

ccfd  
v1.0

Generated by Doxygen 1.8.18

Thu May 7 2020 10:34:50



<b>1 @image{inline} html img/ccfd.png "ccfd"</b>	<b>1</b>
1.1 Dependencies . . . . .	1
1.2 Installation . . . . .	1
1.2.1 Linux . . . . .	2
1.2.2 MacOS . . . . .	2
1.2.3 Windows . . . . .	3
1.3 Usage . . . . .	3
<b>2 Bug List</b>	<b>5</b>
<b>3 Data Structure Index</b>	<b>7</b>
3.1 Data Structures . . . . .	7
<b>4 File Index</b>	<b>9</b>
4.1 File List . . . . .	9
<b>5 Data Structure Documentation</b>	<b>11</b>
5.1 boundary_t Struct Reference . . . . .	11
5.1.1 Detailed Description . . . . .	11
5.1.2 Field Documentation . . . . .	12
5.1.2.1 BCid . . . . .	12
5.1.2.2 BCtype . . . . .	12
5.1.2.3 connection . . . . .	12
5.1.2.4 exactFunc . . . . .	13
5.1.2.5 heatFlux . . . . .	13
5.1.2.6 isAdiabatic . . . . .	13
5.1.2.7 isTemperaturePrescribed . . . . .	13
5.1.2.8 next . . . . .	13
5.1.2.9 pVar . . . . .	14
5.1.2.10 temperature . . . . .	14
5.2 cartMesh_t Struct Reference . . . . .	14
5.2.1 Detailed Description . . . . .	14
5.2.2 Field Documentation . . . . .	14
5.2.2.1 BCrange . . . . .	15
5.2.2.2 BCtype . . . . .	15
5.2.2.3 iMax . . . . .	15
5.2.2.4 jMax . . . . .	15
5.2.2.5 nBC . . . . .	15
5.3 cmd_t Struct Reference . . . . .	16
5.3.1 Detailed Description . . . . .	16
5.3.2 Field Documentation . . . . .	16
5.3.2.1 key . . . . .	16
5.3.2.2 next . . . . .	16
5.3.2.3 prev . . . . .	17

---

5.3.2.4 value	17
5.4 elem_t Struct Reference	17
5.4.1 Detailed Description	18
5.4.2 Field Documentation	18
5.4.2.1 area	18
5.4.2.2 areaq	19
5.4.2.3 bary	19
5.4.2.4 cVar	19
5.4.2.5 cVarStage	19
5.4.2.6 domain	19
5.4.2.7 dt	20
5.4.2.8 dtLoc	20
5.4.2.9 elemType	20
5.4.2.10 firstSide	20
5.4.2.11 id	20
5.4.2.12 innerSides	21
5.4.2.13 next	21
5.4.2.14 nGP	21
5.4.2.15 node	21
5.4.2.16 pVar	21
5.4.2.17 source	22
5.4.2.18 sx	22
5.4.2.19 sy	22
5.4.2.20 u_t	22
5.4.2.21 u_x	22
5.4.2.22 u_y	23
5.4.2.23 venkEps_sq	23
5.4.2.24 wGP	23
5.4.2.25 xGP	23
5.5 node_t Struct Reference	24
5.5.1 Detailed Description	24
5.5.2 Field Documentation	24
5.5.2.1 id	24
5.5.2.2 next	24
5.5.2.3 x	25
5.6 outputTime_t Struct Reference	25
5.6.1 Detailed Description	25
5.6.2 Field Documentation	25
5.6.2.1 iter	26
5.6.2.2 next	26
5.6.2.3 time	26
5.7 recordPoint_t Struct Reference	26

---

5.7.1 Detailed Description	27
5.7.2 Field Documentation	27
5.7.2.1 elem	27
5.7.2.2 ioFile	27
5.7.2.3 nPoints	27
5.7.2.4 x	28
5.8 side_t Struct Reference	28
5.8.1 Detailed Description	29
5.8.2 Field Documentation	29
5.8.2.1 baryBaryDist	29
5.8.2.2 baryBaryVec	29
5.8.2.3 BC	29
5.8.2.4 BCid	30
5.8.2.5 BCtype	30
5.8.2.6 connection	30
5.8.2.7 elem	30
5.8.2.8 flux	30
5.8.2.9 GP	31
5.8.2.10 id	31
5.8.2.11 len	31
5.8.2.12 n	31
5.8.2.13 next	31
5.8.2.14 nextElemSide	32
5.8.2.15 node	32
5.8.2.16 pVar	32
5.8.2.17 w	32
5.9 sideList_t Struct Reference	33
5.9.1 Detailed Description	33
5.9.2 Field Documentation	33
5.9.2.1 BC	33
5.9.2.2 isRotated	34
5.9.2.3 node	34
5.9.2.4 side	34
5.10 sidePtr_t Struct Reference	34
5.10.1 Detailed Description	35
5.10.2 Field Documentation	35
5.10.2.1 next	35
5.10.2.2 side	35
5.11 wing_t Struct Reference	36
5.11.1 Detailed Description	36
5.11.2 Field Documentation	36
5.11.2.1 cd	37

5.11.2.2 cl	37
5.11.2.3 firstPressureSide	37
5.11.2.4 firstSuctionSide	37
5.11.2.5 refLength	37
5.11.2.6 wallId	38
5.11.2.7 wingBC	38
<b>6 File Documentation</b>	<b>39</b>
6.1 src/analyze.c File Reference	39
6.1.1 Detailed Description	40
6.1.2 Function Documentation	40
6.1.2.1 analyze()	40
6.1.2.2 calcErrors()	41
6.1.2.3 evalRecordPoints()	41
6.1.2.4 globalResidual()	42
6.1.3 Variable Documentation	42
6.1.3.1 doCalcWing	42
6.1.3.2 hasExactSolution	42
6.1.3.3 recordPoint	42
6.1.3.4 wing	43
6.2 src/analyze.h File Reference	43
6.2.1 Detailed Description	44
6.2.2 Function Documentation	44
6.2.2.1 analyze()	44
6.2.2.2 calcErrors()	45
6.2.2.3 globalResidual()	45
6.2.3 Variable Documentation	45
6.2.3.1 doCalcWing	46
6.2.3.2 hasExactSolution	46
6.2.3.3 recordPoint	46
6.2.3.4 wing	46
6.3 src/boundary.c File Reference	47
6.3.1 Detailed Description	48
6.3.2 Function Documentation	48
6.3.2.1 boundary()	48
6.3.2.2 setBCatBarys()	48
6.3.2.3 setBCatSides()	49
6.3.3 Variable Documentation	49
6.3.3.1 firstBC	49
6.3.3.2 isPeriodic	49
6.3.3.3 nBC	50
6.4 src/boundary.h File Reference	50

6.4.1 Detailed Description . . . . .	51
6.4.2 Function Documentation . . . . .	51
6.4.2.1 boundary() . . . . .	51
6.4.2.2 setBCatBarys() . . . . .	52
6.4.2.3 setBCatSides() . . . . .	52
6.4.3 Variable Documentation . . . . .	53
6.4.3.1 firstBC . . . . .	53
6.4.3.2 isPeriodic . . . . .	53
6.4.3.3 nBC . . . . .	53
6.5 src/equation.c File Reference . . . . .	53
6.5.1 Detailed Description . . . . .	54
6.5.2 Variable Documentation . . . . .	54
6.5.2.1 cp . . . . .	54
6.5.2.2 doCalcSource . . . . .	55
6.5.2.3 gam . . . . .	55
6.5.2.4 gam1 . . . . .	55
6.5.2.5 gam1q . . . . .	55
6.5.2.6 gam2 . . . . .	55
6.5.2.7 iFlux . . . . .	56
6.5.2.8 intExactFunc . . . . .	56
6.5.2.9 mu . . . . .	56
6.5.2.10 pi . . . . .	56
6.5.2.11 Pr . . . . .	56
6.5.2.12 R . . . . .	57
6.5.2.13 sourceFunc . . . . .	57
6.5.2.14 sqrt2 . . . . .	57
6.5.2.15 sqrt3 . . . . .	57
6.5.2.16 sqrt3q . . . . .	57
6.6 src/equation.h File Reference . . . . .	58
6.6.1 Detailed Description . . . . .	59
6.6.2 Variable Documentation . . . . .	59
6.6.2.1 cp . . . . .	59
6.6.2.2 doCalcSource . . . . .	59
6.6.2.3 gam . . . . .	59
6.6.2.4 gam1 . . . . .	60
6.6.2.5 gam1q . . . . .	60
6.6.2.6 gam2 . . . . .	60
6.6.2.7 iFlux . . . . .	60
6.6.2.8 intExactFunc . . . . .	60
6.6.2.9 mu . . . . .	61
6.6.2.10 pi . . . . .	61
6.6.2.11 Pr . . . . .	61

6.6.2.12 R	61
6.6.2.13 sourceFunc	61
6.6.2.14 sqrt2	62
6.6.2.15 sqrt3	62
6.6.2.16 sqrt3q	62
6.7 src/equationOfState.c File Reference	62
6.7.1 Detailed Description	63
6.7.2 Function Documentation	63
6.7.2.1 charCons()	63
6.7.2.2 consChar()	64
6.7.2.3 consPrim()	64
6.7.2.4 primCons()	64
6.8 src/equationOfState.h File Reference	65
6.8.1 Detailed Description	66
6.8.2 Function Documentation	66
6.8.2.1 charCons()	66
6.8.2.2 consChar()	66
6.8.2.3 consPrim()	67
6.8.2.4 primCons()	67
6.9 src/exactFunction.c File Reference	67
6.9.1 Detailed Description	68
6.9.2 Function Documentation	68
6.9.2.1 exactFunc()	69
6.10 src/exactFunction.h File Reference	69
6.10.1 Detailed Description	70
6.10.2 Function Documentation	70
6.10.2.1 exactFunc()	70
6.11 src/exactRiemann.c File Reference	71
6.11.1 Detailed Description	72
6.11.2 Function Documentation	72
6.11.2.1 exactRiemann()	72
6.11.2.2 preFun()	73
6.11.3 Variable Documentation	73
6.11.3.1 G	73
6.11.3.2 nlter	73
6.11.3.3 tol	74
6.12 src/exactRiemann.h File Reference	74
6.12.1 Detailed Description	74
6.12.2 Function Documentation	74
6.12.2.1 exactRiemann()	75
6.13 src/finiteVolume.c File Reference	75
6.13.1 Detailed Description	76



6.13.2 Function Documentation . . . . .	76
6.13.2.1 fvTimeDerivative() . . . . .	76
6.13.3 Variable Documentation . . . . .	77
6.13.3.1 fluxFunction . . . . .	77
6.13.3.2 spatialOrder . . . . .	77
6.14 src/finiteVolume.h File Reference . . . . .	77
6.14.1 Detailed Description . . . . .	78
6.14.2 Function Documentation . . . . .	78
6.14.2.1 fvTimeDerivative() . . . . .	78
6.14.3 Variable Documentation . . . . .	78
6.14.3.1 fluxFunction . . . . .	78
6.14.3.2 spatialOrder . . . . .	79
6.15 src/fluxCalculation.c File Reference . . . . .	79
6.15.1 Detailed Description . . . . .	80
6.15.2 Function Documentation . . . . .	80
6.15.2.1 convectiveFlux() . . . . .	81
6.15.2.2 flux_ausmd() . . . . .	81
6.15.2.3 flux_ausmdv() . . . . .	82
6.15.2.4 flux_cen() . . . . .	83
6.15.2.5 flux_god() . . . . .	83
6.15.2.6 flux_hll() . . . . .	84
6.15.2.7 flux_hllc() . . . . .	85
6.15.2.8 flux_hlle() . . . . .	85
6.15.2.9 flux_lxf() . . . . .	86
6.15.2.10 flux_roe() . . . . .	86
6.15.2.11 flux_stw() . . . . .	87
6.15.2.12 flux_vanleer() . . . . .	88
6.15.2.13 fluxCalculation() . . . . .	88
6.16 src/fluxCalculation.h File Reference . . . . .	89
6.16.1 Detailed Description . . . . .	89
6.16.2 Function Documentation . . . . .	89
6.16.2.1 fluxCalculation() . . . . .	89
6.17 src/initialCondition.c File Reference . . . . .	90
6.17.1 Detailed Description . . . . .	91
6.17.2 Variable Documentation . . . . .	91
6.17.2.1 alpha . . . . .	91
6.17.2.2 domainID . . . . .	91
6.17.2.3 icType . . . . .	91
6.17.2.4 nDomains . . . . .	92
6.17.2.5 refState . . . . .	92
6.17.2.6 rp1Dinterface . . . . .	92
6.18 src/initialCondition.h File Reference . . . . .	92

6.18.1 Detailed Description	93
6.18.2 Variable Documentation	93
6.18.2.1 alpha	93
6.18.2.2 domainID	93
6.18.2.3 icType	93
6.18.2.4 nDomains	94
6.18.2.5 refState	94
6.18.2.6 rp1DInterface	94
6.19 src/linearSolver.c File Reference	94
6.19.1 Detailed Description	95
6.19.2 Function Documentation	96
6.19.2.1 buildMatrix()	96
6.19.2.2 calcDinv()	96
6.19.2.3 GMRES_M()	96
6.19.2.4 LUSGS()	97
6.19.2.5 matrixVector()	98
6.19.2.6 vectorDotProduct()	98
6.19.3 Variable Documentation	98
6.19.3.1 D	99
6.19.3.2 deltaXstar	99
6.19.3.3 Dinv	99
6.19.3.4 dRdU	99
6.19.3.5 eps2newton	99
6.19.3.6 eps2newton_sq	100
6.19.3.7 epsGMRES	100
6.19.3.8 gamEW	100
6.19.3.9 nGMRESiterGlobal	100
6.19.3.10 nInnerGMRES	100
6.19.3.11 nInnerNewton	101
6.19.3.12 nKdim	101
6.19.3.13 nNewtonIter	101
6.19.3.14 nNewtonIterGlobal	101
6.19.3.15 R0	101
6.19.3.16 R_XK	102
6.19.3.17 rEps0	102
6.19.3.18 srEps0	102
6.19.3.19 usePrecond	102
6.19.3.20 V	102
6.19.3.21 W	103
6.19.3.22 XK	103
6.19.3.23 Z	103
6.20 src/linearSolver.h File Reference	103

6.20.1 Detailed Description . . . . .	104
6.20.2 Function Documentation . . . . .	105
6.20.2.1 GMRES_M() . . . . .	105
6.20.2.2 vectorDotProduct() . . . . .	105
6.20.3 Variable Documentation . . . . .	106
6.20.3.1 eps2newton . . . . .	106
6.20.3.2 eps2newton_sq . . . . .	106
6.20.3.3 epsGMRES . . . . .	106
6.20.3.4 gamEW . . . . .	106
6.20.3.5 nGMRESiterGlobal . . . . .	107
6.20.3.6 nInnerGMRES . . . . .	107
6.20.3.7 nInnerNewton . . . . .	107
6.20.3.8 nKdim . . . . .	107
6.20.3.9 nNewtonIter . . . . .	107
6.20.3.10 nNewtonIterGlobal . . . . .	108
6.20.3.11 R_XK . . . . .	108
6.20.3.12 rEps0 . . . . .	108
6.20.3.13 srEps0 . . . . .	108
6.20.3.14 usePrecond . . . . .	108
6.20.3.15 XK . . . . .	109
6.21 src/main.c File Reference . . . . .	109
6.21.1 Detailed Description . . . . .	109
6.21.2 Function Documentation . . . . .	110
6.21.2.1 main() . . . . .	110
6.22 src/main.h File Reference . . . . .	110
6.22.1 Detailed Description . . . . .	112
6.22.2 Macro Definition Documentation . . . . .	112
6.22.2.1 STRLEN . . . . .	112
6.22.3 Enumeration Type Documentation . . . . .	112
6.22.3.1 boundaryConditionType . . . . .	112
6.22.3.2 cartesianMeshSides . . . . .	113
6.22.3.3 clcdResiduals . . . . .	113
6.22.3.4 conservativeVariables . . . . .	113
6.22.3.5 directions . . . . .	114
6.22.3.6 fluxFunction . . . . .	114
6.22.3.7 generalParameters . . . . .	114
6.22.3.8 ioFormat . . . . .	116
6.22.3.9 limiterFunction . . . . .	116
6.22.3.10 meshType . . . . .	116
6.22.3.11 primitiveVariables . . . . .	117
6.23 src/memTools.c File Reference . . . . .	117
6.23.1 Detailed Description . . . . .	118

6.23.2 Function Documentation	118
6.23.2.1 dyn2DcgsiArray()	118
6.23.2.2 dyn2DdblArray()	119
6.23.2.3 dyn2DintArray()	119
6.23.2.4 dyn3DdblArray()	119
6.23.2.5 dyn3DintArray()	121
6.23.2.6 dyn4DdblArray()	121
6.23.2.7 dynStringArray()	122
6.24 src/memTools.h File Reference	122
6.24.1 Detailed Description	123
6.24.2 Function Documentation	123
6.24.2.1 dyn2DcgsiArray()	123
6.24.2.2 dyn2DdblArray()	124
6.24.2.3 dyn2DintArray()	124
6.24.2.4 dyn3DdblArray()	124
6.24.2.5 dyn3DintArray()	126
6.24.2.6 dyn4DdblArray()	126
6.24.2.7 dynStringArray()	127
6.25 src/mesh.c File Reference	127
6.25.1 Detailed Description	129
6.25.2 Function Documentation	129
6.25.2.1 compare()	129
6.25.2.2 createCartMesh()	130
6.25.2.3 createElemInfo()	130
6.25.2.4 createMesh()	131
6.25.2.5 createReconstructionInfo()	131
6.25.2.6 createSideInfo()	132
6.25.2.7 readCGNS()	132
6.25.2.8 readEMC2()	132
6.25.2.9 readGmsh()	133
6.25.3 Variable Documentation	134
6.25.3.1 BCside	134
6.25.3.2 cartMesh	134
6.25.3.3 dxRef	134
6.25.3.4 elem	134
6.25.3.5 firstBCside	135
6.25.3.6 firstElem	135
6.25.3.7 firstNode	135
6.25.3.8 firstSide	135
6.25.3.9 gridFile	135
6.25.3.10 meshFormat	136
6.25.3.11 meshType	136

6.25.3.12 nBCsides . . . . .	136
6.25.3.13 nElems . . . . .	136
6.25.3.14 nInnerSides . . . . .	136
6.25.3.15 nNodes . . . . .	137
6.25.3.16 nQuads . . . . .	137
6.25.3.17 nSides . . . . .	137
6.25.3.18 nTrias . . . . .	137
6.25.3.19 parameterFile . . . . .	137
6.25.3.20 side . . . . .	138
6.25.3.21 strIniCondFile . . . . .	138
6.25.3.22 strMeshFile . . . . .	138
6.25.3.23 strMeshFormat . . . . .	138
6.25.3.24 totalArea_q . . . . .	138
6.25.3.25 xMax . . . . .	139
6.25.3.26 xMin . . . . .	139
6.25.3.27 yMax . . . . .	139
6.25.3.28 yMin . . . . .	139
6.26 src/mesh.h File Reference . . . . .	140
6.26.1 Detailed Description . . . . .	141
6.26.2 Variable Documentation . . . . .	142
6.26.2.1 BCside . . . . .	142
6.26.2.2 cartMesh . . . . .	142
6.26.2.3 dxRef . . . . .	142
6.26.2.4 elem . . . . .	142
6.26.2.5 firstBCside . . . . .	143
6.26.2.6 firstElem . . . . .	143
6.26.2.7 firstNode . . . . .	143
6.26.2.8 firstSide . . . . .	143
6.26.2.9 gridFile . . . . .	143
6.26.2.10 meshFormat . . . . .	144
6.26.2.11 meshType . . . . .	144
6.26.2.12 nBCsides . . . . .	144
6.26.2.13 nElems . . . . .	144
6.26.2.14 nInnerSides . . . . .	144
6.26.2.15 nNodes . . . . .	145
6.26.2.16 nQuads . . . . .	145
6.26.2.17 nSides . . . . .	145
6.26.2.18 nTrias . . . . .	145
6.26.2.19 parameterFile . . . . .	145
6.26.2.20 side . . . . .	146
6.26.2.21 strIniCondFile . . . . .	146
6.26.2.22 strMeshFile . . . . .	146

6.26.2.23 strMeshFormat . . . . .	146
6.26.2.24 totalArea_q . . . . .	146
6.26.2.25 xMax . . . . .	147
6.26.2.26 xMin . . . . .	147
6.26.2.27 yMax . . . . .	147
6.26.2.28 yMin . . . . .	147
6.27 src/output.c File Reference . . . . .	148
6.27.1 Detailed Description . . . . .	149
6.27.2 Function Documentation . . . . .	149
6.27.2.1 cgnsFinalizeOutput() . . . . .	149
6.27.2.2 cgnsOutput() . . . . .	149
6.27.2.3 csvOutput() . . . . .	150
6.27.2.4 curveOutput() . . . . .	150
6.27.2.5 dataOutput() . . . . .	151
6.27.3 Variable Documentation . . . . .	151
6.27.3.1 doErrorOutput . . . . .	151
6.27.3.2 IOiterInterval . . . . .	151
6.27.3.3 IOtimeInterval . . . . .	152
6.27.3.4 iVisuProg . . . . .	152
6.27.3.5 outputTimes . . . . .	152
6.27.3.6 parameterFile . . . . .	152
6.27.3.7 resFile . . . . .	152
6.27.3.8 strOutFile . . . . .	153
6.28 src/output.h File Reference . . . . .	153
6.28.1 Detailed Description . . . . .	154
6.28.2 Function Documentation . . . . .	154
6.28.2.1 dataOutput() . . . . .	154
6.28.3 Variable Documentation . . . . .	155
6.28.3.1 doErrorOutput . . . . .	155
6.28.3.2 IOiterInterval . . . . .	155
6.28.3.3 IOtimeInterval . . . . .	155
6.28.3.4 iVisuProg . . . . .	155
6.28.3.5 outputTimes . . . . .	156
6.28.3.6 parameterFile . . . . .	156
6.28.3.7 resFile . . . . .	156
6.28.3.8 strOutFile . . . . .	156
6.29 src/readInTools.c File Reference . . . . .	156
6.29.1 Detailed Description . . . . .	157
6.29.2 Function Documentation . . . . .	158
6.29.2.1 countKeys() . . . . .	158
6.29.2.2 deleteCmd() . . . . .	158
6.29.2.3 fillCmds() . . . . .	159

6.29.2.4 findCmd()	159
6.29.2.5 getBool()	160
6.29.2.6 getDbI()	160
6.29.2.7 getDbIArray()	161
6.29.2.8 getInt()	161
6.29.2.9 getIntArray()	162
6.29.2.10 getStr()	162
6.29.3 Variable Documentation	163
6.29.3.1 firstCmd	163
6.30 src/readInTools.h File Reference	163
6.30.1 Detailed Description	164
6.30.2 Function Documentation	164
6.30.2.1 countKeys()	164
6.30.2.2 fillCmds()	165
6.30.2.3 getBool()	165
6.30.2.4 getDbI()	165
6.30.2.5 getDbIArray()	166
6.30.2.6 getInt()	167
6.30.2.7 getIntArray()	167
6.30.2.8 getStr()	168
6.31 src/reconstruction.c File Reference	168
6.31.1 Detailed Description	169
6.31.2 Function Documentation	169
6.31.2.1 limiterBarthJespersen()	169
6.31.2.2 limiterVenkatakrishnan()	170
6.31.2.3 spatialReconstruction()	170
6.31.3 Variable Documentation	170
6.31.3.1 limiter	171
6.31.3.2 venk_k	171
6.32 src/reconstruction.h File Reference	171
6.32.1 Detailed Description	172
6.32.2 Function Documentation	172
6.32.2.1 spatialReconstruction()	172
6.32.3 Variable Documentation	172
6.32.3.1 limiter	172
6.32.3.2 venk_k	173
6.33 src/source.c File Reference	173
6.33.1 Detailed Description	174
6.33.2 Function Documentation	174
6.33.2.1 calcSource()	174
6.33.2.2 evalSource()	174
6.34 src/source.h File Reference	175

6.34.1 Detailed Description	175
6.34.2 Function Documentation	175
6.34.2.1 calcSource()	175
6.35 src/timeDiscretization.c File Reference	176
6.35.1 Detailed Description	177
6.35.2 Function Documentation	177
6.35.2.1 calcTimeStep()	177
6.35.2.2 explicitTimeStepEuler()	178
6.35.2.3 explicitTimeStepRK()	178
6.35.2.4 implicitTimeStep()	179
6.35.2.5 timeDisc()	179
6.35.3 Variable Documentation	179
6.35.3.1 abortResidual	180
6.35.3.2 abortVariable	180
6.35.3.3 abortVariableName	180
6.35.3.4 cdAbortResidual	180
6.35.3.5 cfl	180
6.35.3.6 clAbortResidual	181
6.35.3.7 deltaX	181
6.35.3.8 dfl	181
6.35.3.9 doAbortOnCdResidual	181
6.35.3.10 doAbortOnClResidual	181
6.35.3.11 F_X0	182
6.35.3.12 F_XK	182
6.35.3.13 inIterationNumber	182
6.35.3.14 isImplicit	182
6.35.3.15 isRestart	182
6.35.3.16 isStationary	183
6.35.3.17 isTimeStep1D	183
6.35.3.18 maxIter	183
6.35.3.19 nRKstages	183
6.35.3.20 printIter	183
6.35.3.21 printTime	184
6.35.3.22 Q	184
6.35.3.23 restartTime	184
6.35.3.24 RKcoeff	184
6.35.3.25 startTime	184
6.35.3.26 stopTime	185
6.35.3.27 t	185
6.35.3.28 timeOrder	185
6.35.3.29 timeOverall	185
6.36 src/timeDiscretization.h File Reference	186



6.36.1 Detailed Description . . . . .	187
6.36.2 Function Documentation . . . . .	187
6.36.2.1 timeDisc() . . . . .	187
6.36.3 Variable Documentation . . . . .	188
6.36.3.1 abortResidual . . . . .	188
6.36.3.2 abortVariable . . . . .	188
6.36.3.3 abortVariableName . . . . .	188
6.36.3.4 cdAbortResidual . . . . .	188
6.36.3.5 cfl . . . . .	188
6.36.3.6 clAbortResidual . . . . .	189
6.36.3.7 dfl . . . . .	189
6.36.3.8 doAbortOnCdResidual . . . . .	189
6.36.3.9 doAbortOnClResidual . . . . .	189
6.36.3.10 inIterationNumber . . . . .	189
6.36.3.11 isImplicit . . . . .	190
6.36.3.12 isRestart . . . . .	190
6.36.3.13 isStationary . . . . .	190
6.36.3.14 isTimeStep1D . . . . .	190
6.36.3.15 maxIter . . . . .	190
6.36.3.16 nRKstages . . . . .	191
6.36.3.17 printIter . . . . .	191
6.36.3.18 printTime . . . . .	191
6.36.3.19 restartTime . . . . .	191
6.36.3.20 RKcoeff . . . . .	191
6.36.3.21 startTime . . . . .	192
6.36.3.22 stopTime . . . . .	192
6.36.3.23 t . . . . .	192
6.36.3.24 timeOrder . . . . .	192
6.36.3.25 timeOverall . . . . .	192
<b>Index</b>	<b>193</b>



# Chapter 1

@image{inline} html img/ccfd.png "ccfd"

## ccfd

The `ccfd` code is a drop-in replacement for `cfdv`, a CFD code written in Fortran by the [Institute of Aerodynamics and Gas Dynamics](#) at the University of Stuttgart for a CFD programming course. This code itself is not available online, as far as I know, but it features a similar code structure to [FLEXI](#). The program uses [CGNS](#), for storing the calculation results.



## 1.1 Dependencies

- `git`
- `gcc`
- `make`
- `cmake`
- `gnuplot` (optional, for displaying calculation residuals, available [here](#))
- `gmsh` (optional, for mesh generation, available [here](#))
- `paraview` (optional, for post-processing the results, available [here](#))

## 1.2 Installation

The installation process is easiest on Linux, but possible on MacOS and Windows.

### 1.2.1 Linux

First make sure that all necessary dependencies are all installed. These can usually be obtained through your distributions package manager, on Arch based systems the following command should suffice

```
# pacman -S git base-devel cmake
```

For an Ubuntu based system the following command should be enough

```
# apt-get install git build-essential cmake libomp-dev
```

Next, navigate to the directory where you want to keep `ccfd`, clone the git repository and compile the code

```
$ cd path/to/directory
$ git clone https://github.com/hhh95/ccfd.git
$ cd ccfd
$ make
```

There should now be two new folders, `obj` and `bin`, the last one containing the `ccfd` executable.

You can test the compiled binary file, by executing the following command

```
$ make check
```

This will execute `ccfd` in the directory `check` on some small cases that test specific functions of the program.

Continue with [Usage](#).

### 1.2.2 MacOS

I only had access to MacOS High Sierra, so some things might have changed, but the general procedure should still be the same on any MacOS version. First, install a package manager that can install all the necessary software for you. I suggest [Homebrew](#). Head on over to their website and follow the installation instructions. Once you're done, install the necessary software to compile `ccfd`

```
$ brew update
$ brew upgrade
$ brew install git gcc make cmake libomp
```

Due to the fact, that on MacOS `gcc` is linked to `clang` per default, which does not work well with OpenMP, you will have to make a minor edit to the `Makefile`. Open the file with your favorite editor and find the line that defines the C compiler to be used

```
CC      = gcc
```

Replace `gcc` with the version that you have installed, most likely it is `gcc-9`. To find out which version you have, do the following

```
$ ls /usr/local/bin/gcc*
```

Next, navigate to the directory where you want to keep `ccfd`, clone the git repository and compile the code

```
$ cd path/to/directory
$ git clone https://github.com/hhh95/ccfd.git
$ cd ccfd
$ make
```

There should now be two new folders, `obj` and `bin`, the last one containing the `ccfd` executable.

You can test the compiled binary file, by executing the following command

```
$ make check
```

This will execute `ccfd` in the directory `check` on some small cases that test specific functions of the program.

Continue with [Usage](#). When installing ParaView, do not choose the Linux version, but the MacOS version.

When using `ccfd` it is very handy to have the ability to open a terminal in a folder from Finder. This is possible, but has to be activated first. Go to *System-settings->\*Keyboard\*->\*Services\** and then check the box in front of the option *New Terminal at Folder*. Now, you can right click a folder in Finder and open a terminal in that folder.

### 1.2.3 Windows

I am still working on installing it on Windows directly, but have not yet managed to do so. For now it only works with a [Linux Bash shell](#). Get the latest Ubuntu shell and complete the installation process. Next, start the Ubuntu shell and install the necessary utilities (if you have never used Linux before, \$ in front of a command means the command can be executed as a normal user and # in front of a command means, you need administrative rights; these can be obtained by typing `sudo` in front of the command and entering the password)

```
# apt update
# apt upgrade
# apt install git build-essential cmake libomp-dev
```

Now change to your desired working directory

```
$ git clone https://github.com/hhh95/ccfd.git
$ cd ccfd
$ make
```

There should now be two new folders, `obj` and `bin`, the last one containing the `ccfd` executable.

You can test the compiled binary file, by executing the following command

```
$ make check
```

This will execute `ccfd` in the directory `check` on some small cases that test specific functions of the program.

After everything is set up, continue with [Usage](#). However, when installing ParaView, do not install it in the Ubuntu shell, but rather install it normally for [Windows](#). When you want to access the files created by `ccfd` from Windows, just type the following into the Ubuntu shell

```
$ explorer.exe .
```

This will open the directory in the Windows explorer and you can easily access the file with ParaView.

If, after trying to open ParaView, you get an error that a 'VCOMP140.DLL' library is missing, follow the explainatins of this [Forum Post](#).

## 1.3 Usage

As a first step you should add the `ccfd` executable to your path. You can do so by running

```
$ source ccfdrc
```

Next, navigate to the `calc` folder. Here you will find example case files, contained in folder. First, try the Riemann problems. Navigate to the `riemann` folder with

```
$ cd riemann
```

Here you will find the SOD test case, as well as different versions of the case. Start a calculation with

```
$ ccfd sod.ini
```

and observe the output. There should be four new files. The initial condition of the case and the calculation results at  $t = 0.25$  s. They should all be `.csv` files. You can examine them with any spreadsheet program you like. Alternatively you can use ParaView, a free post-processing program, that can visualize 1D, 2D, and 3D data. For Arch-based distributions you can install it from the package manager

```
# pacman -S paraview
```

On Ubuntu, the ParaView program in the repositories does not read CGNS files correctly for some reason. You will need to download ParaView 5.8 from the [ParaView website](#). Next do the following

```
$ cd folder/where/you/downloaded/paraview
$ tar -xvf ParaView-5.8.0-MPI-Linux-Python3.7-64bit.tar.gz
# mv ParaView-5.8.0-MPI-Linux-Python3.7-64bit.tar.gz /opt
$ echo "export PATH:$PATH:/opt/ParaView-5.8.0-MPI-Linux-Python3.7-64bit/bin" » ~/.bashrc
```

Next open ParaView in the directory where you performed the calculations

```
$ cd path/to/ccfd/calc/riemann
$ paraview &
```

Now, click on *File->\*Open\** and select both sets of `.csv` files. Because the results are 1D data, you need to change from *Render View* to *Line Chart View*. In the top right of the viewing area, click on the `X` button. Now select *Line Chart View* from the list. You should now see an empty grid with an  $x$ -, and a  $y$ -axis. In the *Pipeline Browser* to the left, click on the eye icons in front of the loaded files. A plot should appear on the axis grid. It will probably show the initial state. In the top bar, click on the play button. Now, the final state should be shown. You will see the analytical, or exact, solution, as well as the numerical solution.

For more information on the theory, maybe have a look at [Wikipedia](#).

The procedure for running the other cases is the same. However, if the solution data is 2D, then you do not need to switch to *Line Chart View*. The 2D CGNS output files will usually have more than just the solution file. You can load everything at once by selecting the file that has `_Master` in its name. After loading the file, select all *Cell Arrays* in the *Pipeline Browser* and click on *Apply*. Then you can look at the different fields of the solution, by selecting them in the top bar (where it first says *Solid Color*).

Some files can only be run with the Navier-Stokes equations. In order to switch between Euler and Navier-Stokes equations, open the `Makefile` and change the `EQNSYS` parameter.

