL, [131](#)
  - modelWrite\_TEOS, [132](#)

- setDEDMClamp, [132](#)
- setupLocalGrid, [133](#)
- updateLocalBoundaries, [133](#)
- updateLocalBoundariesNewGrid, [134](#)
- updateLocalBoundaryVelocitiesNewGrid\_R, [134](#)
- updateLocalBoundaryVelocitiesNewGrid\_RT, [135](#)
- updateLocalBoundaryVelocitiesNewGrid\_RTP, [135](#)
- updateNewGridWithOld, [135](#)
- updateOldGrid, [135](#)
- dataManipulation.h
  - average3DTo1DBoundariesNew, [137](#)
  - average3DTo1DBoundariesOld, [137](#)
  - fin, [138](#)
  - init, [138](#)
  - initImplicitCalculation, [139](#)
  - initUpdateLocalBoundaries, [140](#)
  - modelRead, [141](#)
  - modelWrite\_GL, [141](#)
  - modelWrite\_TEOS, [142](#)
  - setDEDMClamp, [142](#)
  - setupLocalGrid, [143](#)
  - updateLocalBoundaries, [143](#)
  - updateLocalBoundariesNewGrid, [144](#)
  - updateLocalBoundaryVelocitiesNewGrid\_R, [144](#)
  - updateLocalBoundaryVelocitiesNewGrid\_RT, [145](#)
  - updateLocalBoundaryVelocitiesNewGrid\_RTP, [145](#)
  - updateNewGridWithOld, [145](#)
  - updateOldGrid, [145](#)
- dataMonitoring.cpp
  - finWatchZones, [147](#)
  - initWatchZones, [147](#)
  - writeWatchZones\_R\_GL, [147](#)
  - writeWatchZones\_R\_TEOS, [148](#)
  - writeWatchZones\_RT\_GL, [148](#)
  - writeWatchZones\_RT\_TEOS, [148](#)
  - writeWatchZones\_RTP\_GL, [149](#)
  - writeWatchZones\_RTP\_TEOS, [149](#)
- dataMonitoring.h
  - finWatchZones, [150](#)
  - initWatchZones, [151](#)
  - writeWatchZones\_R\_GL, [151](#)
  - writeWatchZones\_R\_TEOS, [152](#)
  - writeWatchZones\_RT\_GL, [152](#)
  - writeWatchZones\_RT\_TEOS, [152](#)
  - writeWatchZones\_RTP\_GL, [153](#)
  - writeWatchZones\_RTP\_TEOS, [153](#)
- datafile.DataFile, [31](#)
- datafile::DataFile
  - readFile, [31](#)
  - readFileFixed, [31](#)
  - readFileUnFixed, [31](#)
- dt
  - Time, [121](#)
- dump.dump, [34](#)
- dump::dump
  - \_\_init\_\_, [34](#)
  - printHeader, [34](#)
  - printVar, [34](#)
  - read, [35](#)
  - readAsciiVar, [35](#)
  - readBinaryVar, [35](#)
  - readHeader, [35](#)
  - readHeaderAscii, [35](#)
  - readHeaderBinary, [36](#)
  - setVarIDs, [36](#)
- en/main.cpp
  - EOS\_GL, [164](#)
  - calculateFirstShell\_SEDOV, [163](#)
  - calculateFirstShell\_TEOS, [163](#)
  - dPartialDerivativeVar1, [164](#)
  - dPartialDerivativeVar2, [164](#)
  - generateModel\_GL, [164](#)
  - generateModel\_SEDOV, [165](#)
  - generateModel\_TEOS, [165](#)
  - main, [165](#)
  - readConfig, [165](#)
  - readUPProfile, [166](#)
  - writeModel\_Bin\_R\_GL, [166](#)
  - writeModel\_Bin\_R\_TEOS, [167](#)
  - writeModel\_Bin\_RT\_GL, [167](#)
  - writeModel\_Bin\_RT\_TEOS, [167](#)
  - writeModel\_Bin\_RTP\_GL, [167](#)
  - writeModel\_Bin\_RTP\_TEOS, [168](#)
  - writeModel\_R\_GL, [168](#)
  - writeModel\_R\_TEOS, [168](#)
  - writeModel\_RT\_GL, [168](#)
  - writeModel\_RT\_TEOS, [169](#)
  - writeModel\_RTP\_GL, [169](#)
  - writeModel\_RTP\_TEOS, [169](#)
  - writeModelToScreen\_GL, [170](#)
  - writeModelToScreen\_TEOS, [170](#)

