

# FOORT

1.0

Generated by Doxygen 1.12.0



<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Namespace Documentation</b>	<b>9</b>
5.1 Config Namespace Reference	9
5.1.1 Typedef Documentation	10
5.1.1.1 ConfigCollection	10
5.1.1.2 SettingError	10
5.1.2 Function Documentation	10
5.1.2.1 GetGeodesicIntegrator()	10
5.1.2.2 GetMesh()	10
5.1.2.3 GetMetric()	10
5.1.2.4 GetOutputHandler()	10
5.1.2.5 GetSource()	11
5.1.2.6 GetViewScreen()	11
5.1.2.7 InitializeDiagnostics()	11
5.1.2.8 InitializeScreenOutput()	11
5.1.2.9 InitializeTerminations()	11
5.1.3 Variable Documentation	11
5.1.3.1 Output_Important_Default	11
5.1.3.2 Output_Other_Default	12
5.2 ConfigReader Namespace Reference	12
5.2.1 Variable Documentation	12
5.2.1.1 MaxStreamSize	12
5.3 Integrators Namespace Reference	12
5.3.1 Function Documentation	12
5.3.1.1 GetAdaptiveStep()	12
5.3.1.2 GetFullIntegratorDescription()	13
5.3.1.3 IntegrateGeodesicStep_RK4()	13
5.3.1.4 IntegrateGeodesicStep_Verlet()	13
5.3.2 Variable Documentation	13
5.3.2.1 delta_nodiv0	13
5.3.2.2 Derivative_hval	13
5.3.2.3 epsilon	13
5.3.2.4 IntegratorDescription	13

5.3.2.5 SmallestPossibleStepsize . . . . .	14
5.3.2.6 VerletVelocityTolerance . . . . .	14
5.4 tk Namespace Reference . . . . .	14
5.5 tk::internal Namespace Reference . . . . .	14
5.6 Utilities Namespace Reference . . . . .	14
5.6.1 Function Documentation . . . . .	14
5.6.1.1 GetDiagNameStrings() . . . . .	14
5.6.1.2 GetFirstLineInfoString() . . . . .	14
5.6.1.3 GetTimeStampString() . . . . .	14
<b>6 Class Documentation</b>	<b>15</b>
6.1 BosonStarMetric Class Reference . . . . .	15
6.1.1 Constructor & Destructor Documentation . . . . .	16
6.1.1.1 BosonStarMetric() . . . . .	16
6.1.2 Member Function Documentation . . . . .	16
6.1.2.1 getFullDescriptionStr() . . . . .	16
6.1.2.2 getMetric_dd() . . . . .	16
6.1.2.3 getMetric_uu() . . . . .	16
6.1.3 Member Data Documentation . . . . .	16
6.1.3.1 m_mSpline . . . . .	16
6.1.3.2 m_PhiSpline . . . . .	17
6.2 BoundarySphereTermination Class Reference . . . . .	17
6.2.1 Constructor & Destructor Documentation . . . . .	18
6.2.1.1 BoundarySphereTermination() . . . . .	18
6.2.2 Member Function Documentation . . . . .	18
6.2.2.1 CheckTermination() . . . . .	18
6.2.2.2 getFullDescriptionStr() . . . . .	18
6.2.3 Member Data Documentation . . . . .	18
6.2.3.1 TermOptions . . . . .	18
6.3 BoundarySphereTermOptions Struct Reference . . . . .	18
6.3.1 Constructor & Destructor Documentation . . . . .	19
6.3.1.1 BoundarySphereTermOptions() . . . . .	19
6.3.2 Member Data Documentation . . . . .	19
6.3.2.1 rLogScale . . . . .	19
6.3.2.2 SphereRadius . . . . .	19
6.4 ClosestRadiusDiagnostic Class Reference . . . . .	20
6.4.1 Constructor & Destructor Documentation . . . . .	21
6.4.1.1 ClosestRadiusDiagnostic() . . . . .	21
6.4.2 Member Function Documentation . . . . .	21
6.4.2.1 FinalDataValDistance() . . . . .	21
6.4.2.2 getFinalDataVal() . . . . .	21
6.4.2.3 getFullDataStr() . . . . .	21