## Chapter 2

## Bug List

Global [flux\\_ausmdv](#) (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])

This function produces incorrect output, refrain from using it for the time being





## Chapter 3

# Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">boundary_t</a>	Structure that holds the information of a boundary condition . . . . .	11
<a href="#">cartMesh_t</a>	Structure holding the information for a cartesian mesh . . . . .	14
<a href="#">cmd_t</a>	A structure used to store the commands, read in from the parameter file . . . . .	16
<a href="#">elem_t</a>	Structure for a single element in the global element list . . . . .	17
<a href="#">node_t</a>	Structure for a single node in a linked list of nodes . . . . .	24
<a href="#">outputTime_t</a>	Output times linked list . . . . .	25
<a href="#">recordPoint_t</a>	Recording point structure, used to output flow field at specific points . . . . .	26
<a href="#">side_t</a>	Structure for a single side in the global side list . . . . .	28
<a href="#">sideList_t</a>	Helper structure for reading in the sides and deviding them into BC sides and non-BC sides . .	33
<a href="#">sidePtr_t</a>	Secondary side lists used for various things . . . . .	34
<a href="#">wing_t</a>	Collection of all necessary values for the calculation of CL and CD . . . . .	36



## Chapter 4

# File Index

### 4.1 File List

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

src/ <a href="#">analyze.c</a>	
Contains functions for analyzing flow results . . . . .	39
src/ <a href="#">analyze.h</a>	
Contains the structure definitions of <a href="#">wing_t</a> and <a href="#">recordPoint_t</a> . . . . .	43
src/ <a href="#">boundary.c</a>	
Contains the functions for initializing and applying boundary conditions . . . . .	47
src/ <a href="#">boundary.h</a>	
Contains the structure definition of a boundary . . . . .	50
src/ <a href="#">equation.c</a>	
Contains the function for initializing the physical constants . . . . .	53
src/ <a href="#">equation.h</a> . . . . .	58
src/ <a href="#">equationOfState.c</a>	
Contains conversion functions between the different variable types . . . . .	62
src/ <a href="#">equationOfState.h</a> . . . . .	65
src/ <a href="#">exactFunction.c</a>	
Contains the exact function evaluation function . . . . .	67
src/ <a href="#">exactFunction.h</a> . . . . .	69
src/ <a href="#">exactRiemann.c</a>	
Contains the function to calculate the exact Riemann flux . . . . .	71
src/ <a href="#">exactRiemann.h</a> . . . . .	74
src/ <a href="#">finiteVolume.c</a>	
Finite volume time derivative functions . . . . .	75
src/ <a href="#">finiteVolume.h</a> . . . . .	77
src/ <a href="#">fluxCalculation.c</a>	
Contains the flux calculation functions . . . . .	79
src/ <a href="#">fluxCalculation.h</a> . . . . .	89
src/ <a href="#">initialCondition.c</a>	
Functions involving the initialization and application of initial conditions . . . . .	90
src/ <a href="#">initialCondition.h</a> . . . . .	92
src/ <a href="#">linearSolver.c</a>	
Contains the functions for solving the linear system of equations during implicit calculations . . . . .	94
src/ <a href="#">linearSolver.h</a> . . . . .	103
src/ <a href="#">main.c</a>	
Contains the main function of <code>ccfd</code> . . . . .	109
src/ <a href="#">main.h</a>	
Contains the global constants and definitions . . . . .	110

src/ <a href="#">memTools.c</a>	
Memory management functions	117
src/ <a href="#">memTools.h</a>	122
src/ <a href="#">mesh.c</a>	
Contains all the functions for reading and creating meshes	127
src/ <a href="#">mesh.h</a>	
Contains the definitions of all structs for the mesh handling	140
src/ <a href="#">output.c</a>	
Contains all functions used for writing flow solutions	148
src/ <a href="#">output.h</a>	
Contains <a href="#">outputTime_t</a> struct definition	153
src/ <a href="#">readInTools.c</a>	
Provides functions for reading data from the <code>.ini</code> parameter file	156
src/ <a href="#">readInTools.h</a>	163
src/ <a href="#">reconstruction.c</a>	
Contains the reconstruction and limiter functions	168
src/ <a href="#">reconstruction.h</a>	171
src/ <a href="#">source.c</a>	
Contains the functions for initializing and evaluating the source term	173
src/ <a href="#">source.h</a>	175
src/ <a href="#">timeDiscretization.c</a>	
Contains the functions for performing the time stepping process	176
src/ <a href="#">timeDiscretization.h</a>	186

## Chapter 5

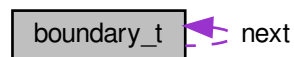
# Data Structure Documentation

### 5.1 boundary\_t Struct Reference

Structure that holds the information of a boundary condition.

```
#include <boundary.h>
```

Collaboration diagram for boundary\_t:



#### Data Fields

- int BCtype
- int BCid
- int exactFunc
- double pVar [NVAR]
- bool isAdiabatic
- bool isTemperaturePrescribed
- double temperature
- double heatFlux
- double \* connection
- boundary\_t \* next

#### 5.1.1 Detailed Description

Structure that holds the information of a boundary condition.

## 5.1.2 Field Documentation

### 5.1.2.1 BCid

```
int boundary_t::BCid
```

boundary condition ID

Referenced by `cgnsWriteMesh()`, `connectPeriodicBC()`, `createMesh()`, `initBoundary()`, and `initWing()`.

### 5.1.2.2 BCtype

```
int boundary_t::BCtype
```

boundary type:

- 1: slip wall
- 2: Navier-Stokes wall
- 3: supersonic inflow
- 4: supersonic outflow
- 5: characteristic
- 6: exact solution
- 7: periodic boundary condition
- 8: pressure outflow

Referenced by `boundary()`, `cgnsWriteMesh()`, `connectPeriodicBC()`, `createMesh()`, `initBoundary()`, and `initWing()`.

### 5.1.2.3 connection

```
double* boundary_t::connection
```

connection coordinates for periodic BC

Referenced by `connectPeriodicBC()`, and `initBoundary()`.

#### 5.1.2.4 exactFunc

```
int boundary_t::exactFunc
```

exact boundary function identifier

Referenced by `boundary()`, and `initBoundary()`.

#### 5.1.2.5 heatFlux

```
double boundary_t::heatFlux
```

wall heat flux

#### 5.1.2.6 isAdiabatic

```
bool boundary_t::isAdiabatic
```

adiabatic wall flag

Referenced by `initBoundary()`.

#### 5.1.2.7 isTemperaturePrescribed

```
bool boundary_t::isTemperaturePrescribed
```

is the temperature prescribed flag

Referenced by `initBoundary()`.

#### 5.1.2.8 next

```
boundary_t* boundary_t::next
```

pointer to next boundary condition

Referenced by `cgnsWriteMesh()`, `createMesh()`, `freeBoundary()`, `initBoundary()`, and `initWing()`.

### 5.1.2.9 pVar

```
double boundary_t::pVar[NVAR]
```

inflow state

Referenced by `boundary()`, and `initBoundary()`.

### 5.1.2.10 temperature

```
double boundary_t::temperature
```

wall temperature

Referenced by `initBoundary()`.

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

- `src/boundary.h`

## 5.2 cartMesh\_t Struct Reference

Structure holding the information for a cartesian mesh.

```
#include <mesh.h>
```

### Data Fields

- int `iMax`
- int `jMax`
- int \* `nBC`
- int `BType` [2 \* `NDIM`][`NBC`]
- int `BCrange` [2 \* `NDIM`][`NBC`][2]

### 5.2.1 Detailed Description

Structure holding the information for a cartesian mesh.

### 5.2.2 Field Documentation



### 5.2.2.1 BCrange

```
int cartMesh_t::BCrange[2 *NDIM][NBC][2]
```

list of BC ranges per side

Referenced by createCartMesh(), and readMesh().

### 5.2.2.2 BCtype

```
int cartMesh_t::BCtype[2 *NDIM][NBC]
```

list of BC types per side

Referenced by createCartMesh(), and readMesh().

### 5.2.2.3 iMax

```
int cartMesh_t::iMax
```

number of cells in x-direction

Referenced by createCartMesh(), and readMesh().

### 5.2.2.4 jMax

```
int cartMesh_t::jMax
```

number of cells in y-direction

Referenced by createCartMesh(), and readMesh().

### 5.2.2.5 nBC

```
int* cartMesh_t::nBC
```

number of different BC per side

Referenced by createCartMesh(), and readMesh().

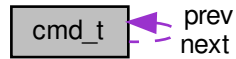
The documentation for this struct was generated from the following file:

- [src/mesh.h](#)

## 5.3 cmd\_t Struct Reference

A structure used to store the commands, read in from the parameter file.

Collaboration diagram for cmd\_t:



### Data Fields

- char [key](#) [STRLEN]
- char [value](#) [STRLEN]
- [cmd\\_t](#) \* [next](#)
- [cmd\\_t](#) \* [prev](#)

### 5.3.1 Detailed Description

A structure used to store the commands, read in from the parameter file.

### 5.3.2 Field Documentation

#### 5.3.2.1 key

```
char cmd_t::key[STRLEN]
```

the key word of the command

Referenced by `fillCmds()`, and `findCmd()`.

#### 5.3.2.2 next

```
cmd_t* cmd_t::next
```

next command

Referenced by `countKeys()`, `deleteCmd()`, `fillCmds()`, `findCmd()`, `freeCmds()`, and `ignoredCmds()`.

## 5.3.2.3 prev

```
cmd_t* cmd_t::prev
```

previous command

Referenced by deleteCmd(), fillCmds(), and freeCmds().

## 5.3.2.4 value

```
char cmd_t::value[STRLEN]
```

the vale of the command

Referenced by fillCmds(), and findCmd().

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

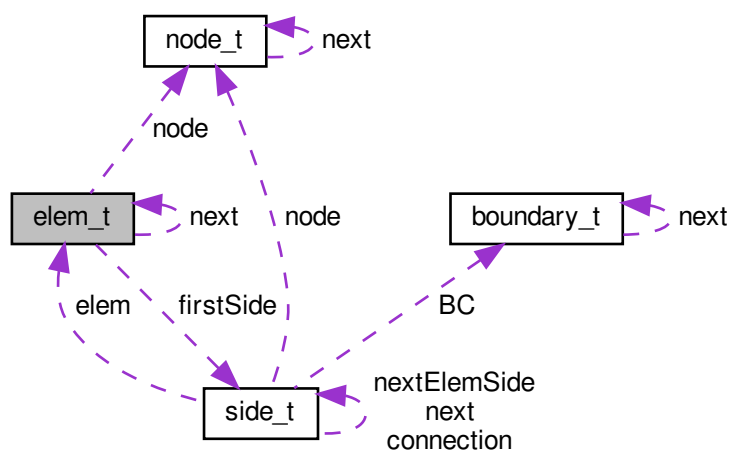
- src/readInTools.c

## 5.4 elem\_t Struct Reference

Structure for a single element in the global element list.

```
#include <mesh.h>
```

Collaboration diagram for elem\_t:



## Data Fields

- int `elemType`
- long `id`
- int `domain`
- double `bary` [NDIM]
- double `sx`
- double `sy`
- double `area`
- double `areaq`
- double `pVar` [NVAR]
- double `cVar` [NVAR]
- double `cVarStage` [NVAR]
- double `u_x` [NVAR]
- double `u_y` [NVAR]
- double `u_t` [NVAR]
- double `source` [NVAR]
- double `dt`
- double `dtLoc`
- double `venkEps_sq`
- int `innerSides`
- int `nGP`
- double \*\* `xGP`
- double \* `wGP`
- `side_t` \* `firstSide`
- `elem_t` \* `next`
- `node_t` \*\* `node`

### 5.4.1 Detailed Description

Structure for a single element in the global element list.

### 5.4.2 Field Documentation

#### 5.4.2.1 `area`

```
double elem_t::area
```

area of the element

Referenced by `calcTimeStep()`, `createElemInfo()`, `createReconstructionInfo()`, and `globalResidual()`.

#### 5.4.2.2 areaq

```
double elem_t::areaq
```

inverse of element area

Referenced by `createElemInfo()`, and `fvTimeDerivative()`.

#### 5.4.2.3 bary

```
double elem_t::bary[NDIM]
```

coordinates of element barycenter

Referenced by `calcCoef()`, `calcErrors()`, `cgnsOutput()`, `connectPeriodicBC()`, `createElemInfo()`, `createMesh()`, `createReconstructionInfo()`, `createSideInfo()`, `csvOutput()`, `curveOutput()`, `initWing()`, `setBCatBarys()`, `setBCatSides()`, and `setInitialCondition()`.

#### 5.4.2.4 cVar

```
double elem_t::cVar[NVAR]
```

conservative variables of element

Referenced by `explicitTimeStepEuler()`, `explicitTimeStepRK()`, `implicitTimeStep()`, `matrixVector()`, and `setInitialCondition()`.

#### 5.4.2.5 cVarStage

```
double elem_t::cVarStage[NVAR]
```

conservative variables at initial Runge-Kutta stage

Referenced by `explicitTimeStepRK()`.

#### 5.4.2.6 domain

```
int elem_t::domain
```

flow domain number

Referenced by `createMesh()`, and `setInitialCondition()`.

#### 5.4.2.7 dt

```
double elem_t::dt
```

element time step

Referenced by analyze(), calcTimeStep(), evalRecordPoints(), and fvTimeDerivative().

#### 5.4.2.8 dtLoc

```
double elem_t::dtLoc
```

local element time step

Referenced by fvTimeDerivative().

#### 5.4.2.9 elemType

```
int elem_t::elemType
```

element type: triangle (3) or quadrangle (4)

Referenced by cgnsWriteMesh(), createElemInfo(), createMesh(), createReconstructionInfo(), and initRecord↵  
Points().

#### 5.4.2.10 firstSide

```
side_t* elem_t::firstSide
```

pointer to the first side of the element

Referenced by buildMatrix(), createMesh(), createReconstructionInfo(), freeMesh(), fvTimeDerivative(), limiter↵  
BarthJespersen(), limiterVenkatakrishnan(), LUSGS(), and spatialReconstruction().

#### 5.4.2.11 id

```
long elem_t::id
```

unique element Id

Referenced by buildMatrix(), cgnsOutput(), cgnsReadSolution(), createMesh(), LUSGS(), and setInitialCondition().

#### 5.4.2.12 innerSides

```
int elem_t::innerSides
```

number of non-BC sides of element

#### 5.4.2.13 next

```
elem_t* elem_t::next
```

pointer to the next element in global element list

Referenced by `cgnsOutput()`, `cgnsReadSolution()`, `cgnsWriteMesh()`, `createMesh()`, `csvOutput()`, `curveOutput()`, `initRecordPoints()`, and `setInitialCondition()`.

#### 5.4.2.14 nGP

```
int elem_t::nGP
```

number of Gaussian integration points

Referenced by `calcErrors()`, `calcSource()`, and `createReconstructionInfo()`.

#### 5.4.2.15 node

```
node_t** elem_t::node
```

pointer array of the element's nodes

Referenced by `cgnsWriteMesh()`, `createElemInfo()`, `createMesh()`, `createReconstructionInfo()`, `freeMesh()`, and `initRecordPoints()`.

#### 5.4.2.16 pVar

```
double elem_t::pVar[NVAR]
```

primitive variables of element

Referenced by `buildMatrix()`, `calcErrors()`, `calcTimeStep()`, `cgnsOutput()`, `cgnsReadSolution()`, `csvOutput()`, `curveOutput()`, `evalRecordPoints()`, `explicitTimeStepEuler()`, `explicitTimeStepRK()`, `fluxCalculation()`, `implicitTimeStep()`, `limiterBarthJespersen()`, `limiterVenkatakrishnan()`, `matrixVector()`, `setBCatBarys()`, `setInitialCondition()`, and `spatialReconstruction()`.

#### 5.4.2.17 source

```
double elem_t::source[NVAR]
```

source term

Referenced by `calcSource()`, `fvTimeDerivative()`, and `initFV()`.

#### 5.4.2.18 sx

```
double elem_t::sx
```

cell extension in x-direction

Referenced by `calcTimeStep()`, and `createElemInfo()`.

#### 5.4.2.19 sy

```
double elem_t::sy
```

cell extension in y-direction

Referenced by `calcTimeStep()`, and `createElemInfo()`.

#### 5.4.2.20 u\_t

```
double elem_t::u_t[NVAR]
```

t-gradient of primitive variables

Referenced by `buildMatrix()`, `explicitTimeStepEuler()`, `explicitTimeStepRK()`, `fvTimeDerivative()`, `globalResidual()`, `implicitTimeStep()`, `matrixVector()`, and `spatialReconstruction()`.

#### 5.4.2.21 u\_x

```
double elem_t::u_x[NVAR]
```

x-gradient of primitive variables

Referenced by `calcErrors()`, `fluxCalculation()`, `limiterBarthJespersen()`, `limiterVenkatakrishnan()`, and `spatialReconstruction()`.



#### 5.4.2.22 u\_y

```
double elem_t::u_y[NVAR]
```

y-gradient of primitive variables

Referenced by `calcErrors()`, `fluxCalculation()`, `limiterBarthJespersen()`, `limiterVenkatakrishnan()`, and `spatialReconstruction()`.

#### 5.4.2.23 venkEps\_sq

```
double elem_t::venkEps_sq
```

Venkatakrishnan limiter constant for element

Referenced by `initFV()`, and `limiterVenkatakrishnan()`.

#### 5.4.2.24 wGP

```
double* elem_t::wGP
```

Gaussian weights

Referenced by `calcErrors()`, `calcSource()`, `createReconstructionInfo()`, and `freeMesh()`.

#### 5.4.2.25 xGP

```
double** elem_t::xGP
```

Gaussian points for volume integral

Referenced by `calcErrors()`, `calcSource()`, `createReconstructionInfo()`, and `freeMesh()`.

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

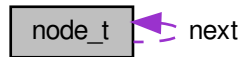
- `src/mesh.h`

## 5.5 node\_t Struct Reference

Structure for a single node in a linked list of nodes.

```
#include <mesh.h>
```

Collaboration diagram for node\_t:



### Data Fields

- long `id`
- double `x` [`NDIM`]
- `node_t` \* `next`

#### 5.5.1 Detailed Description

Structure for a single node in a linked list of nodes.

#### 5.5.2 Field Documentation

##### 5.5.2.1 id

```
long node_t::id
```

unique node ID

Referenced by `cgnsWriteMesh()`, and `createMesh()`.

##### 5.5.2.2 next

```
node_t* node_t::next
```

next node in the list

Referenced by `cgnsWriteMesh()`, `createMesh()`, and `freeMesh()`.

### 5.5.2.3 x

```
double node_t::x[NDIM]
```

coordinates of the node

Referenced by `cgnsWriteMesh()`, `createElemInfo()`, `createMesh()`, `createReconstructionInfo()`, `createSideInfo()`, and `initRecordPoints()`.

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

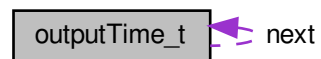
- `src/mesh.h`

## 5.6 outputTime\_t Struct Reference

Output times linked list.

```
#include <output.h>
```

Collaboration diagram for `outputTime_t`:



### Data Fields

- long `iter`
- double `time`
- `outputTime_t` \* `next`

### 5.6.1 Detailed Description

Output times linked list.

### 5.6.2 Field Documentation

### 5.6.2.1 iter

```
long outputTime_t::iter
```

iteration number at output

Referenced by `cgnsFinalizeOutput()`, and `dataOutput()`.

### 5.6.2.2 next

```
outputTime_t* outputTime_t::next
```

pointer to next output time

Referenced by `cgnsFinalizeOutput()`, `dataOutput()`, and `freeOutputTimes()`.

### 5.6.2.3 time

```
double outputTime_t::time
```

computational time at output

Referenced by `cgnsFinalizeOutput()`, and `dataOutput()`.

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

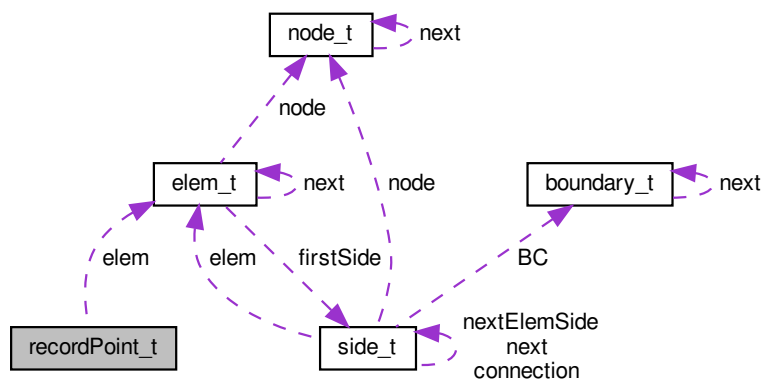
- [src/output.h](#)

## 5.7 recordPoint\_t Struct Reference

Recording point structure, used to output flow field at specific points.

```
#include <analyze.h>
```

Collaboration diagram for `recordPoint_t`:



## Data Fields

- int [nPoints](#)
- double \*\* [x](#)
- [elem\\_t](#) \*\* [elem](#)
- FILE \*\* [ioFile](#)

### 5.7.1 Detailed Description

Recording point structure, used to output flow field at specific points.

### 5.7.2 Field Documentation

#### 5.7.2.1 elem

```
elem\_t** recordPoint_t::elem
```

array of elements that contain a RP

Referenced by [evalRecordPoints\(\)](#), and [initRecordPoints\(\)](#).

#### 5.7.2.2 ioFile

```
FILE** recordPoint_t::ioFile
```

array of output file pointers

Referenced by [evalRecordPoints\(\)](#), [initRecordPoints\(\)](#), and [timeDisc\(\)](#).

#### 5.7.2.3 nPoints

```
int recordPoint_t::nPoints
```

number of recording points

Referenced by [analyze\(\)](#), [evalRecordPoints\(\)](#), [initAnalyze\(\)](#), [initRecordPoints\(\)](#), and [timeDisc\(\)](#).

#### 5.7.2.4 x

```
double** recordPoint_t::x
```

nPointsxNDIM array of RP coordinates

Referenced by initRecordPoints().

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

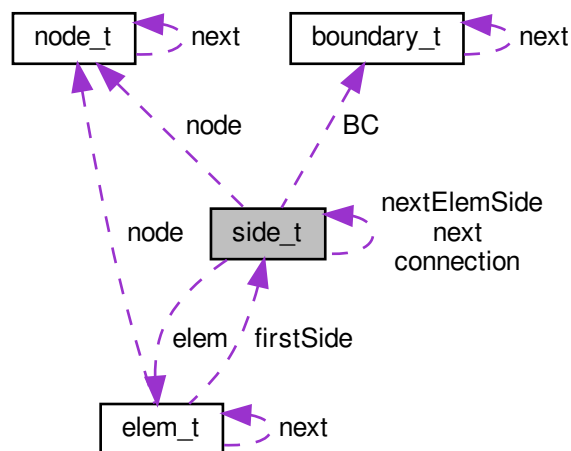
- src/[analyze.h](#)

## 5.8 side\_t Struct Reference

Structure for a single side in the global side list.

```
#include <mesh.h>
```

Collaboration diagram for side\_t:



### Data Fields

- long `id`
- int `BCtype`
- int `BCid`
- `boundary_t *` `BC`
- double `pVar` [`NVAR`]
- double `n` [`NDIM`]
- double `len`
- double `baryBaryVec` [`NDIM`]

- double [baryBaryDist](#)
- double [GP](#) [[NDIM](#)]
- double [w](#) [[NDIM](#)]
- double [flux](#) [[NVAR](#)]
- [side\\_t](#) \* [connection](#)
- [side\\_t](#) \* [nextElemSide](#)
- [side\\_t](#) \* [next](#)
- [node\\_t](#) \* [node](#) [2]
- [elem\\_t](#) \* [elem](#)

### 5.8.1 Detailed Description

Structure for a single side in the global side list.

### 5.8.2 Field Documentation

#### 5.8.2.1 baryBaryDist

```
double side_t::baryBaryDist
```

length of [baryBaryVec](#)

Referenced by [createReconstructionInfo\(\)](#), and [fluxCalculation\(\)](#).

#### 5.8.2.2 baryBaryVec

```
double side_t::baryBaryVec[NDIM]
```

vector from element barycenter to barycenter of neighbor element

Referenced by [createReconstructionInfo\(\)](#), and [fluxCalculation\(\)](#).

#### 5.8.2.3 BC

```
boundary\_t* side_t::BC
```

pointer to the boundary condition

Referenced by [boundary\(\)](#), [cgnsWriteMesh\(\)](#), [connectPeriodicBC\(\)](#), [createMesh\(\)](#), [createSideInfo\(\)](#), and [initWing\(\)](#).

#### 5.8.2.4 BCid

```
int side_t::BCid
```

boundary condition Sub-ID

#### 5.8.2.5 BCtype

```
int side_t::BCtype
```

boundary condition type

#### 5.8.2.6 connection

```
side_t* side_t::connection
```

neighbor side

Referenced by `boundary()`, `buildMatrix()`, `connectPeriodicBC()`, `createMesh()`, `createReconstructionInfo()`, `createSideInfo()`, `fluxCalculation()`, `initWing()`, `limiterBarthJespersen()`, `limiterVenkatakrishnan()`, `LUSGS()`, `setBCatBarys()`, `setBCatSides()`, and `spatialReconstruction()`.

#### 5.8.2.7 elem

```
elem_t* side_t::elem
```

pointer to the element of the side

Referenced by `buildMatrix()`, `calcCoef()`, `connectPeriodicBC()`, `createMesh()`, `createSideInfo()`, `fluxCalculation()`, `freeMesh()`, `initWing()`, `limiterBarthJespersen()`, `limiterVenkatakrishnan()`, `LUSGS()`, `setBCatBarys()`, `setBCatSides()`, and `spatialReconstruction()`.

#### 5.8.2.8 flux

```
double side_t::flux[NVAR]
```

numerical flux of the side

Referenced by `fluxCalculation()`, and `fvTimeDerivative()`.



### 5.8.2.9 GP

```
double side_t::GP[NDIM]
```

vector from element barycenter to the Gaussian point of the side

Referenced by `calcCoef()`, `connectPeriodicBC()`, `createMesh()`, `createReconstructionInfo()`, `createSideInfo()`, `initWing()`, `limiterBarthJespersen()`, `limiterVenkatakrishnan()`, `setBCatSides()`, and `spatialReconstruction()`.

### 5.8.2.10 id

```
long side_t::id
```

unique side ID

Referenced by `createMesh()`.

### 5.8.2.11 len

```
double side_t::len
```

length of the side

Referenced by `calcCoef()`, `createSideInfo()`, and `fluxCalculation()`.

### 5.8.2.12 n

```
double side_t::n[NDIM]
```

normal vector of side

Referenced by `boundary()`, `calcCoef()`, `createMesh()`, `createSideInfo()`, `fluxCalculation()`, and `initWing()`.

### 5.8.2.13 next

```
side_t* side_t::next
```

point to the next side in the global side list

Referenced by `connectPeriodicBC()`, and `createMesh()`.

#### 5.8.2.14 nextElemSide

```
side_t* side_t::nextElemSide
```

pointer to the next side of the element

Referenced by buildMatrix(), createMesh(), createReconstructionInfo(), freeMesh(), fvTimeDerivative(), limiter← BarthJespersen(), limiterVenkatakrishnan(), LUSGS(), and spatialReconstruction().

#### 5.8.2.15 node

```
node_t* side_t::node[2]
```

pointer array to the nodes of the side

Referenced by cgnsWriteMesh(), createMesh(), and createSideInfo().

#### 5.8.2.16 pVar

```
double side_t::pVar[NVAR]
```

primitive variables state at side

Referenced by calcCoef(), fluxCalculation(), setBCatSides(), and spatialReconstruction().

#### 5.8.2.17 w

```
double side_t::w[NDIM]
```

omegaX and omegaY entries for 2nd order gradient reconstruction

Referenced by createReconstructionInfo(), and spatialReconstruction().

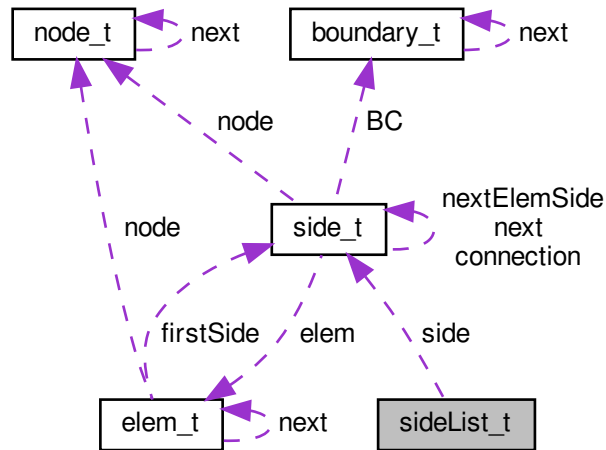
The documentation for this struct was generated from the following file:

- src/mesh.h

## 5.9 sideList\_t Struct Reference

Helper structure for reading in the sides and deviding them into BC sides and non-BC sides.

Collaboration diagram for sideList\_t:



### Data Fields

- long `node` [2]
- bool `BC`
- `side_t` \* `side`
- bool `isRotated`

### 5.9.1 Detailed Description

Helper structure for reading in the sides and deviding them into BC sides and non-BC sides.

### 5.9.2 Field Documentation

#### 5.9.2.1 BC

```
bool sideList_t::BC
```

flag for if the side is a BC side

Referenced by `compare()`, and `createMesh()`.

### 5.9.2.2 isRotated

```
bool sideList_t::isRotated
```

flag for if the side is rotated

Referenced by createMesh().

### 5.9.2.3 node

```
long sideList_t::node[2]
```

node ID array of the side

Referenced by compare(), and createMesh().

### 5.9.2.4 side

```
side_t* sideList_t::side
```

pointer to the side

Referenced by createMesh().

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

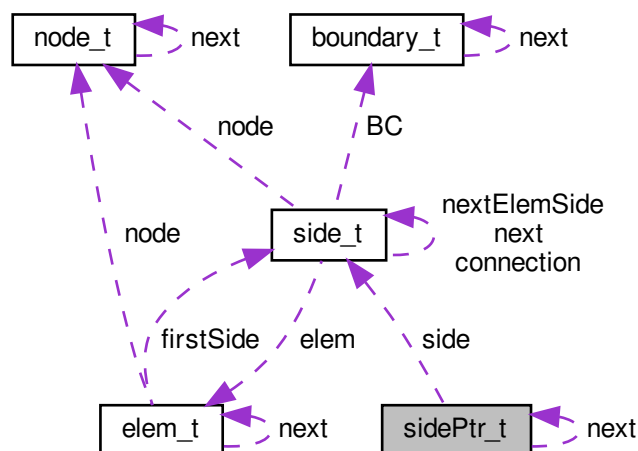
- src/mesh.c

## 5.10 sidePtr\_t Struct Reference

Secondary side lists used for various things.

```
#include <mesh.h>
```

Collaboration diagram for sidePtr\_t:



## Data Fields

- [side\\_t](#) \* [side](#)
- [sidePtr\\_t](#) \* [next](#)

### 5.10.1 Detailed Description

Secondary side lists used for various things.

### 5.10.2 Field Documentation

#### 5.10.2.1 next

```
sidePtr_t* sidePtr_t::next
```

pointer to the next side in the secondary list

Referenced by [calcCoef\(\)](#), [cgnsWriteMesh\(\)](#), [connectPeriodicBC\(\)](#), [createMesh\(\)](#), [freeAnalyze\(\)](#), [freeMesh\(\)](#), and [initWing\(\)](#).

#### 5.10.2.2 side

```
side_t* sidePtr_t::side
```

pointer to a side

Referenced by [calcCoef\(\)](#), [cgnsWriteMesh\(\)](#), [connectPeriodicBC\(\)](#), [createMesh\(\)](#), [freeMesh\(\)](#), and [initWing\(\)](#).

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

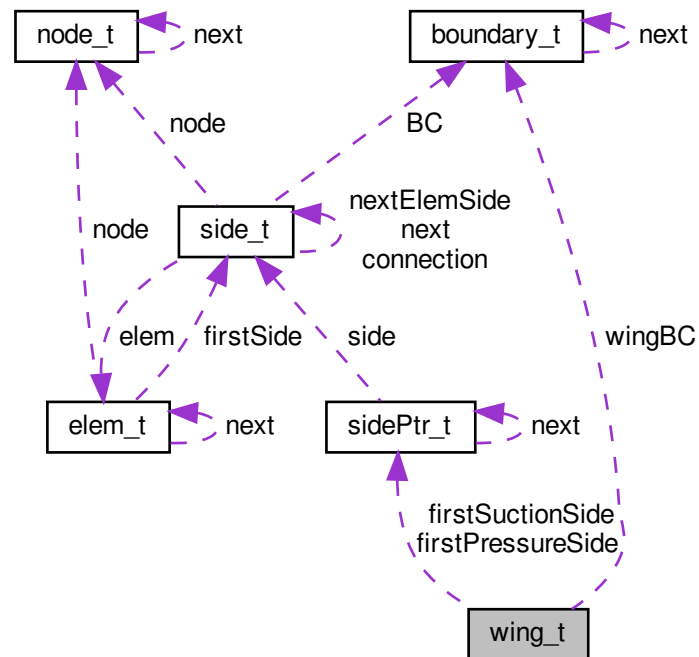
- [src/mesh.h](#)

## 5.11 wing\_t Struct Reference

Collection of all necessary values for the calculation of CL and CD.

```
#include <analyze.h>
```

Collaboration diagram for wing\_t:



### Data Fields

- double `refLength`
- int `wallId`
- double `cl`
- double `cd`
- `boundary_t *` `wingBC`
- `sidePtr_t *` `firstPressureSide`
- `sidePtr_t *` `firstSuctionSide`

#### 5.11.1 Detailed Description

Collection of all necessary values for the calculation of CL and CD.

#### 5.11.2 Field Documentation

### 5.11.2.1 cd

```
double wing_t::cd
```

drag coefficient

Referenced by analyze(), calcCoef(), and timeDisc().

### 5.11.2.2 cl

```
double wing_t::cl
```

lift coefficient

Referenced by analyze(), calcCoef(), and timeDisc().

### 5.11.2.3 firstPressureSide

```
sidePtr_t* wing_t::firstPressureSide
```

pointer to the first pressure side

Referenced by calcCoef(), freeAnalyze(), and initWing().

### 5.11.2.4 firstSuctionSide

```
sidePtr_t* wing_t::firstSuctionSide
```

pointer to the first suction side

Referenced by calcCoef(), freeAnalyze(), and initWing().

### 5.11.2.5 refLength

```
double wing_t::refLength
```

reference length

Referenced by calcCoef(), and readWing().

#### 5.11.2.6 wallId

```
int wing_t::wallId
```

BCid of the wall that represents the wing

Referenced by `initWing()`, and `readWing()`.