---

en/main.h  
   EOS\_GL, 182  
   bAutoDeltaM, 188  
   bBinaryOutput, 188  
   bGammaLawEOS, 188  
   bIsSedov, 188  
   bWriteToScreen, 189  
   calculateFirstShell\_SEDOV, 181  
   calculateFirstShell\_TEOS, 181  
   dAlpha, 189  
   dDeltaPhi, 189  
   dDeltaTheta, 189  
   dEPro, 190  
   dEProM, 190  
   dEng, 189  
   dEngCent, 189  
   dG, 190  
   dGamma, 190  
   dL, 190  
   dLSun, 190  
   dMDelta, 190  
   dMDeltaDelta, 191  
   dMSun, 191  
   dMTotal, 191  
   dPartialDerivativeVar1, 182  
   dPartialDerivativeVar2, 182  
   dPi, 191  
   dRDelta, 191  
   dRMin, 191  
   dRSun, 191  
   dRSurf, 192  
   dRho, 191  
   dSigma, 192  
   dTeff, 192  
   dTimeStepFactor, 192  
   dTolerance, 192  
   dU, 192  
   dU0, 193  
   dUPro, 193  
   dUProR, 193  
   dUSurf, 193  
   dV, 193  
   dW, 193  
   eosTable, 194  
   generateModel\_GL, 183  
   generateModel\_SEDOV, 183  
   generateModel\_TEOS, 183  
   main, 183  
   nNumCellsCent, 194  
   nNumDims, 194  
   nNumEProPoints, 194  
   nNumGhostCells, 194  
   nNumIters, 194  
   nNumPhi, 194  
   nNumR, 195  
   nNumTheta, 195  
   nNumUProPoints, 195  
   nNumZones1D, 195  
   nPeriodic, 196  
   nPrecision, 196  
   readConfig, 184  
   readUProfile, 184  
   sEOSFile, 196  
   sOutPutfile, 196  
   sUDistType, 196  
   vecdE, 197  
   vecdKappa, 197  
   vecdM, 197  
   vecdMDEL, 197  
   vecdP, 197  
   vecdR, 198  
   vecdRho, 198  
   vecdT, 198  
   vectVelDist, 198  
   writeModel\_Bin\_R\_GL, 185  
   writeModel\_Bin\_R\_TEOS, 185  
   writeModel\_Bin\_RT\_GL, 185  
   writeModel\_Bin\_RT\_TEOS, 185  
   writeModel\_Bin\_RTP\_GL, 186  
   writeModel\_Bin\_RTP\_TEOS, 186  
   writeModel\_R\_GL, 186  
   writeModel\_R\_TEOS, 186  
   writeModel\_RT\_GL, 187  
   writeModel\_RT\_TEOS, 187  
   writeModel\_RTP\_GL, 187  
   writeModel\_RTP\_TEOS, 187  
   writeModelToScreen\_GL, 188  
   writeModelToScreen\_TEOS, 188  
 eos, 36  
   ~eos, 38  
   dDRhoDP, 38  
   dGetEnergy, 38  
   dGetOpacity, 39  
   dGetPressure, 39  
   dLogE, 45  
   dLogKappa, 45  
   dLogP, 46  
   dLogRhoDelta, 46  
   dLogRhoMin, 46  
   dLogTDelta, 46

- dLogTMin, 46
- dSoundSpeed, 40
- dXMassFrac, 47
- dYMassFrac, 47
- eos, 38
- gamma1DelAdC\_v, 40
- getDlnPDlnTDlnPDlnPDEDT, 41
- getEAndDTDE, 41
- getEKappa, 41
- getPAndDRhoDP, 42
- getPEKappa, 42
- getPEKappaGamma, 42
- getPEKappaGammaCp, 43
- getPKappaGamma, 43
- nNumRho, 47
- nNumT, 47
- operator=, 44
- readAscii, 44
- readBin, 44
- readBobsAscii, 44
- writeAscii, 45
- writeBin, 45
- eos\_interp.eosTable, 47
- eos\_interp.eosTableManager, 50
- eos\_interp.interpTable, 88
- eos\_interp.interpTableManager, 90
- eos\_interp.opacityTable, 95
- eos\_interp.opacityTableManager, 96
- eos\_interp::eosTable
  - \_\_init\_\_, 48
  - interpolate, 48
  - load, 48
  - plotLogE, 49
  - plotLogP, 49
  - write, 49
- eos\_interp::eosTableManager
  - \_\_init\_\_, 50
  - getTableFromComp, 50
  - interpComp, 50
  - load, 51
  - plotGrid, 51
- eos\_interp::interpTable
  - \_\_init\_\_, 88
  - interpolate, 89
  - plotLogE, 89
  - plotLogK, 89
  - plotLogP, 89
  - read, 90
- eos\_interp::interpTableManager
  - \_\_init\_\_, 90
  - createTables, 91
- eos\_interp::opacityTable
  - \_\_init\_\_, 95
  - fillInDepNans, 95
  - interpolate, 95
  - load, 96
  - plotLogK, 96
- eos\_interp::opacityTableManager
  - \_\_init\_\_, 97
  - getTableFromComp, 97
  - interpComp, 97
  - load, 97
  - plotGrids, 98
- eosTable
  - en/main.h, 194
  - Parameters, 106
- exception2, 51
- fillInDepNans
  - eos\_interp::opacityTable, 95
- fin
  - dataManipulation.cpp, 128
  - dataManipulation.h, 138
- finWatchZones
  - dataMonitoring.cpp, 147
  - dataMonitoring.h, 150
- fpCalculateAveDensities
  - Functions, 55
- fpCalculateDeltat
  - Functions, 55
- fpCalculateNewAV
  - Functions, 55
- fpCalculateNewDensities
  - Functions, 56
- fpCalculateNewEOSVars
  - Functions, 56
- fpCalculateNewEddyVisc
  - Functions, 56
- fpCalculateNewEnergies
  - Functions, 56
- fpCalculateNewGridVelocities
  - Functions, 56
- fpCalculateNewRadii
  - Functions, 56
- fpCalculateNewVelocities
  - Functions, 56
- fpImplicitSolve
  - Functions, 57
- fpModelWrite
  - Functions, 57