6.4.2.4 getFullDescriptionStr()	21
6.4.2.5 getNameStr()	21
6.4.2.6 Reset()	21
6.4.2.7 UpdateData()	22
6.4.3 Member Data Documentation	22
6.4.3.1 DiagOptions	22
6.4.3.2 m_ClosestRadius	22
6.5 ClosestRadiusOptions Struct Reference	22
6.5.1 Constructor & Destructor Documentation	23
6.5.1.1 ClosestRadiusOptions()	23
6.5.2 Member Data Documentation	23
6.5.2.1 RLogScale	23
6.6 ConfigReader::ConfigCollection Class Reference	23
6.6.1 Member Typedef Documentation	24
6.6.1.1 ConfigSettingValue	24
6.6.2 Member Function Documentation	24
6.6.2.1 DisplayCollection()	24
6.6.2.2 DisplaySetting()	24
6.6.2.3 DisplayTabs()	24
6.6.2.4 Exists()	25
6.6.2.5 GetSettingIndex()	25
6.6.2.6 IsCollection() [1/2]	25
6.6.2.7 IsCollection() [2/2]	25
6.6.2.8 LookupValue() [1/2]	25
6.6.2.9 LookupValue() [2/2]	25
6.6.2.10 LookupValueInteger() [1/7]	25
6.6.2.11 LookupValueInteger() [2/7]	25
6.6.2.12 LookupValueInteger() [3/7]	26
6.6.2.13 LookupValueInteger() [4/7]	26
6.6.2.14 LookupValueInteger() [5/7]	26
6.6.2.15 LookupValueInteger() [6/7]	26
6.6.2.16 LookupValueInteger() [7/7]	26
6.6.2.17 NrSettings()	26
6.6.2.18 operator[]() [1/2]	26
6.6.2.19 operator[]() [2/2]	26
6.6.2.20 ReadCollection()	27
6.6.2.21 ReadFile()	27
6.6.2.22 ReadSettingName()	27
6.6.2.23 ReadSettingSpecificChar()	27
6.6.2.24 ReadSettingValue()	27
6.6.3 Member Data Documentation	27
6.6.3.1 m_Settings	27

6.7 ConfigReader::ConfigReaderException Class Reference . . . . .	27
6.7.1 Constructor & Destructor Documentation . . . . .	28
6.7.1.1 ConfigReaderException() . . . . .	28
6.7.2 Member Function Documentation . . . . .	28
6.7.2.1 trace() . . . . .	28
6.7.3 Member Data Documentation . . . . .	28
6.7.3.1 m_settingtrace . . . . .	28
6.8 ConfigReader::ConfigCollection::ConfigSetting Struct Reference . . . . .	28
6.8.1 Member Data Documentation . . . . .	28
6.8.1.1 SettingName . . . . .	28
6.8.1.2 SettingValue . . . . .	29
6.9 Diagnostic Class Reference . . . . .	29
6.9.1 Constructor & Destructor Documentation . . . . .	29
6.9.1.1 Diagnostic() [1/2] . . . . .	29
6.9.1.2 Diagnostic() [2/2] . . . . .	30
6.9.1.3 ~Diagnostic() . . . . .	30
6.9.2 Member Function Documentation . . . . .	30
6.9.2.1 DecideUpdate() . . . . .	30
6.9.2.2 FinalDataValDistance() . . . . .	30
6.9.2.3 getFinalDataVal() . . . . .	30
6.9.2.4 getFullDataStr() . . . . .	30
6.9.2.5 getFullDescriptionStr() . . . . .	30
6.9.2.6 getNameStr() . . . . .	31
6.9.2.7 Reset() . . . . .	31
6.9.2.8 UpdateData() . . . . .	31
6.9.3 Member Data Documentation . . . . .	31
6.9.3.1 m_OwnerGeodesic . . . . .	31
6.9.3.2 m_StepsSinceUpdated . . . . .	31
6.10 DiagnosticOptions Struct Reference . . . . .	31
6.10.1 Constructor & Destructor Documentation . . . . .	32
6.10.1.1 DiagnosticOptions() . . . . .	32
6.10.1.2 ~DiagnosticOptions() . . . . .	32
6.10.2 Member Data Documentation . . . . .	32
6.10.2.1 theUpdateFrequency . . . . .	32
6.11 EmissionModel Struct Reference . . . . .	32
6.11.1 Constructor & Destructor Documentation . . . . .	33
6.11.1.1 ~EmissionModel() . . . . .	33
6.11.2 Member Function Documentation . . . . .	33
6.11.2.1 GetEmission() . . . . .	33
6.11.2.2 getFullDescriptionStr() . . . . .	33
6.12 EquatorialEmissionDiagnostic Class Reference . . . . .	33
6.12.1 Constructor & Destructor Documentation . . . . .	35