#### 5.11.2.7 wingBC

```
boundary_t* wing_t::wingBC
```

pointer to the BC of the wing

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

- [src/analyze.h](#)



## Chapter 6

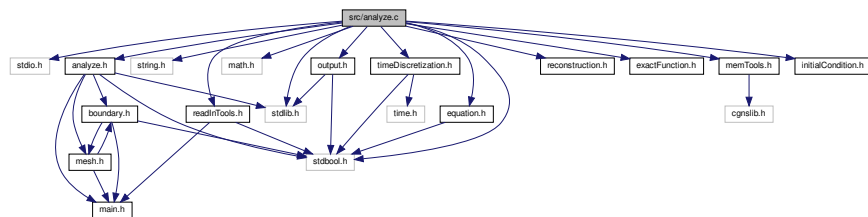
# File Documentation

### 6.1 src/analyze.c File Reference

Contains functions for analyzing flow results.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include "analyze.h"
#include "readInTools.h"
#include "timeDiscretization.h"
#include "output.h"
#include "memTools.h"
#include "reconstruction.h"
#include "exactFunction.h"
#include "equation.h"
#include "initialCondition.h"
```

Include dependency graph for analyze.c:



### Functions

- void `initRecordPoints` (void)  
*Initialize recording points.*
- void `initWing` (void)  
*Initialize required data for calculation of CL and CD.*
- void `readWing` (void)

- *Get parameters for calculating aerodynamic coefficients.*
- void `initAnalyze` (void)
  - Initialize the analysis function.*
- void `calcCoef` (void)
  - Calculate CL and CD around the specified wall.*
- void `evalRecordPoints` (double time)
  - Evaluation of recording points.*
- void `analyze` (double time, long iter, double resIter[NVAR+2])
  - Compute aerodynamic coefficients and extract values at record points.*
- void `calcErrors` (double time)
  - Calculate L1, L2, and Linf error norms, between the flow solution and the exact solution.*
- void `globalResidual` (double resIter[NVAR+2])
  - Calculate the global residual of the conservative variables.*
- void `freeAnalyze` (void)
  - Free all memory that was allocated for the wing evaluation.*

## Variables

- bool `doCalcWing`
- `wing_t` wing
- `recordPoint_t` recordPoint
- bool `hasExactSolution`

### 6.1.1 Detailed Description

Contains functions for analyzing flow results.

#### Author

hhh

#### Date

Sun 29 Mar 2020 06:29:36 PM CEST

### 6.1.2 Function Documentation

#### 6.1.2.1 analyze()

```
void analyze (
    double time,
    long iter,
    double resIter[NVAR+2] )
```

Compute aerodynamic coefficients and extract values at record points.

## Parameters

in	<i>time</i>	Calculation time at output
in	<i>iter</i>	Iteration count at output
in, out	<i>resIter</i>	The residual vector containing the CL and CD residuals at 4th and 5th index position

References abortVariable, calcCoef(), wing\_t::cd, wing\_t::cl, doCalcWing, elem\_t::dt, E, evalRecordPoints(), firstElem, isStationary, recordPoint\_t::nPoints, recordPoint, resFile, RHO, VX, VY, and wing.

Referenced by timeDisc().

## 6.1.2.2 calcErrors()

```
void calcErrors (
    double time )
```

Calculate L1, L2, and Linf error norms, between the flow solution and the exact solution.

## Parameters

in	<i>time</i>	Calculation time at output
----	-------------	----------------------------

References elem\_t::bary, elem, exactFunc(), intExactFunc, NDIM, nElems, elem\_t::nGP, NVAR, P, elem\_t::pVar, RHO, spatialReconstruction(), totalArea\_q, elem\_t::u\_x, elem\_t::u\_y, VX, VY, elem\_t::wGP, X, elem\_t::xGP, and Y.

Referenced by timeDisc().

## 6.1.2.3 evalRecordPoints()

```
void evalRecordPoints (
    double time )
```

Evaluation of recording points.

## Parameters

in	<i>time</i>	Calculation time at output
----	-------------	----------------------------

References elem\_t::dt, recordPoint\_t::elem, recordPoint\_t::ioFile, recordPoint\_t::nPoints, P, elem\_t::pVar, recordPoint, RHO, VX, and VY.

Referenced by analyze().

#### 6.1.2.4 globalResidual()

```
void globalResidual (
    double resIter[NVAR+2] )
```

Calculate the global residual of the conservative variables.

##### Parameters

<code>in, out</code>	<code>resIter</code>	The residual vector containing the residuals of the conservative variables at the first four positions
----------------------	----------------------	--

References `elem_t::area`, `E`, `elem`, `MX`, `MY`, `nElems`, `NVAR`, `RHO`, `totalArea_q`, and `elem_t::u_t`.

Referenced by `explicitTimeStepEuler()`, `explicitTimeStepRK()`, and `implicitTimeStep()`.

### 6.1.3 Variable Documentation

#### 6.1.3.1 doCalcWing

```
bool doCalcWing
```

calculate CL CD flag

Referenced by `analyze()`, `freeAnalyze()`, `initAnalyze()`, and `timeDisc()`.

#### 6.1.3.2 hasExactSolution

```
bool hasExactSolution
```

exact solution existence flag

Referenced by `dataOutput()`, `initAnalyze()`, and `timeDisc()`.

#### 6.1.3.3 recordPoint

```
recordPoint_t recordPoint
```

record flow field at a specific point

Referenced by `analyze()`, `evalRecordPoints()`, `initAnalyze()`, `initRecordPoints()`, and `timeDisc()`.

### 6.1.3.4 wing

`wing_t` wing

holds data for coefficient calculation

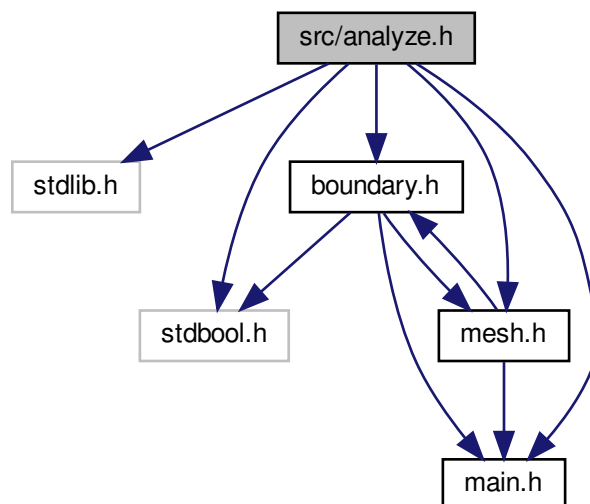
Referenced by `analyze()`, `calcCoef()`, `freeAnalyze()`, `initWing()`, `readWing()`, and `timeDisc()`.

## 6.2 src/analyze.h File Reference

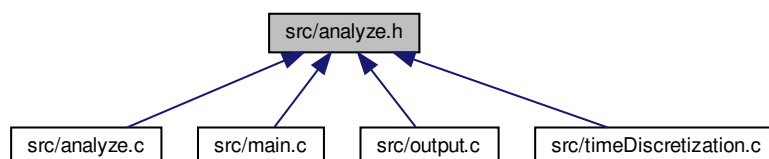
Contains the structure definitions of `wing_t` and `recordPoint_t`

```
#include <stdlib.h>
#include <stdbool.h>
#include "main.h"
#include "boundary.h"
#include "mesh.h"
```

Include dependency graph for `analyze.h`:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [wing\\_t](#)  
*Collection of all necessary values for the calculation of CL and CD.*
- struct [recordPoint\\_t](#)  
*Recording point structure, used to output flow field at specific points.*

## Functions

- void [initAnalyze](#) (void)  
*Initialize the analysis function.*
- void [analyze](#) (double time, long iter, double resIter[NVAR+2])  
*Compute aerodynamic coefficients and extract values at record points.*
- void [calcErrors](#) (double time)  
*Calculate L1, L2, and Linf error norms, between the flow solution and the exact solution.*
- void [globalResidual](#) (double resIter[NVAR+2])  
*Calculate the global residual of the conservative variables.*
- void [freeAnalyze](#) (void)  
*Free all memory that was allocated for the wing evaluation.*

## Variables

- bool [doCalcWing](#)
- [wing\\_t](#) wing
- [recordPoint\\_t](#) recordPoint
- bool [hasExactSolution](#)

### 6.2.1 Detailed Description

Contains the structure definitions of [wing\\_t](#) and [recordPoint\\_t](#)

#### Author

hhh

#### Date

Sun 29 Mar 2020 05:51:23 PM CEST

### 6.2.2 Function Documentation

#### 6.2.2.1 analyze()

```
void analyze (  
    double time,  
    long iter,  
    double resIter[NVAR+2] )
```

Compute aerodynamic coefficients and extract values at record points.

## Parameters

in	<i>time</i>	Calculation time at output
in	<i>iter</i>	Iteration count at output
in, out	<i>resIter</i>	The residual vector containing the CL and CD residuals at 4th and 5th index position

References abortVariable, calcCoef(), wing\_t::cd, wing\_t::cl, doCalcWing, elem\_t::dt, E, evalRecordPoints(), firstElem, isStationary, recordPoint\_t::nPoints, recordPoint, resFile, RHO, VX, VY, and wing.

Referenced by timeDisc().

## 6.2.2.2 calcErrors()

```
void calcErrors (
    double time )
```

Calculate L1, L2, and Linf error norms, between the flow solution and the exact solution.

## Parameters

in	<i>time</i>	Calculation time at output
----	-------------	----------------------------

References elem\_t::bary, elem, exactFunc(), intExactFunc, NDIM, nElems, elem\_t::nGP, NVAR, P, elem\_t::pVar, RHO, spatialReconstruction(), totalArea\_q, elem\_t::u\_x, elem\_t::u\_y, VX, VY, elem\_t::wGP, X, elem\_t::xGP, and Y.

Referenced by timeDisc().

## 6.2.2.3 globalResidual()

```
void globalResidual (
    double resIter[NVAR+2] )
```

Calculate the global residual of the conservative variables.

## Parameters

in, out	<i>resIter</i>	The residual vector containing the residuals of the conservative variables at the first four positions
---------	----------------	--

References elem\_t::area, E, elem, MX, MY, nElems, NVAR, RHO, totalArea\_q, and elem\_t::u\_t.

Referenced by explicitTimeStepEuler(), explicitTimeStepRK(), and implicitTimeStep().

## 6.2.3 Variable Documentation

### 6.2.3.1 doCalcWing

```
bool doCalcWing
```

calculate CL CD flag

Referenced by analyze(), freeAnalyze(), initAnalyze(), and timeDisc().

### 6.2.3.2 hasExactSolution

```
bool hasExactSolution
```

exact solution existence flag

Referenced by dataOutput(), initAnalyze(), and timeDisc().

### 6.2.3.3 recordPoint

```
recordPoint_t recordPoint
```

record flow field at a specific point

Referenced by analyze(), evalRecordPoints(), initAnalyze(), initRecordPoints(), and timeDisc().

### 6.2.3.4 wing

```
wing_t wing
```

holds data for coefficient calculation

Referenced by analyze(), calcCoef(), freeAnalyze(), initWing(), readWing(), and timeDisc().

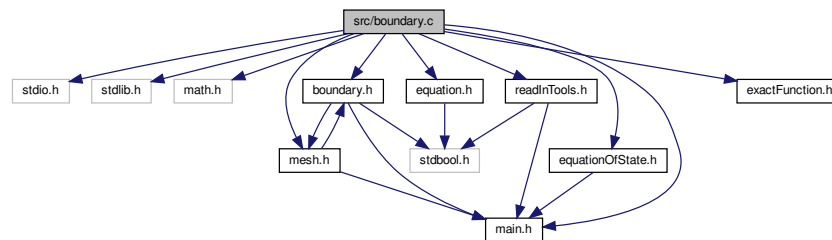


## 6.3 src/boundary.c File Reference

Contains the functions for initializing and applying boundary conditions.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "main.h"
#include "boundary.h"
#include "readInTools.h"
#include "equation.h"
#include "mesh.h"
#include "equationOfState.h"
#include "exactFunction.h"
```

Include dependency graph for boundary.c:



### Functions

- void `initBoundary` (void)  
*Initialize boundary conditions.*
- void `boundary` (`side_t` \*aSide, double time, double int\_pVar[NVAR], double ghost\_pVar[NVAR], double x[NDIM])  
*Set boundary condition value at x.*
- void `setBCatSides` (double time)  
*Set the ghost values at sides.*
- void `setBCatBarys` (double time)  
*Set the ghost values at elements.*
- void `freeBoundary` (void)  
*Free all memory that was allocated for the boundary conditions.*

### Variables

- `boundary_t` \*firstBC
- int nBC
- bool isPeriodic

### 6.3.1 Detailed Description

Contains the functions for initializing and applying boundary conditions.

#### Author

hhh

#### Date

Tue 24 Mar 2020 10:10:51 AM CET

### 6.3.2 Function Documentation

#### 6.3.2.1 boundary()

```
void boundary (
    side_t * aSide,
    double time,
    double int_pVar[NVAR],
    double ghost_pVar[NVAR],
    double x[NDIM] )
```

Set boundary condition value at x.

#### Parameters

in	<i>aSide</i>	Pinter to a boundary side
in	<i>time</i>	Computation time at calculation
in	<i>int_pVar</i>	Internal cell primitive variables state
out	<i>ghost_pVar</i>	Ghost cell primitive variables state
in	<i>x</i>	Barycenter coordinates of the ghost cell

References `side_t::BC`, `boundary_t::BCtype`, `CHARACTERISTIC`, `charCons()`, `side_t::connection`, `consChar()`, `consPrim()`, `boundary_t::exactFunc`, `exactFunc()`, `EXACTSOL`, `gam`, `INFLOW`, `mu`, `MY`, `side_t::n`, `NDIM`, `NVAR`, `OUTFLOW`, `P`, `PRESSURE_OUT`, `primCons()`, `boundary_t::pVar`, `RHO`, `SLIPWALL`, `VX`, `VY`, `WALL`, `X`, and `Y`.

Referenced by `setBCatBarys()`, and `setBCatSides()`.

#### 6.3.2.2 setBCatBarys()

```
void setBCatBarys (
    double time )
```

Set the ghost values at elements.

**Parameters**

<code>in</code>	<code>time</code>	Computation time at calculation
-----------------	-------------------	---------------------------------

References `elem_t::bary`, `BCside`, `boundary()`, `side_t::connection`, `side_t::elem`, `nBCsides`, and `elem_t::pVar`.

Referenced by `spatialReconstruction()`.

**6.3.2.3 setBCatSides()**

```
void setBCatSides (
    double time )
```

Set the ghost values at sides.

**Parameters**

<code>in</code>	<code>time</code>	Computation time at calculation
-----------------	-------------------	---------------------------------

References `elem_t::bary`, `BCside`, `boundary()`, `side_t::connection`, `side_t::elem`, `side_t::GP`, `nBCsides`, `NDIM`, `side_t::pVar`, `X`, and `Y`.

Referenced by `fvTimeDerivative()`.

**6.3.3 Variable Documentation****6.3.3.1 firstBC**

```
boundary_t* firstBC
```

pointer to the first boundary condition

Referenced by `cgnsWriteMesh()`, `createMesh()`, `freeBoundary()`, `initBoundary()`, and `initWing()`.

**6.3.3.2 isPeriodic**

```
bool isPeriodic
```

periodic boundary condition flag

Referenced by `cgnsWriteMesh()`, and `connectPeriodicBC()`.

### 6.3.3.3 nBC

```
int nBC
```

number of boundary conditions

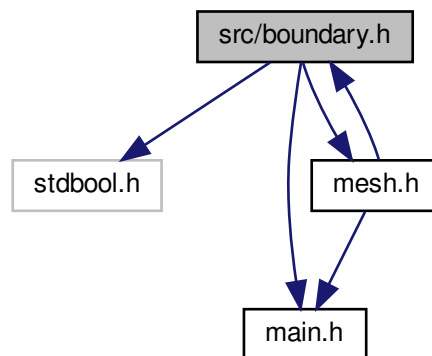
Referenced by `cgnsWriteMesh()`, and `initBoundary()`.

## 6.4 src/boundary.h File Reference

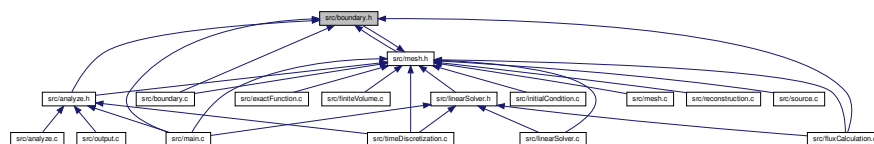
Contains the structure definition of a boundary.

```
#include <stdbool.h>
#include "main.h"
#include "mesh.h"
```

Include dependency graph for boundary.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [boundary\\_t](#)

Structure that holds the information of a boundary condition.

## Functions

- void `initBoundary` (void)  
*Initialize boundary conditions.*
- void `setBCatSides` (double time)  
*Set the ghost values at sides.*
- void `setBCatBarys` (double time)  
*Set the ghost values at elements.*
- void `boundary` (`side_t` \*aSide, double time, double int\_pVar[NVAR], double ghost\_pVar[NVAR], double x[NDIM])  
*Set boundary condition value at x.*
- void `freeBoundary` (void)  
*Free all memory that was allocated for the boundary conditions.*

## Variables

- `boundary_t` \*firstBC
- int nBC
- bool isPeriodic

### 6.4.1 Detailed Description

Contains the structure definition of a boundary.

#### Author

hhh

#### Date

Tue 24 Mar 2020 10:02:10 AM CET

### 6.4.2 Function Documentation

#### 6.4.2.1 `boundary()`

```
void boundary (
    side_t * aSide,
    double time,
    double int_pVar[NVAR],
    double ghost_pVar[NVAR],
    double x[NDIM] )
```

Set boundary condition value at x.

**Parameters**

in	<i>aSide</i>	Pinter to a boundary side
in	<i>time</i>	Computation time at calculation
in	<i>int_pVar</i>	Internal cell primitive variables state
out	<i>ghost_pVar</i>	Ghost cell primitive variables state
in	<i>x</i>	Barycenter coordinates of the ghost cell

References `side_t::BC`, `boundary_t::BCtype`, `CHARACTERISTIC`, `charCons()`, `side_t::connection`, `consChar()`, `consPrim()`, `boundary_t::exactFunc`, `exactFunc()`, `EXACTSOL`, `gam`, `INFLOW`, `mu`, `MY`, `side_t::n`, `NDIM`, `NVAR`, `OUTFLOW`, `P`, `PRESSURE_OUT`, `primCons()`, `boundary_t::pVar`, `RHO`, `SLIPWALL`, `VX`, `VY`, `WALL`, `X`, and `Y`.

Referenced by `setBCatBarys()`, and `setBCatSides()`.

**6.4.2.2 setBCatBarys()**

```
void setBCatBarys (
    double time )
```

Set the ghost values at elements.

**Parameters**

in	<i>time</i>	Computation time at calculation
----	-------------	---------------------------------

References `elem_t::bary`, `BCside`, `boundary()`, `side_t::connection`, `side_t::elem`, `nBCsides`, and `elem_t::pVar`.

Referenced by `spatialReconstruction()`.

**6.4.2.3 setBCatSides()**

```
void setBCatSides (
    double time )
```

Set the ghost values at sides.

**Parameters**

in	<i>time</i>	Computation time at calculation
----	-------------	---------------------------------

References `elem_t::bary`, `BCside`, `boundary()`, `side_t::connection`, `side_t::elem`, `side_t::GP`, `nBCsides`, `NDIM`, `side_t::pVar`, `X`, and `Y`.

Referenced by `fvTimeDerivative()`.

### 6.4.3 Variable Documentation

#### 6.4.3.1 firstBC

```
boundary_t* firstBC
```

pointer to the first boundary condition

Referenced by `cgnsWriteMesh()`, `createMesh()`, `freeBoundary()`, `initBoundary()`, and `initWing()`.

#### 6.4.3.2 isPeriodic

```
bool isPeriodic
```

periodic boundary condition flag

Referenced by `cgnsWriteMesh()`, and `connectPeriodicBC()`.

#### 6.4.3.3 nBC

```
int nBC
```

number of boundary conditions

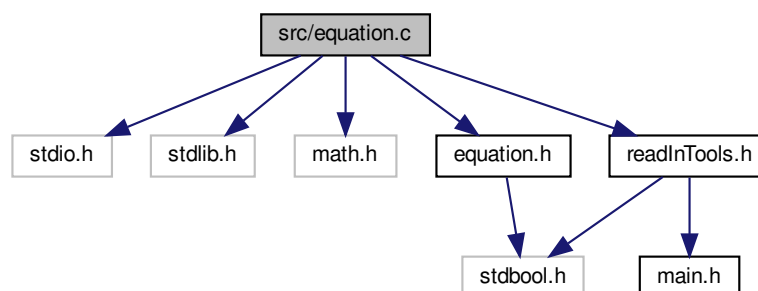
Referenced by `cgnsWriteMesh()`, and `initBoundary()`.

## 6.5 src/equation.c File Reference

Contains the function for initializing the physical constants.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "equation.h"
#include "readInTools.h"
```

Include dependency graph for `equation.c`:



## Functions

- void `initEquation` (void)  
*Initialize equations.*

## Variables

- double `pi`
- bool `doCalcSource`
- double `R`
- double `gam`
- double `gam1`
- double `gam2`
- double `gam1q`
- double `cp`
- double `Pr`
- double `mu`
- int `iFlux`
- int `intExactFunc`
- int `sourceFunc`
- double `sqrt2`
- double `sqrt3`
- double `sqrt3q`

### 6.5.1 Detailed Description

Contains the function for initializing the physical constants.

#### Date

Tue 24 Mar 2020 08:30:28 AM CET

#### Author

hhh

### 6.5.2 Variable Documentation

#### 6.5.2.1 `cp`

`double cp`

specific heat capacity

Referenced by `calcCoef()`.



### 6.5.2.2 doCalcSource

`bool doCalcSource`

calculate source flag

Referenced by `fvTimeDerivative()`, and `initEquation()`.

### 6.5.2.3 gam

`double gam`

specific heat ratio

Referenced by `boundary()`, `calcTimeStep()`, `charCons()`, `consChar()`, `evalSource()`, `exactFunc()`, `exactRiemann()`, `flux_ausmd()`, `flux_ausmdv()`, `flux_god()`, `flux_hll()`, `flux_hllc()`, `flux_hlle()`, `flux_lxf()`, `flux_roe()`, `flux_stw()`, `flux_vanleer()`, `GMRES_M()`, `initBoundary()`, `initEquation()`, and `initInitialCondition()`.

### 6.5.2.4 gam1

`double gam1`

`gam - 1`

Referenced by `consPrim()`, `evalSource()`, `exactFunc()`, `flux_god()`, `flux_hll()`, `flux_hllc()`, `flux_roe()`, `flux_stw()`, `flux_vanleer()`, and `initEquation()`.

### 6.5.2.5 gam1q

`double gam1q`

`1.0 / (gam - 1)`

Referenced by `charCons()`, `consChar()`, `flux_ausmd()`, `flux_ausmdv()`, `flux_cen()`, `flux_hll()`, `flux_hllc()`, `flux_hlle()`, `flux_lxf()`, `flux_roe()`, `flux_stw()`, `flux_vanleer()`, `initEquation()`, and `primCons()`.

### 6.5.2.6 gam2

`double gam2`

`gam - 2`

Referenced by `flux_roe()`, and `initEquation()`.

#### 6.5.2.7 iFlux

`int iFlux`

flux function control

Referenced by `convectiveFlux()`, and `initEquation()`.

#### 6.5.2.8 intExactFunc

`int intExactFunc`

exact function control

Referenced by `calcErrors()`, `cgnsOutput()`, `csvOutput()`, `curveOutput()`, `initInitialCondition()`, and `setInitialCondition()`.

#### 6.5.2.9 mu

`double mu`

dynamic viscosity

Referenced by `boundary()`, `calcTimeStep()`, `evalSource()`, and `initEquation()`.

#### 6.5.2.10 pi

`double pi`

pi

Referenced by `calcCoef()`, `evalSource()`, `exactFunc()`, `initBoundary()`, `initEquation()`, and `initInitialCondition()`.

#### 6.5.2.11 Pr

`double Pr`

Prandtl number

Referenced by `calcTimeStep()`, `evalSource()`, and `initEquation()`.

#### 6.5.2.12 R

`double R`

specific gas constant

Referenced by `exactFunc()`, and `initEquation()`.

#### 6.5.2.13 sourceFunc

`int sourceFunc`

source function control

Referenced by `calcSource()`, and `initEquation()`.

#### 6.5.2.14 sqrt2

`double sqrt2`

`sqrt(2.0)`

Referenced by `initEquation()`.

#### 6.5.2.15 sqrt3

`double sqrt3`

`sqrt(3.0)`

Referenced by `initEquation()`.

#### 6.5.2.16 sqrt3q

`double sqrt3q`

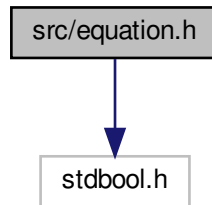
`1.0 / sqrt(3.0)`

Referenced by `exactFunc()`, and `initEquation()`.

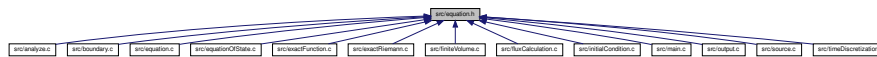
## 6.6 src/equation.h File Reference

```
#include <stdbool.h>
```

Include dependency graph for equation.h:



This graph shows which files directly or indirectly include this file:



### Functions

- void [initEquation](#) (void)  
*Initialize equations.*

### Variables

- double [pi](#)
- bool [doCalcSource](#)
- double [R](#)
- double [gam](#)
- double [gam1](#)
- double [gam2](#)
- double [gam1q](#)
- double [cp](#)
- double [Pr](#)
- double [mu](#)
- int [iFlux](#)
- int [intExactFunc](#)
- int [sourceFunc](#)
- double [sqrt2](#)
- double [sqrt3](#)
- double [sqrt3q](#)

## 6.6.1 Detailed Description

Author

hhh

Date

Tue 24 Mar 2020 08:23:21 AM CET

## 6.6.2 Variable Documentation

### 6.6.2.1 cp

double cp

specific heat capacity

Referenced by calcCoef().

### 6.6.2.2 doCalcSource

bool doCalcSource

calculate source flag

Referenced by fvTimeDerivative(), and initEquation().

### 6.6.2.3 gam

double gam

specific heat ratio

Referenced by boundary(), calcTimeStep(), charCons(), consChar(), evalSource(), exactFunc(), exactRiemann(), flux\_ausmd(), flux\_ausmdv(), flux\_god(), flux\_hll(), flux\_hllc(), flux\_hlle(), flux\_lxf(), flux\_roe(), flux\_stw(), flux\_↔vanleer(), GMRES\_M(), initBoundary(), initEquation(), and initInitialCondition().

#### 6.6.2.4 gam1

```
double gam1
```

```
gam - 1
```

Referenced by consPrim(), evalSource(), exactFunc(), flux\_god(), flux\_hll(), flux\_hllc(), flux\_roe(), flux\_stw(), flux\_vanleer(), and initEquation().

#### 6.6.2.5 gam1q

```
double gam1q
```

```
1.0 / (gam - 1)
```

Referenced by charCons(), consChar(), flux\_ausmd(), flux\_ausmdv(), flux\_cen(), flux\_hll(), flux\_hllc(), flux\_hlle(), flux\_lxf(), flux\_roe(), flux\_stw(), flux\_vanleer(), initEquation(), and primCons().

#### 6.6.2.6 gam2

```
double gam2
```

```
gam - 2
```

Referenced by flux\_roe(), and initEquation().

#### 6.6.2.7 iFlux

```
int iFlux
```

```
flux function control
```

Referenced by convectiveFlux(), and initEquation().

#### 6.6.2.8 intExactFunc

```
int intExactFunc
```

```
exact function control
```

Referenced by calcErrors(), cgnsOutput(), csvOutput(), curveOutput(), initInitialCondition(), and setInitialCondition().

### 6.6.2.9 mu

```
double mu
```

dynamic viscosity

Referenced by `boundary()`, `calcTimeStep()`, `evalSource()`, and `initEquation()`.

### 6.6.2.10 pi

```
double pi
```

pi

Referenced by `calcCoef()`, `evalSource()`, `exactFunc()`, `initBoundary()`, `initEquation()`, and `initInitialCondition()`.

### 6.6.2.11 Pr

```
double Pr
```

Prandtl number

Referenced by `calcTimeStep()`, `evalSource()`, and `initEquation()`.

### 6.6.2.12 R

```
double R
```

specific gas constant

Referenced by `exactFunc()`, and `initEquation()`.

### 6.6.2.13 sourceFunc

```
int sourceFunc
```

source function control

Referenced by `calcSource()`, and `initEquation()`.

#### 6.6.2.14 `sqrt2`

```
double sqrt2
```

```
sqrt(2.0)
```

Referenced by `initEquation()`.

#### 6.6.2.15 `sqrt3`

```
double sqrt3
```

```
sqrt(3.0)
```

Referenced by `initEquation()`.

#### 6.6.2.16 `sqrt3q`

```
double sqrt3q
```

```
1.0 / sqrt(3.0)
```

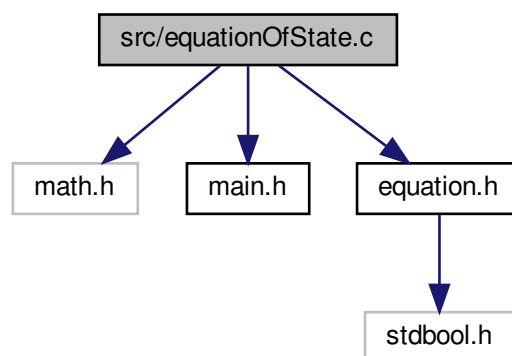
Referenced by `exactFunc()`, and `initEquation()`.

## 6.7 `src/equationOfState.c` File Reference

Contains conversion functions between the different variable types.

```
#include <math.h>
#include "main.h"
#include "equation.h"
```

Include dependency graph for `equationOfState.c`:





## Functions

- void [primCons](#) (const double pVar[NVAR], double cVar[NVAR])  
*Convert primitive variables into conservative variables.*
- void [consPrim](#) (const double cVar[NVAR], double pVar[NVAR])  
*Convert conservative variables into primitive variables.*
- void [consChar](#) (double cVar[NVAR], double charac[3], double pVarRef[NVAR])  
*Convert conservative variables to characteristic variables.*
- void [charCons](#) (double charac[3], double cVar[NVAR], double pVarRef[NVAR])  
*Convert characteristic variables to conservative variables.*

### 6.7.1 Detailed Description

Contains conversion functions between the different variable types.

#### Author

hhh

#### Date

Sat 28 Mar 2020 09:45:30 PM CET

### 6.7.2 Function Documentation

#### 6.7.2.1 charCons()

```
void charCons (
    double charac[3],
    double cVar[NVAR],
    double pVarRef[NVAR] )
```

Convert characteristic variables to conservative variables.

#### Parameters

in	<i>charac</i>	Characteristic variable vector
out	<i>cVar</i>	Conservative variable vector
in	<i>pVarRef</i>	Reference primitive variable vector

References E, gam, gam1q, MX, P, RHO, and VX.

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

### 6.7.2.2 consChar()

```
void consChar (
    double cVar[NVAR],
    double charac[3],
    double pVarRef[NVAR] )
```

Convert conservative variables to characteristic variables.

#### Parameters

in	<i>cVar</i>	Conservative variable vector
out	<i>charac</i>	Characteristic variable vector
in	<i>pVarRef</i>	Reference primitive variable vector

References E, gam, gam1q, P, RHO, and VX.

Referenced by boundary().

### 6.7.2.3 consPrim()

```
void consPrim (
    const double cVar[NVAR],
    double pVar[NVAR] )
```

Convert conservative variables into primitive variables.

This function is used during reconstruction, therefore it has to be checked if the resulting primitive variables are negative. If that is the case, they are set to zero.

#### Parameters

in	<i>cVar</i>	Conservative variable vector
out	<i>pVar</i>	Primitive variable vector

References E, gam1, MX, MY, P, RHO, VX, and VY.

Referenced by boundary(), exactFunc(), explicitTimeStepEuler(), explicitTimeStepRK(), implicitTimeStep(), and matrixVector().

### 6.7.2.4 primCons()

```
void primCons (
    const double pVar[NVAR],
    double cVar[NVAR] )
```

Convert primitive variables into conservative variables.

## Parameters

in	<i>pVar</i>	Primitive variable vector
out	<i>cVar</i>	Conservative variable vector

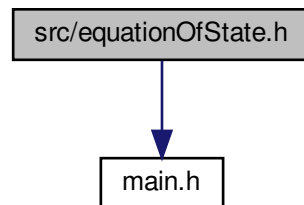
References E, gam1q, MX, MY, P, RHO, VX, and VY.

Referenced by boundary(), and setInitialCondition().

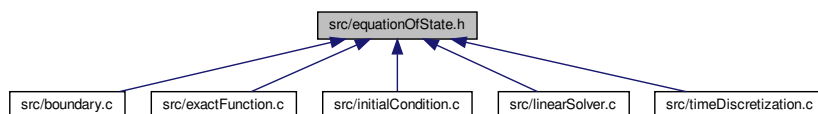
## 6.8 src/equationOfState.h File Reference

```
#include "main.h"
```

Include dependency graph for equationOfState.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void [primCons](#) (const double pVar[NVAR], double cVar[NVAR])  
Convert primitive variables into conservative variables.
- void [consPrim](#) (const double cVar[NVAR], double pVar[NVAR])  
Convert conservative variables into primitive variables.
- void [consChar](#) (double cVar[NVAR], double charac[3], double pVarRef[NVAR])  
Convert conservative variables to characteristic variables.
- void [charCons](#) (double charac[3], double cVar[NVAR], double pVarRef[NVAR])  
Convert characteristic variables to conservative variables.

## 6.8.1 Detailed Description

### Author

hhh

### Date

Sat 28 Mar 2020 09:45:50 PM CET

## 6.8.2 Function Documentation

### 6.8.2.1 charCons()

```
void charCons (
    double charac[3],
    double cVar[NVAR],
    double pVarRef[NVAR] )
```

Convert characteristic variables to conservative variables.

#### Parameters

in	<i>charac</i>	Characteristic variable vector
out	<i>cVar</i>	Conservative variable vector
in	<i>pVarRef</i>	Reference primitive variable vector

References E, gam, gam1q, MX, P, RHO, and VX.

Referenced by boundary().

### 6.8.2.2 consChar()

```
void consChar (
    double cVar[NVAR],
    double charac[3],
    double pVarRef[NVAR] )
```

Convert conservative variables to characteristic variables.

#### Parameters

in	<i>cVar</i>	Conservative variable vector
out	<i>charac</i>	Characteristic variable vector
in	<i>pVarRef</i>	Reference primitive variable vector

References E, gam, gam1q, P, RHO, and VX.

Referenced by boundary().

### 6.8.2.3 consPrim()

```
void consPrim (
    const double cVar[NVAR],
    double pVar[NVAR] )
```

Convert conservative variables into primitive variables.

This function is used during reconstruction, therefore it has to be checked if the resulting primitive variables are negative. If that is the case, they are set to zero.

#### Parameters

in	<i>cVar</i>	Conservative variable vector
out	<i>pVar</i>	Primitive variable vector

References E, gam1, MX, MY, P, RHO, VX, and VY.

Referenced by boundary(), exactFunc(), explicitTimeStepEuler(), explicitTimeStepRK(), implicitTimeStep(), and matrixVector().

### 6.8.2.4 primCons()

```
void primCons (
    const double pVar[NVAR],
    double cVar[NVAR] )
```

Convert primitive variables into conservative variables.

#### Parameters

in	<i>pVar</i>	Primitive variable vector
out	<i>cVar</i>	Conservative variable vector

References E, gam1q, MX, MY, P, RHO, VX, and VY.

Referenced by boundary(), and setInitialCondition().

## 6.9 src/exactFunction.c File Reference

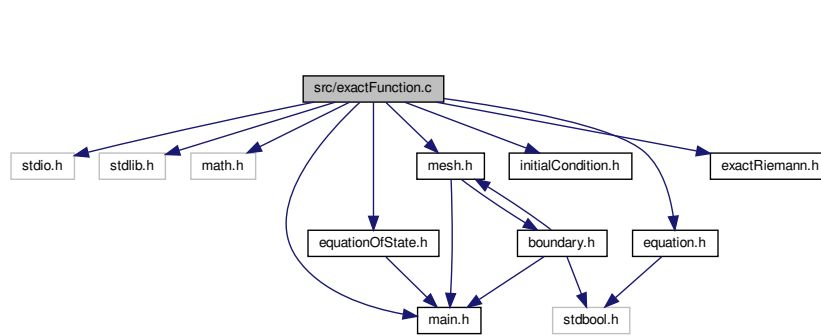
Contains the exact function evaluation function.

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "main.h"
#include "mesh.h"
#include "equation.h"
#include "initialCondition.h"
#include "equationOfState.h"
#include "exactRiemann.h"

```

Include dependency graph for exactFunction.c:



## Functions

- void [exactFunc](#) (int iExactFunc, double x[[NDIM](#)], double time, double pVar[[NVAR](#)])  
Calculate an exact function.

### 6.9.1 Detailed Description

Contains the exact function evaluation function.

Author

hhh

Date

Sat 28 Mar 2020 05:31:15 PM CET

### 6.9.2 Function Documentation

### 6.9.2.1 exactFunc()

```
void exactFunc (
    int iExactFunc,
    double x[NDIM],
    double time,
    double pVar[NVAR] )
```

Calculate an exact function.

This function contains the following exact functions:

- 1: Richtmyer-Meshkov
- 2: Gaussian pressure pulse
- 3: Sinewave
- 4: Double mach reflection
- 5: 1D Riemann problem
- 6: 1D sine wave

#### Parameters

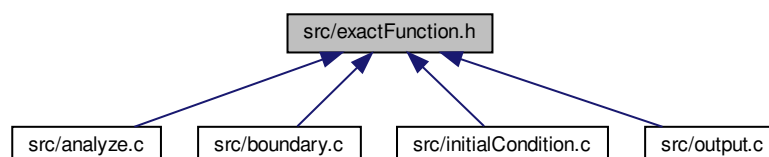
in	<i>iExactFunc</i>	The exact function control
in	<i>x</i>	The coordinates for which to calculate the exact function
in	<i>time</i>	The time for which to compute the exact value
out	<i>pVar</i>	The resulting vector of primitive variables

References consPrim(), dxRef, E, exactRiemann(), gam, gam1, NVAR, P, pi, R, refState, RHO, rp1Dinterface, sqrt3q, VX, VY, X, xMax, xMin, Y, yMax, and yMin.

Referenced by boundary(), calcErrors(), cgnsOutput(), csvOutput(), curveOutput(), and setInitialCondition().

## 6.10 src/exactFunction.h File Reference

This graph shows which files directly or indirectly include this file:



## Functions

- void `exactFunc` (int `iExactFunc`, double `x`[`NDIM`], double `time`, double `pVar`[`NVAR`])  
*Calculate an exact function.*

### 6.10.1 Detailed Description

#### Author

hhh

#### Date

Sat 28 Mar 2020 05:55:25 PM CET

### 6.10.2 Function Documentation

#### 6.10.2.1 `exactFunc()`

```
void exactFunc (
    int iExactFunc,
    double x[NDIM],
    double time,
    double pVar[NVAR] )
```

Calculate an exact function.