- fpUpdateLocalBoundaryVelocitiesNew-Grid
  - Functions, [57](#)
- fpWriteWatchZones
  - Functions, [57](#)
- functions
  - Global, [58](#)
- gamma1DelAdC\_v
  - eos, [40](#)
- generateModel\_GL
  - en/main.cpp, [164](#)
  - en/main.h, [183](#)
- generateModel\_SEDOV
  - en/main.cpp, [165](#)
  - en/main.h, [183](#)
- generateModel\_TEOS
  - en/main.cpp, [165](#)
  - en/main.h, [183](#)
- getCurve
  - plot\_file::DataSet, [32](#)
  - plot\_profile::DataSet, [33](#)
- getDlnPDlnTDlnPDlnPDEDT
  - eos, [41](#)
- getEAndDTDE
  - eos, [41](#)
- getEKappa
  - eos, [41](#)
- getPAndDRhoDP
  - eos, [42](#)
- getPEKappa
  - eos, [42](#)
- getPEKappaGamma
  - eos, [42](#)
- getPEKappaGammaCp
  - eos, [43](#)
- getPKappaGamma
  - eos, [43](#)
- getTableFromComp
  - eos\_interp::eosTableManager, [50](#)
  - eos\_interp::opacityTableManager, [97](#)
- global.h
  - DEBUG\_EQUATIONS, [155](#)
  - DEDEM\_CLAMP, [155](#)
  - DUMP\_VERSION, [155](#)
  - SEDOV, [155](#)
  - SIGNEGDN, [155](#)
  - SIGNEGNG, [156](#)
  - SIGNEGTEMP, [156](#)
  - TRACKMAXSOLVERERROR, [156](#)
  - VISCOUS\_ENERGY\_EQ, [156](#)
- grid
  - Global, [58](#)
- implicit
  - Global, [58](#)
- implicitSolve\_None
  - physEquations.cpp, [254](#)
  - physEquations.h, [315](#)
- implicitSolve\_R
  - physEquations.cpp, [254](#)
  - physEquations.h, [315](#)
- implicitSolve\_RT
  - physEquations.cpp, [254](#)
  - physEquations.h, [315](#)
- implicitSolve\_RTP
  - physEquations.cpp, [255](#)
  - physEquations.h, [316](#)
- init
  - dataManipulation.cpp, [128](#)
  - dataManipulation.h, [138](#)
- initDonorFracAndMaxConVel\_R\_GL
  - physEquations.cpp, [255](#)
  - physEquations.h, [316](#)
- initDonorFracAndMaxConVel\_R\_TEOS
  - physEquations.cpp, [256](#)
  - physEquations.h, [317](#)
- initDonorFracAndMaxConVel\_RT\_GL
  - physEquations.cpp, [256](#)
  - physEquations.h, [317](#)
- initDonorFracAndMaxConVel\_RT\_TEOS
  - physEquations.cpp, [256](#)
  - physEquations.h, [317](#)
- initDonorFracAndMaxConVel\_RTP\_GL
  - physEquations.cpp, [257](#)
  - physEquations.h, [318](#)
- initDonorFracAndMaxConVel\_RTP\_TEO-S
  - physEquations.cpp, [257](#)
  - physEquations.h, [318](#)
- initImplicitCalculation
  - dataManipulation.cpp, [129](#)
  - dataManipulation.h, [139](#)
- initInternalVars
  - physEquations.cpp, [257](#)
  - physEquations.h, [318](#)
- initUpdateLocalBoundaries
  - dataManipulation.cpp, [130](#)
  - dataManipulation.h, [140](#)
- initWatchZones

- dataMonitoring.cpp, 147
- dataMonitoring.h, 151
- interpComp
  - eos\_interp::eosTableManager, 50
  - eos\_interp::opacityTableManager, 97
- interpolate
  - eos\_interp::eosTable, 48
  - eos\_interp::interpTable, 89
  - eos\_interp::opacityTable, 95
- kspContext
  - Implicit, 85
- light\_curve.LightCurve, 91
- light\_curve::LightCurve
  - calculateCurve, 91
  - readBoloCorr, 91
  - readProfiles, 92
  - write, 92
- load
  - eos\_interp::eosTable, 48
  - eos\_interp::eosTableManager, 51
  - eos\_interp::opacityTable, 96
  - eos\_interp::opacityTableManager, 97
  - plot\_2DSlices::File2DSlice, 51
  - plot\_file::Axis, 28
  - plot\_file::Curve, 29
  - plot\_file::DataSet, 32
  - plot\_file::Plot, 109
  - plot\_profile::Axis, 28
  - plot\_profile::Curve, 30
  - plot\_profile::DataSet, 33
  - plot\_profile::Plot, 110
- main
  - en/main.cpp, 165
  - en/main.h, 183
  - main.cpp, 157
  - main.h, 170
  - nal/main.cpp, 159
- main.cpp
  - main, 157
  - signalHandler, 158
- main.h
  - main, 170
  - signalHandler, 172
- make\_hdf.fileSet, 52
- make\_hdf.hdfFile, 83
- make\_hdf.interpVar, 91
- make\_hdf.variable, 122
- make\_hdf2.fileSet, 53
- make\_hdf2::fileSet
  - \_\_init\_\_, 53
  - convertDumpToHDF, 54
  - makeHDFFiles, 54
- make\_hdf::fileSet
  - \_\_init\_\_, 52
  - convertDumpToHDF, 52
  - makeHDFFiles, 53
- make\_hdf::hdfFile
  - write, 83
- makeHDFFiles
  - make\_hdf2::fileSet, 54
  - make\_hdf::fileSet, 53
- matCoeff
  - Implicit, 85
- messPass
  - Global, 58
- modelRead
  - dataManipulation.cpp, 131
  - dataManipulation.h, 141
- modelWrite\_GL
  - dataManipulation.cpp, 131
  - dataManipulation.h, 141
- modelWrite\_TEOS
  - dataManipulation.cpp, 132
  - dataManipulation.h, 142
- nC
  - nal/main.h, 175
- nCenIntOffset
  - Grid, 65
- nCoords
  - ProcTop, 111
- nCotThetaJK
  - Grid, 66
- nCotThetaJp1halfK
  - Grid, 66
- nCurrentNumIterations
  - Implicit, 85
- nD
  - Grid, 66
  - nal/main.h, 175
- nDCosThetaJK
  - Grid, 67
- nDM
  - Grid, 68
  - nal/main.h, 175
- nDPhi
  - Grid, 68