This function contains the following exact functions:

- 1: Richtmyer-Meshkov
- 2: Gaussian pressure pulse
- 3: Sinewave
- 4: Double mach reflection
- 5: 1D Riemann problem
- 6: 1D sine wave

#### Parameters

in	<i>iExactFunc</i>	The exact function control
in	<i>x</i>	The coordinates for which to calculate the exact function
in	<i>time</i>	The time for which to compute the exact value
out	<i>pVar</i>	The resulting vector of primitive variables

References `consPrim()`, `dxRef`, `E`, `exactRiemann()`, `gam`, `gam1`, `NVAR`, `P`, `pi`, `R`, `refState`, `RHO`, `rp1Dinterface`,



sqrt3q, VX, VY, X, xMax, xMin, Y, yMax, and yMin.

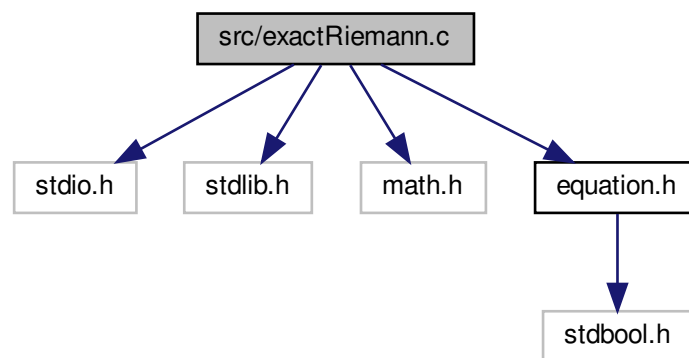
Referenced by boundary(), calcErrors(), cgnsOutput(), csvOutput(), curveOutput(), and setInitialCondition().

## 6.11 src/exactRiemann.c File Reference

Contains the function to calculate the exact Riemann flux.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "equation.h"
```

Include dependency graph for exactRiemann.c:



### Functions

- void [preFun](#) (double \*f, double \*fd, double p, double rhok, double pk, double ck)  
*Helper function for exactRiemann*
- void [exactRiemann](#) (double rhoL, double rhoR, double \*rho, double uL, double uR, double \*u, double pL, double pR, double \*p, double aL, double aR, double s)  
*Calculate the exact solution to the Riemann problem.*

### Variables

- double [G](#) [9]
- double [tol](#) = 1e-6
- int [nlter](#) = 1000

### 6.11.1 Detailed Description

Contains the function to calculate the exact Riemann flux.

#### Author

hhh

#### Date

Sun 29 Mar 2020 12:35:17 PM CEST

### 6.11.2 Function Documentation

#### 6.11.2.1 exactRiemann()

```
void exactRiemann (
    double rhol,
    double rhor,
    double * rho,
    double ul,
    double ur,
    double * u,
    double pl,
    double pr,
    double * p,
    double al,
    double ar,
    double s )
```

Calculate the exact solution to the Riemann problem.

#### Parameters

in	<i>rho</i> <sub>l</sub>	Left side density
in	<i>rho</i> <sub>r</sub>	Right side density
out	<i>rho</i>	The resulting density
in	<i>u</i> <sub>l</sub>	Left side velocity
in	<i>u</i> <sub>r</sub>	Right side velocity
out	<i>u</i>	Resulting velocity
in	<i>p</i> <sub>l</sub>	Left side pressure
in	<i>p</i> <sub>r</sub>	Right side pressure
out	<i>p</i>	Resulting pressure
in	<i>a</i> <sub>l</sub>	Left side speed of sound
in	<i>a</i> <sub>r</sub>	Right side speed of sound
in	<i>s</i>	Speed of the discontinuity

References G, gam, nlter, preFun(), and tol.

Referenced by `exactFunc()`, and `flux_god()`.

### 6.11.2.2 preFun()

```
void preFun (
    double * f,
    double * fd,
    double p,
    double rhok,
    double pk,
    double ck )
```

Helper function for `exactRiemann`

#### Parameters

out	<i>f</i>	Flux
out	<i>fd</i>	Flux difference
in	<i>p</i>	Pressure
in	<i>rhok</i>	Critical density
in	<i>pk</i>	Critical pressure
in	<i>ck</i>	Critical speed of sound

References G.

Referenced by `exactRiemann()`.

## 6.11.3 Variable Documentation

### 6.11.3.1 G

```
double G[9]
```

gamma vector

Referenced by `exactRiemann()`, and `preFun()`.

### 6.11.3.2 nIter

```
int nIter = 1000
```

maximum number of iterations

Referenced by `exactRiemann()`.

### 6.11.3.3 tol

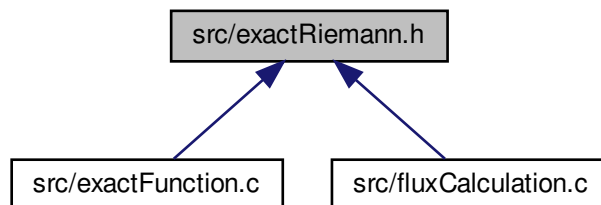
```
double tol = 1e-6
```

tolerance for the iteration

Referenced by `exactRiemann()`.

## 6.12 src/exactRiemann.h File Reference

This graph shows which files directly or indirectly include this file:



## Functions

- void `exactRiemann` (double rho1, double rho2, double \*rho, double ul, double ur, double \*u, double pl, double pr, double \*p, double al, double ar, double s)

*Calculate the exact solution to the Riemann problem.*

### 6.12.1 Detailed Description

Date

Sun 29 Mar 2020 12:35:26 PM CEST

Author

hhh

### 6.12.2 Function Documentation

### 6.12.2.1 exactRiemann()

```
void exactRiemann (
    double rhol,
    double rhor,
    double * rho,
    double ul,
    double ur,
    double * u,
    double pl,
    double pr,
    double * p,
    double al,
    double ar,
    double s )
```

Calculate the exact solution to the Riemann problem.

#### Parameters

in	<i>rho</i> <sub>l</sub>	Left side density
in	<i>rho</i> <sub>r</sub>	Right side density
out	<i>rho</i>	The resulting density
in	<i>u</i> <sub>l</sub>	Left side velocity
in	<i>u</i> <sub>r</sub>	Right side velocity
out	<i>u</i>	Resulting velocity
in	<i>p</i> <sub>l</sub>	Left side pressure
in	<i>p</i> <sub>r</sub>	Right side pressure
out	<i>p</i>	Resulting pressure
in	<i>a</i> <sub>l</sub>	Left side speed of sound
in	<i>a</i> <sub>r</sub>	Right side speed of sound
in	<i>s</i>	Speed of the discontinuity

References `G`, `gam`, `nIter`, `preFun()`, and `tol`.

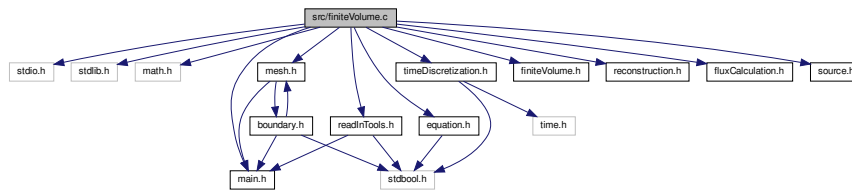
Referenced by `exactFunc()`, and `flux_god()`.

## 6.13 src/finiteVolume.c File Reference

Finite volume time derivative functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "main.h"
#include "finiteVolume.h"
#include "mesh.h"
#include "readInTools.h"
#include "reconstruction.h"
#include "timeDiscretization.h"
#include "fluxCalculation.h"
```

```
#include "equation.h"
#include "source.h"
Include dependency graph for finiteVolume.c:
```



## Functions

- void `initFV` (void)  
*Initialize the finite volume method.*
- void `fvTimeDerivative` (double time)  
*Perform the spacial operator of the finite volume scheme.*

## Variables

- int `spatialOrder`
- int `fluxFunction`

### 6.13.1 Detailed Description

Finite volume time derivative functions.

Author

hhh

Date

Fri 27 Mar 2020 05:10:43 PM CET

### 6.13.2 Function Documentation

#### 6.13.2.1 `fvTimeDerivative()`

```
void fvTimeDerivative (
    double time )
```

Perform the spacial operator of the finite volume scheme.

First, the local time step is calculated, then spacial gradients inside of the cells are reconstructed. Following that, the boundary conditions at the sides are applied and the numerical flux is calculated, using the specified flux function. Finally, the source term is evaluated and the time derivatives of all the elements are calculated.

## Parameters

<code>in</code>	<code>time</code>	Calculation time at which to perform the finite volume differentiation
-----------------	-------------------	--

References `elem_t::areaq`, `calcSource()`, `doCalcSource`, `elem_t::dt`, `elem_t::dtLoc`, `E`, `elem`, `elem_t::firstSide`, `side_t::flux`, `fluxCalculation()`, `nElems`, `side_t::nextElemSide`, `RHO`, `setBCatSides()`, `elem_t::source`, `spatialReconstruction()`, `timeOrder`, `elem_t::u_t`, `VX`, and `VY`.

Referenced by `buildMatrix()`, `explicitTimeStepEuler()`, `explicitTimeStepRK()`, `implicitTimeStep()`, and `matrixVector()`.

### 6.13.3 Variable Documentation

#### 6.13.3.1 fluxFunction

```
int fluxFunction
```

the flux function to be used

#### 6.13.3.2 spatialOrder

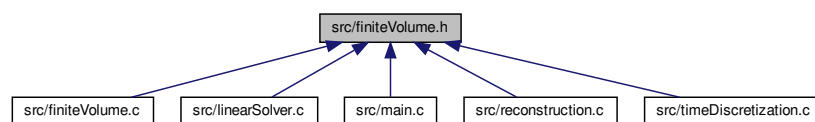
```
int spatialOrder
```

the spacial order to be used

Referenced by `initFV()`, and `spatialReconstruction()`.

## 6.14 src/finiteVolume.h File Reference

This graph shows which files directly or indirectly include this file:



## Functions

- void `initFV` (void)  
*Initialize the finite volume method.*
- void `fvTimeDerivative` (double time)  
*Perform the spacial operator of the finite volume scheme.*

## Variables

- int `spatialOrder`
- int `fluxFunction`

### 6.14.1 Detailed Description

#### Author

hhh

#### Date

Fri 27 Mar 2020 05:09:57 PM CET

### 6.14.2 Function Documentation

#### 6.14.2.1 `fvTimeDerivative()`

```
void fvTimeDerivative (
    double time )
```

Perform the spacial operator of the finite volume scheme.

First, the local time step is calculated, then spacial gradients inside of the cells are reconstructed. Following that, the boundary conditions at the sides are applied and the numerical flux is calculated, using the specified flux function. Finally, the source term is evaluated and the time derivatives of all the elements are calculated.

#### Parameters

<code>in</code>	<code>time</code>	Calculation time at which to perform the finite volume differentiation
-----------------	-------------------	--

References `elem_t::areaq`, `calcSource()`, `doCalcSource`, `elem_t::dt`, `elem_t::dtLoc`, `E`, `elem`, `elem_t::firstSide`, `side_t::flux`, `fluxCalculation()`, `nElems`, `side_t::nextElemSide`, `RHO`, `setBCatSides()`, `elem_t::source`, `spatial↔Reconstruction()`, `timeOrder`, `elem_t::u_t`, `VX`, and `VY`.

Referenced by `buildMatrix()`, `explicitTimeStepEuler()`, `explicitTimeStepRK()`, `implicitTimeStep()`, and `matrixVector()`.

### 6.14.3 Variable Documentation

#### 6.14.3.1 `fluxFunction`

```
int fluxFunction
```

the flux function to be used



### 6.14.3.2 spatialOrder

```
int spatialOrder
```

the spacial order to be used

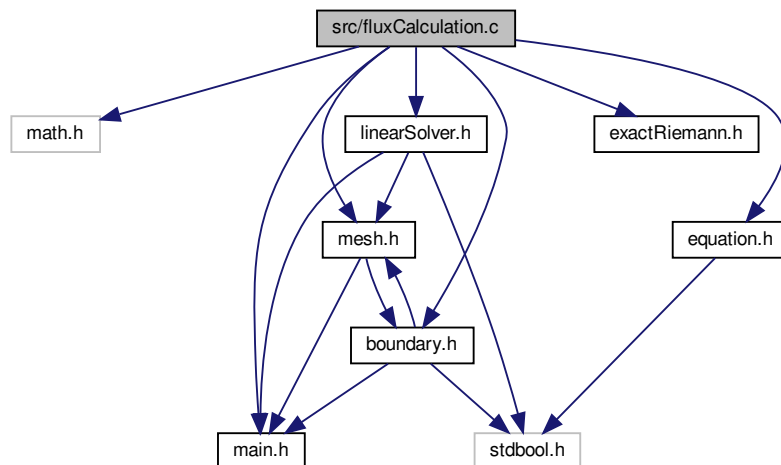
Referenced by `initFV()`, and `spatialReconstruction()`.

## 6.15 src/fluxCalculation.c File Reference

Contains the flux calculation functions.

```
#include <math.h>
#include "main.h"
#include "mesh.h"
#include "equation.h"
#include "exactRiemann.h"
#include "boundary.h"
#include "linearSolver.h"
```

Include dependency graph for `fluxCalculation.c`:



## Functions

- void `flux_god` (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])  
*Godunov flux, which is the exact flux.*
- void `flux_roe` (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])  
*Roe flux.*
- void `flux_hll` (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])  
*HLL flux.*

- void `flux_hlle` (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])  
*HLLC flux.*
- void `flux_hllc` (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])  
*HLLC flux.*
- void `flux_lxf` (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])  
*Lax-Friedrichs flux.*
- void `flux_stw` (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])  
*Steger-Warming flux.*
- void `flux_cen` (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])  
*Central flux.*
- void `flux_ausmd` (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])  
*AUSMD flux.*
- void `flux_ausmdv` (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])  
*AUSMDV flux.*
- void `flux_vanleer` (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])  
*Van Leer flux.*
- void `convectiveFlux` (double rhoL, double rhoR, double vxL, double vxR, double vyL, double vyR, double pL, double pR, double fluxLoc[4])  
*Select the convective flux.*
- void `fluxCalculation` (void)  
*Perform the flux calculation.*

### 6.15.1 Detailed Description

Contains the flux calculation functions.

Author

hhh

Date

Tue 31 Mar 2020 05:18:40 PM CEST

### 6.15.2 Function Documentation

### 6.15.2.1 convectiveFlux()

```
void convectiveFlux (
    double rhoL,
    double rhoR,
    double vxL,
    double vxR,
    double vyL,
    double vyR,
    double pL,
    double pR,
    double fluxLoc[4] )
```

Select the convective flux.

#### Parameters

in	<i>rhoL</i>	Left side density
in	<i>rhoR</i>	Right side density
in	<i>vxL</i>	Left side x-velocity
in	<i>vxR</i>	Right side x-velocity
in	<i>vyL</i>	Left side y-velocity
in	<i>vyR</i>	Right side y-velocity
in	<i>pL</i>	Left side pressure
in	<i>pR</i>	Right side pressure
out	<i>fluxLoc</i>	The local numeric flux

References AUSMD, AUSMDV, CEN, flux\_ausmd(), flux\_ausmdv(), flux\_cen(), flux\_god(), flux\_hll(), flux\_hllc(), flux\_hlle(), flux\_lxf(), flux\_roe(), flux\_stw(), flux\_vanleer(), GOD, HLL, HLLC, HLLE, iFlux, LXF, ROE, STW, and VANLEER.

Referenced by fluxCalculation().

### 6.15.2.2 flux\_ausmd()

```
void flux_ausmd (
    double rhoL,
    double rhoR,
    double vxL,
    double vxR,
    double vyL,
    double vyR,
    double pL,
    double pR,
    double fluxLoc[4] )
```

AUSMD flux.

#### Parameters

in	<i>rhoL</i>	Left side density
----	-------------	-------------------

**Parameters**

in	<i>rhoR</i>	Right side density
in	<i>vxL</i>	Left side x-velocity
in	<i>vxR</i>	Right side x-velocity
in	<i>vyL</i>	Left side y-velocity
in	<i>vyR</i>	Right side y-velocity
in	<i>pL</i>	Left side pressure
in	<i>pR</i>	Right side pressure
out	<i>fluxLoc</i>	The local numeric flux

References gam, and gam1q.

Referenced by convectiveFlux().

**6.15.2.3 flux\_ausmdv()**

```
void flux_ausmdv (
    double rhoL,
    double rhoR,
    double vxL,
    double vxR,
    double vyL,
    double vyR,
    double pL,
    double pR,
    double fluxLoc[4] )
```

AUSMDV flux.

**Bug** This function produces incorrect output, refrain from using it for the time being

**Parameters**

in	<i>rhoL</i>	Left side density
in	<i>rhoR</i>	Right side density
in	<i>vxL</i>	Left side x-velocity
in	<i>vxR</i>	Right side x-velocity
in	<i>vyL</i>	Left side y-velocity
in	<i>vyR</i>	Right side y-velocity
in	<i>pL</i>	Left side pressure
in	<i>pR</i>	Right side pressure
out	<i>fluxLoc</i>	The local numeric flux

References gam, and gam1q.

Referenced by convectiveFlux().

### 6.15.2.4 flux\_cen()

```
void flux_cen (
    double rhoL,
    double rhoR,
    double vxL,
    double vxR,
    double vyL,
    double vyR,
    double pL,
    double pR,
    double fluxLoc[4] )
```

Central flux.

#### Note

This flux is unconditionally unstable, it can be stabilized by adding artificial viscosity (Jameson method). This is not implemented, however.

#### Parameters

in	<i>rhoL</i>	Left side density
in	<i>rhoR</i>	Right side density
in	<i>vxL</i>	Left side x-velocity
in	<i>vxR</i>	Right side x-velocity
in	<i>vyL</i>	Left side y-velocity
in	<i>vyR</i>	Right side y-velocity
in	<i>pL</i>	Left side pressure
in	<i>pR</i>	Right side pressure
out	<i>fluxLoc</i>	The local numeric flux

References gam1q.

Referenced by convectiveFlux().

### 6.15.2.5 flux\_god()

```
void flux_god (
    double rhoL,
    double rhoR,
    double vxL,
    double vxR,
    double vyL,
    double vyR,
    double pL,
    double pR,
    double fluxLoc[4] )
```

Godunov flux, which is the exact flux.

**Parameters**

in	<i>rhoL</i>	Left side density
in	<i>rhoR</i>	Right side density
in	<i>vxL</i>	Left side x-velocity
in	<i>vxR</i>	Right side x-velocity
in	<i>vyL</i>	Left side y-velocity
in	<i>vyR</i>	Right side y-velocity
in	<i>pL</i>	Left side pressure
in	<i>pR</i>	Right side pressure
out	<i>fluxLoc</i>	The local numeric flux

References `exactRiemann()`, `gam`, and `gam1`.

Referenced by `convectiveFlux()`.

**6.15.2.6 flux\_hll()**

```
void flux_hll (
    double rhoL,
    double rhoR,
    double vxL,
    double vxR,
    double vyL,
    double vyR,
    double pL,
    double pR,
    double fluxLoc[4] )
```

HLL flux.

**Parameters**

in	<i>rhoL</i>	Left side density
in	<i>rhoR</i>	Right side density
in	<i>vxL</i>	Left side x-velocity
in	<i>vxR</i>	Right side x-velocity
in	<i>vyL</i>	Left side y-velocity
in	<i>vyR</i>	Right side y-velocity
in	<i>pL</i>	Left side pressure
in	<i>pR</i>	Right side pressure
out	<i>fluxLoc</i>	The local numeric flux

References `gam`, `gam1`, `gam1q`, `MX`, and `NVAR`.

Referenced by `convectiveFlux()`.

### 6.15.2.7 flux\_hllc()

```
void flux_hllc (
    double rhoL,
    double rhoR,
    double vxL,
    double vxR,
    double vyL,
    double vyR,
    double pL,
    double pR,
    double fluxLoc[4] )
```

HLLC flux.

#### Parameters

in	<i>rhoL</i>	Left side density
in	<i>rhoR</i>	Right side density
in	<i>vxL</i>	Left side x-velocity
in	<i>vxR</i>	Right side x-velocity
in	<i>vyL</i>	Left side y-velocity
in	<i>vyR</i>	Right side y-velocity
in	<i>pL</i>	Left side pressure
in	<i>pR</i>	Right side pressure
out	<i>fluxLoc</i>	The local numeric flux

References gam, gam1, gam1q, MX, and NVAR.

Referenced by convectiveFlux().

### 6.15.2.8 flux\_hlle()

```
void flux_hlle (
    double rhoL,
    double rhoR,
    double vxL,
    double vxR,
    double vyL,
    double vyR,
    double pL,
    double pR,
    double fluxLoc[4] )
```

HLLE flux.

#### Parameters

in	<i>rhoL</i>	Left side density
in	<i>rhoR</i>	Right side density
in	<i>vxL</i>	Left side x-velocity

**Parameters**

in	$vxR$	Right side x-velocity
in	$vyL$	Left side y-velocity
in	$vyR$	Right side y-velocity
in	$pL$	Left side pressure
in	$pR$	Right side pressure
out	$fluxLoc$	The local numeric flux

References  $\gamma$ ,  $\gamma_1q$ ,  $MX$ , and  $NVAR$ .

Referenced by `convectiveFlux()`.

**6.15.2.9 flux\_lxf()**

```
void flux_lxf (
    double rhoL,
    double rhoR,
    double vxL,
    double vxR,
    double vyL,
    double vyR,
    double pL,
    double pR,
    double fluxLoc[4] )
```

Lax-Friedrichs flux.

**Parameters**

in	$\rho L$	Left side density
in	$\rho R$	Right side density
in	$vxL$	Left side x-velocity
in	$vxR$	Right side x-velocity
in	$vyL$	Left side y-velocity
in	$vyR$	Right side y-velocity
in	$pL$	Left side pressure
in	$pR$	Right side pressure
out	$fluxLoc$	The local numeric flux

References  $\gamma$ ,  $\gamma_1q$ , and  $NVAR$ .

Referenced by `convectiveFlux()`.

**6.15.2.10 flux\_roe()**

```
void flux_roe (
    double rhoL,
```



```

double rhoR,
double vxL,
double vxR,
double vyL,
double vyR,
double pL,
double pR,
double fluxLoc[4] )

```

Roe flux.

#### Parameters

in	<i>rhoL</i>	Left side density
in	<i>rhoR</i>	Right side density
in	<i>vxL</i>	Left side x-velocity
in	<i>vxR</i>	Right side x-velocity
in	<i>vyL</i>	Left side y-velocity
in	<i>vyR</i>	Right side y-velocity
in	<i>pL</i>	Left side pressure
in	<i>pR</i>	Right side pressure
out	<i>fluxLoc</i>	The local numeric flux

References gam, gam1, gam1q, and gam2.

Referenced by convectiveFlux().

#### 6.15.2.11 flux\_stw()

```

void flux_stw (
    double rhoL,
    double rhoR,
    double vxL,
    double vxR,
    double vyL,
    double vyR,
    double pL,
    double pR,
    double fluxLoc[4] )

```

Steger-Warming flux.

#### Parameters

in	<i>rhoL</i>	Left side density
in	<i>rhoR</i>	Right side density
in	<i>vxL</i>	Left side x-velocity
in	<i>vxR</i>	Right side x-velocity
in	<i>vyL</i>	Left side y-velocity
in	<i>vyR</i>	Right side y-velocity
in	<i>pL</i>	Left side pressure
in	<i>pR</i>	Right side pressure
out	<i>fluxLoc</i>	The local numeric flux

References gam, gam1, and gam1q.

Referenced by convectiveFlux().

#### 6.15.2.12 flux\_vanleer()

```
void flux_vanleer (
    double rhoL,
    double rhoR,
    double vxL,
    double vxR,
    double vyL,
    double vyR,
    double pL,
    double pR,
    double fluxLoc[4] )
```

Van Leer flux.

##### Parameters

in	<i>rhoL</i>	Left side density
in	<i>rhoR</i>	Right side density
in	<i>vxL</i>	Left side x-velocity
in	<i>vxR</i>	Right side x-velocity
in	<i>vyL</i>	Left side y-velocity
in	<i>vyR</i>	Right side y-velocity
in	<i>pL</i>	Left side pressure
in	<i>pR</i>	Right side pressure
out	<i>fluxLoc</i>	The local numeric flux

References gam, gam1, and gam1q.

Referenced by convectiveFlux().

#### 6.15.2.13 fluxCalculation()

```
void fluxCalculation (
    void )
```

Perform the flux calculation.

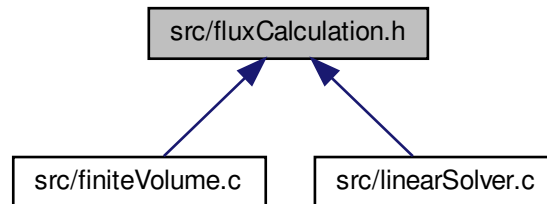
Calculation of left and right state, the velocity vector is transformed into the normal system of the cell interfaces. The function finishes with a back rotation of the velocity vector into global coordinate system.

References side\_t::baryBaryDist, side\_t::baryBaryVec, side\_t::connection, convectiveFlux(), E, side\_t::elem, side\_t::flux, side\_t::len, MX, MY, side\_t::n, NDIM, nSides, NVAR, P, side\_t::pVar, elem\_t::pVar, RHO, side, elem\_t::u\_x, elem\_t::u\_y, VX, VY, X, and Y.

Referenced by fvTimeDerivative().

## 6.16 src/fluxCalculation.h File Reference

This graph shows which files directly or indirectly include this file:



### Functions

- void `fluxCalculation` (void)  
*Perform the flux calculation.*

#### 6.16.1 Detailed Description

Author

hhh

Date

Tue 31 Mar 2020 05:18:46 PM CEST

#### 6.16.2 Function Documentation

##### 6.16.2.1 fluxCalculation()

```
void fluxCalculation (
    void )
```

Perform the flux calculation.

Calculation of left and right state, the velocity vector is transformed into the normal system of the cell interfaces. The function finishes with a back rotation of the velocity vector into global coordinate system.

References `side_t::baryBaryDist`, `side_t::baryBaryVec`, `side_t::connection`, `convectiveFlux()`, `E`, `side_t::elem`, `side_t::flux`, `side_t::len`, `MX`, `MY`, `side_t::n`, `NDIM`, `nSides`, `NVAR`, `P`, `side_t::pVar`, `elem_t::pVar`, `RHO`, `side`, `elem_t::u_x`, `elem_t::u_y`, `VX`, `VY`, `X`, and `Y`.

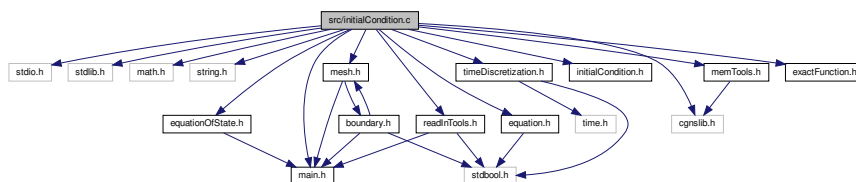
Referenced by `fvTimeDerivative()`.

## 6.17 src/initialCondition.c File Reference

Functions involving the initialization and application of initial conditions.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "main.h"
#include "readInTools.h"
#include "initialCondition.h"
#include "equation.h"
#include "memTools.h"
#include "timeDiscretization.h"
#include "mesh.h"
#include "exactFunction.h"
#include "equationOfState.h"
#include "cgnslib.h"
```

Include dependency graph for initialCondition.c:



### Functions

- void [initInitialCondition](#) (void)  
*Get initial flow conditions from the parameter file.*
- void [cgnsReadSolution](#) (void)  
*Read a solution from a CGNS file, used at restart.*
- void [setInitialCondition](#) (void)  
*Set initial flow field in all cells.*
- void [freeInitialCondition](#) (void)  
*Free all memory that was allocated for the reference state.*

### Variables

- int [icType](#)
- int [nDomains](#)
- int \* [domainID](#)
- double [rp1Dinterface](#)
- double [alpha](#)
- double \*\* [refState](#)

### 6.17.1 Detailed Description

Functions involving the initialization and application of initial conditions.

Author

hhh

Date

Fri 27 Mar 2020 02:31:24 PM CET

### 6.17.2 Variable Documentation

#### 6.17.2.1 alpha

double alpha

incident angle of the flow

Referenced by calcCoef(), GMRES\_M(), implicitTimeStep(), initBoundary(), initInitialCondition(), and matrixVector().

#### 6.17.2.2 domainID

int\* domainID

domain ID vector, from 1 to nDomains

Referenced by initInitialCondition().

#### 6.17.2.3 icType

int icType

type of initial condition

Referenced by initInitialCondition(), and setInitialCondition().

#### 6.17.2.4 nDomains

```
int nDomains
```

number of domains where initial conditions are applied

Referenced by `initInitialCondition()`, and `setInitialCondition()`.

#### 6.17.2.5 refState

```
double** refState
```

primitive variable state in domain

Referenced by `calcCoef()`, `exactFunc()`, `freeInitialCondition()`, `initInitialCondition()`, and `setInitialCondition()`.

#### 6.17.2.6 rp1Dinterface

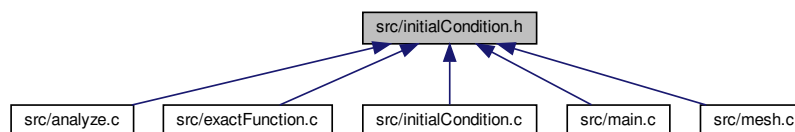
```
double rp1Dinterface
```

interface for a Riemann problem

Referenced by `exactFunc()`, and `initInitialCondition()`.

## 6.18 src/initialCondition.h File Reference

This graph shows which files directly or indirectly include this file:



## Functions

- void `initInitialCondition` (void)  
*Get initial flow conditions from the parameter file.*
- void `setInitialCondition` (void)  
*Set initial flow field in all cells.*
- void `freeInitialCondition` (void)  
*Free all memory that was allocated for the reference state.*

## Variables

- int `icType`
- int `nDomains`
- int \* `domainID`
- double `rp1Dinterface`
- double `alpha`
- double \*\* `refState`

### 6.18.1 Detailed Description

Author

hhh

Date

Fri 27 Mar 2020 02:28:25 PM CET

### 6.18.2 Variable Documentation

#### 6.18.2.1 `alpha`

`double alpha`

incident angle of the flow

Referenced by `calcCoef()`, `GMRES_M()`, `implicitTimeStep()`, `initBoundary()`, `initInitialCondition()`, and `matrixVector()`.

#### 6.18.2.2 `domainID`

`int* domainID`

domain ID vector, from 1 to `nDomains`

Referenced by `initInitialCondition()`.

#### 6.18.2.3 `icType`

`int icType`

type of initial condition

Referenced by `initInitialCondition()`, and `setInitialCondition()`.

### 6.18.2.4 nDomains

```
int nDomains
```

number of domains where initial conditions are applied

Referenced by `initInitialCondition()`, and `setInitialCondition()`.

### 6.18.2.5 refState

```
double** refState
```

primitive variable state in domain

Referenced by `calcCoef()`, `exactFunc()`, `freeInitialCondition()`, `initInitialCondition()`, and `setInitialCondition()`.

### 6.18.2.6 rp1Dinterface

```
double rp1Dinterface
```

interface for a Riemann problem

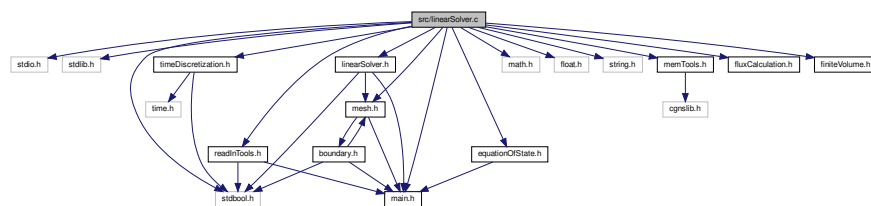
Referenced by `exactFunc()`, and `initInitialCondition()`.

## 6.19 src/linearSolver.c File Reference

Contains the functions for solving the linear system of equations during implicit calculations.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include <float.h>
#include <string.h>
#include "main.h"
#include "linearSolver.h"
#include "readInTools.h"
#include "mesh.h"
#include "timeDiscretization.h"
#include "memTools.h"
#include "fluxCalculation.h"
#include "equationOfState.h"
#include "finiteVolume.h"
```

Include dependency graph for `linearSolver.c`:





## Functions

- void `initLinearSolver` (void)  
*Initialize linear solver.*
- double `vectorDotProduct` (double \*\*A, double \*\*B)  
*Compute dot product for two state vectors A and B for every element.*
- bool `calcDinv` (double \*\*A, double \*\*Ainv)  
*Compute inverse of a 4x4 matrix.*
- void `buildMatrix` (double time, double dt)  
*Compute the global Jacobian matrix by use of finite differences.*
- void `LUSGS` (double \*\*B, double \*\*delX)  
*LUSGS preconditioner.*
- void `matrixVector` (double time, double dt, double `alpha`, double \*\*v, double \*\*res)  
*Computes matrix vector product using spatial operator and finite differences.*
- void `GMRES_M` (double time, double dt, double `alpha`, double \*\*B, double normB, double \*abortCrit, double \*\*delX)  
*Uses matrix free to solve the linear system.*
- void `freeLinearSolver` (void)  
*Free all memory that was allocated for.*

## Variables

- int `nKdim`
- int `nNewtonIter`
- int `nNewtonIterGlobal`
- int `nGMRESiterGlobal`
- int `nInnerNewton`
- int `nInnerGMRES`
- bool `usePrecond`
- double `rEps0`
- double `srEps0`
- double `eps2newton`
- double `eps2newton_sq`
- double `epsGMRES`
- double `gamEW`
- double \*\* `XK`
- double \*\* `R_XK`
- double \*\*\* `D`
- double \*\*\* `Dinv`
- double \*\* `dRdU`
- double \*\*\* `V`
- double \*\*\* `Z`
- double \*\* `R0`
- double \*\* `W`
- double \*\* `deltaXstar`

### 6.19.1 Detailed Description

Contains the functions for solving the linear system of equations during implicit calculations.

Author

hhh

Date

Sat 28 Mar 2020 03:41:56 PM CET

## 6.19.2 Function Documentation

### 6.19.2.1 buildMatrix()

```
void buildMatrix (
    double time,
    double dt )
```

Compute the global Jacobian matrix by use of finite differences.

#### Parameters

in	<i>time</i>	Computation time at calculation
in	<i>dt</i>	Time step at calculation

References calcDinv(), side\_t::connection, D, Dinv, dRdU, side\_t::elem, elem, elem\_t::firstSide, fvTimeDerivative(), elem\_t::id, nElems, side\_t::nextElemSide, NVAR, elem\_t::pVar, R\_XK, rEps0, srEps0, and elem\_t::u\_t.

Referenced by GMRES\_M().

### 6.19.2.2 calcDinv()

```
bool calcDinv (
    double ** A,
    double ** Ainv )
```

Compute inverse of a 4x4 matrix.

#### Parameters

in	<i>A</i>	The 4x4 matrix to be inverted
out	<i>Ainv</i>	The 4x4 inverse matrix of A

#### Returns

0 = Inverse does not exist, 1 = Inverse computed

Referenced by buildMatrix().

### 6.19.2.3 GMRES\_M()

```
void GMRES_M (
    double time,
```

```
double dt,
double alpha,
double ** B,
double normB,
double * abortCrit,
double ** delX )
```

Uses matrix free to solve the linear system.

#### Parameters

in	<i>time</i>	Computation time at calculation
in	<i>dt</i>	Time step at calculation
in	<i>alpha</i>	Relaxation parameter
in	<i>B</i>	Right hand side
in	<i>normB</i>	Norm of right hand side
in, out	<i>abortCrit</i>	GMRES abort criterium
out	<i>delX</i>	Resulting x vector of the linear system

References `alpha`, `buildMatrix()`, `E`, `epsGMRES`, `gam`, `LUSGS()`, `matrixVector()`, `MX`, `MY`, `nElems`, `nGMRESiter`, `Global`, `nInnerGMRES`, `nKdim`, `R0`, `RHO`, `t`, `usePrecond`, `V`, `vectorDotProduct()`, `W`, and `Z`.

Referenced by `implicitTimeStep()`.

#### 6.19.2.4 LUSGS()

```
void LUSGS (
    double ** B,
    double ** delX )
```

LUSGS preconditioner.

#### Parameters

in	<i>B</i>	Old vector, to be preconditioned
out	<i>delX</i>	Preconditioned vector

#### Note

This function is slow to execute, it might be possible to speed it up via the use of omp locks for every element

References `side_t::connection`, `deltaXstar`, `Dinv`, `dRdU`, `side_t::elem`, `elem`, `elem_t::firstSide`, `elem_t::id`, `nElems`, `side_t::nextElemSide`, and `NVAR`.

Referenced by `GMRES_M()`.

### 6.19.2.5 matrixVector()

```
void matrixVector (
    double time,
    double dt,
    double alpha,
    double ** v,
    double ** res )
```

Computes matrix vector product using spatial operator and finite differences.

#### Parameters

in	<i>time</i>	Computation time at calculation
in	<i>dt</i>	Time step at calculation
in	<i>alpha</i>	Relaxation parameter
in	<i>v</i>	Input vector for the matrix vector product
out	<i>res</i>	Resulting vector of the matrix vector product

References `alpha`, `consPrim()`, `elem_t::cVar`, `E`, `elem`, `fvTimeDerivative()`, `MX`, `MY`, `nElems`, `elem_t::pVar`, `R_XK`, `rEps0`, `RHO`, `elem_t::u_t`, `vectorDotProduct()`, and `XK`.

Referenced by `GMRES_M()`.

### 6.19.2.6 vectorDotProduct()

```
double vectorDotProduct (
    double ** A,
    double ** B )
```

Compute dot product for two state vectors A and B for every element.

#### Parameters

in	<i>A</i>	First state vector for every element
in	<i>B</i>	Second state vector for every element

#### Returns

`sum(sum(A_ij * B_ij, i = RHO,MX,MY,E), i = 1..nElems)`

References `E`, `MX`, `MY`, `nElems`, and `RHO`.

Referenced by `GMRES_M()`, `implicitTimeStep()`, and `matrixVector()`.

## 6.19.3 Variable Documentation

### 6.19.3.1 D

`double*** D`

D-Matrix of LUSGS procedure

Referenced by `buildMatrix()`, `freeLinearSolver()`, and `initLinearSolver()`.

### 6.19.3.2 deltaXstar

`double** deltaXstar`

temporary array, used in LUSGS

Referenced by `freeLinearSolver()`, `initLinearSolver()`, and `LUSGS()`.

### 6.19.3.3 Dinv

`double*** Dinv`

inverse of D-Matrix

Referenced by `buildMatrix()`, `freeLinearSolver()`, `initLinearSolver()`, and `LUSGS()`.

### 6.19.3.4 dRdU

`double** dRdU`

$dR / dU$

Referenced by `buildMatrix()`, `freeLinearSolver()`, `initLinearSolver()`, and `LUSGS()`.

### 6.19.3.5 eps2newton

`double eps2newton`

square of newton relative epsilon

Referenced by `implicitTimeStep()`, and `initLinearSolver()`.

#### 6.19.3.6 eps2newton\_sq

`double eps2newton_sq`

newton relative epsilon

Referenced by `implicitTimeStep()`, and `initLinearSolver()`.

#### 6.19.3.7 epsGMRES

`double epsGMRES`

GMRES relative epsilon

Referenced by `GMRES_M()`, and `initLinearSolver()`.

#### 6.19.3.8 gamEW

`double gamEW`

gamma parameter for Eisenstat Walker

Referenced by `implicitTimeStep()`, and `initLinearSolver()`.

#### 6.19.3.9 nGMRESiterGlobal

`int nGMRESiterGlobal`

global number of GMRES iterations

Referenced by `GMRES_M()`, `initLinearSolver()`, and `timeDisc()`.

#### 6.19.3.10 nInnerGMRES

`int nInnerGMRES`

maximum number of GMRES iterations for one stage

Referenced by `GMRES_M()`, and `initLinearSolver()`.

#### 6.19.3.11 nInnerNewton

```
int nInnerNewton
```

maximum number of Newton iterations for one stage

Referenced by `implicitTimeStep()`, and `initLinearSolver()`.

#### 6.19.3.12 nKdim

```
int nKdim
```

number Krylov spaces

Referenced by `GMRES_M()`, and `initLinearSolver()`.

#### 6.19.3.13 nNewtonIter

```
int nNewtonIter
```

maximum number of Newton iterations

Referenced by `implicitTimeStep()`, and `initLinearSolver()`.

#### 6.19.3.14 nNewtonIterGlobal

```
int nNewtonIterGlobal
```

global number of Newton iterations

Referenced by `implicitTimeStep()`, `initLinearSolver()`, and `timeDisc()`.

#### 6.19.3.15 R0

```
double** R0
```

temporary array, used in GMRES

Referenced by `freeLinearSolver()`, `GMRES_M()`, and `initLinearSolver()`.

#### 6.19.3.16 R\_XK

`double** R_XK`

residual of kth vector array

Referenced by `buildMatrix()`, `freeLinearSolver()`, `implicitTimeStep()`, `initLinearSolver()`, and `matrixVector()`.

#### 6.19.3.17 rEps0

`double rEps0`

`DBL_EPSILON`

Referenced by `buildMatrix()`, `initLinearSolver()`, and `matrixVector()`.

#### 6.19.3.18 srEps0

`double srEps0`

`sqrt(DBL_EPSILON)`

Referenced by `buildMatrix()`, and `initLinearSolver()`.

#### 6.19.3.19 usePrecond

`bool usePrecond`

use LUSGS preconditioner flag

Referenced by `freeLinearSolver()`, `GMRES_M()`, and `initLinearSolver()`.

#### 6.19.3.20 V

`double*** V`

temporary array, used in GMRES

Referenced by `freeLinearSolver()`, `GMRES_M()`, and `initLinearSolver()`.



**6.19.3.21 W**

double\*\* W

temporary array, used in GMRES

Referenced by freeLinearSolver(), GMRES\_M(), and initLinearSolver().

**6.19.3.22 XK**

double\*\* XK

kth X vector array

Referenced by freeLinearSolver(), implicitTimeStep(), initLinearSolver(), and matrixVector().

**6.19.3.23 Z**

double\*\*\* Z

temporary array, used in GMRES

Referenced by freeLinearSolver(), GMRES\_M(), and initLinearSolver().

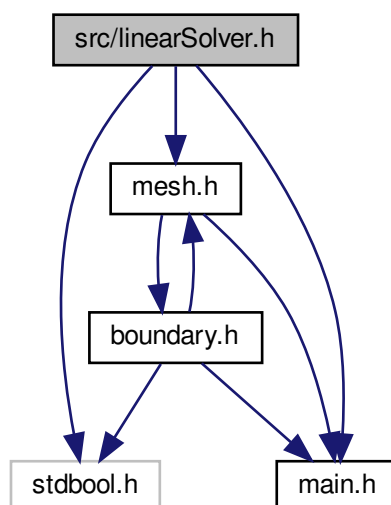
**6.20 src/linearSolver.h File Reference**

```
#include <stdbool.h>
```

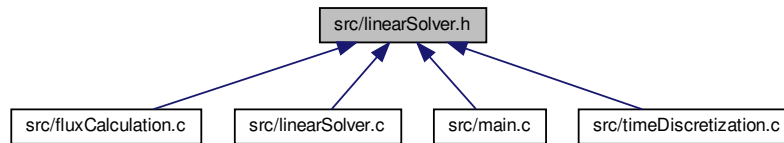
```
#include "main.h"
```

```
#include "mesh.h"
```

Include dependency graph for linearSolver.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void [initLinearSolver](#) (void)  
*Initialize linear solver.*
- double [vectorDotProduct](#) (double \*\*A, double \*\*B)  
*Compute dot product for two state vectors A and B for every element.*
- void [GMRES\\_M](#) (double time, double dt, double [alpha](#), double \*\*B, double normB, double \*abortCrit, double \*\*[deltaX](#))  
*Uses matrix free to solve the linear system.*
- void [freeLinearSolver](#) (void)  
*Free all memory that was allocated for.*

## Variables

- int [nKdim](#)
- int [nNewtonIter](#)
- int [nNewtonIterGlobal](#)
- int [nGMRESIterGlobal](#)
- int [nInnerNewton](#)
- int [nInnerGMRES](#)
- bool [usePrecond](#)
- double [rEps0](#)
- double [srEps0](#)
- double [eps2newton](#)
- double [eps2newton\\_sq](#)
- double [epsGMRES](#)
- double [gamEW](#)
- double \*\* [XK](#)
- double \*\* [R\\_XK](#)

### 6.20.1 Detailed Description

Author

hhh

Date

Sat 28 Mar 2020 03:09:46 PM CET

## 6.20.2 Function Documentation

### 6.20.2.1 GMRES\_M()

```
void GMRES_M (
    double time,
    double dt,
    double alpha,
    double ** B,
    double normB,
    double * abortCrit,
    double ** delX )
```

Uses matrix free to solve the linear system.

#### Parameters

in	<i>time</i>	Computation time at calculation
in	<i>dt</i>	Time step at calculation
in	<i>alpha</i>	Relaxation parameter
in	<i>B</i>	Right hand side
in	<i>normB</i>	Norm of right hand side
in, out	<i>abortCrit</i>	GMRES abort criterium
out	<i>delX</i>	Resulting x vector of the linear system

References `alpha`, `buildMatrix()`, `E`, `epsGMRES`, `gam`, `LUSGS()`, `matrixVector()`, `MX`, `MY`, `nElems`, `nGMRESiter`↔`Global`, `nInnerGMRES`, `nKdim`, `R0`, `RHO`, `t`, `usePrecond`, `V`, `vectorDotProduct()`, `W`, and `Z`.

Referenced by `implicitTimeStep()`.

### 6.20.2.2 vectorDotProduct()

```
double vectorDotProduct (
    double ** A,
    double ** B )
```

Compute dot product for two state vectors A and B for every element.

#### Parameters

in	<i>A</i>	First state vector for every element
in	<i>B</i>	Second state vector for every element

**Returns**

`sum(sum(A_ij * B_ij, i = RHO,MX,MY,E), i = 1..nElems)`

References E, MX, MY, nElems, and RHO.

Referenced by GMRES\_M(), implicitTimeStep(), and matrixVector().

### 6.20.3 Variable Documentation

#### 6.20.3.1 eps2newton

`double eps2newton`

square of newton relative epsilon

Referenced by implicitTimeStep(), and initLinearSolver().

#### 6.20.3.2 eps2newton\_sq

`double eps2newton_sq`

newton relative epsilon

Referenced by implicitTimeStep(), and initLinearSolver().

#### 6.20.3.3 epsGMRES

`double epsGMRES`

GMRES relative epsilon

Referenced by GMRES\_M(), and initLinearSolver().

#### 6.20.3.4 gamEW

`double gamEW`

gamma parameter for Eisenstat Walker

Referenced by implicitTimeStep(), and initLinearSolver().

### 6.20.3.5 nGMRESiterGlobal

```
int nGMRESiterGlobal
```

global number of GMRES iterations

Referenced by GMRES\_M(), initLinearSolver(), and timeDisc().

### 6.20.3.6 nInnerGMRES

```
int nInnerGMRES
```

maximum number of GMRES iterations for one stage

Referenced by GMRES\_M(), and initLinearSolver().

### 6.20.3.7 nInnerNewton

```
int nInnerNewton
```

maximum number of Newton iterations for one stage

Referenced by implicitTimeStep(), and initLinearSolver().

### 6.20.3.8 nKdim

```
int nKdim
```

number Krylov spaces

Referenced by GMRES\_M(), and initLinearSolver().

### 6.20.3.9 nNewtonIter

```
int nNewtonIter
```

maximum number of Newton iterations

Referenced by implicitTimeStep(), and initLinearSolver().

#### 6.20.3.10 nNewtonIterGlobal

```
int nNewtonIterGlobal
```

global number of Newton iterations

Referenced by `implicitTimeStep()`, `initLinearSolver()`, and `timeDisc()`.

#### 6.20.3.11 R\_XK

```
double** R_XK
```

residual of kth vector array

Referenced by `buildMatrix()`, `freeLinearSolver()`, `implicitTimeStep()`, `initLinearSolver()`, and `matrixVector()`.

#### 6.20.3.12 rEps0

```
double rEps0
```

DBL\_EPSILON

Referenced by `buildMatrix()`, `initLinearSolver()`, and `matrixVector()`.

#### 6.20.3.13 srEps0

```
double srEps0
```

`sqrt(DBL_EPSILON)`

Referenced by `buildMatrix()`, and `initLinearSolver()`.

#### 6.20.3.14 usePrecond

```
bool usePrecond
```

use LUSGS preconditioner flag

Referenced by `freeLinearSolver()`, `GMRES_M()`, and `initLinearSolver()`.

### 6.20.3.15 XK

double\*\* XK

kth X vector array

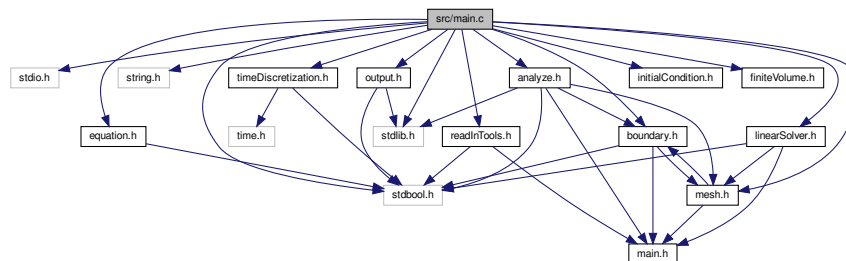
Referenced by freeLinearSolver(), implicitTimeStep(), initLinearSolver(), and matrixVector().

## 6.21 src/main.c File Reference

Contains the main function of `ccfd`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "readInTools.h"
#include "timeDiscretization.h"
#include "output.h"
#include "equation.h"
#include "boundary.h"
#include "mesh.h"
#include "initialCondition.h"
#include "finiteVolume.h"
#include "linearSolver.h"
#include "analyze.h"
```

Include dependency graph for main.c:



## Functions

- int [main](#) (int argc, char \*argv[])  
*Main function.*

### 6.21.1 Detailed Description

Contains the main function of `ccfd`

Author

hhh

Date

Sat 21 Mar 2020 10:41:51 AM CET

## 6.21.2 Function Documentation

### 6.21.2.1 main()

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

Main function.

This function starts the program by initializing all the necessary global variables and initializing the time discretization loop. The function finishes by deallocating all the allocated memory.

#### Parameters

in	<i>argc</i>	The number of command line arguments passed to <code>main</code>
in	<i>argv</i>	The argument vector containing the command line arguments

#### Returns

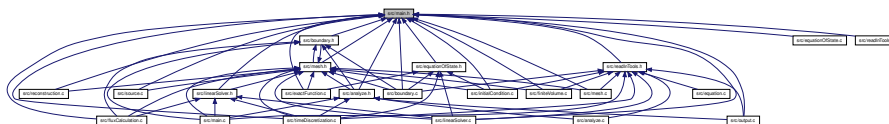
0 = Success, 1 = Error during execution

References `fillCmds()`, `freeAnalyze()`, `freeBoundary()`, `freeInitialCondition()`, `freeLinearSolver()`, `freeMesh()`, `freeOutputTimes()`, `getBool()`, `ignoredCmds()`, `initerationNumber`, `initAnalyze()`, `initBoundary()`, `initEquation()`, `initFV()`, `initInitialCondition()`, `initLinearSolver()`, `initMesh()`, `initOutput()`, `initTimeDisc()`, `isRestart`, `isStationary`, `outputTimes`, `restartTime`, `setInitialCondition()`, `startTime`, `strIniCondFile`, and `timeDisc()`.

## 6.22 src/main.h File Reference

Contains the global constants and definitions.

This graph shows which files directly or indirectly include this file:



### Macros

- `#define STRLEN 256`



## Enumerations

- enum `conservativeVariables` {  
`RHO`,  
`MX`,  
`MY`,  
`E` }

*Index aliases for the conservative variables vector.*

- enum `primitiveVariables` {  
`VX` = 1,  
`VY`,  
`P` }

*Index aliases for the primitive variables vector.*

- enum `directions` {  
`X`,  
`Y` }

*Index aliases for the x- and y-components of a vector.*

- enum `boundaryConditionType` {  
`SLIPWALL` = 1,  
`WALL`,  
`INFLOW`,  
`OUTFLOW`,  
`CHARACTERISTIC`,  
`EXACTSOL`,  
`PERIODIC`,  
`PRESSURE_OUT` }

*Aliases for the different boundary condition types.*

- enum `cartesianMeshSides` {  
`BOTTOM`,  
`RIGHT`,  
`TOP`,  
`LEFT` }

*Aliases for the sides of a cartesian mesh.*

- enum `meshType` {  
`UNSTRUCTURED`,  
`CARTESIAN` }

*Flag for the mesh type.*

- enum `fluxFunction` {  
`GOD` = 1,  
`ROE`,  
`HLL`,  
`HLLC`,  
`HLLC`,  
`LXF`,  
`STW`,  
`CEN`,  
`AUSMD`,  
`AUSMDV`,  
`VANLEER` }

*Aliases for the different flux functions.*

- enum `limiterFunction` {  
`BARTHJESPERSEN` = 1,  
`VENKATAKRISHNAN` }

*Flag for the limiter function.*

- enum `generalParameters` {  
`NDIM = 2,`  
`NVAR = 4,`  
`NBC = 20 }`  
*General parameters for the Program.*
- enum `ioFormat` {  
`CGNS = 1,`  
`CURVE,`  
`CSV }`  
*Output format for the results.*
- enum `clcdResiduals` {  
`CL = 4,`  
`CD }`  
*Index aliases for the residual vector.*

### 6.22.1 Detailed Description

Contains the global constants and definitions.

Author

hhh

Date

Sat 21 Mar 2020 10:44:50 AM CET

### 6.22.2 Macro Definition Documentation

#### 6.22.2.1 STRLEN

```
#define STRLEN 256
```

string length

### 6.22.3 Enumeration Type Documentation

#### 6.22.3.1 boundaryConditionType

```
enum boundaryConditionType
```

Aliases for the different boundary condition types.

## Enumerator

SLIPWALL	slip wall, or Euler wall
WALL	no slip wall, or Navier-Stokes wall
INFLOW	supersonic inflow
OUTFLOW	supersonic outflow
CHARACTERISTIC	characteristic BC, or subsonic inflow
EXACTSOL	exact solution
PERIODIC	periodic BC
PRESSURE_OUT	subsonic pressure outflow

### 6.22.3.2 cartesianMeshSides

enum `cartesianMeshSides`

Aliases for the sides of a cartesian mesh.

## Enumerator

BOTTOM	bottom side
RIGHT	right side
TOP	top side
LEFT	left side

### 6.22.3.3 clcdResiduals

enum `clcdResiduals`

Index aliases for the residual vector.

## Enumerator

CL	lift coefficient
CD	drag coefficient

### 6.22.3.4 conservativeVariables

enum `conservativeVariables`

Index aliases for the conservative variables vector.

**Enumerator**

RHO	density
MX	momentum in x-direction
MY	momentum in y-direction
E	energy

**6.22.3.5 directions**

enum `directions`

Index aliases for the x- and y-components of a vector.

**Enumerator**

X	x-direction
Y	y-direction

**6.22.3.6 fluxFunction**

enum `fluxFunction`

Aliases for the different flux functions.

**Enumerator**

GOD	Godunov flux function
ROE	Roe flux function
HLL	HLL flux function
HLLE	HLLE flux function
HLLC	HLLC flux function
LXF	Lax-Friedrichs flux function
STW	Steger-Warming flux function
CEN	central flux function
AUSMD	AUSMD flux function
AUSMDV	AUSMDV flux function
VANLEER	Van Leer flux function

**6.22.3.7 generalParameters**

enum `generalParameters`

General parameters for the Program.

**Enumerator**

NDIM	number of dimensions, cannot be changed
NVAR	number of variables, primitive or conservative
NBC	maximum number of boundary conditions

**6.22.3.8 ioFormat**

enum `ioFormat`

Output format for the results.

**Enumerator**

CGNS	.CGNS file format
CURVE	.curve file format
CSV	.csv file format

**6.22.3.9 limiterFunction**

enum `limiterFunction`

Flag for the limiter function.

**Enumerator**

BARTHJESPERSEN	Barth & Jespersen limiter
VENKATAKRISHNAN	Venkatakrishnan limiter

**6.22.3.10 meshType**

enum `meshType`

Flag for the mesh type.

**Enumerator**

UNSTRUCTURED	unstructured
CARTESIAN	cartesian

### 6.22.3.11 primitiveVariables

enum `primitiveVariables`

Index aliases for the primitive variables vector.

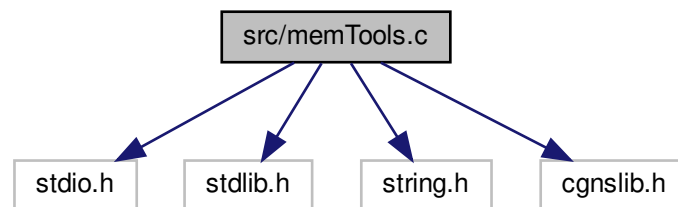
Enumerator

VX	velocity in x-direction
VY	velocity in y-direction
P	pressure

## 6.23 src/memTools.c File Reference

Memory management functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cgnslib.h"
Include dependency graph for memTools.c:
```



### Functions

- long \*\* `dyn2DintArray` (long I, long J)  
*Allocate a dynamic 2D array of integers.*
- cgsize\_t \*\* `dyn2DcgsizeArray` (long I, long J)  
*Allocate a dynamic 2D array of cgsize\_t.*
- double \*\* `dyn2DdblArray` (long I, long J)  
*Allocate a dynamic 2D array of doubles.*
- long \*\*\* `dyn3DintArray` (long I, long J, long K)  
*Allocate a dynamic 3D array of integers.*

- double \*\*\* [dyn3DdblArray](#) (long I, long J, long K)  
*Allocate a dynamic 3D array of doubles.*
- double \*\*\*\* [dyn4DdblArray](#) (long I, long J, long K, long L)  
*Allocate a dynamic 4D array of doubles.*
- char \*\* [dynStringArray](#) (long I, long J)  
*Allocate a dynamic array of strings.*

### 6.23.1 Detailed Description

Memory management functions.

#### Author

hhh

#### Date

Fri 27 Mar 2020 02:38:03 PM CET

#### Note

Manual Memory Management:

"The manual type involves malloc and free, and is where most of your segfaults happen. This memory model is why Jesus weeps when he has to code in C."

- Ben Klemens

### 6.23.2 Function Documentation

#### 6.23.2.1 dyn2DcgsizeArray()

```
cgsize_t** dyn2DcgsizeArray (
    long I,
    long J )
```

Allocate a dynamic 2D array of cgsize\_t.

#### Parameters

in	<i>I</i>	Number of elements in the first dimension
in	<i>J</i>	Number of elements in the second dimension

#### Returns

Pointer to a 2D cgsize\_t array

Referenced by cgnsWriteMesh().



**6.23.2.2 dyn2DdblArray()**

```
double** dyn2DdblArray (
    long I,
    long J )
```

Allocate a dynamic 2D array of doubles.

**Parameters**

in	<i>I</i>	Number of elements in the first dimension
in	<i>J</i>	Number of elements in the second dimension

**Returns**

Pointer to a 2D double array

Referenced by cgnsWriteMesh(), createCartMesh(), createReconstructionInfo(), csvOutput(), curveOutput(), initInitialCondition(), initLinearSolver(), initRecordPoints(), initTimeDisc(), readCGNS(), readEMC2(), and readGmsh().

**6.23.2.3 dyn2DintArray()**

```
long** dyn2DintArray (
    long I,
    long J )
```

Allocate a dynamic 2D array of integers.

**Parameters**

in	<i>I</i>	Number of elements in the first dimension
in	<i>J</i>	Number of elements in the second dimension

**Returns**

Pointer to a 2D integer array

Referenced by createCartMesh(), readCGNS(), readEMC2(), and readGmsh().

**6.23.2.4 dyn3DdblArray()**

```
double*** dyn3DdblArray (
    long I,
```

```
long J,  
long K )
```

Allocate a dynamic 3D array of doubles.

**Parameters**

in	<i>I</i>	Number of elements in the first dimension
in	<i>J</i>	Number of elements in the second dimension
in	<i>K</i>	Number of elements in the third dimension

**Returns**

Pointer to a 3D double array

Referenced by `initLinearSolver()`.

**6.23.2.5 dyn3DintArray()**

```
long*** dyn3DintArray (
    long I,
    long J,
    long K )
```

Allocate a dynamic 3D array of integers.

**Parameters**

in	<i>I</i>	Number of elements in the first dimension
in	<i>J</i>	Number of elements in the second dimension
in	<i>K</i>	Number of elements in the third dimension

**Returns**

Pointer to a 3D integer array

**6.23.2.6 dyn4DdblArray()**

```
double**** dyn4DdblArray (
    long I,
    long J,
    long K,
    long L )
```

Allocate a dynamic 4D array of doubles.

**Parameters**

in	<i>I</i>	Number of elements in the first dimension
in	<i>J</i>	Number of elements in the second dimension
in	<i>K</i>	Number of elements in the third dimension
in	<i>L</i>	Number of elements in the fourth dimension

**Returns**

Pointer to a 4D double array

**6.23.2.7 dynStringArray()**

```
char** dynStringArray (
    long I,
    long J )
```

Allocate a dynamic array of strings.

**Parameters**

in	I	Number of elements in the first dimension
in	J	String length of each element

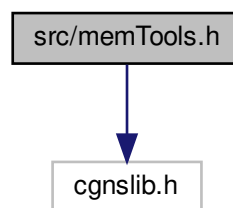
**Returns**

Pointer to a string array

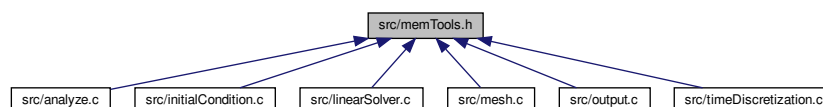
**6.24 src/memTools.h File Reference**

```
#include "cgnslib.h"
```

Include dependency graph for memTools.h:



This graph shows which files directly or indirectly include this file:



## Functions

- long \*\* [dyn2DintArray](#) (long I, long J)  
*Allocate a dynamic 2D array of integers.*
- csize\_t \*\* [dyn2DcsizeArray](#) (long I, long J)  
*Allocate a dynamic 2D array of csize\_t.*
- double \*\* [dyn2DdblArray](#) (long I, long J)  
*Allocate a dynamic 2D array of doubles.*
- long \*\*\* [dyn3DintArray](#) (long I, long J, long K)  
*Allocate a dynamic 3D array of integers.*
- double \*\*\* [dyn3DdblArray](#) (long I, long J, long K)  
*Allocate a dynamic 3D array of doubles.*
- double \*\*\*\* [dyn4DdblArray](#) (long I, long J, long K, long L)  
*Allocate a dynamic 4D array of doubles.*
- char \*\* [dynStringArray](#) (long I, long J)  
*Allocate a dynamic array of strings.*

### 6.24.1 Detailed Description

#### Author

hhh

#### Date

Fri 27 Mar 2020 02:38:45 PM CET

### 6.24.2 Function Documentation

#### 6.24.2.1 dyn2DcsizeArray()

```
csize_t** dyn2DcsizeArray (
    long I,
    long J )
```

Allocate a dynamic 2D array of csize\_t.

#### Parameters

in	<i>I</i>	Number of elements in the first dimension
in	<i>J</i>	Number of elements in the second dimension

#### Returns

Pointer to a 2D csize\_t array

Referenced by `cgnsWriteMesh()`.

### 6.24.2.2 dyn2DdblArray()

```
double** dyn2DdblArray (
    long I,
    long J )
```

Allocate a dynamic 2D array of doubles.

#### Parameters

in	<i>I</i>	Number of elements in the first dimension
in	<i>J</i>	Number of elements in the second dimension

#### Returns

Pointer to a 2D double array

Referenced by cgnsWriteMesh(), createCartMesh(), createReconstructionInfo(), csvOutput(), curveOutput(), initInitialCondition(), initLinearSolver(), initRecordPoints(), initTimeDisc(), readCGNS(), readEMC2(), and readGmsh().

### 6.24.2.3 dyn2DintArray()

```
long** dyn2DintArray (
    long I,
    long J )
```

Allocate a dynamic 2D array of integers.

#### Parameters

in	<i>I</i>	Number of elements in the first dimension
in	<i>J</i>	Number of elements in the second dimension

#### Returns

Pointer to a 2D integer array

Referenced by createCartMesh(), readCGNS(), readEMC2(), and readGmsh().

### 6.24.2.4 dyn3DdblArray()

```
double*** dyn3DdblArray (
    long I,
```

```
long J,  
long K )
```

Allocate a dynamic 3D array of doubles.

**Parameters**

in	$I$	Number of elements in the first dimension
in	$J$	Number of elements in the second dimension
in	$K$	Number of elements in the third dimension

**Returns**

Pointer to a 3D double array

Referenced by `initLinearSolver()`.

**6.24.2.5 dyn3DintArray()**

```
long*** dyn3DintArray (
    long  $I$ ,
    long  $J$ ,
    long  $K$  )
```

Allocate a dynamic 3D array of integers.

**Parameters**

in	$I$	Number of elements in the first dimension
in	$J$	Number of elements in the second dimension
in	$K$	Number of elements in the third dimension

**Returns**

Pointer to a 3D integer array

**6.24.2.6 dyn4DdblArray()**

```
double**** dyn4DdblArray (
    long  $I$ ,
    long  $J$ ,
    long  $K$ ,
    long  $L$  )
```

Allocate a dynamic 4D array of doubles.

**Parameters**

in	$I$	Number of elements in the first dimension
in	$J$	Number of elements in the second dimension
in	$K$	Number of elements in the third dimension
in	$L$	Number of elements in the fourth dimension





## Data Structures

- struct [sideList\\_t](#)

*Helper structure for reading in the sides and deviding them into BC sides and non-BC sides.*

## Functions

- void [createReconstructionInfo](#) ([elem\\_t](#) \*aElem)  
*Compute required vectors for reconstruction.*
- void [connectPeriodicBC](#) (void)  
*Create connection for periodic BCs.*
- void [createSideInfo](#) ([side\\_t](#) \*aSide)  
*Create side info: normal vector, side length, and ghost cells.*
- void [createElemInfo](#) ([elem\\_t](#) \*aElem)  
*Compute the cell specific values: barycenter, area, projection length of an element.*
- int [compare](#) (const void \*a, const void \*b)  
*Compare two elements of the `sideList` list, used for sorting the list.*
- void [createCartMesh](#) (double \*\*\*vertex, long \*nVertices, long \*\*\*BCedge, long \*nBCedges, long \*\*\*quad)  
*Create a cartesian mesh.*
- void [readGmsh](#) (char fileName[STRLEN], double \*\*\*vertex, long \*nVertices, long \*\*\*BCedge, long \*nBCedges, long \*\*\*tria, long \*\*\*quad)  
*Read in a gmsh mesh file.*
- void [readEMC2](#) (char fileName[STRLEN], double \*\*\*vertex, long \*nVertices, long \*\*\*BCedge, long \*nBCedges, long \*\*\*tria, long \*\*\*quad)  
*Read in EMC2 mesh file.*
- void [readCGNS](#) (char fileName[STRLEN], double \*\*\*vertex, long \*nVertices, long \*\*\*BCedge, long \*nBCedges, long \*\*\*tria, long \*\*\*quad)  
*Read in a CGNS mesh.*
- void [createMesh](#) (void)  
*Create a cartesian or structured mesh.*
- void [readMesh](#) (void)  
*Read in and store the mesh parameters from the parameter file.*
- void [initMesh](#) (void)  
*Initialize the mesh and call `readMesh`*
- void [freeMesh](#) (void)  
*Free all allocated memory of the mesh.*

## Variables

- char [parameterFile](#) [STRLEN]
- char [strMeshFormat](#) [STRLEN]
- char [strMeshFile](#) [STRLEN]
- char [strIniCondFile](#) [STRLEN]
- [cartMesh\\_t](#) cartMesh
- char [gridFile](#) [STRLEN]
- int [meshType](#)
- int [meshFormat](#)
- long [nNodes](#)
- long [nElems](#)
- long [nTrias](#)
- long [nQuads](#)

- long `nSides`
- long `nBCsides`
- long `nInnerSides`
- double `totalArea_q`
- double `xMin`
- double `xMax`
- double `yMin`
- double `yMax`
- double `dxRef`
- `elem_t` \* `elem`
- `side_t` \* `side`
- `side_t` \* `BCside`
- `elem_t` \* `firstElem`
- `node_t` \* `firstNode`
- `side_t` \* `firstSide`
- `sidePtr_t` \* `firstBCside`

### 6.25.1 Detailed Description

Contains all the functions for reading and creating meshes.

#### Author

hhh

#### Date

Tue 24 Mar 2020 12:45:57 PM CET

### 6.25.2 Function Documentation

#### 6.25.2.1 `compare()`

```
int compare (
    const void * a,
    const void * b )
```

Compare two elements of the `sideList` list, used for sorting the list.

Sort `sideList` array in order to retrieve the connectivity info more efficiently. Basically we want to sort the sides on three different levels: first, by their first node ID, secondly, by their second node ID, and thirdly by if they are a boundary condition side. The result might look something like this: 3 3 1 1 2 0 2 1 1 1 3 0 1 2 0 --> 2 1 1 3 1 0 2 2 1 2 2 1 3 3 0 3 3 0 3 3 1

#### Parameters

in	<i>a</i>	Pointer an element of the side list
in	<i>b</i>	Pointer an element of the side list

**Returns**

The difference of a and b, if the values are the same, the move to next sorting metric

References sideList\_t::BC, and sideList\_t::node.

Referenced by createMesh().

**6.25.2.2 createCartMesh()**

```
void createCartMesh (
    double *** vertex,
    long * nVertices,
    long *** BCedge,
    long * nBCedges,
    long *** quad )
```

Create a cartesian mesh.

**Parameters**

in, out	<i>vertex</i>	Pointer to 2D array, used for the vertices
in, out	<i>nVertices</i>	Pointer to the number of total vertices in <i>vertex</i>
in, out	<i>BCedge</i>	Pointer to 2D array, used for the BC edges
in, out	<i>nBCedges</i>	Pointer to the number of total BC edges
in, out	<i>quad</i>	Pointer to a 2D array, used for the quadrangles

References cartMesh\_t::BCrange, cartMesh\_t::BCtype, BOTTOM, cartMesh, dyn2DdblArray(), dyn2DintArray(), cartMesh\_t::iMax, cartMesh\_t::jMax, LEFT, cartMesh\_t::nBC, nQuads, RIGHT, TOP, X, xMax, xMin, Y, yMax, and yMin.

Referenced by createMesh().

**6.25.2.3 createElemInfo()**

```
void createElemInfo (
    elem_t * aElem )
```

Compute the cell specific values: barycenter, area, projection length of an element.

**Parameters**

in	<i>aElem</i>	A pointer to an element
----	--------------	-------------------------

References elem\_t::area, elem\_t::areaq, elem\_t::bary, elem\_t::elemType, elem\_t::node, elem\_t::sx, elem\_t::sy, totalArea\_q, node\_t::x, X, and Y.

Referenced by createMesh().

#### 6.25.2.4 createMesh()

```
void createMesh (
    void )
```

Create a cartesian or structured mesh.