- nDTheta
  - Grid, [69](#)
- nDenAve
  - Grid, [67](#)
- nDonorCellFrac
  - Grid, [68](#)
- nDumpFileVersion
  - nal/main.h, [175](#)
- nDumpFrequencyStep
  - Output, [100](#)
- nE
  - Grid, [69](#)
  - nal/main.h, [175](#)
- nEddyVisc
  - Grid, [70](#)
- nEndGhostUpdateExplicit
  - Grid, [70](#)
- nEndGhostUpdateImplicit
  - Grid, [71](#)
- nEndTimeStep
  - Time, [121](#)
- nEndUpdateExplicit
  - Grid, [71](#)
- nEndUpdateImplicit
  - Grid, [71](#)
- nF\_con
  - nal/main.h, [176](#)
- nGamma
  - Grid, [72](#)
  - nal/main.h, [176](#)
- nGlobalGridDims
  - Grid, [72](#)
- nGlobalGridPositionLocalGrid
  - Grid, [72](#)
- nKE
  - nal/main.h, [176](#)
- nKappa
  - Grid, [72](#)
  - nal/main.h, [176](#)
- nL\_con
  - nal/main.h, [176](#)
- nL\_rad
  - nal/main.h, [176](#)
- nLocDer
  - Implicit, [85](#)
- nLocFun
  - Implicit, [86](#)
- nLocalGridDims
  - Grid, [73](#)
- nM
  - Grid, [73](#)
  - nal/main.h, [176](#)
- nMaxIterations
  - Parameters, [107](#)
- nMaxNumIterations
  - Implicit, [86](#)
- nMaxNumSolverIterations
  - Implicit, [86](#)
- nMaxNumZones
  - profileData, [114](#)
- nNeighborRanks
  - ProcTop, [111](#)
- nNum1DZones
  - Grid, [73](#)
- nNumCellsCent
  - en/main.h, [194](#)
- nNumDerPerRow
  - Implicit, [86](#)
- nNumDims
  - en/main.h, [194](#)
  - Grid, [73](#)
- nNumEProPoints
  - en/main.h, [194](#)
- nNumGhostCells
  - en/main.h, [194](#)
  - Grid, [74](#)
- nNumImplicitZones
  - Implicit, [86](#)
- nNumIntVars
  - Grid, [74](#)
- nNumIters
  - en/main.h, [194](#)
- nNumNeighbors
  - ProcTop, [111](#)
- nNumPhi
  - en/main.h, [194](#)
- nNumProcs
  - ProcTop, [112](#)
- nNumR
  - en/main.h, [195](#)
- nNumRadialNeighbors
  - ProcTop, [112](#)
- nNumRho
  - eos, [47](#)
- nNumRowsALocal
  - Implicit, [87](#)
- nNumRowsALocalSB
  - Implicit, [87](#)
- nNumT
  - eos, [47](#)

- nNumTheta
  - en/main.h, [195](#)
- nNumTimeStepsSinceLastPrint
  - Output, [100](#)
- nNumUProPoints
  - en/main.h, [195](#)
- nNumVars
  - Grid, [74](#)
- nNumZones1D
  - en/main.h, [195](#)
- nP
  - Grid, [74](#)
  - nal/main.h, [176](#)
- nPeriodic
  - en/main.h, [196](#)
  - ProcTop, [112](#)
- nPhi
  - Grid, [75](#)
  - nal/main.h, [177](#)
- nPrecision
  - en/main.h, [196](#)
- nPrecisionAscii
  - nal/main.h, [177](#)
- nPrintFrequencyStep
  - Output, [100](#)
- nPrintMode
  - Output, [100](#)
- nProcDims
  - ProcTop, [112](#)
- nQ
  - nal/main.h, [177](#)
- nQ0
  - Grid, [75](#)
- nQ1
  - Grid, [76](#)
- nQ2
  - Grid, [76](#)
- nR
  - Grid, [77](#)
  - nal/main.h, [177](#)
- nRadialNeighborNeighborIDs
  - ProcTop, [112](#)
- nRadialNeighborRanks
  - ProcTop, [113](#)
- nRank
  - ProcTop, [113](#)
- nSinThetaJK
  - Grid, [77](#)
- nSinThetaJp1halfK
  - Grid, [78](#)
- nStartGhostUpdateExplicit
  - Grid, [78](#)
- nStartGhostUpdateImplicit
  - Grid, [78](#)
- nStartUpdateExplicit
  - Grid, [79](#)
- nStartUpdateImplicit
  - Grid, [79](#)
- nT
  - Grid, [80](#)
  - nal/main.h, [177](#)
- nTheta
  - Grid, [80](#)
  - nal/main.h, [177](#)
- nTimeStepIndex
  - Time, [121](#)
- nTypeDer
  - Implicit, [87](#)
- nTypeTurbulenceMod
  - Parameters, [107](#)
- nU
  - Grid, [80](#)
  - nal/main.h, [177](#)
- nU0
  - Grid, [81](#)
  - nal/main.h, [178](#)
- nV
  - Grid, [81](#)
  - nal/main.h, [178](#)
- nVariables
  - Grid, [82](#)
- nW
  - Grid, [82](#)
  - nal/main.h, [178](#)
- nal/main.cpp
  - bFileExists, [159](#)
  - computeFourierTransFromList, [159](#)
  - convertBinToHDF4, [159](#)
  - dCalRhoAve2D, [159](#)
  - main, [159](#)
  - printHelp, [161](#)
- nal/main.h
  - bExtraInfoInProfile, [174](#)
  - bFileExists, [173](#)
  - bScientific, [174](#)
  - computeFourierTransFromList, [173](#)
  - convertBinToHDF4, [174](#)
  - dCalRhoAve2D, [174](#)
  - dLSun, [175](#)
  - dPi, [175](#)

- dSigma, [175](#)
- nC, [175](#)
- nD, [175](#)
- nDM, [175](#)
- nDumpFileVersion, [175](#)
- nE, [175](#)
- nF\_con, [176](#)
- nGamma, [176](#)
- nKE, [176](#)
- nKappa, [176](#)
- nL\_con, [176](#)
- nL\_rad, [176](#)
- nM, [176](#)
- nP, [176](#)
- nPhi, [177](#)
- nPrecisionAscii, [177](#)
- nQ, [177](#)
- nR, [177](#)
- nT, [177](#)
- nTheta, [177](#)
- nU, [177](#)
- nU0, [178](#)
- nV, [178](#)
- nW, [178](#)
- printHelp, [174](#)
- sEOSFile, [178](#)
- ofWatchZoneFiles
  - Output, [100](#)
- operator=
  - eos, [44](#)
- output
  - Global, [59](#)
- parameters
  - Global, [59](#)
- parseXML
  - work\_plot::Settings, [116](#)
- performance
  - Global, [59](#)
- physEquations.cpp
  - calDelt\_CONST, [203](#)
  - calDelt\_R\_GL, [203](#)
  - calDelt\_R\_TEOS, [203](#)
  - calDelt\_RT\_GL, [204](#)
  - calDelt\_RT\_TEOS, [204](#)
  - calDelt\_RTP\_GL, [205](#)
  - calDelt\_RTP\_TEOS, [206](#)
  - calNewD\_R, [206](#)
  - calNewD\_RT, [207](#)
  - calNewD\_RTP, [207](#)
  - calNewDenave\_None, [208](#)
  - calNewDenave\_R, [208](#)
  - calNewDenave\_RT, [208](#)
  - calNewDenave\_RTP, [209](#)
  - calNewE\_R\_AD, [209](#)
  - calNewE\_R\_NA, [209](#)
  - calNewE\_R\_NA\_LES, [210](#)
  - calNewE\_RT\_AD, [211](#)
  - calNewE\_RT\_NA, [211](#)
  - calNewE\_RT\_NA\_LES, [212](#)
  - calNewE\_RTP\_AD, [213](#)
  - calNewE\_RTP\_NA, [214](#)
  - calNewE\_RTP\_NA\_LES, [214](#)
  - calNewEddyVisc\_None, [215](#)
  - calNewEddyVisc\_R\_CN, [216](#)
  - calNewEddyVisc\_R\_SM, [216](#)
  - calNewEddyVisc\_RT\_CN, [216](#)
  - calNewEddyVisc\_RT\_SM, [217](#)
  - calNewEddyVisc\_RTP\_CN, [217](#)
  - calNewEddyVisc\_RTP\_SM, [217](#)
  - calNewP\_GL, [218](#)
  - calNewPEKappaGamma\_TEOS, [218](#)
  - calNewQ0\_R\_GL, [218](#)
  - calNewQ0\_R\_TEOS, [219](#)
  - calNewQ0Q1\_RT\_GL, [219](#)
  - calNewQ0Q1\_RT\_TEOS, [220](#)
  - calNewQ0Q1Q2\_RTP\_GL, [220](#)
  - calNewQ0Q1Q2\_RTP\_TEOS, [221](#)
  - calNewR, [221](#)
  - calNewTPKappaGamma\_TEOS, [221](#)
  - calNewU0\_R, [222](#)
  - calNewU0\_RT, [222](#)
  - calNewU0\_RTP, [223](#)
  - calNewU\_R, [224](#)
  - calNewU\_R\_LES, [224](#)
  - calNewU\_RT, [225](#)
  - calNewU\_RT\_LES, [226](#)
  - calNewU\_RTP, [227](#)
  - calNewU\_RTP\_LES, [228](#)
  - calNewV\_RT, [229](#)
  - calNewV\_RT\_LES, [230](#)
  - calNewV\_RTP, [230](#)
  - calNewV\_RTP\_LES, [231](#)
  - calNewVelocities\_R, [232](#)
  - calNewVelocities\_R\_LES, [232](#)
  - calNewVelocities\_RT, [232](#)
  - calNewVelocities\_RT\_LES, [233](#)

- calNewVelocities\_RTP, [233](#)
- calNewVelocities\_RTP\_LES, [234](#)
- calNewW\_RTP, [234](#)
- calNewW\_RTP\_LES, [235](#)
- calOldDenave\_None, [235](#)
- calOldDenave\_R, [236](#)
- calOldDenave\_RT, [236](#)
- calOldDenave\_RTP, [236](#)
- calOldEddyVisc\_R\_CN, [237](#)
- calOldEddyVisc\_R\_SM, [237](#)
- calOldEddyVisc\_RT\_CN, [237](#)
- calOldEddyVisc\_RT\_SM, [238](#)
- calOldEddyVisc\_RTP\_CN, [238](#)
- calOldEddyVisc\_RTP\_SM, [238](#)
- calOldP\_GL, [239](#)
- calOldPEKappaGamma\_TEOS, [239](#)
- calOldQ0\_R\_GL, [240](#)
- calOldQ0\_R\_TEOS, [240](#)
- calOldQ0Q1\_RT\_GL, [241](#)
- calOldQ0Q1\_RT\_TEOS, [241](#)
- calOldQ0Q1Q2\_RTP\_GL, [241](#)
- calOldQ0Q1Q2\_RTP\_TEOS, [242](#)
- dEOS\_GL, [242](#)
- dET4, [243](#)
- dImplicitEnergyFunction\_None, [243](#)
- dImplicitEnergyFunction\_R, [243](#)
- dImplicitEnergyFunction\_R\_LES, [244](#)
- dImplicitEnergyFunction\_R\_LES\_SB, [245](#)
- dImplicitEnergyFunction\_R\_SB, [245](#)
- dImplicitEnergyFunction\_RT, [246](#)
- dImplicitEnergyFunction\_RT\_LES, [247](#)
- dImplicitEnergyFunction\_RT\_LES\_SB, [248](#)
- dImplicitEnergyFunction\_RT\_SB, [249](#)
- dImplicitEnergyFunction\_RTP, [250](#)
- dImplicitEnergyFunction\_RTP\_LES, [250](#)
- dImplicitEnergyFunction\_RTP\_LES\_SB, [251](#)
- dImplicitEnergyFunction\_RTP\_SB, [253](#)
- implicitSolve\_None, [254](#)
- implicitSolve\_R, [254](#)
- implicitSolve\_RT, [254](#)
- implicitSolve\_RTP, [255](#)
- initDonorFracAndMaxConVel\_R\_GL, [255](#)
- initDonorFracAndMaxConVel\_R\_TEOS, [256](#)
- initDonorFracAndMaxConVel\_RT\_GL, [256](#)
- initDonorFracAndMaxConVel\_RT\_TEOS, [256](#)
- initDonorFracAndMaxConVel\_RTP\_GL, [257](#)
- initDonorFracAndMaxConVel\_RTP\_TEOS, [257](#)
- initInternalVars, [257](#)
- setInternalVarInf, [258](#)
- setMainFunctions, [259](#)
- physEquations.h
  - calDelt\_CONST, [264](#)
  - calDelt\_R\_GL, [264](#)
  - calDelt\_R\_TEOS, [264](#)
  - calDelt\_RT\_GL, [265](#)
  - calDelt\_RT\_TEOS, [265](#)
  - calDelt\_RTP\_GL, [266](#)
  - calDelt\_RTP\_TEOS, [267](#)
  - calNewD\_R, [267](#)
  - calNewD\_RT, [268](#)
  - calNewD\_RTP, [268](#)
  - calNewDenave\_None, [269](#)
  - calNewDenave\_R, [269](#)
  - calNewDenave\_RT, [269](#)
  - calNewDenave\_RTP, [270](#)
  - calNewE\_R\_AD, [270](#)
  - calNewE\_R\_NA, [270](#)
  - calNewE\_R\_NA\_LES, [271](#)
  - calNewE\_RT\_AD, [272](#)
  - calNewE\_RT\_NA, [272](#)
  - calNewE\_RT\_NA\_LES, [273](#)
  - calNewE\_RTP\_AD, [274](#)
  - calNewE\_RTP\_NA, [275](#)
  - calNewE\_RTP\_NA\_LES, [275](#)
  - calNewEddyVisc\_None, [276](#)
  - calNewEddyVisc\_R\_CN, [277](#)
  - calNewEddyVisc\_R\_SM, [277](#)
  - calNewEddyVisc\_RT\_CN, [277](#)
  - calNewEddyVisc\_RT\_SM, [278](#)
  - calNewEddyVisc\_RTP\_CN, [278](#)
  - calNewEddyVisc\_RTP\_SM, [278](#)
  - calNewP\_GL, [279](#)
  - calNewPEKappaGamma\_TEOS, [279](#)
  - calNewQ0\_R\_GL, [279](#)