Read in of all supported mesh types:

- \*.msh
- \*.msh2
- \*.msh4
- \*.emc2
- \*.cgns

References elem\_t::bary, side\_t::BC, sideList\_t::BC, boundary\_t::BCid, BCside, boundary\_t::BCtype, CARTESIAN, compare(), side\_t::connection, connectPeriodicBC(), createCartMesh(), createElemInfo(), createReconstructionInfo(), createSideInfo(), elem\_t::domain, side\_t::elem, elem, elem\_t::elemType, firstBC, firstBCside, firstElem, firstNode, firstSide, elem\_t::firstSide, side\_t::GP, node\_t::id, side\_t::id, elem\_t::id, sideList\_t::isRotated, side\_t::n, nBCsides, nElems, node\_t::next, boundary\_t::next, side\_t::next, sidePtr\_t::next, elem\_t::next, side\_t::nextElemSide, nInnerSides, nNodes, side\_t::node, sideList\_t::node, elem\_t::node, nQuads, nSides, nTrias, PERIODIC, readCGNS(), readEMC2(), readGmsh(), side, sidePtr\_t::side, sideList\_t::side, strMeshFile, strMeshFormat, totalArea\_q, UNSTRUCTURED, node\_t::x, X, xMax, xMin, Y, yMax, and yMin.

Referenced by initMesh().

#### 6.25.2.5 createReconstructionInfo()

```
void createReconstructionInfo (
    elem_t * aElem )
```

Compute required vectors for reconstruction.

##### Parameters

in	<i>aElem</i>	A pointer to an element
----	--------------	-------------------------

References elem\_t::area, elem\_t::bary, side\_t::baryBaryDist, side\_t::baryBaryVec, side\_t::connection, dyn2DdblArray(), elem\_t::elemType, elem\_t::firstSide, side\_t::GP, NDIM, side\_t::nextElemSide, elem\_t::nGP, elem\_t::node, side\_t::w, elem\_t::wGP, node\_t::x, X, elem\_t::xGP, and Y.

Referenced by createMesh().

### 6.25.2.6 createSideInfo()

```
void createSideInfo (
    side_t * aSide )
```

Create side info: normal vector, side length, and ghost cells.

#### Parameters

in	<i>aSide</i>	A pointer to a side
----	--------------	---------------------

References `elem_t::bary`, `side_t::BC`, `side_t::connection`, `side_t::elem`, `side_t::GP`, `side_t::len`, `side_t::n`, `NDIM`, `side_t::node`, `node_t::x`, `X`, and `Y`.

Referenced by `createMesh()`.

### 6.25.2.7 readCGNS()

```
void readCGNS (
    char fileName[STRLEN],
    double *** vertex,
    long * nVertices,
    long *** BCedge,
    long * nBCedges,
    long *** tria,
    long *** quad )
```

Read in a CGNS mesh.

#### Parameters

in	<i>fileName</i>	Name of the mesh file
in, out	<i>vertex</i>	Pointer to 2D array, used for the vertices
in, out	<i>nVertices</i>	Pointer to the number of total vertices in <code>vertex</code>
in, out	<i>BCedge</i>	Pointer to 2D array, used for the BC edges
in, out	<i>nBCedges</i>	Pointer to the number of total BC edges
in, out	<i>tria</i>	Pointer to a 2D array, used for the triangles
in, out	<i>quad</i>	Pointer to a 2D array, used for the quadrangles

References `dyn2DdblArray()`, `dyn2DintArray()`, `nElems`, `nQuads`, `nTrias`, `X`, and `Y`.

Referenced by `createMesh()`.

### 6.25.2.8 readEMC2()

```
void readEMC2 (
    char fileName[STRLEN],
```

```

double *** vertex,
long * nVertices,
long *** BCedge,
long * nBCedges,
long *** tria,
long *** quad )

```

Read in EMC2 mesh file.

#### Parameters

in	<i>fileName</i>	Name of the mesh file
in, out	<i>vertex</i>	Pointer to 2D array, used for the vertices
in, out	<i>nVertices</i>	Pointer to the number of total vertices in <i>vertex</i>
in, out	<i>BCedge</i>	Pointer to 2D array, used for the BC edges
in, out	<i>nBCedges</i>	Pointer to the number of total BC edges
in, out	<i>tria</i>	Pointer to a 2D array, used for the triangles
in, out	<i>quad</i>	Pointer to a 2D array, used for the quadrangles

References `dyn2DdblArray()`, `dyn2DintArray()`, `nQuads`, `nTrias`, `STRLEN`, `X`, and `Y`.

Referenced by `createMesh()`.

#### 6.25.2.9 readGmsh()

```

void readGmsh (
    char fileName[STRLEN],
    double *** vertex,
    long * nVertices,
    long *** BCedge,
    long * nBCedges,
    long *** tria,
    long *** quad )

```

Read in a gmsh mesh file.

#### Parameters

in	<i>fileName</i>	Name of the mesh file
in, out	<i>vertex</i>	Pointer to 2D array, used for the vertices
in, out	<i>nVertices</i>	Pointer to the number of total vertices in <i>vertex</i>
in, out	<i>BCedge</i>	Pointer to 2D array, used for the BC edges
in, out	<i>nBCedges</i>	Pointer to the number of total BC edges
in, out	<i>tria</i>	Pointer to a 2D array, used for the triangles
in, out	<i>quad</i>	Pointer to a 2D array, used for the quadrangles

References `dyn2DdblArray()`, `dyn2DintArray()`, `nQuads`, `nTrias`, `STRLEN`, `X`, and `Y`.

Referenced by `createMesh()`.

### 6.25.3 Variable Documentation

#### 6.25.3.1 BCside

```
side_t** BCside
```

global BC side pointer array

Referenced by createMesh(), freeBoundary(), setBCatBarys(), and setBCatSides().

#### 6.25.3.2 cartMesh

```
cartMesh_t cartMesh
```

cartesian mesh structure

Referenced by createCartMesh(), and readMesh().

#### 6.25.3.3 dxRef

```
double dxRef
```

reference x length

Referenced by exactFunc(), and initMesh().

#### 6.25.3.4 elem

```
elem_t** elem
```

global element pointer array

Referenced by buildMatrix(), calcErrors(), calcSource(), calcTimeStep(), createMesh(), explicitTimeStepEuler(), explicitTimeStepRK(), freeMesh(), fvTimeDerivative(), globalResidual(), implicitTimeStep(), initFV(), LUSGS(), matrixVector(), and spatialReconstruction().



### 6.25.3.5 firstBCside

```
sidePtr_t* firstBCside
```

pointer to first BC side

Referenced by `cgnsWriteMesh()`, `connectPeriodicBC()`, `createMesh()`, `freeMesh()`, and `initWing()`.

### 6.25.3.6 firstElem

```
elem_t* firstElem
```

pointer to first element

Referenced by `analyze()`, `cgnsOutput()`, `cgnsReadSolution()`, `cgnsWriteMesh()`, `createMesh()`, `csvOutput()`, `curve↔Output()`, `initRecordPoints()`, and `setInitialCondition()`.

### 6.25.3.7 firstNode

```
node_t* firstNode
```

pointer to first node

Referenced by `cgnsWriteMesh()`, `createMesh()`, and `freeMesh()`.

### 6.25.3.8 firstSide

```
side_t* firstSide
```

pointer to first side

Referenced by `connectPeriodicBC()`, and `createMesh()`.

### 6.25.3.9 gridFile

```
char gridFile[STRLEN]
```

complete name of the output mesh file

Referenced by `cgnsFinalizeOutput()`, `cgnsOutput()`, `cgnsWriteMesh()`, and `initMesh()`.

#### 6.25.3.10 meshFormat

```
int meshFormat
```

code for the mesh format

#### 6.25.3.11 meshType

```
int meshType
```

code for the mesh type

#### 6.25.3.12 nBCsides

```
long nBCsides
```

global number of BC sides

Referenced by `cgnsWriteMesh()`, `createMesh()`, `setBCatBarys()`, and `setBCatSides()`.

#### 6.25.3.13 nElems

```
long nElems
```

global number of elements

Referenced by `buildMatrix()`, `calcErrors()`, `calcSource()`, `calcTimeStep()`, `cgnsFinalizeOutput()`, `cgnsOutput()`, `cgnsReadSolution()`, `cgnsWriteMesh()`, `createMesh()`, `csvOutput()`, `curveOutput()`, `explicitTimeStepEuler()`, `explicitTimeStepRK()`, `freeMesh()`, `fvTimeDerivative()`, `globalResidual()`, `GMRES_M()`, `implicitTimeStep()`, `initFV()`, `initLinearSolver()`, `initMesh()`, `initTimeDisc()`, `LUSGS()`, `matrixVector()`, `readCGNS()`, `spatialReconstruction()`, and `vectorDotProduct()`.

#### 6.25.3.14 nInnerSides

```
long nInnerSides
```

global number of non BC sides

Referenced by `createMesh()`.

### 6.25.3.15 nNodes

```
long nNodes
```

global number of nodes

Referenced by `cgnsFinalizeOutput()`, `cgnsOutput()`, `cgnsWriteMesh()`, and `createMesh()`.

### 6.25.3.16 nQuads

```
long nQuads
```

global number of quadrangles

Referenced by `cgnsFinalizeOutput()`, `cgnsOutput()`, `cgnsWriteMesh()`, `createCartMesh()`, `createMesh()`, `readCGNS()`, `readEMC2()`, and `readGmsh()`.

### 6.25.3.17 nSides

```
long nSides
```

global number of sides

Referenced by `connectPeriodicBC()`, `createMesh()`, and `fluxCalculation()`.

### 6.25.3.18 nTrias

```
long nTrias
```

global number of triangles

Referenced by `cgnsFinalizeOutput()`, `cgnsOutput()`, `cgnsWriteMesh()`, `createMesh()`, `readCGNS()`, `readEMC2()`, and `readGmsh()`.

### 6.25.3.19 parameterFile

```
char parameterFile[STRLEN]
```

parameter file name

### 6.25.3.20 side

```
side_t** side
```

global side pointer array

Referenced by createMesh(), fluxCalculation(), and freeMesh().

### 6.25.3.21 strIniCondFile

```
char strIniCondFile[STRLEN]
```

file name of the initial conditions file

Referenced by cgnsReadSolution(), and main().

### 6.25.3.22 strMeshFile

```
char strMeshFile[STRLEN]
```

mesh file base name

Referenced by createMesh(), and readMesh().

### 6.25.3.23 strMeshFormat

```
char strMeshFormat[STRLEN]
```

mesh format string

Referenced by createMesh(), and readMesh().

### 6.25.3.24 totalArea\_q

```
double totalArea_q
```

inverse of the global area of the mesh

Referenced by calcErrors(), createElemInfo(), createMesh(), globalResidual(), and initMesh().

### 6.25.3.25 xMax

```
double xMax
```

minimum x-direction extension

Referenced by `createCartMesh()`, `createMesh()`, `exactFunc()`, and `readMesh()`.

### 6.25.3.26 xMin

```
double xMin
```

maximum x-direction extension

Referenced by `createCartMesh()`, `createMesh()`, `exactFunc()`, and `readMesh()`.

### 6.25.3.27 yMax

```
double yMax
```

minimum y-direction extension

Referenced by `createCartMesh()`, `createMesh()`, `exactFunc()`, and `readMesh()`.

### 6.25.3.28 yMin

```
double yMin
```

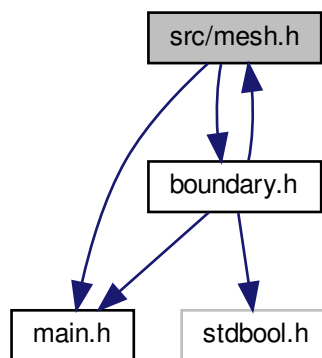
maximum y-direction extension

Referenced by `createCartMesh()`, `createMesh()`, `exactFunc()`, and `readMesh()`.

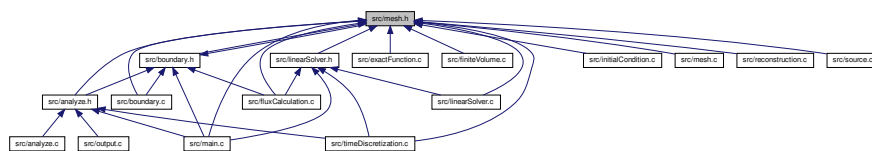
## 6.26 src/mesh.h File Reference

Contains the definitions of all structs for the mesh handling.

```
#include "main.h"
#include "boundary.h"
Include dependency graph for mesh.h:
```



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `node_t`  
Structure for a single node in a linked list of nodes.
- struct `side_t`  
Structure for a single side in the global side list.
- struct `sidePtr_t`  
Secondary side lists used for various things.
- struct `elem_t`  
Structure for a single element in the global element list.
- struct `cartMesh_t`  
Structure holding the information for a cartesian mesh.

## Functions

- void `initMesh` (void)  
*Initialize the mesh and call `readMesh`*
- void `freeMesh` (void)  
*Free all allocated memory of the mesh.*

## Variables

- char `parameterFile` [STRLEN]
- char `strMeshFormat` [STRLEN]
- char `strMeshFile` [STRLEN]
- char `strIniCondFile` [STRLEN]
- `cartMesh_t` `cartMesh`
- char `gridFile` [STRLEN]
- int `meshType`
- int `meshFormat`
- long `nNodes`
- long `nElems`
- long `nTrias`
- long `nQuads`
- long `nSides`
- long `nBCsides`
- long `nInnerSides`
- double `totalArea_q`
- double `xMin`
- double `xMax`
- double `yMin`
- double `yMax`
- double `dxRef`
- `elem_t` \*\* `elem`
- `side_t` \*\* `side`
- `side_t` \*\* `BCside`
- `elem_t` \* `firstElem`
- `node_t` \* `firstNode`
- `side_t` \* `firstSide`
- `sidePtr_t` \* `firstBCside`

### 6.26.1 Detailed Description

Contains the definitions of all structs for the mesh handling.

Author

hhh

Date

Tue 24 Mar 2020 12:45:49 PM CET

## 6.26.2 Variable Documentation

### 6.26.2.1 BCside

```
side_t** BCside
```

global BC side pointer array

Referenced by createMesh(), freeBoundary(), setBCatBarys(), and setBCatSides().

### 6.26.2.2 cartMesh

```
cartMesh_t cartMesh
```

cartesian mesh structure

Referenced by createCartMesh(), and readMesh().

### 6.26.2.3 dxRef

```
double dxRef
```

reference x length

Referenced by exactFunc(), and initMesh().

### 6.26.2.4 elem

```
elem_t** elem
```

global element pointer array

Referenced by buildMatrix(), calcErrors(), calcSource(), calcTimeStep(), createMesh(), explicitTimeStepEuler(), explicitTimeStepRK(), freeMesh(), fvTimeDerivative(), globalResidual(), implicitTimeStep(), initFV(), LUSGS(), matrixVector(), and spatialReconstruction().



### 6.26.2.5 firstBCside

```
sidePtr_t* firstBCside
```

pointer to first BC side

Referenced by `cgnsWriteMesh()`, `connectPeriodicBC()`, `createMesh()`, `freeMesh()`, and `initWing()`.

### 6.26.2.6 firstElem

```
elem_t* firstElem
```

pointer to first element

Referenced by `analyze()`, `cgnsOutput()`, `cgnsReadSolution()`, `cgnsWriteMesh()`, `createMesh()`, `csvOutput()`, `curve↔Output()`, `initRecordPoints()`, and `setInitialCondition()`.

### 6.26.2.7 firstNode

```
node_t* firstNode
```

pointer to first node

Referenced by `cgnsWriteMesh()`, `createMesh()`, and `freeMesh()`.

### 6.26.2.8 firstSide

```
side_t* firstSide
```

pointer to first side

Referenced by `connectPeriodicBC()`, and `createMesh()`.

### 6.26.2.9 gridFile

```
char gridFile[STRLEN]
```

complete name of the output mesh file

Referenced by `cgnsFinalizeOutput()`, `cgnsOutput()`, `cgnsWriteMesh()`, and `initMesh()`.

#### 6.26.2.10 meshFormat

```
int meshFormat
```

code for the mesh format

#### 6.26.2.11 meshType

```
int meshType
```

code for the mesh type

#### 6.26.2.12 nBCsides

```
long nBCsides
```

global number of BC sides

Referenced by `cgnsWriteMesh()`, `createMesh()`, `setBCatBarys()`, and `setBCatSides()`.

#### 6.26.2.13 nElems

```
long nElems
```

global number of elements

Referenced by `buildMatrix()`, `calcErrors()`, `calcSource()`, `calcTimeStep()`, `cgnsFinalizeOutput()`, `cgnsOutput()`, `cgnsReadSolution()`, `cgnsWriteMesh()`, `createMesh()`, `csvOutput()`, `curveOutput()`, `explicitTimeStepEuler()`, `explicitTimeStepRK()`, `freeMesh()`, `fvTimeDerivative()`, `globalResidual()`, `GMRES_M()`, `implicitTimeStep()`, `initFV()`, `initLinearSolver()`, `initMesh()`, `initTimeDisc()`, `LUSGS()`, `matrixVector()`, `readCGNS()`, `spatialReconstruction()`, and `vectorDotProduct()`.

#### 6.26.2.14 nInnerSides

```
long nInnerSides
```

global number of non BC sides

Referenced by `createMesh()`.

### 6.26.2.15 nNodes

```
long nNodes
```

global number of nodes

Referenced by `cgnsFinalizeOutput()`, `cgnsOutput()`, `cgnsWriteMesh()`, and `createMesh()`.

### 6.26.2.16 nQuads

```
long nQuads
```

global number of quadrangles

Referenced by `cgnsFinalizeOutput()`, `cgnsOutput()`, `cgnsWriteMesh()`, `createCartMesh()`, `createMesh()`, `readCGNS()`, `readEMC2()`, and `readGmsh()`.

### 6.26.2.17 nSides

```
long nSides
```

global number of sides

Referenced by `connectPeriodicBC()`, `createMesh()`, and `fluxCalculation()`.

### 6.26.2.18 nTrias

```
long nTrias
```

global number of triangles

Referenced by `cgnsFinalizeOutput()`, `cgnsOutput()`, `cgnsWriteMesh()`, `createMesh()`, `readCGNS()`, `readEMC2()`, and `readGmsh()`.

### 6.26.2.19 parameterFile

```
char parameterFile[STRLEN]
```

parameter file name

name of the parameter file

#### 6.26.2.20 side

```
side_t** side
```

global side pointer array

Referenced by createMesh(), fluxCalculation(), and freeMesh().

#### 6.26.2.21 strIniCondFile

```
char strIniCondFile[STRLEN]
```

file name of the initial conditions file

Referenced by cgnsReadSolution(), and main().

#### 6.26.2.22 strMeshFile

```
char strMeshFile[STRLEN]
```

mesh file base name

Referenced by createMesh(), and readMesh().

#### 6.26.2.23 strMeshFormat

```
char strMeshFormat[STRLEN]
```

mesh format string

Referenced by createMesh(), and readMesh().

#### 6.26.2.24 totalArea\_q

```
double totalArea_q
```

inverse of the global area of the mesh

Referenced by calcErrors(), createElemInfo(), createMesh(), globalResidual(), and initMesh().

### 6.26.2.25 xMax

```
double xMax
```

minimum x-direction extension

Referenced by `createCartMesh()`, `createMesh()`, `exactFunc()`, and `readMesh()`.

### 6.26.2.26 xMin

```
double xMin
```

maximum x-direction extension

Referenced by `createCartMesh()`, `createMesh()`, `exactFunc()`, and `readMesh()`.

### 6.26.2.27 yMax

```
double yMax
```

minimum y-direction extension

Referenced by `createCartMesh()`, `createMesh()`, `exactFunc()`, and `readMesh()`.

### 6.26.2.28 yMin

```
double yMin
```

maximum y-direction extension

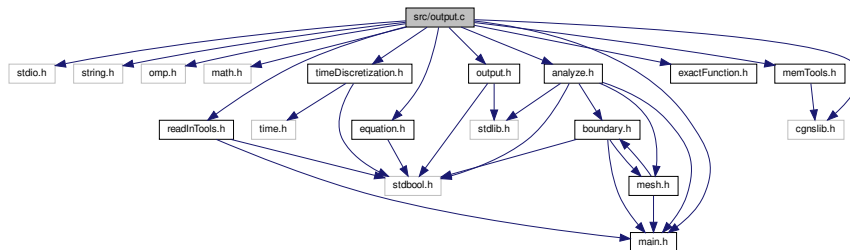
Referenced by `createCartMesh()`, `createMesh()`, `exactFunc()`, and `readMesh()`.

## 6.27 src/output.c File Reference

Contains all functions used for writing flow solutions.

```
#include <stdio.h>
#include <string.h>
#include <omp.h>
#include <math.h>
#include "main.h"
#include "output.h"
#include "readInTools.h"
#include "timeDiscretization.h"
#include "analyze.h"
#include "equation.h"
#include "exactFunction.h"
#include "cgnslib.h"
#include "memTools.h"
```

Include dependency graph for output.c:



## Functions

- void **initOutput** (void)  
*Initialize output.*
- void **csvOutput** (char fileName[STRLEN], double time, bool doExact)  
*Tabular CSV output, only for 1D data.*
- void **cgnsOutput** (char fileName[STRLEN], double time, bool doExact)  
*Write solution to CGNS file.*
- void **curveOutput** (char fileName[STRLEN], double time, bool doExact)  
*Curve data output, only for 1D data.*
- void **dataOutput** (double time, long iter)  
*Perform a data output, dependent on the output format.*
- void **cgnsFinalizeOutput** (void)  
*Finalize CGNS output.*
- void **finalizeDataOutput** (void)  
*Finalize the data output, if necessary.*
- void **cgnsWriteMesh** (void)  
*Write the generated mesh to a CGNS mesh file.*
- void **freeOutputTimes** (void)  
*Free all memory that was allocated for the output times.*

## Variables

- char `strOutFile` [STRLEN]
- double `IOtimeInterval`
- int `IOiterInterval`
- int `iVisuProg`
- char `parameterFile` [STRLEN]
- FILE \* `resFile`
- bool `doErrorOutput`
- `outputTime_t` \* `outputTimes`

### 6.27.1 Detailed Description

Contains all functions used for writing flow solutions.

#### Author

hhh

#### Date

Mon 23 Mar 2020 10:42:06 PM CET

### 6.27.2 Function Documentation

#### 6.27.2.1 `cgnsFinalizeOutput()`

```
void cgnsFinalizeOutput (
    void )
```

Finalize CGNS output.

This function creates a `<case>_Master.cgns` file that has links to all the output times of previous flow solutions. This makes it easier to load an entire case into ParaView.

References `gridFile`, `isStationary`, `outputTime_t::iter`, `nElems`, `outputTime_t::next`, `nNodes`, `nQuads`, `nTrias`, `outputTimes`, `STRLEN`, `strOutFile`, and `outputTime_t::time`.

Referenced by `finalizeDataOutput()`.

#### 6.27.2.2 `cgnsOutput()`

```
void cgnsOutput (
    char fileName[STRLEN],
    double time,
    bool doExact )
```

Write solution to CGNS file.

**Parameters**

in	<i>fileName</i>	The name of the output file
in	<i>time</i>	The computational time of the output result
in	<i>doExact</i>	If the exact exact solution should be written, instead of the computed flow results

References elem\_t::bary, exactFunc(), firstElem, gridFile, elem\_t::id, intExactFunc, nElems, elem\_t::next, nNodes, nQuads, nTrias, NVAR, P, elem\_t::pVar, RHO, STRLEN, timeOverall, VX, and VY.

Referenced by dataOutput().

**6.27.2.3 csvOutput()**

```
void csvOutput (
    char fileName[STRLEN],
    double time,
    bool doExact )
```

Tabular CSV output, only for 1D data.

**Parameters**

in	<i>fileName</i>	The name of the output file
in	<i>time</i>	The computational time of the output result
in	<i>doExact</i>	If the exact exact solution should be written, instead of the computed flow results

References elem\_t::bary, dyn2DdblArray(), exactFunc(), firstElem, intExactFunc, nElems, elem\_t::next, NVAR, P, elem\_t::pVar, RHO, VX, and X.

Referenced by dataOutput().

**6.27.2.4 curveOutput()**

```
void curveOutput (
    char fileName[STRLEN],
    double time,
    bool doExact )
```

Curve data output, only for 1D data.

**Parameters**

in	<i>fileName</i>	The name of the output file
in	<i>time</i>	The computational time of the output result
in	<i>doExact</i>	If the exact exact solution should be written, instead of the computed flow results



References `elem_t::bary`, `dyn2DdblArray()`, `exactFunc()`, `firstElem`, `intExactFunc`, `nElems`, `elem_t::next`, `NVAR`, `P`, `elem_t::pVar`, `RHO`, `VX`, and `X`.

Referenced by `dataOutput()`.

### 6.27.2.5 dataOutput()

```
void dataOutput (
    double time,
    long iter )
```

Perform a data output, dependent on the output format.

#### Parameters

in	<i>time</i>	The computational time of the output result
in	<i>iter</i>	The iteration number of the output result

References `CGNS`, `cgnsOutput()`, `CSV`, `csvOutput()`, `CURVE`, `curveOutput()`, `hasExactSolution`, `isStationary`, `outputTime_t::iter`, `iVisuProg`, `outputTime_t::next`, `outputTimes`, `STRLEN`, `strOutFile`, and `outputTime_t::time`.

Referenced by `timeDisc()`.

## 6.27.3 Variable Documentation

### 6.27.3.1 doErrorOutput

```
bool doErrorOutput
```

error output flag

### 6.27.3.2 IOiterInterval

```
int IOiterInterval
```

iteration interval for data output

Referenced by `initOutput()`, `initTimeDisc()`, and `timeDisc()`.

### 6.27.3.3 IOtimeInterval

```
double IOtimeInterval
```

time interval for data output

Referenced by `initOutput()`, `initTimeDisc()`, and `timeDisc()`.

### 6.27.3.4 iVisuProg

```
int iVisuProg
```

output format code

Referenced by `dataOutput()`, `finalizeDataOutput()`, `initMesh()`, and `initOutput()`.

### 6.27.3.5 outputTimes

```
outputTime_t* outputTimes
```

the first output time object

Referenced by `cgnsFinalizeOutput()`, `dataOutput()`, `freeOutputTimes()`, and `main()`.

### 6.27.3.6 parameterFile

```
char parameterFile[STRLEN]
```

name of the parameter file

### 6.27.3.7 resFile

```
FILE* resFile
```

residual file pointer

Referenced by `analyze()`, `initAnalyze()`, and `timeDisc()`.

### 6.27.3.8 strOutFile

```
char strOutFile[STRLEN]
```

name of the output file

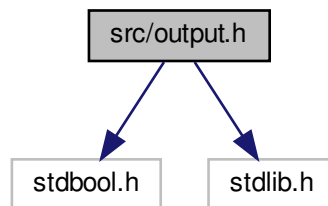
Referenced by `calcCoef()`, `cgnsFinalizeOutput()`, `dataOutput()`, `initAnalyze()`, `initMesh()`, `initOutput()`, and `initRecordPoints()`.

## 6.28 src/output.h File Reference

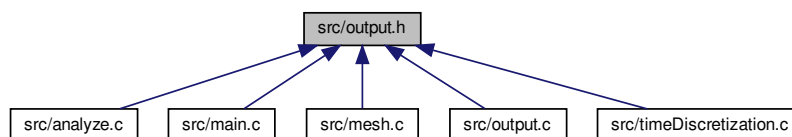
Contains `outputTime_t` struct definition.

```
#include <stdbool.h>
#include <stdlib.h>
```

Include dependency graph for output.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct `outputTime_t`

*Output times linked list.*

## Functions

- void `initOutput` (void)  
*Initialize output.*
- void `dataOutput` (double time, long iter)  
*Perform a data output, dependent on the output format.*
- void `finalizeDataOutput` (void)  
*Finalize the data output, if necessary.*
- void `cgnsWriteMesh` (void)  
*Write the generated mesh to a CGNS mesh file.*
- void `freeOutputTimes` (void)  
*Free all memory that was allocated for the output times.*

## Variables

- char `strOutFile` [STRLEN]
- double `IOtimeInterval`
- int `IOiterInterval`
- int `iVisuProg`
- char `parameterFile` [STRLEN]
- FILE \* `resFile`
- bool `doErrorOutput`
- `outputTime_t` \* `outputTimes`

### 6.28.1 Detailed Description

Contains `outputTime_t` struct definition.

#### Author

hhh

#### Date

Mon 23 Mar 2020 10:37:31 PM CET

### 6.28.2 Function Documentation

#### 6.28.2.1 `dataOutput()`

```
void dataOutput (
    double time,
    long iter )
```

Perform a data output, dependent on the output format.

**Parameters**

<code>in</code>	<code>time</code>	The computational time of the output result
<code>in</code>	<code>iter</code>	The iteration number of the output result

References CGNS, cgnsOutput(), CSV, csvOutput(), CURVE, curveOutput(), hasExactSolution, isStationary, outputTime\_t::iter, iVisuProg, outputTime\_t::next, outputTimes, STRLEN, strOutFile, and outputTime\_t::time.

Referenced by timeDisc().

### 6.28.3 Variable Documentation

#### 6.28.3.1 doErrorOutput

```
bool doErrorOutput
```

error output flag

#### 6.28.3.2 IOiterInterval

```
int IOiterInterval
```

iteration interval for data output

Referenced by initOutput(), initTimeDisc(), and timeDisc().

#### 6.28.3.3 IotimeInterval

```
double IotimeInterval
```

time interval for data output

Referenced by initOutput(), initTimeDisc(), and timeDisc().

#### 6.28.3.4 iVisuProg

```
int iVisuProg
```

output format code

Referenced by dataOutput(), finalizeDataOutput(), initMesh(), and initOutput().

### 6.28.3.5 outputTimes

`outputTime_t*` outputTimes

the first output time object

Referenced by `cgnsFinalizeOutput()`, `dataOutput()`, `freeOutputTimes()`, and `main()`.

### 6.28.3.6 parameterFile

`char` parameterFile[STRLEN]

parameter file name

name of the parameter file

### 6.28.3.7 resFile

`FILE*` resFile

residual file pointer

Referenced by `analyze()`, `initAnalyze()`, and `timeDisc()`.

### 6.28.3.8 strOutFile

`char` strOutFile[STRLEN]

name of the output file

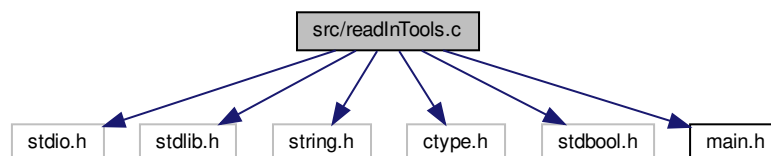
Referenced by `calcCoef()`, `cgnsFinalizeOutput()`, `dataOutput()`, `initAnalyze()`, `initMesh()`, `initOutput()`, and `initRecordPoints()`.

## 6.29 src/readInTools.c File Reference

Provides functions for reading data from the `.ini` parameter file.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <stdbool.h>
#include "main.h"
```

Include dependency graph for `readInTools.c`:



## Data Structures

- struct `cmd_t`

*A structure used to store the commands, read in from the parameter file.*

## Functions

- void `fillCmds` (char iniFileName[STRLEN])  
*Read parameter file and create commands list.*
- void `deleteCmd` (cmd\_t \*aCmd)  
*Delete a single node of the command list.*
- char \* `findCmd` (const char \*key, char defMsg[8], const char \*proposal)  
*Find a command in the commands list.*
- char \* `getStr` (const char \*key, const char \*proposal)  
*Get a string from the commands list.*
- int `countKeys` (const char \*key, const int proposal)  
*Count how often a key appears.*
- int `getInt` (const char \*key, const char \*proposal)  
*Get an integer from the commands list.*
- double `getDb` (const char \*key, const char \*proposal)  
*Get a double from the commands list.*
- bool `getBool` (const char \*key, const char \*proposal)  
*Get a boolean from the commands list.*
- int \* `getIntArray` (const char \*key, const int N, const char \*proposal)  
*Get an integer array from the commands list.*
- double \* `getDbArray` (const char \*key, const int N, const char \*proposal)  
*Get a double array from the commands list.*
- void `freeCmds` (void)  
*Delete all commands in the commands list.*
- void `ignoredCmds` (void)  
*Print out all remaining commands in the list.*

## Variables

- `cmd_t * firstCmd`

### 6.29.1 Detailed Description

Provides functions for reading data from the `.ini` parameter file.

Author

hhh

Date

Sat 21 Mar 2020 10:46:32 AM CET

## 6.29.2 Function Documentation

### 6.29.2.1 countKeys()

```
int countKeys (
    const char * key,
    const int proposal )
```

Count how often a key appears.

Count all occurrences of key in parameter file and return them. If the key is not specified, the proposal will be returned. If the proposal is -1, but the key is not in the list, an error will be thrown.

#### Parameters

in	<i>key</i>	Key string of the command to be found
in	<i>proposal</i>	The default value that is used if the key was not

#### Returns

How often the *key* appeared in the command list

References firstCmd, and cmd\_t::next.

Referenced by initBoundary().

### 6.29.2.2 deleteCmd()

```
void deleteCmd (
    cmd_t * aCmd )
```

Delete a single node of the command list.

Before deleting the command, the previous command is connected to the next command and vice versa.

#### Parameters

in	<i>aCmd</i>	A pointer to the command that is to be deleted
----	-------------	--

References firstCmd, cmd\_t::next, and cmd\_t::prev.

Referenced by findCmd().



### 6.29.2.3 fillCmds()

```
void fillCmds (
    char iniFileName[STRLEN] )
```

Read parameter file and create commands list.

Read .ini file and parse each line into a `cmd_t` object. All `cmd_t` objects are connected in a list of commands starting with `firstCmd`.

#### Parameters

in	<i>iniFileName</i>	The name of the parameter file
----	--------------------	--------------------------------

References `firstCmd`, `cmd_t::key`, `cmd_t::next`, `cmd_t::prev`, `STRLEN`, and `cmd_t::value`.

Referenced by `main()`.

### 6.29.2.4 findCmd()

```
char* findCmd (
    const char * key,
    char defMsg[8],
    const char * proposal )
```

Find a command in the commands list.

Find the provided key in the list of commands, and return the address of the corresponding value string. Return NULL if key was not found. Once a key was read from the commands list, it is deleted from the list.

#### Parameters

in	<i>key</i>	Key string of the command to be found
out	<i>defMsg</i>	String, that indicates if an actual value or the proposal was returned
in	<i>proposal</i>	The default value that is used if the key was not specified

#### Returns

Pointer to the `value` string, or NULL

References `deleteCmd()`, `firstCmd`, `cmd_t::key`, `cmd_t::next`, and `cmd_t::value`.

Referenced by `getBool()`, `getDbI()`, `getDbIArray()`, `getInt()`, `getIntArray()`, and `getStr()`.

### 6.29.2.5 getBool()

```
bool getBool (
    const char * key,
    const char * proposal )
```

Get a boolean from the commands list.

Find the key in the command list and return the corresponding value. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown. The value in the parameter file is accepted as true, if it is a 'T', 't', 'True', or 'true', otherwise it is false.

#### Parameters

in	<i>key</i>	Key string of the command to be found
in	<i>proposal</i>	The default value that is used if the key was not

#### Returns

The value of the parameter, or the default value

References findCmd().

Referenced by initAnalyze(), initBoundary(), initEquation(), initLinearSolver(), initTimeDisc(), and main().

### 6.29.2.6 getDbl()

```
double getDbl (
    const char * key,
    const char * proposal )
```

Get a double from the commands list.

Find the key in the command list and return the corresponding value. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

#### Parameters

in	<i>key</i>	Key string of the command to be found
in	<i>proposal</i>	The default value that is used if the key was not

#### Returns

The value of the parameter, or the default value

References findCmd().

Referenced by initBoundary(), initEquation(), initFV(), initInitialCondition(), initLinearSolver(), initOutput(), initTimeDisc(), and readWing().

**6.29.2.7 getDbIArray()**

```
double* getDbIArray (
    const char * key,
    const int N,
    const char * proposal )
```

Get a double array from the commands list.

Find the key in the command list and return the corresponding integer array. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

**Parameters**

in	<i>key</i>	Key string of the command to be found
in	<i>N</i>	Length of the array that is to be read in
in	<i>proposal</i>	The default value that is used if the key was not

**Returns**

A pointer to the value array of the parameter, or the default value array

References findCmd(), and STRLEN.

Referenced by initBoundary(), initInitialCondition(), initRecordPoints(), and readMesh().

**6.29.2.8 getInt()**