- calNewQ0\_R\_TEOS, [280](#)
- calNewQ0Q1\_RT\_GL, [280](#)
- calNewQ0Q1\_RT\_TEOS, [281](#)
- calNewQ0Q1Q2\_RTP\_GL, [281](#)
- calNewQ0Q1Q2\_RTP\_TEOS, [282](#)
- calNewR, [282](#)
- calNewTPKappaGamma\_TEOS, [282](#)
- calNewU0\_R, [283](#)
- calNewU0\_RT, [283](#)
- calNewU0\_RTP, [284](#)
- calNewU\_R, [285](#)
- calNewU\_R\_LES, [285](#)
- calNewU\_RT, [286](#)
- calNewU\_RT\_LES, [287](#)
- calNewU\_RTP, [288](#)
- calNewU\_RTP\_LES, [289](#)
- calNewV\_RT, [290](#)
- calNewV\_RT\_LES, [291](#)
- calNewV\_RTP, [291](#)
- calNewV\_RTP\_LES, [292](#)
- calNewVelocities\_R, [293](#)
- calNewVelocities\_R\_LES, [293](#)
- calNewVelocities\_RT, [293](#)
- calNewVelocities\_RT\_LES, [294](#)
- calNewVelocities\_RTP, [294](#)
- calNewVelocities\_RTP\_LES, [295](#)
- calNewW\_RTP, [295](#)
- calNewW\_RTP\_LES, [296](#)
- calOldDenave\_None, [296](#)
- calOldDenave\_R, [297](#)
- calOldDenave\_RT, [297](#)
- calOldDenave\_RTP, [297](#)
- calOldEddyVisc\_R\_CN, [298](#)
- calOldEddyVisc\_R\_SM, [298](#)
- calOldEddyVisc\_RT\_CN, [298](#)
- calOldEddyVisc\_RT\_SM, [299](#)
- calOldEddyVisc\_RTP\_CN, [299](#)
- calOldEddyVisc\_RTP\_SM, [299](#)
- calOldP\_GL, [300](#)
- calOldPEKappaGamma\_TEOS, [300](#)
- calOldQ0\_R\_GL, [301](#)
- calOldQ0\_R\_TEOS, [301](#)
- calOldQ0Q1\_RT\_GL, [302](#)
- calOldQ0Q1\_RT\_TEOS, [302](#)
- calOldQ0Q1Q2\_RTP\_GL, [302](#)
- calOldQ0Q1Q2\_RTP\_TEOS, [303](#)
- dEOS\_GL, [303](#)
- dET4, [304](#)
- dImplicitEnergyFunction\_None, [304](#)
- dImplicitEnergyFunction\_R, [304](#)
- dImplicitEnergyFunction\_R\_LES, [305](#)
- dImplicitEnergyFunction\_R\_LES\_SB, [306](#)
- dImplicitEnergyFunction\_R\_SB, [306](#)
- dImplicitEnergyFunction\_RT, [307](#)
- dImplicitEnergyFunction\_RT\_LES, [308](#)
- dImplicitEnergyFunction\_RT\_LES\_SB, [309](#)
- dImplicitEnergyFunction\_RT\_SB, [310](#)
- dImplicitEnergyFunction\_RTP, [311](#)
- dImplicitEnergyFunction\_RTP\_LES, [311](#)
- dImplicitEnergyFunction\_RTP\_LES\_SB, [312](#)
- dImplicitEnergyFunction\_RTP\_SB, [314](#)
- implicitSolve\_None, [315](#)
- implicitSolve\_R, [315](#)
- implicitSolve\_RT, [315](#)
- implicitSolve\_RTP, [316](#)
- initDonorFracAndMaxConVel\_R\_GL, [316](#)
- initDonorFracAndMaxConVel\_R\_TEOS, [317](#)
- initDonorFracAndMaxConVel\_RT\_GL, [317](#)
- initDonorFracAndMaxConVel\_RT\_TEOS, [317](#)
- initDonorFracAndMaxConVel\_RTP\_GL, [318](#)
- initDonorFracAndMaxConVel\_RTP\_TEOS, [318](#)
- initInternalVars, [318](#)
- setInternalVarInf, [319](#)
- setMainFunctions, [320](#)
- plot\_2DSlices.File2DSlice, [51](#)
- plot\_2DSlices::File2DSlice load, [51](#)
- plot\_file.Axis, [27](#)
- plot\_file.Curve, [29](#)
- plot\_file.DataSet, [32](#)
- plot\_file.Plot, [109](#)
- plot\_file.Text, [117](#)
- plot\_file::Axis \_\_init\_\_, [27](#)
- load, [28](#)