```
int getInt (
    const char * key,
    const char * proposal )
```

Get an integer from the commands list.

Find the key in the command list and return the corresponding value. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

**Parameters**

in	<i>key</i>	Key string of the command to be found
in	<i>proposal</i>	The default value that is used if the key was not

**Returns**

The value of the parameter, or the default value

References findCmd().

Referenced by initAnalyze(), initBoundary(), initEquation(), initFV(), initInitialCondition(), initLinearSolver(), init←Output(), initTimeDisc(), readMesh(), and readWing().

### 6.29.2.9 getIntArray()

```
int* getIntArray (
    const char * key,
    const int N,
    const char * proposal )
```

Get an integer array from the commands list.

Find the key in the command list and return the corresponding integer array. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

#### Parameters

in	<i>key</i>	Key string of the command to be found
in	<i>N</i>	Length of the array that is to be read in
in	<i>proposal</i>	The default value that is used if the key was not

#### Returns

A pointer to the value array of the parameter, or the default value array

References findCmd(), and STRLEN.

Referenced by readMesh().

### 6.29.2.10 getStr()

```
char* getStr (
    const char * key,
    const char * proposal )
```

Get a string from the commands list.

Find the key in the command list and return the corresponding value. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

#### Parameters

in	<i>key</i>	Key string of the command to be found
in	<i>proposal</i>	The default value that is used if the key was not

#### Returns

Pointer to the `value` string, containing the parameter, or the default value, if the parameter was not specified

References findCmd().

Referenced by initOutput(), and readMesh().

### 6.29.3 Variable Documentation

#### 6.29.3.1 firstCmd

```
cmd_t* firstCmd
```

first command of the list

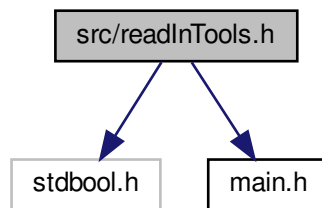
Referenced by countKeys(), deleteCmd(), fillCmds(), findCmd(), freeCmds(), and ignoredCmds().

## 6.30 src/readInTools.h File Reference

```
#include <stdbool.h>
```

```
#include "main.h"
```

Include dependency graph for readInTools.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void [fillCmds](#) (char iniFileName[STRLEN])  
*Read parameter file and create commands list.*
- char \* [getStr](#) (const char \*key, const char \*proposal)  
*Get a string from the commands list.*
- int [countKeys](#) (const char \*key, const int proposal)  
*Count how often a key appears.*
- int [getInt](#) (const char \*key, const char \*proposal)

- Get an integer from the commands list.*
- double `getDbI` (const char \*key, const char \*proposal)
- Get a double from the commands list.*
- bool `getDbOol` (const char \*key, const char \*proposal)
- Get a boolean from the commands list.*
- int \* `getDbIntArray` (const char \*key, const int N, const char \*proposal)
- Get an integer array from the commands list.*
- double \* `getDbDblArray` (const char \*key, const int N, const char \*proposal)
- Get a double array from the commands list.*
- void `ignoredCmds` (void)
- Print out all remaining commands in the list.*

### 6.30.1 Detailed Description

Author

hhh

Date

Sat 21 Mar 2020 10:51:13 AM CET

### 6.30.2 Function Documentation

#### 6.30.2.1 countKeys()

```
int countKeys (
    const char * key,
    const int proposal )
```

Count how often a key appears.

Count all occurrences of key in parameter file and return them. If the key is not specified, the proposal will be returned. If the proposal is -1, but the key is not in the list, an error will be thrown.

Parameters

in	<i>key</i>	Key string of the command to be found
in	<i>proposal</i>	The default value that is used if the key was not

Returns

How often the `key` appeared in the command list

References `firstCmd`, and `cmd_t::next`.

Referenced by `initBoundary()`.

### 6.30.2.2 fillCmds()

```
void fillCmds (
    char iniFileName[STRLEN] )
```

Read parameter file and create commands list.

Read .ini file and parse each line into a `cmd_t` object. All `cmd_t` objects are connected in a list of commands starting with `firstCmd`.

#### Parameters

in	<i>iniFileName</i>	The name of the parameter file
----	--------------------	--------------------------------

References `firstCmd`, `cmd_t::key`, `cmd_t::next`, `cmd_t::prev`, `STRLEN`, and `cmd_t::value`.

Referenced by `main()`.

### 6.30.2.3 getBool()

```
bool getBool (
    const char * key,
    const char * proposal )
```

Get a boolean from the commands list.

Find the key in the command list and return the corresponding value. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown. The value in the parameter file is accepted as true, if it is a 'T', 't', 'True', or 'true', otherwise it is false.

#### Parameters

in	<i>key</i>	Key string of the command to be found
in	<i>proposal</i>	The default value that is used if the key was not

#### Returns

The value of the parameter, or the default value

References `findCmd()`.

Referenced by `initAnalyze()`, `initBoundary()`, `initEquation()`, `initLinearSolver()`, `initTimeDisc()`, and `main()`.

### 6.30.2.4 getDbI()

```
double getDbI (
    const char * key,
    const char * proposal )
```

Get a double from the commands list.

Find the key in the command list and return the corresponding value. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

#### Parameters

in	<i>key</i>	Key string of the command to be found
in	<i>proposal</i>	The default value that is used if the key was not

#### Returns

The value of the parameter, or the default value

References findCmd().

Referenced by initBoundary(), initEquation(), initFV(), initInitialCondition(), initLinearSolver(), initOutput(), initTime← Disc(), and readWing().

### 6.30.2.5 getDbIArray()

```
double* getDbIArray (
    const char * key,
    const int N,
    const char * proposal )
```

Get a double array from the commands list.

Find the key in the command list and return the corresponding integer array. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

#### Parameters

in	<i>key</i>	Key string of the command to be found
in	<i>N</i>	Length of the array that is to be read in
in	<i>proposal</i>	The default value that is used if the key was not

#### Returns

A pointer to the value array of the parameter, or the default value array

References findCmd(), and STRLEN.

Referenced by initBoundary(), initInitialCondition(), initRecordPoints(), and readMesh().



### 6.30.2.6 getInt()

```
int getInt (
    const char * key,
    const char * proposal )
```

Get an integer from the commands list.

Find the key in the command list and return the corresponding value. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

#### Parameters

in	<i>key</i>	Key string of the command to be found
in	<i>proposal</i>	The default value that is used if the key was not

#### Returns

The value of the parameter, or the default value

References findCmd().

Referenced by initAnalyze(), initBoundary(), initEquation(), initFV(), initInitialCondition(), initLinearSolver(), initOutput(), initTimeDisc(), readMesh(), and readWing().

### 6.30.2.7 getIntArray()

```
int* getIntArray (
    const char * key,
    const int N,
    const char * proposal )
```

Get an integer array from the commands list.

Find the key in the command list and return the corresponding integer array. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

#### Parameters

in	<i>key</i>	Key string of the command to be found
in	<i>N</i>	Length of the array that is to be read in
in	<i>proposal</i>	The default value that is used if the key was not

#### Returns

A pointer to the value array of the parameter, or the default value array

References findCmd(), and STRLEN.

Referenced by readMesh().

### 6.30.2.8 getStr()

```
char* getStr (
    const char * key,
    const char * proposal )
```

Get a string from the commands list.

Find the key in the command list and return the corresponding value. If the key is not specified, the proposal will be returned. If the proposal is NULL, but the key is not in the list, an error will be thrown.

#### Parameters

in	<i>key</i>	Key string of the command to be found
in	<i>proposal</i>	The default value that is used if the key was not

#### Returns

Pointer to the `value` string, containing the parameter, or the default value, if the parameter was not specified

References `findCmd()`.

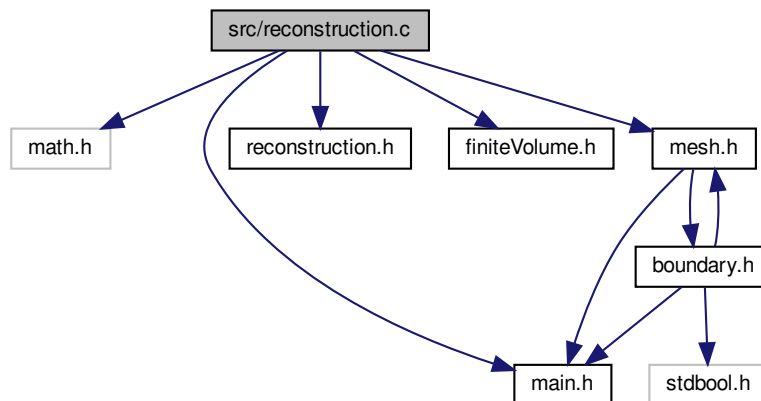
Referenced by `initOutput()`, and `readMesh()`.

## 6.31 src/reconstruction.c File Reference

Contains the reconstruction and limiter functions.

```
#include <math.h>
#include "main.h"
#include "reconstruction.h"
#include "finiteVolume.h"
#include "mesh.h"
```

Include dependency graph for `reconstruction.c`:



## Functions

- void `limiterBarthJespersen` (`elem_t` \*`aElem`)  
*Limiter after Barth & Jespersen.*
- void `limiterVenkatakrishnan` (`elem_t` \*`aElem`)  
*Limiter after Venkatakrishnan, with additional limiting parameter  $k$ .*
- void `spatialReconstruction` (double time)  
*Compute the gradients of  $dU/dx$ .*

## Variables

- int `limiter`
- double `venk_k`

### 6.31.1 Detailed Description

Contains the reconstruction and limiter functions.

#### Author

hhh

#### Date

Sat 28 Mar 2020 10:17:02 AM CET

### 6.31.2 Function Documentation

#### 6.31.2.1 `limiterBarthJespersen()`

```
void limiterBarthJespersen (
    elem_t * aElem )
```

Limiter after Barth & Jespersen.

#### Note

2D, unstructured limiter

#### Parameters

in	<i>aElem</i>	Pointer to an element
----	--------------	-----------------------

References `side_t::connection`, `side_t::elem`, `elem_t::firstSide`, `side_t::GP`, `side_t::nextElemSide`, `NVAR`, `P`, `elem_t::pVar`, `RHO`, `elem_t::u_x`, `elem_t::u_y`, `VX`, `VY`, `X`, and `Y`.

Referenced by `spatialReconstruction()`.

### 6.31.2.2 limiterVenkatakrishnan()

```
void limiterVenkatakrishnan (
    elem_t * aElem )
```

Limiter after Venkatakrishnan, with additional limiting parameter k.

#### Note

2D, unstructured limiter

#### Parameters

in	<i>aElem</i>	Pointer to an element
----	--------------	-----------------------

References `side_t::connection`, `side_t::elem`, `elem_t::firstSide`, `side_t::GP`, `side_t::nextElemSide`, `NVAR`, `P`, `elem_t::pVar`, `RHO`, `elem_t::u_x`, `elem_t::u_y`, `elem_t::venkEps_sq`, `VX`, `VY`, `X`, and `Y`.

Referenced by `spatialReconstruction()`.

### 6.31.2.3 spatialReconstruction()

```
void spatialReconstruction (
    double time )
```

Compute the gradients of  $dU/dx$ .

#### Parameters

in	<i>time</i>	Calculation time at which to perform the spatial reconstruction
----	-------------	---

References `BARTHJESPERSEN`, `side_t::connection`, `side_t::elem`, `elem`, `elem_t::firstSide`, `side_t::GP`, `limiter`, `limiterBarthJespersen()`, `limiterVenkatakrishnan()`, `nElems`, `side_t::nextElemSide`, `NVAR`, `P`, `side_t::pVar`, `elem_t::pVar`, `RHO`, `setBCatBarys()`, `spatialOrder`, `elem_t::u_t`, `elem_t::u_x`, `elem_t::u_y`, `VENKATAKRISHNAN`, `VX`, `VY`, `side_t::w`, `X`, and `Y`.

Referenced by `calcErrors()`, and `fvTimeDerivative()`.

## 6.31.3 Variable Documentation

### 6.31.3.1 limiter

`int limiter`

limiter selection

Referenced by `initFV()`, and `spatialReconstruction()`.

### 6.31.3.2 venk\_k

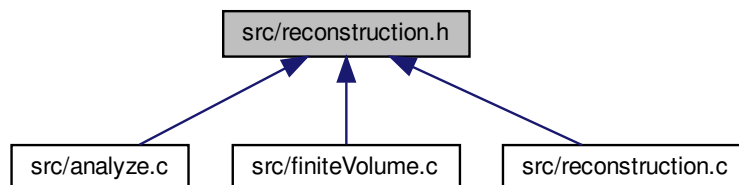
`double venk_k`

constant for Venkatakrishnan limiter

Referenced by `initFV()`.

## 6.32 src/reconstruction.h File Reference

This graph shows which files directly or indirectly include this file:



## Functions

- void `spatialReconstruction` (double time)  
*Compute the gradients of  $dU/dx$ .*

## Variables

- int `limiter`
- double `venk_k`

### 6.32.1 Detailed Description

Author

hhh

Date

Sat 28 Mar 2020 10:16:16 AM CET

### 6.32.2 Function Documentation

#### 6.32.2.1 spatialReconstruction()

```
void spatialReconstruction (
    double time )
```

Compute the gradients of dU/dx.

Parameters

in	time	Calculation time at which to perform the spatial reconstruction
----	------	---

References BARTHJESPERSEN, side\_t::connection, side\_t::elem, elem, elem\_t::firstSide, side\_t::GP, limiter, limiterBarthJespersen(), limiterVenkatakrishnan(), nElems, side\_t::nextElemSide, NVAR, P, side\_t::pVar, elem\_t::pVar, RHO, setBCatBarys(), spatialOrder, elem\_t::u\_t, elem\_t::u\_x, elem\_t::u\_y, VENKATAKRISHNAN, VX, VY, side\_t::w, X, and Y.

Referenced by calcErrors(), and fvTimeDerivative().

### 6.32.3 Variable Documentation

#### 6.32.3.1 limiter

```
int limiter
```

limiter selection

Referenced by initFV(), and spatialReconstruction().

### 6.32.3.2 venk\_k

```
double venk_k
```

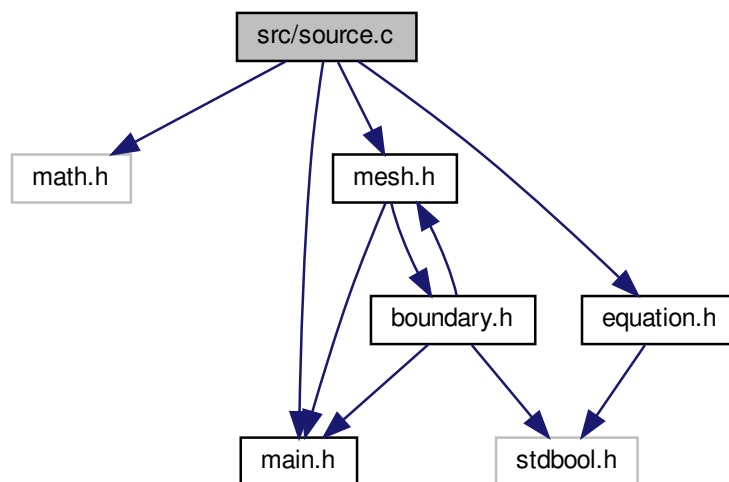
constant for Venkatakrishnan limiter

Referenced by `initFV()`.

## 6.33 src/source.c File Reference

Contains the functions for initializing and evaluating the source term.

```
#include <math.h>
#include "main.h"
#include "equation.h"
#include "mesh.h"
Include dependency graph for source.c:
```



## Functions

- void `evalSource` (int `iSource`, double `x[NDIM]`, double `time`, double `source[NVAR]`)  
*Evaluate the source term.*
- void `calcSource` (double `time`)  
*Calculate the contribution of the source terms.*

### 6.33.1 Detailed Description

Contains the functions for initializing and evaluating the source term.

#### Author

hhh

#### Date

Wed 01 Apr 2020 12:25:11 PM CEST

### 6.33.2 Function Documentation

#### 6.33.2.1 calcSource()

```
void calcSource (
    double time )
```

Calculate the contribution of the source terms.

##### Parameters

in	<i>time</i>	The computation time at which the evaluate the source term
----	-------------	--

References E, elem, evalSource(), nElems, elem\_t::nGP, NVAR, RHO, elem\_t::source, sourceFunc, VX, VY, elem\_t::wGP, and elem\_t::xGP.

Referenced by fvTimeDerivative().

#### 6.33.2.2 evalSource()

```
void evalSource (
    int iSource,
    double x[NDIM],
    double time,
    double source[NVAR] )
```

Evaluate the source term.

##### Parameters

in	<i>iSource</i>	The source control integer
in	<i>x</i>	The coordinates at which to evaluate the source term
in	<i>time</i>	The computation time at which the evaluate the source term
out	<i>source</i>	The source term contribution

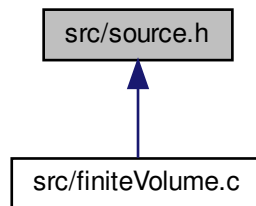


References E, gam, gam1, mu, pi, Pr, RHO, VX, VY, X, and Y.

Referenced by calcSource().

## 6.34 src/source.h File Reference

This graph shows which files directly or indirectly include this file:



### Functions

- void [calcSource](#) (double time)  
*Calculate the contribution of the source terms.*

#### 6.34.1 Detailed Description

Author

hhh

Date

Wed 01 Apr 2020 12:25:19 PM CEST

#### 6.34.2 Function Documentation

##### 6.34.2.1 calcSource()

```
void calcSource (  
    double time )
```

Calculate the contribution of the source terms.

## Parameters

in	time	The computation time at which the evaluate the source term
----	------	--

References E, elem, evalSource(), nElems, elem\_t::nGP, NVAR, RHO, elem\_t::source, sourceFunc, VX, VY, elem\_t::wGP, and elem\_t::xGP.

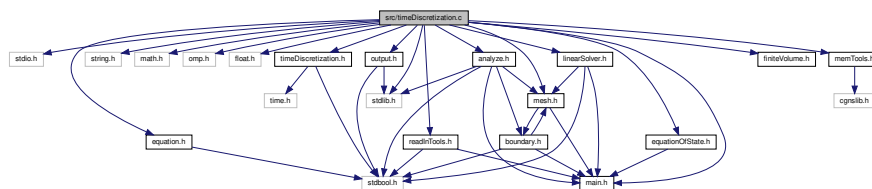
Referenced by fvTimeDerivative().

## 6.35 src/timeDiscretization.c File Reference

Contains the functions for performing the time stepping process.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <omp.h>
#include <float.h>
#include "main.h"
#include "timeDiscretization.h"
#include "readInTools.h"
#include "output.h"
#include "mesh.h"
#include "equation.h"
#include "analyze.h"
#include "linearSolver.h"
#include "equationOfState.h"
#include "finiteVolume.h"
#include "memTools.h"
```

Include dependency graph for timeDiscretization.c:



## Functions

- void **initTimeDisc** (void)  
*Initialize the time discretization.*
- void **calcTimeStep** (double pTime, double \*dt, bool \*viscousTimeStepDominates)  
*Compute the time step.*
- void **explicitTimeStepEuler** (double time, double dt, double reslter[NVAR+2])  
*Performs explicit time step using Euler scheme.*
- void **explicitTimeStepRK** (double time, double dt, double reslter[NVAR+2])  
*Performs explicit time step using Runge-Kutta scheme nRKstages stages.*
- void **implicitTimeStep** (double time, double dt, double reslter[NVAR+2])  
*Euler implicit time integration.*
- void **timeDisc** (void)  
*Main time discretization loop.*

## Variables

- double `cfl`
- double `dfl`
- double `t`
- double `timeOverall`
- int `timeOrder`
- bool `isTimeStep1D`
- bool `isStationary`
- long `maxIter`
- double `stopTime`
- long `iniliterationNumber`
- double `startTime`
- double `abortResidual`
- int `abortVariable`
- char `abortVariableName` [4]
- double `clAbortResidual`
- double `cdAbortResidual`
- bool `doAbortOnClResidual`
- bool `doAbortOnCdResidual`
- bool `isRestart`
- double `restartTime`
- int `printIter`
- double `printTime`
- int `nRKstages`
- double `RKcoeff` [6] = {0.0}
- bool `isImplicit`
- double \*\* `deltaX`
- double \*\* `Q`
- double \*\* `F_X0`
- double \*\* `F_XK`

### 6.35.1 Detailed Description

Contains the functions for performing the time stepping process.

#### Author

hhh

#### Date

Sat 21 Mar 2020 07:52:42 PM CET

### 6.35.2 Function Documentation

#### 6.35.2.1 `calcTimeStep()`

```
void calcTimeStep (  
    double pTime,  
    double * dt,  
    bool * viscousTimeStepDominates )
```

Compute the time step.

## Parameters

in	<i>pTime</i>	The print time interval
out	<i>dt</i>	The resulting time step
out	<i>viscousTimeStepDominates</i>	Flag for if the viscous time step is dominating

References *elem\_t::area*, *cfl*, *dfl*, *elem\_t::dt*, *elem*, *gam*, *isTimeStep1D*, *mu*, *nElems*, *P*, *Pr*, *elem\_t::pVar*, *RHO*, *stopTime*, *elem\_t::sx*, *elem\_t::sy*, *t*, *VX*, and *VY*.

Referenced by *timeDisc()*.

### 6.35.2.2 explicitTimeStepEuler()

```
void explicitTimeStepEuler (
    double time,
    double dt,
    double resIter[NVAR+2] )
```

Performs explicit time step using Euler scheme.

## Parameters

in	<i>time</i>	Computation time at calculation
in	<i>dt</i>	Time step at calculation
out	<i>resIter</i>	Residual vector for time step

References *consPrim()*, *elem\_t::cVar*, *E*, *elem*, *fvTimeDerivative()*, *globalResidual()*, *MX*, *MY*, *nElems*, *elem\_t::pVar*, *RHO*, and *elem\_t::u\_t*.

Referenced by *timeDisc()*.

### 6.35.2.3 explicitTimeStepRK()

```
void explicitTimeStepRK (
    double time,
    double dt,
    double resIter[NVAR+2] )
```

Performs explicit time step using Runge-Kutta scheme *nRKstages* stages.

## Parameters

in	<i>time</i>	Computation time at calculation
in	<i>dt</i>	Time step at calculation
out	<i>resIter</i>	Residual vector for time step

References `consPrim()`, `elem_t::cVar`, `elem_t::cVarStage`, `E`, `elem`, `fvTimeDerivative()`, `globalResidual()`, `MX`, `MY`, `nElems`, `nRKstages`, `elem_t::pVar`, `RHO`, `RKcoeff`, and `elem_t::u_t`.

Referenced by `timeDisc()`.

#### 6.35.2.4 implicitTimeStep()

```
void implicitTimeStep (
    double time,
    double dt,
    double resIter[NVAR+2] )
```

Euler implicit time integration.

The non-linear equations require the use of a Newton method with internal sub-iteration, using a GMRES method.

##### Parameters

in	<i>time</i>	Computation time at calculation
in	<i>dt</i>	Time step at calculation
out	<i>resIter</i>	Residual vector for time step

References `alpha`, `consPrim()`, `elem_t::cVar`, `deltaX`, `E`, `elem`, `eps2newton`, `eps2newton_sq`, `F_X0`, `F_XK`, `fvTimeDerivative()`, `gamEW`, `globalResidual()`, `GMRES_M()`, `MX`, `MY`, `nElems`, `nInnerNewton`, `nNewtonIter`, `nNewtonIterGlobal`, `elem_t::pVar`, `Q`, `R_XK`, `RHO`, `t`, `elem_t::u_t`, `vectorDotProduct()`, and `XK`.

Referenced by `timeDisc()`.

#### 6.35.2.5 timeDisc()

```
void timeDisc (
    void )
```

Main time discretization loop.

Selection of temporal integration method, as well as management of data output and analysis tools.

References `abortResidual`, `abortVariable`, `abortVariableName`, `analyze()`, `calcErrors()`, `calcTimeStep()`, `wing_t::cd`, `cdAbortResidual`, `wing_t::cl`, `clAbortResidual`, `CPU_TIME`, `dataOutput()`, `deltaX`, `doAbortOnCdResidual`, `doAbortOnClResidual`, `doCalcWing`, `E`, `explicitTimeStepEuler()`, `explicitTimeStepRK()`, `F_X0`, `F_XK`, `finalizeDataOutput()`, `hasExactSolution`, `implicitTimeStep()`, `inIterationNumber`, `recordPoint_t::ioFile`, `IOiterInterval`, `IOtimeInterval`, `isImplicit`, `isStationary`, `maxIter`, `MX`, `MY`, `nGMRESiterGlobal`, `nNewtonIterGlobal`, `recordPoint_t::nPoints`, `nRKstages`, `NVAR`, `printIter`, `printTime`, `Q`, `recordPoint`, `resFile`, `RHO`, `stopTime`, `t`, `timeOrder`, and `wing`.

Referenced by `main()`.

### 6.35.3 Variable Documentation

#### 6.35.3.1 abortResidual

```
double abortResidual
```

residual at which to abort the calculation

Referenced by `initTimeDisc()`, and `timeDisc()`.

#### 6.35.3.2 abortVariable

```
int abortVariable
```

abort variable

Referenced by `analyze()`, `initTimeDisc()`, and `timeDisc()`.

#### 6.35.3.3 abortVariableName

```
char abortVariableName[4]
```

string of the abort variable

Referenced by `initAnalyze()`, `initTimeDisc()`, and `timeDisc()`.

#### 6.35.3.4 cdAbortResidual

```
double cdAbortResidual
```

CD abort residual

Referenced by `initTimeDisc()`, and `timeDisc()`.

#### 6.35.3.5 cfl

```
double cfl
```

Courant-Friedrichs-Lewy number

Referenced by `calcTimeStep()`, and `initTimeDisc()`.

### 6.35.3.6 clAbortResidual

```
double clAbortResidual
```

CL abort residual

Referenced by `initTimeDisc()`, and `timeDisc()`.

### 6.35.3.7 deltaX

```
double** deltaX
```

variable used in implicit calculation

Referenced by `implicitTimeStep()`, `initTimeDisc()`, and `timeDisc()`.

### 6.35.3.8 dfl

```
double dfl
```

diffusive Courant-Friedrichs-Lewy number

Referenced by `calcTimeStep()`, and `initTimeDisc()`.

### 6.35.3.9 doAbortOnCdResidual

```
bool doAbortOnCdResidual
```

CD abort flag

Referenced by `initTimeDisc()`, and `timeDisc()`.

### 6.35.3.10 doAbortOnClResidual

```
bool doAbortOnClResidual
```

CL abort flag

Referenced by `initTimeDisc()`, and `timeDisc()`.

#### 6.35.3.11 F\_X0

```
double** F_X0
```

variable used in implicit calculation

Referenced by `implicitTimeStep()`, `initTimeDisc()`, and `timeDisc()`.

#### 6.35.3.12 F\_XK

```
double** F_XK
```

variable used in implicit calculation

Referenced by `implicitTimeStep()`, `initTimeDisc()`, and `timeDisc()`.

#### 6.35.3.13 iniIterationNumber

```
long iniIterationNumber
```

initial iteration number

Referenced by `initTimeDisc()`, `main()`, and `timeDisc()`.

#### 6.35.3.14 isImplicit

```
bool isImplicit
```

implicit calculation flag

Referenced by `freeLinearSolver()`, `initLinearSolver()`, `initTimeDisc()`, and `timeDisc()`.

#### 6.35.3.15 isRestart

```
bool isRestart
```

restart flag

Referenced by `initAnalyze()`, `initMesh()`, `initTimeDisc()`, `main()`, and `setInitialCondition()`.



### 6.35.3.16 isStationary

```
bool isStationary
```

flag for stationary problem

Referenced by analyze(), cgnsFinalizeOutput(), dataOutput(), initAnalyze(), initTimeDisc(), main(), and timeDisc().

### 6.35.3.17 isTimeStep1D

```
bool isTimeStep1D
```

flag for 1D problem

Referenced by calcTimeStep(), and initTimeDisc().

### 6.35.3.18 maxIter

```
long maxIter
```

maximum number of iterations

Referenced by initTimeDisc(), and timeDisc().

### 6.35.3.19 nRKstages

```
int nRKstages
```

number of Runge-Kutta stages

Referenced by explicitTimeStepRK(), initTimeDisc(), and timeDisc().

### 6.35.3.20 printIter

```
int printIter
```

iterations after which to output

Referenced by initTimeDisc(), and timeDisc().

#### 6.35.3.21 printTime

```
double printTime
```

calculation time after which to output

Referenced by `initTimeDisc()`, and `timeDisc()`.

#### 6.35.3.22 Q

```
double** Q
```

variable used in implicit calculation

Referenced by `implicitTimeStep()`, `initTimeDisc()`, and `timeDisc()`.

#### 6.35.3.23 restartTime

```
double restartTime
```

calculation time for restart

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

#### 6.35.3.24 RKcoeff

```
double RKcoeff[6] = {0.0}
```

array of Runge-Kutta coefficients

Referenced by `explicitTimeStepRK()`, and `initTimeDisc()`.

#### 6.35.3.25 startTime

```
double startTime
```

starting time

Referenced by `main()`.

### 6.35.3.26 stopTime

```
double stopTime
```

simulation end time

Referenced by calcTimeStep(), initTimeDisc(), and timeDisc().

### 6.35.3.27 t

```
double t
```

global calculation time

Referenced by calcTimeStep(), cgnsReadSolution(), GMRES\_M(), implicitTimeStep(), initTimeDisc(), and timeDisc().

### 6.35.3.28 timeOrder

```
int timeOrder
```

order of the time integration

Referenced by fvTimeDerivative(), initTimeDisc(), and timeDisc().

### 6.35.3.29 timeOverall

```
double timeOverall
```

overall time

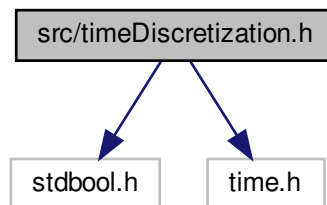
Referenced by cgnsOutput(), and cgnsReadSolution().

## 6.36 src/timeDiscretization.h File Reference

```
#include <stdbool.h>
```

```
#include <time.h>
```

Include dependency graph for timeDiscretization.h:



This graph shows which files directly or indirectly include this file:



### Macros

- `#define CPU_TIME()` `((double)clock() / (double)CLOCKS_PER_SEC)`  
Get the CPU time for a serial program.

### Functions

- void `initTimeDisc` (void)  
Initialize the time discretization.
- void `timeDisc` (void)  
Main time discretization loop.

### Variables

- double `cfl`
- double `dfl`
- double `t`
- double `timeOverall`
- int `timeOrder`
- bool `isTimeStep1D`
- bool `isStationary`

- long [maxIter](#)
- double [stopTime](#)
- long [inIterationNumber](#)
- double [startTime](#)
- double [abortResidual](#)
- int [abortVariable](#)
- char [abortVariableName](#) [4]
- double [clAbortResidual](#)
- double [cdAbortResidual](#)
- bool [doAbortOnClResidual](#)
- bool [doAbortOnCdResidual](#)
- bool [isRestart](#)
- double [restartTime](#)
- int [printIter](#)
- double [printTime](#)
- int [nRKstages](#)
- double [RKcoeff](#) [6]
- bool [isImplicit](#)

### 6.36.1 Detailed Description

#### Author

hhh

#### Date

Sat 21 Mar 2020 07:48:34 PM CET

### 6.36.2 Function Documentation

#### 6.36.2.1 timeDisc()

```
void timeDisc (
    void )
```

Main time discretization loop.

Selection of temporal integration method, as well as management of data output and analysis tools.

References [abortResidual](#), [abortVariable](#), [abortVariableName](#), [analyze\(\)](#), [calcErrors\(\)](#), [calcTimeStep\(\)](#), [wing\\_t::cd](#), [cdAbortResidual](#), [wing\\_t::cl](#), [clAbortResidual](#), [CPU\\_TIME](#), [dataOutput\(\)](#), [deltaX](#), [doAbortOnCdResidual](#), [doAbortOnClResidual](#), [doCalcWing](#), [E](#), [explicitTimeStepEuler\(\)](#), [explicitTimeStepRK\(\)](#), [F\\_X0](#), [F\\_XK](#), [finalizeDataOutput\(\)](#), [hasExactSolution](#), [implicitTimeStep\(\)](#), [inIterationNumber](#), [recordPoint\\_t::ioFile](#), [IOiterInterval](#), [IOtimeInterval](#), [isImplicit](#), [isStationary](#), [maxIter](#), [MX](#), [MY](#), [nGMRESiterGlobal](#), [nNewtonIterGlobal](#), [recordPoint\\_t::nPoints](#), [nRKstages](#), [NVAR](#), [printIter](#), [printTime](#), [Q](#), [recordPoint](#), [resFile](#), [RHO](#), [stopTime](#), [t](#), [timeOrder](#), and [wing](#).

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

### 6.36.3 Variable Documentation

#### 6.36.3.1 abortResidual

```
double abortResidual
```

residual at which to abort the calculation

Referenced by `initTimeDisc()`, and `timeDisc()`.

#### 6.36.3.2 abortVariable

```
int abortVariable
```

abort variable

Referenced by `analyze()`, `initTimeDisc()`, and `timeDisc()`.

#### 6.36.3.3 abortVariableName

```
char abortVariableName[4]
```

string of the abort variable

Referenced by `initAnalyze()`, `initTimeDisc()`, and `timeDisc()`.

#### 6.36.3.4 cdAbortResidual

```
double cdAbortResidual
```

CD abort residual

Referenced by `initTimeDisc()`, and `timeDisc()`.

#### 6.36.3.5 cfl

```
double cfl
```

Courant-Friedrichs-Lewy number

Referenced by `calcTimeStep()`, and `initTimeDisc()`.

### 6.36.3.6 clAbortResidual

```
double clAbortResidual
```

CL abort residual

Referenced by `initTimeDisc()`, and `timeDisc()`.

### 6.36.3.7 dfl

```
double dfl
```

diffusive Courant-Friedrichs-Lewy number

Referenced by `calcTimeStep()`, and `initTimeDisc()`.

### 6.36.3.8 doAbortOnCdResidual

```
bool doAbortOnCdResidual
```

CD abort flag

Referenced by `initTimeDisc()`, and `timeDisc()`.

### 6.36.3.9 doAbortOnClResidual

```
bool doAbortOnClResidual
```

CL abort flag

Referenced by `initTimeDisc()`, and `timeDisc()`.

### 6.36.3.10 iniIterationNumber

```
long iniIterationNumber
```

initial iteration number

Referenced by `initTimeDisc()`, `main()`, and `timeDisc()`.

#### 6.36.3.11 isImplicit

```
bool isImplicit
```

implicit calculation flag

Referenced by freeLinearSolver(), initLinearSolver(), initTimeDisc(), and timeDisc().

#### 6.36.3.12 isRestart

```
bool isRestart
```

restart flag

Referenced by initAnalyze(), initMesh(), initTimeDisc(), main(), and setInitialCondition().

#### 6.36.3.13 isStationary

```
bool isStationary
```

flag for stationary problem

Referenced by analyze(), cgnsFinalizeOutput(), dataOutput(), initAnalyze(), initTimeDisc(), main(), and timeDisc().

#### 6.36.3.14 isTimeStep1D

```
bool isTimeStep1D
```

flag for 1D problem

Referenced by calcTimeStep(), and initTimeDisc().

#### 6.36.3.15 maxIter

```
long maxIter
```

maximum number of iterations

Referenced by initTimeDisc(), and timeDisc().



### 6.36.3.16 nRKstages

```
int nRKstages
```

number of Runge-Kutta stages

Referenced by `explicitTimeStepRK()`, `initTimeDisc()`, and `timeDisc()`.

### 6.36.3.17 printIter

```
int printIter
```

iterations after which to output

Referenced by `initTimeDisc()`, and `timeDisc()`.

### 6.36.3.18 printTime

```
double printTime
```

calculation time after which to output

Referenced by `initTimeDisc()`, and `timeDisc()`.

### 6.36.3.19 restartTime

```
double restartTime
```

calculation time for restart

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

### 6.36.3.20 RKcoeff

```
double RKcoeff[6]
```

array of Runge-Kutta coefficients

Referenced by `explicitTimeStepRK()`, and `initTimeDisc()`.

#### 6.36.3.21 **startTime**

```
double startTime
```

starting time

Referenced by `main()`.