---

- plot\_file::Curve
  - \_\_init\_\_, 29
  - load, 29
- plot\_file::DataSet
  - \_\_init\_\_, 32
  - getCurve, 32
  - load, 32
- plot\_file::Plot
  - \_\_init\_\_, 109
  - load, 109
- plot\_file::Text
  - \_\_init\_\_, 117
- plot\_profile.Axis, 28
- plot\_profile.Curve, 30
- plot\_profile.DataSet, 33
- plot\_profile.Plot, 110
- plot\_profile::Axis
  - \_\_init\_\_, 28
  - load, 28
- plot\_profile::Curve
  - \_\_init\_\_, 30
  - load, 30
- plot\_profile::DataSet
  - \_\_init\_\_, 33
  - getCurve, 33
  - load, 33
- plot\_profile::Plot
  - \_\_init\_\_, 110
  - load, 110
- plotGrid
  - eos\_interp::eosTableManager, 51
- plotGrids
  - eos\_interp::opacityTableManager, 98
- plotLogE
  - eos\_interp::eosTable, 49
  - eos\_interp::interpTable, 89
- plotLogK
  - eos\_interp::interpTable, 89
  - eos\_interp::opacityTable, 96
- plotLogP
  - eos\_interp::eosTable, 49
  - eos\_interp::interpTable, 89
- printHeader
  - dump::dump, 34
- printHelp
  - nal/main.cpp, 161
  - nal/main.h, 174
- printVar
  - dump::dump, 34
- procTop
  - Global, 59
- profileData, 113
  - clear, 114
  - nMaxNumZones, 114
  - profileData, 114
  - profileData, 114
  - set, 114
  - setMax, 114
  - setMaxAbs, 115
  - setSum, 115
  - test, 115
  - toFile, 115
- read
  - dump::dump, 35
  - eos\_interp::interpTable, 90
- readAscii
  - eos, 44
- readAsciiVar
  - dump::dump, 35
- readBin
  - eos, 44
- readBinaryVar
  - dump::dump, 35
- readBobsAscii
  - eos, 44
- readBoloCorr
  - light\_curve::LightCurve, 91
- readConfig
  - en/main.cpp, 165
  - en/main.h, 184
- readFile
  - datafile::DataFile, 31
- readFileFixed
  - datafile::DataFile, 31
- readFileUnFixed
  - datafile::DataFile, 31
- readHeader
  - dump::dump, 35
- readHeaderAscii
  - dump::dump, 35
- readHeaderBinary
  - dump::dump, 36
- readProfiles
  - light\_curve::LightCurve, 92
- readUProfile
  - en/main.cpp, 166
  - en/main.h, 184
- requestRecv
  - MessPass, 93

---

- requestSend
  - MessPass, [93](#)
- sBaseOutputFileName
  - Output, [100](#)
- sDebugProfileOutput
  - Parameters, [107](#)
- sEOSFile
  - en/main.h, [196](#)
  - nal/main.h, [178](#)
- sEOSFileName
  - Parameters, [107](#)
- sOutPutfile
  - en/main.h, [196](#)
- sUDistType
  - en/main.h, [196](#)
- scripts/ Directory Reference, [23](#)
- scripts/cp\_files.py, [125](#)
- set
  - profileData, [114](#)
- setDEDMClamp
  - dataManipulation.cpp, [132](#)
  - dataManipulation.h, [142](#)
- setInternalVarInf
  - physEquations.cpp, [258](#)
  - physEquations.h, [319](#)
- setMainFunctions
  - physEquations.cpp, [259](#)
  - physEquations.h, [320](#)
- setMax
  - profileData, [114](#)
- setMaxAbs
  - profileData, [115](#)
- setSum
  - profileData, [115](#)
- setVarIDs
  - dump::dump, [36](#)
- setupLocalGrid
  - dataManipulation.cpp, [133](#)
  - dataManipulation.h, [143](#)
- signalHandler
  - main.cpp, [158](#)
  - main.h, [172](#)
- src/ Directory Reference, [25](#)
- src/SPHERLS/ Directory Reference, [24](#)
- src/SPHERLS/dataManipulation.cpp, [126](#)
- src/SPHERLS/dataManipulation.h, [136](#)
- src/SPHERLS/dataMonitoring.cpp, [146](#)
- src/SPHERLS/dataMonitoring.h, [150](#)
- src/SPHERLS/global.cpp, [154](#)
- src/SPHERLS/global.h, [154](#)
- src/SPHERLS/main.cpp, [156](#)
- src/SPHERLS/main.h, [170](#)
- src/SPHERLS/physEquations.cpp, [199](#)
- src/SPHERLS/physEquations.h, [260](#)
- src/SPHERLS/procTop.cpp, [321](#)
- src/SPHERLS/procTop.h, [321](#)
- src/SPHERLS/profileData.h, [321](#)
- src/SPHERLS/time.cpp, [322](#)
- src/SPHERLS/time.h, [322](#)
- src/SPHERLS/watchzone.cpp, [322](#)
- src/SPHERLS/watchzone.h, [322](#)
- src/SPHERLSanal/ Directory Reference, [25](#)
- src/SPHERLSanal/main.cpp, [158](#)
- src/SPHERLSanal/main.h, [172](#)
- src/SPHERLSanal/userguide.h, [323](#)
- src/SPHERLSgen/ Directory Reference, [25](#)
- src/SPHERLSgen/main.cpp, [161](#)
- src/SPHERLSgen/main.h, [178](#)
- src/SPHERLSgen/userguide.h, [323](#)
- src/eos.cpp, [125](#)
- src/eos.h, [125](#)
- src/pythonextensions/ Directory Reference, [23](#)
- src/pythonextensions/hdfwrapper/ Directory Reference, [23](#)
- statusRecv
  - MessPass, [93](#)
- statusSend
  - MessPass, [93](#)
- term, [116](#)
  - dCoeff, [117](#)
  - dPower, [117](#)
- test
  - profileData, [115](#)
- time
  - Global, [59](#)
- toFile
  - profileData, [115](#)
- typeRecvNewVar
  - MessPass, [94](#)
- typeRecvOldGrid
  - MessPass, [94](#)
- typeSendNewGrid
  - MessPass, [94](#)
- typeSendNewVar
  - MessPass, [94](#)