#### 6.36.3.22 **stopTime**

```
double stopTime
```

simulation end time

Referenced by `calcTimeStep()`, `initTimeDisc()`, and `timeDisc()`.

#### 6.36.3.23 **t**

```
double t
```

global calculation time

Referenced by `calcTimeStep()`, `cgnsReadSolution()`, `GMRES_M()`, `implicitTimeStep()`, `initTimeDisc()`, and `timeDisc()`.

#### 6.36.3.24 **timeOrder**

```
int timeOrder
```

order of the time integration

Referenced by `fvTimeDerivative()`, `initTimeDisc()`, and `timeDisc()`.

#### 6.36.3.25 **timeOverall**

```
double timeOverall
```

overall time

Referenced by `cgnsOutput()`, and `cgnsReadSolution()`.

# Index

- abortResidual
  - timeDiscretization.c, [179](#)
  - timeDiscretization.h, [188](#)
- abortVariable
  - timeDiscretization.c, [180](#)
  - timeDiscretization.h, [188](#)
- abortVariableName
  - timeDiscretization.c, [180](#)
  - timeDiscretization.h, [188](#)
- alpha
  - initialCondition.c, [91](#)
  - initialCondition.h, [93](#)
- analyze
  - analyze.c, [40](#)
  - analyze.h, [44](#)
- analyze.c
  - analyze, [40](#)
  - calcErrors, [41](#)
  - doCalcWing, [42](#)
  - evalRecordPoints, [41](#)
  - globalResidual, [41](#)
  - hasExactSolution, [42](#)
  - recordPoint, [42](#)
  - wing, [42](#)
- analyze.h
  - analyze, [44](#)
  - calcErrors, [45](#)
  - doCalcWing, [45](#)
  - globalResidual, [45](#)
  - hasExactSolution, [46](#)
  - recordPoint, [46](#)
  - wing, [46](#)
- area
  - elem\_t, [18](#)
- areaq
  - elem\_t, [18](#)
- AUSMD
  - main.h, [114](#)
- AUSMDV
  - main.h, [114](#)
- BARTHJESPERSEN
  - main.h, [116](#)
- bary
  - elem\_t, [19](#)
- baryBaryDist
  - side\_t, [29](#)
- baryBaryVec
  - side\_t, [29](#)
- BC
  - side\_t, [29](#)
  - sideList\_t, [33](#)
- BCid
  - boundary\_t, [12](#)
  - side\_t, [29](#)
- BCrange
  - cartMesh\_t, [14](#)
- BCside
  - mesh.c, [134](#)
  - mesh.h, [142](#)
- BCtype
  - boundary\_t, [12](#)
  - cartMesh\_t, [15](#)
  - side\_t, [30](#)
- BOTTOM
  - main.h, [113](#)
- boundary
  - boundary.c, [48](#)
  - boundary.h, [51](#)
- boundary.c
  - boundary, [48](#)
  - firstBC, [49](#)
  - isPeriodic, [49](#)
  - nBC, [49](#)
  - setBCatBarys, [48](#)
  - setBCatSides, [49](#)
- boundary.h
  - boundary, [51](#)
  - firstBC, [53](#)
  - isPeriodic, [53](#)
  - nBC, [53](#)
  - setBCatBarys, [52](#)
  - setBCatSides, [52](#)
- boundary\_t, [11](#)
  - BCid, [12](#)
  - BCtype, [12](#)
  - connection, [12](#)
  - exactFunc, [12](#)
  - heatFlux, [13](#)
  - isAdiabatic, [13](#)
  - isTemperaturePrescribed, [13](#)
  - next, [13](#)
  - pVar, [13](#)
  - temperature, [14](#)
- boundaryConditionType
  - main.h, [112](#)
- buildMatrix
  - linearSolver.c, [96](#)
- calcDinv

- linearSolver.c, 96
- calcErrors
  - analyze.c, 41
  - analyze.h, 45
- calcSource
  - source.c, 174
  - source.h, 175
- calcTimeStep
  - timeDiscretization.c, 177
- CARTESIAN
  - main.h, 116
- cartesianMeshSides
  - main.h, 113
- cartMesh
  - mesh.c, 134
  - mesh.h, 142
- cartMesh\_t, 14
  - BCrange, 14
  - BCtype, 15
  - iMax, 15
  - jMax, 15
  - nBC, 15
- CD
  - main.h, 113
- cd
  - wing\_t, 36
- cdAbortResidual
  - timeDiscretization.c, 180
  - timeDiscretization.h, 188
- CEN
  - main.h, 114
- cfl
  - timeDiscretization.c, 180
  - timeDiscretization.h, 188
- CGNS
  - main.h, 116
- cgnsFinalizeOutput
  - output.c, 149
- cgnsOutput
  - output.c, 149
- CHARACTERISTIC
  - main.h, 113
- charCons
  - equationOfState.c, 63
  - equationOfState.h, 66
- CL
  - main.h, 113
- cl
  - wing\_t, 37
- clAbortResidual
  - timeDiscretization.c, 180
  - timeDiscretization.h, 188
- clcdResiduals
  - main.h, 113
- cmd\_t, 16
  - key, 16
  - next, 16
  - prev, 16
  - value, 17
- compare
  - mesh.c, 129
- connection
  - boundary\_t, 12
  - side\_t, 30
- consChar
  - equationOfState.c, 63
  - equationOfState.h, 66
- conservativeVariables
  - main.h, 113
- consPrim
  - equationOfState.c, 64
  - equationOfState.h, 67
- convectiveFlux
  - fluxCalculation.c, 80
- countKeys
  - readInTools.c, 158
  - readInTools.h, 164
- cp
  - equation.c, 54
  - equation.h, 59
- createCartMesh
  - mesh.c, 130
- createElemInfo
  - mesh.c, 130
- createMesh
  - mesh.c, 131
- createReconstructionInfo
  - mesh.c, 131
- createSideInfo
  - mesh.c, 131
- CSV
  - main.h, 116
- csvOutput
  - output.c, 150
- CURVE
  - main.h, 116
- curveOutput
  - output.c, 150
- cVar
  - elem\_t, 19
- cVarStage
  - elem\_t, 19
- D
  - linearSolver.c, 98
- dataOutput
  - output.c, 151
  - output.h, 154
- deleteCmd
  - readInTools.c, 158
- deltaX
  - timeDiscretization.c, 181
- deltaXstar
  - linearSolver.c, 99
- dfl
  - timeDiscretization.c, 181
  - timeDiscretization.h, 189

Dinv  
     linearSolver.c, 99  
 directions  
     main.h, 114  
 doAbortOnCdResidual  
     timeDiscretization.c, 181  
     timeDiscretization.h, 189  
 doAbortOnCIResidual  
     timeDiscretization.c, 181  
     timeDiscretization.h, 189  
 doCalcSource  
     equation.c, 54  
     equation.h, 59  
 doCalcWing  
     analyze.c, 42  
     analyze.h, 45  
 doErrorOutput  
     output.c, 151  
     output.h, 155  
 domain  
     elem\_t, 19  
 domainID  
     initialCondition.c, 91  
     initialCondition.h, 93  
 dRdU  
     linearSolver.c, 99  
 dt  
     elem\_t, 19  
 dtLoc  
     elem\_t, 20  
 dxRef  
     mesh.c, 134  
     mesh.h, 142  
 dyn2DcgsizeArray  
     memTools.c, 118  
     memTools.h, 123  
 dyn2DdblArray  
     memTools.c, 119  
     memTools.h, 124  
 dyn2DintArray  
     memTools.c, 119  
     memTools.h, 124  
 dyn3DdblArray  
     memTools.c, 119  
     memTools.h, 124  
 dyn3DintArray  
     memTools.c, 121  
     memTools.h, 126  
 dyn4DdblArray  
     memTools.c, 121  
     memTools.h, 126  
 dynStringArray  
     memTools.c, 122  
     memTools.h, 127  
 E  
     main.h, 114  
 elem  
     mesh.c, 134  
     mesh.h, 142  
     recordPoint\_t, 27  
     side\_t, 30  
 elem\_t, 17  
     area, 18  
     areaq, 18  
     bary, 19  
     cVar, 19  
     cVarStage, 19  
     domain, 19  
     dt, 19  
     dtLoc, 20  
     elemType, 20  
     firstSide, 20  
     id, 20  
     innerSides, 20  
     next, 21  
     nGP, 21  
     node, 21  
     pVar, 21  
     source, 21  
     sx, 22  
     sy, 22  
     u\_t, 22  
     u\_x, 22  
     u\_y, 22  
     venkEps\_sq, 23  
     wGP, 23  
     xGP, 23  
 elemType  
     elem\_t, 20  
 eps2newton  
     linearSolver.c, 99  
     linearSolver.h, 106  
 eps2newton\_sq  
     linearSolver.c, 99  
     linearSolver.h, 106  
 epsGMRES  
     linearSolver.c, 100  
     linearSolver.h, 106  
 equation.c  
     cp, 54  
     doCalcSource, 54  
     gam, 55  
     gam1, 55  
     gam1q, 55  
     gam2, 55  
     iFlux, 55  
     intExactFunc, 56  
     mu, 56  
     pi, 56  
     Pr, 56  
     R, 56  
     sourceFunc, 57  
     sqrt2, 57  
     sqrt3, 57  
     sqrt3q, 57  
 equation.h

- cp, 59
- doCalcSource, 59
- gam, 59
- gam1, 59
- gam1q, 60
- gam2, 60
- iFlux, 60
- intExactFunc, 60
- mu, 60
- pi, 61
- Pr, 61
- R, 61
- sourceFunc, 61
- sqrt2, 61
- sqrt3, 62
- sqrt3q, 62
- equationOfState.c
  - charCons, 63
  - consChar, 63
  - consPrim, 64
  - primCons, 64
- equationOfState.h
  - charCons, 66
  - consChar, 66
  - consPrim, 67
  - primCons, 67
- evalRecordPoints
  - analyze.c, 41
- evalSource
  - source.c, 174
- exactFunc
  - boundary\_t, 12
  - exactFunction.c, 68
  - exactFunction.h, 70
- exactFunction.c
  - exactFunc, 68
- exactFunction.h
  - exactFunc, 70
- exactRiemann
  - exactRiemann.c, 72
  - exactRiemann.h, 74
- exactRiemann.c
  - exactRiemann, 72
  - G, 73
  - nIter, 73
  - preFun, 73
  - tol, 73
- exactRiemann.h
  - exactRiemann, 74
- EXACTSOL
  - main.h, 113
- explicitTimeStepEuler
  - timeDiscretization.c, 178
- explicitTimeStepRK
  - timeDiscretization.c, 178
- F\_X0
  - timeDiscretization.c, 181
- F\_XK
  - timeDiscretization.c, 182
- fillCmds
  - readInTools.c, 158
  - readInTools.h, 164
- findCmd
  - readInTools.c, 159
- finiteVolume.c
  - fluxFunction, 77
  - fvTimeDerivative, 76
  - spatialOrder, 77
- finiteVolume.h
  - fluxFunction, 78
  - fvTimeDerivative, 78
  - spatialOrder, 78
- firstBC
  - boundary.c, 49
  - boundary.h, 53
- firstBCside
  - mesh.c, 134
  - mesh.h, 142
- firstCmd
  - readInTools.c, 163
- firstElem
  - mesh.c, 135
  - mesh.h, 143
- firstNode
  - mesh.c, 135
  - mesh.h, 143
- firstPressureSide
  - wing\_t, 37
- firstSide
  - elem\_t, 20
  - mesh.c, 135
  - mesh.h, 143
- firstSuctionSide
  - wing\_t, 37
- flux
  - side\_t, 30
- flux\_ausmd
  - fluxCalculation.c, 81
- flux\_ausmdv
  - fluxCalculation.c, 82
- flux\_cen
  - fluxCalculation.c, 82
- flux\_god
  - fluxCalculation.c, 83
- flux\_hll
  - fluxCalculation.c, 84
- flux\_hllc
  - fluxCalculation.c, 84
- flux\_hlle
  - fluxCalculation.c, 85
- flux\_lxf
  - fluxCalculation.c, 86
- flux\_roe
  - fluxCalculation.c, 86
- flux\_stw
  - fluxCalculation.c, 87

- flux\_vanleer
  - fluxCalculation.c, 88
- fluxCalculation
  - fluxCalculation.c, 88
  - fluxCalculation.h, 89
- fluxCalculation.c
  - convectiveFlux, 80
  - flux\_ausmd, 81
  - flux\_ausmdv, 82
  - flux\_cen, 82
  - flux\_god, 83
  - flux\_hll, 84
  - flux\_hllc, 84
  - flux\_hlle, 85
  - flux\_lxf, 86
  - flux\_roe, 86
  - flux\_stw, 87
  - flux\_vanleer, 88
  - fluxCalculation, 88
- fluxCalculation.h
  - fluxCalculation, 89
- fluxFunction
  - finiteVolume.c, 77
  - finiteVolume.h, 78
  - main.h, 114
- fvTimeDerivative
  - finiteVolume.c, 76
  - finiteVolume.h, 78
- G
  - exactRiemann.c, 73
- gam
  - equation.c, 55
  - equation.h, 59
- gam1
  - equation.c, 55
  - equation.h, 59
- gam1q
  - equation.c, 55
  - equation.h, 60
- gam2
  - equation.c, 55
  - equation.h, 60
- gamEW
  - linearSolver.c, 100
  - linearSolver.h, 106
- generalParameters
  - main.h, 114
- getBool
  - readInTools.c, 159
  - readInTools.h, 165
- getDbI
  - readInTools.c, 160
  - readInTools.h, 165
- getDbIArray
  - readInTools.c, 160
  - readInTools.h, 166
- getInt
  - readInTools.c, 161
- readInTools.h, 166
- getIntArray
  - readInTools.c, 161
  - readInTools.h, 167
- getStr
  - readInTools.c, 162
  - readInTools.h, 167
- globalResidual
  - analyze.c, 41
  - analyze.h, 45
- GMRES\_M
  - linearSolver.c, 96
  - linearSolver.h, 105
- GOD
  - main.h, 114
- GP
  - side\_t, 30
- gridFile
  - mesh.c, 135
  - mesh.h, 143
- hasExactSolution
  - analyze.c, 42
  - analyze.h, 46
- heatFlux
  - boundary\_t, 13
- HLL
  - main.h, 114
- HLLC
  - main.h, 114
- HLLE
  - main.h, 114
- icType
  - initialCondition.c, 91
  - initialCondition.h, 93
- id
  - elem\_t, 20
  - node\_t, 24
  - side\_t, 31
- iFlux
  - equation.c, 55
  - equation.h, 60
- iMax
  - cartMesh\_t, 15
- implicitTimeStep
  - timeDiscretization.c, 179
- INFLOW
  - main.h, 113
- inIiterationNumber
  - timeDiscretization.c, 182
  - timeDiscretization.h, 189
- initialCondition.c
  - alpha, 91
  - domainID, 91
  - icType, 91
  - nDomains, 91
  - refState, 92
  - rp1Dinterface, 92

- initialCondition.h
  - alpha, [93](#)
  - domainID, [93](#)
  - icType, [93](#)
  - nDomains, [93](#)
  - refState, [94](#)
  - rp1Dinterface, [94](#)
- innerSides
  - elem\_t, [20](#)
- intExactFunc
  - equation.c, [56](#)
  - equation.h, [60](#)
- ioFile
  - recordPoint\_t, [27](#)
- ioFormat
  - main.h, [116](#)
- IOiterInterval
  - output.c, [151](#)
  - output.h, [155](#)
- IOtimeInterval
  - output.c, [151](#)
  - output.h, [155](#)
- isAdiabatic
  - boundary\_t, [13](#)
- isImplicit
  - timeDiscretization.c, [182](#)
  - timeDiscretization.h, [189](#)
- isPeriodic
  - boundary.c, [49](#)
  - boundary.h, [53](#)
- isRestart
  - timeDiscretization.c, [182](#)
  - timeDiscretization.h, [190](#)
- isRotated
  - sideList\_t, [33](#)
- isStationary
  - timeDiscretization.c, [182](#)
  - timeDiscretization.h, [190](#)
- isTemperaturePrescribed
  - boundary\_t, [13](#)
- isTimeStep1D
  - timeDiscretization.c, [183](#)
  - timeDiscretization.h, [190](#)
- iter
  - outputTime\_t, [25](#)
- iVisuProg
  - output.c, [152](#)
  - output.h, [155](#)
- jMax
  - cartMesh\_t, [15](#)
- key
  - cmd\_t, [16](#)
- LEFT
  - main.h, [113](#)
- len
  - side\_t, [31](#)
- limiter
  - reconstruction.c, [170](#)
  - reconstruction.h, [172](#)
- limiterBarthJespersen
  - reconstruction.c, [169](#)
- limiterFunction
  - main.h, [116](#)
- limiterVenkatakrishnan
  - reconstruction.c, [170](#)
- linearSolver.c
  - buildMatrix, [96](#)
  - calcDinv, [96](#)
  - D, [98](#)
  - deltaXstar, [99](#)
  - Dinv, [99](#)
  - dRdU, [99](#)
  - eps2newton, [99](#)
  - eps2newton\_sq, [99](#)
  - epsGMRES, [100](#)
  - gamEW, [100](#)
  - GMRES\_M, [96](#)
  - LUSGS, [97](#)
  - matrixVector, [97](#)
  - nGMRESiterGlobal, [100](#)
  - nInnerGMRES, [100](#)
  - nInnerNewton, [100](#)
  - nKdim, [101](#)
  - nNewtonIter, [101](#)
  - nNewtonIterGlobal, [101](#)
  - R0, [101](#)
  - R\_XK, [101](#)
  - rEps0, [102](#)
  - srEps0, [102](#)
  - usePrecond, [102](#)
  - V, [102](#)
  - vectorDotProduct, [98](#)
  - W, [102](#)
  - XK, [103](#)
  - Z, [103](#)
- linearSolver.h
  - eps2newton, [106](#)
  - eps2newton\_sq, [106](#)
  - epsGMRES, [106](#)
  - gamEW, [106](#)
  - GMRES\_M, [105](#)
  - nGMRESiterGlobal, [106](#)
  - nInnerGMRES, [107](#)
  - nInnerNewton, [107](#)
  - nKdim, [107](#)
  - nNewtonIter, [107](#)
  - nNewtonIterGlobal, [107](#)
  - R\_XK, [108](#)
  - rEps0, [108](#)
  - srEps0, [108](#)
  - usePrecond, [108](#)
  - vectorDotProduct, [105](#)
  - XK, [108](#)
- LUSGS



- linearSolver.c, [97](#)
- LXF
  - main.h, [114](#)
- main
  - main.c, [110](#)
- main.c
  - main, [110](#)
- main.h
  - AUSMD, [114](#)
  - AUSMDV, [114](#)
  - BARTHJESPERSEN, [116](#)
  - BOTTOM, [113](#)
  - boundaryConditionType, [112](#)
  - CARTESIAN, [116](#)
  - cartesianMeshSides, [113](#)
  - CD, [113](#)
  - CEN, [114](#)
  - CGNS, [116](#)
  - CHARACTERISTIC, [113](#)
  - CL, [113](#)
  - clcdResiduals, [113](#)
  - conservativeVariables, [113](#)
  - CSV, [116](#)
  - CURVE, [116](#)
  - directions, [114](#)
  - E, [114](#)
  - EXACTSOL, [113](#)
  - fluxFunction, [114](#)
  - generalParameters, [114](#)
  - GOD, [114](#)
  - HLL, [114](#)
  - HLLC, [114](#)
  - HLLE, [114](#)
  - INFLOW, [113](#)
  - ioFormat, [116](#)
  - LEFT, [113](#)
  - limiterFunction, [116](#)
  - LXF, [114](#)
  - meshType, [116](#)
  - MX, [114](#)
  - MY, [114](#)
  - NBC, [116](#)
  - NDIM, [116](#)
  - NVAR, [116](#)
  - OUTFLOW, [113](#)
  - P, [117](#)
  - PERIODIC, [113](#)
  - PRESSURE\_OUT, [113](#)
  - primitiveVariables, [117](#)
  - RHO, [114](#)
  - RIGHT, [113](#)
  - ROE, [114](#)
  - SLIPWALL, [113](#)
  - STRLEN, [112](#)
  - STW, [114](#)
  - TOP, [113](#)
  - UNSTRUCTURED, [116](#)
  - VANLEER, [114](#)
  - VENKATAKRISHNAN, [116](#)
  - VX, [117](#)
  - VY, [117](#)
  - WALL, [113](#)
  - X, [114](#)
  - Y, [114](#)
  - matrixVector
    - linearSolver.c, [97](#)
  - maxIter
    - timeDiscretization.c, [183](#)
    - timeDiscretization.h, [190](#)
  - memTools.c
    - dyn2DcgsizeArray, [118](#)
    - dyn2DdblArray, [119](#)
    - dyn2DintArray, [119](#)
    - dyn3DdblArray, [119](#)
    - dyn3DintArray, [121](#)
    - dyn4DdblArray, [121](#)
    - dynStringArray, [122](#)
  - memTools.h
    - dyn2DcgsizeArray, [123](#)
    - dyn2DdblArray, [124](#)
    - dyn2DintArray, [124](#)
    - dyn3DdblArray, [124](#)
    - dyn3DintArray, [126](#)
    - dyn4DdblArray, [126](#)
    - dynStringArray, [127](#)
  - mesh.c
    - BCside, [134](#)
    - cartMesh, [134](#)
    - compare, [129](#)
    - createCartMesh, [130](#)
    - createElemInfo, [130](#)
    - createMesh, [131](#)
    - createReconstructionInfo, [131](#)
    - createSideInfo, [131](#)
    - dxRef, [134](#)
    - elem, [134](#)
    - firstBCside, [134](#)
    - firstElem, [135](#)
    - firstNode, [135](#)
    - firstSide, [135](#)
    - gridFile, [135](#)
    - meshFormat, [135](#)
    - meshType, [136](#)
    - nBCsides, [136](#)
    - nElems, [136](#)
    - nInnerSides, [136](#)
    - nNodes, [136](#)
    - nQuads, [137](#)
    - nSides, [137](#)
    - nTrias, [137](#)
    - parameterFile, [137](#)
    - readCGNS, [132](#)
    - readEMC2, [132](#)
    - readGmsh, [133](#)
    - side, [137](#)
    - strIniCondFile, [138](#)

- strMeshFile, 138
- strMeshFormat, 138
- totalArea\_q, 138
- xMax, 138
- xMin, 139
- yMax, 139
- yMin, 139
- mesh.h
  - BCside, 142
  - cartMesh, 142
  - dxRef, 142
  - elem, 142
  - firstBCside, 142
  - firstElem, 143
  - firstNode, 143
  - firstSide, 143
  - gridFile, 143
  - meshFormat, 143
  - meshType, 144
  - nBCsides, 144
  - nElems, 144
  - nInnerSides, 144
  - nNodes, 144
  - nQuads, 145
  - nSides, 145
  - nTrias, 145
  - parameterFile, 145
  - side, 145
  - strIniCondFile, 146
  - strMeshFile, 146
  - strMeshFormat, 146
  - totalArea\_q, 146
  - xMax, 146
  - xMin, 147
  - yMax, 147
  - yMin, 147
- meshFormat
  - mesh.c, 135
  - mesh.h, 143
- meshType
  - main.h, 116
  - mesh.c, 136
  - mesh.h, 144
- mu
  - equation.c, 56
  - equation.h, 60
- MX
  - main.h, 114
- MY
  - main.h, 114
- n
  - side\_t, 31
- NBC
  - main.h, 116
- nBC
  - boundary.c, 49
  - boundary.h, 53
  - cartMesh\_t, 15
- nBCsides
  - mesh.c, 136
  - mesh.h, 144
- NDIM
  - main.h, 116
- nDomains
  - initialCondition.c, 91
  - initialCondition.h, 93
- nElems
  - mesh.c, 136
  - mesh.h, 144
- next
  - boundary\_t, 13
  - cmd\_t, 16
  - elem\_t, 21
  - node\_t, 24
  - outputTime\_t, 26
  - side\_t, 31
  - sidePtr\_t, 35
- nextElemSide
  - side\_t, 31
- nGMRESiterGlobal
  - linearSolver.c, 100
  - linearSolver.h, 106
- nGP
  - elem\_t, 21
- nInnerGMRES
  - linearSolver.c, 100
  - linearSolver.h, 107
- nInnerNewton
  - linearSolver.c, 100
  - linearSolver.h, 107
- nInnerSides
  - mesh.c, 136
  - mesh.h, 144
- nIter
  - exactRiemann.c, 73
- nKdim
  - linearSolver.c, 101
  - linearSolver.h, 107
- nNewtonIter
  - linearSolver.c, 101
  - linearSolver.h, 107
- nNewtonIterGlobal
  - linearSolver.c, 101
  - linearSolver.h, 107
- nNodes
  - mesh.c, 136
  - mesh.h, 144
- node
  - elem\_t, 21
  - side\_t, 32
  - sideList\_t, 34
- node\_t, 24
  - id, 24
  - next, 24
  - x, 24
- nPoints

- recordPoint\_t, 27
- nQuads
  - mesh.c, 137
  - mesh.h, 145
- nRKstages
  - timeDiscretization.c, 183
  - timeDiscretization.h, 190
- nSides
  - mesh.c, 137
  - mesh.h, 145
- nTrias
  - mesh.c, 137
  - mesh.h, 145
- NVAR
  - main.h, 116
- OUTFLOW
  - main.h, 113
- output.c
  - cgnsFinalizeOutput, 149
  - cgnsOutput, 149
  - csvOutput, 150
  - curveOutput, 150
  - dataOutput, 151
  - doErrorOutput, 151
  - IOiterInterval, 151
  - IOtimeInterval, 151
  - iVisuProg, 152
  - outputTimes, 152
  - parameterFile, 152
  - resFile, 152
  - strOutFile, 152
- output.h
  - dataOutput, 154
  - doErrorOutput, 155
  - IOiterInterval, 155
  - IOtimeInterval, 155
  - iVisuProg, 155
  - outputTimes, 155
  - parameterFile, 156
  - resFile, 156
  - strOutFile, 156
- outputTime\_t, 25
  - iter, 25
  - next, 26
  - time, 26
- outputTimes
  - output.c, 152
  - output.h, 155
- P
  - main.h, 117
- parameterFile
  - mesh.c, 137
  - mesh.h, 145
  - output.c, 152
  - output.h, 156
- PERIODIC
  - main.h, 113
- pi
  - equation.c, 56
  - equation.h, 61
- Pr
  - equation.c, 56
  - equation.h, 61
- preFun
  - exactRiemann.c, 73
- PRESSURE\_OUT
  - main.h, 113
- prev
  - cmd\_t, 16
- primCons
  - equationOfState.c, 64
  - equationOfState.h, 67
- primitiveVariables
  - main.h, 117
- printIter
  - timeDiscretization.c, 183
  - timeDiscretization.h, 191
- printTime
  - timeDiscretization.c, 183
  - timeDiscretization.h, 191
- pVar
  - boundary\_t, 13
  - elem\_t, 21
  - side\_t, 32
- Q
  - timeDiscretization.c, 184
- R
  - equation.c, 56
  - equation.h, 61
- R0
  - linearSolver.c, 101
- R\_XK
  - linearSolver.c, 101
  - linearSolver.h, 108
- readCGNS
  - mesh.c, 132
- readEMC2
  - mesh.c, 132
- readGmsh
  - mesh.c, 133
- readInTools.c
  - countKeys, 158
  - deleteCmd, 158
  - fillCmds, 158
  - findCmd, 159
  - firstCmd, 163
  - getBool, 159
  - getDbI, 160
  - getDbIArray, 160
  - getIInt, 161
  - getIIntArray, 161
  - getStr, 162
- readInTools.h
  - countKeys, 164

- fillCmds, 164
- getBool, 165
- getDb, 165
- getDbArray, 166
- getInt, 166
- getIntArray, 167
- getStr, 167
- reconstruction.c
  - limiter, 170
  - limiterBarthJespersen, 169
  - limiterVenkatakrishnan, 170
  - spatialReconstruction, 170
  - venk\_k, 171
- reconstruction.h
  - limiter, 172
  - spatialReconstruction, 172
  - venk\_k, 172
- recordPoint
  - analyze.c, 42
  - analyze.h, 46
- recordPoint\_t, 26
  - elem, 27
  - ioFile, 27
  - nPoints, 27
  - x, 27
- refLength
  - wing\_t, 37
- refState
  - initialCondition.c, 92
  - initialCondition.h, 94
- rEps0
  - linearSolver.c, 102
  - linearSolver.h, 108
- resFile
  - output.c, 152
  - output.h, 156
- restartTime
  - timeDiscretization.c, 184
  - timeDiscretization.h, 191
- RHO
  - main.h, 114
- RIGHT
  - main.h, 113
- RKcoeff
  - timeDiscretization.c, 184
  - timeDiscretization.h, 191
- ROE
  - main.h, 114
- rp1Dinterface
  - initialCondition.c, 92
  - initialCondition.h, 94
- setBCatBarys
  - boundary.c, 48
  - boundary.h, 52
- setBCatSides
  - boundary.c, 49
  - boundary.h, 52
- side
  - mesh.c, 137
  - mesh.h, 145
  - sideList\_t, 34
  - sidePtr\_t, 35
- side\_t, 28
  - baryBaryDist, 29
  - baryBaryVec, 29
  - BC, 29
  - BCid, 29
  - BCtype, 30
  - connection, 30
  - elem, 30
  - flux, 30
  - GP, 30
  - id, 31
  - len, 31
  - n, 31
  - next, 31
  - nextElemSide, 31
  - node, 32
  - pVar, 32
  - w, 32
- sideList\_t, 33
  - BC, 33
  - isRotated, 33
  - node, 34
  - side, 34
- sidePtr\_t, 34
  - next, 35
  - side, 35
- SLIPWALL
  - main.h, 113
- source
  - elem\_t, 21
- source.c
  - calcSource, 174
  - evalSource, 174
- source.h
  - calcSource, 175
- sourceFunc
  - equation.c, 57
  - equation.h, 61
- spatialOrder
  - finiteVolume.c, 77
  - finiteVolume.h, 78
- spatialReconstruction
  - reconstruction.c, 170
  - reconstruction.h, 172
- sqrt2
  - equation.c, 57
  - equation.h, 61
- sqrt3
  - equation.c, 57
  - equation.h, 62
- sqrt3q
  - equation.c, 57
  - equation.h, 62
- src/analyze.c, 39

- src/analyze.h, 43
- src/boundary.c, 47
- src/boundary.h, 50
- src/equation.c, 53
- src/equation.h, 58
- src/equationOfState.c, 62
- src/equationOfState.h, 65
- src/exactFunction.c, 67
- src/exactFunction.h, 69
- src/exactRiemann.c, 71
- src/exactRiemann.h, 74
- src/finiteVolume.c, 75
- src/finiteVolume.h, 77
- src/fluxCalculation.c, 79
- src/fluxCalculation.h, 89
- src/initialCondition.c, 90
- src/initialCondition.h, 92
- src/linearSolver.c, 94
- src/linearSolver.h, 103
- src/main.c, 109
- src/main.h, 110
- src/memTools.c, 117
- src/memTools.h, 122
- src/mesh.c, 127
- src/mesh.h, 140
- src/output.c, 148
- src/output.h, 153
- src/readInTools.c, 156
- src/readInTools.h, 163
- src/reconstruction.c, 168
- src/reconstruction.h, 171
- src/source.c, 173
- src/source.h, 175
- src/timeDiscretization.c, 176
- src/timeDiscretization.h, 186
- srEps0
  - linearSolver.c, 102
  - linearSolver.h, 108
- startTime
  - timeDiscretization.c, 184
  - timeDiscretization.h, 191
- stopTime
  - timeDiscretization.c, 184
  - timeDiscretization.h, 192
- strIniCondFile
  - mesh.c, 138
  - mesh.h, 146
- STRLEN
  - main.h, 112
- strMeshFile
  - mesh.c, 138
  - mesh.h, 146
- strMeshFormat
  - mesh.c, 138
  - mesh.h, 146
- strOutFile
  - output.c, 152
  - output.h, 156
- STW
  - main.h, 114
- sx
  - elem\_t, 22
- sy
  - elem\_t, 22
- t
  - timeDiscretization.c, 185
  - timeDiscretization.h, 192
- temperature
  - boundary\_t, 14
- time
  - outputTime\_t, 26
- timeDisc
  - timeDiscretization.c, 179
  - timeDiscretization.h, 187
- timeDiscretization.c
  - abortResidual, 179
  - abortVariable, 180
  - abortVariableName, 180
  - calcTimeStep, 177
  - cdAbortResidual, 180
  - cfl, 180
  - clAbortResidual, 180
  - deltaX, 181
  - dfl, 181
  - doAbortOnCdResidual, 181
  - doAbortOnClResidual, 181
  - explicitTimeStepEuler, 178
  - explicitTimeStepRK, 178
  - F\_X0, 181
  - F\_XK, 182
  - implicitTimeStep, 179
  - iniliterationNumber, 182
  - isImplicit, 182
  - isRestart, 182
  - isStationary, 182
  - isTimeStep1D, 183
  - maxIter, 183
  - nRKstages, 183
  - printIter, 183
  - printTime, 183
  - Q, 184
  - restartTime, 184
  - RKcoeff, 184
  - startTime, 184
  - stopTime, 184
  - t, 185
  - timeDisc, 179
  - timeOrder, 185
  - timeOverall, 185
- timeDiscretization.h
  - abortResidual, 188
  - abortVariable, 188
  - abortVariableName, 188
  - cdAbortResidual, 188
  - cfl, 188
  - clAbortResidual, 188

- dfI, 189
  - doAbortOnCdResidual, 189
  - doAbortOnCIResidual, 189
  - inIterationNumber, 189
  - isImplicit, 189
  - isRestart, 190
  - isStationary, 190
  - isTimeStep1D, 190
  - maxIter, 190
  - nRKstages, 190
  - printIter, 191
  - printTime, 191
  - restartTime, 191
  - RKcoeff, 191
  - startTime, 191
  - stopTime, 192
  - t, 192
  - timeDisc, 187
  - timeOrder, 192
  - timeOverall, 192
- timeOrder
  - timeDiscretization.c, 185
  - timeDiscretization.h, 192
- timeOverall
  - timeDiscretization.c, 185
  - timeDiscretization.h, 192
- tol
  - exactRiemann.c, 73
- TOP
  - main.h, 113
- totalArea\_q
  - mesh.c, 138
  - mesh.h, 146
- u\_t
  - elem\_t, 22
- u\_x
  - elem\_t, 22
- u\_y
  - elem\_t, 22
- UNSTRUCTURED
  - main.h, 116
- usePrecond
  - linearSolver.c, 102
  - linearSolver.h, 108
- V
  - linearSolver.c, 102
- value
  - cmd\_t, 17
- VANLEER
  - main.h, 114
- vectorDotProduct
  - linearSolver.c, 98
  - linearSolver.h, 105
- venk\_k
  - reconstruction.c, 171
  - reconstruction.h, 172
- VENKATAKRISHNAN
  - main.h, 116
- venkEps\_sq
  - elem\_t, 23
- VX
  - main.h, 117
- VY
  - main.h, 117
- W
  - linearSolver.c, 102
- w
  - side\_t, 32
- WALL
  - main.h, 113
- wallId
  - wing\_t, 37
- wGP
  - elem\_t, 23
- wing
  - analyze.c, 42
  - analyze.h, 46
- wing\_t, 36
  - cd, 36
  - cl, 37
  - firstPressureSide, 37
  - firstSuctionSide, 37
  - refLength, 37
  - wallId, 37
  - wingBC, 38
- wingBC
  - wing\_t, 38
- X
  - main.h, 114
- x
  - node\_t, 24
  - recordPoint\_t, 27
- xGP
  - elem\_t, 23
- XK
  - linearSolver.c, 103
  - linearSolver.h, 108
- xMax
  - mesh.c, 138
  - mesh.h, 146
- xMin
  - mesh.c, 139
  - mesh.h, 147
- Y
  - main.h, 114
- yMax
  - mesh.c, 139
  - mesh.h, 147
- yMin
  - mesh.c, 139
  - mesh.h, 147
- Z
  - linearSolver.c, 103