- updateLocalBoundaries
  - dataManipulation.cpp, [133](#)
  - dataManipulation.h, [143](#)
- updateLocalBoundariesNewGrid
  - dataManipulation.cpp, [134](#)
  - dataManipulation.h, [144](#)
- updateLocalBoundaryVelocitiesNewGrid-  
\_R
  - dataManipulation.cpp, [134](#)
  - dataManipulation.h, [144](#)
- updateLocalBoundaryVelocitiesNewGrid-  
\_RT
  - dataManipulation.cpp, [135](#)
  - dataManipulation.h, [145](#)
- updateLocalBoundaryVelocitiesNewGrid-  
\_RTP
  - dataManipulation.cpp, [135](#)
  - dataManipulation.h, [145](#)
- updateNewGridWithOld
  - dataManipulation.cpp, [135](#)
  - dataManipulation.h, [145](#)
- updateOldGrid
  - dataManipulation.cpp, [135](#)
  - dataManipulation.h, [145](#)
- vecRHS
  - Implicit, [87](#)
- vecTCorrections
  - Implicit, [87](#)
- vecTCorrectionsLocal
  - Implicit, [88](#)
- vecdE
  - en/main.h, [197](#)
- vecdKappa
  - en/main.h, [197](#)
- vecdM
  - en/main.h, [197](#)
- vecdMdel
  - en/main.h, [197](#)
- vecdP
  - en/main.h, [197](#)
- vecdR
  - en/main.h, [198](#)
- vecdRho
  - en/main.h, [198](#)
- vecdT
  - en/main.h, [198](#)
- vecscatTCorrections
  - Implicit, [87](#)
- vectVelDist
  - en/main.h, [198](#)
- watchzone, [122](#)
- watchzoneList
  - Output, [101](#)
- work\_plot.PdVPlotSettings, [108](#)
- work\_plot.Settings, [116](#)
- work\_plot.WorkPlotSettings, [122](#)
- work\_plot::Settings
  - \_\_init\_\_, [116](#)
  - parseXML, [116](#)
- write
  - eos\_interp::eosTable, [49](#)
  - light\_curve::LightCurve, [92](#)
  - make\_hdf::hdfFile, [83](#)
- writeAscii
  - eos, [45](#)
- writeBin
  - eos, [45](#)
- writeModel\_Bin\_R\_GL
  - en/main.cpp, [166](#)
  - en/main.h, [185](#)
- writeModel\_Bin\_R\_TEOS
  - en/main.cpp, [167](#)
  - en/main.h, [185](#)
- writeModel\_Bin\_RT\_GL
  - en/main.cpp, [167](#)
  - en/main.h, [185](#)
- writeModel\_Bin\_RT\_TEOS
  - en/main.cpp, [167](#)
  - en/main.h, [185](#)
- writeModel\_Bin\_RTP\_GL
  - en/main.cpp, [167](#)
  - en/main.h, [186](#)
- writeModel\_Bin\_RTP\_TEOS
  - en/main.cpp, [168](#)
  - en/main.h, [186](#)
- writeModel\_R\_GL
  - en/main.cpp, [168](#)
  - en/main.h, [186](#)
- writeModel\_R\_TEOS
  - en/main.cpp, [168](#)
  - en/main.h, [186](#)
- writeModel\_RT\_GL
  - en/main.cpp, [168](#)
  - en/main.h, [187](#)
- writeModel\_RT\_TEOS
  - en/main.cpp, [169](#)
  - en/main.h, [187](#)
- writeModel\_RTP\_GL



---

- en/main.cpp, [169](#)
- en/main.h, [187](#)
- writeModel\_RTP\_TEOS
  - en/main.cpp, [169](#)
  - en/main.h, [187](#)
- writeModelToScreen\_GL
  - en/main.cpp, [170](#)
  - en/main.h, [188](#)
- writeModelToScreen\_TEOS
  - en/main.cpp, [170](#)
  - en/main.h, [188](#)
- writeWatchZones\_R\_GL
  - dataMonitoring.cpp, [147](#)
  - dataMonitoring.h, [151](#)
- writeWatchZones\_R\_TEOS
  - dataMonitoring.cpp, [148](#)
  - dataMonitoring.h, [152](#)
- writeWatchZones\_RT\_GL
  - dataMonitoring.cpp, [148](#)
  - dataMonitoring.h, [152](#)
- writeWatchZones\_RT\_TEOS
  - dataMonitoring.cpp, [148](#)
  - dataMonitoring.h, [152](#)
- writeWatchZones\_RTP\_GL
  - dataMonitoring.cpp, [149](#)
  - dataMonitoring.h, [153](#)
- writeWatchZones\_RTP\_TEOS
  - dataMonitoring.cpp, [149](#)
  - dataMonitoring.h, [153](#)