6.12.1.1 EquatorialEmissionDiagnostic()	35
6.12.2 Member Function Documentation	35
6.12.2.1 FinalDataValDistance()	35
6.12.2.2 getFinalDataVal()	35
6.12.2.3 getFullDataStr()	35
6.12.2.4 getFullDescriptionStr()	35
6.12.2.5 getNameStr()	36
6.12.2.6 Reset()	36
6.12.2.7 UpdateData()	36
6.12.3 Member Data Documentation	36
6.12.3.1 DiagOptions	36
6.12.3.2 m_Intensity	36
6.13 EquatorialEmissionOptions Struct Reference	36
6.13.1 Constructor & Destructor Documentation	37
6.13.1.1 EquatorialEmissionOptions()	37
6.13.2 Member Data Documentation	38
6.13.2.1 EquatPassUpperBound	38
6.13.2.2 GeometricFudgeFactor	38
6.13.2.3 RedShiftPower	38
6.13.2.4 RLogScale	38
6.13.2.5 TheEmissionModel	38
6.13.2.6 TheFluidVelocityModel	38
6.14 EquatorialPassesDiagnostic Class Reference	38
6.14.1 Constructor & Destructor Documentation	39
6.14.1.1 EquatorialPassesDiagnostic()	39
6.14.2 Member Function Documentation	40
6.14.2.1 FinalDataValDistance()	40
6.14.2.2 getFinalDataVal()	40
6.14.2.3 getFullDataStr()	40
6.14.2.4 getFullDescriptionStr()	40
6.14.2.5 getNameStr()	40
6.14.2.6 Reset()	40
6.14.2.7 UpdateData()	40
6.14.3 Member Data Documentation	41
6.14.3.1 DiagOptions	41
6.14.3.2 m_EquatPasses	41
6.14.3.3 m_PrevTheta	41
6.15 EquatorialPassesOptions Struct Reference	41
6.15.1 Constructor & Destructor Documentation	42
6.15.1.1 EquatorialPassesOptions()	42
6.15.2 Member Data Documentation	42
6.15.2.1 Threshold	42

6.16 FlatSpaceMetric Class Reference	42
6.16.1 Constructor & Destructor Documentation	43
6.16.1.1 FlatSpaceMetric()	43
6.16.2 Member Function Documentation	43
6.16.2.1 getFullDescriptionStr()	43
6.16.2.2 getMetric_dd()	43
6.16.2.3 getMetric_uu()	43
6.17 FluidVelocityModel Struct Reference	44
6.17.1 Constructor & Destructor Documentation	44
6.17.1.1 FluidVelocityModel()	44
6.17.1.2 ~FluidVelocityModel()	44
6.17.2 Member Function Documentation	44
6.17.2.1 GetFourVelocityd()	44
6.17.2.2 getFullDescriptionStr()	45
6.17.3 Member Data Documentation	45
6.17.3.1 m_theMetric	45
6.18 FourColorScreenDiagnostic Class Reference	45
6.18.1 Constructor & Destructor Documentation	46
6.18.1.1 FourColorScreenDiagnostic()	46
6.18.2 Member Function Documentation	46
6.18.2.1 FinalDataValDistance()	46
6.18.2.2 getFinalDataVal()	46
6.18.2.3 getFullDataStr()	46
6.18.2.4 getFullDescriptionStr()	47
6.18.2.5 getNameStr()	47
6.18.2.6 Reset()	47
6.18.2.7 UpdateData()	47
6.18.3 Member Data Documentation	47
6.18.3.1 m_quadrant	47
6.19 GeneralCircularRadialFluid Struct Reference	47
6.19.1 Constructor & Destructor Documentation	48
6.19.1.1 GeneralCircularRadialFluid()	48
6.19.2 Member Function Documentation	49
6.19.2.1 FindISCO()	49
6.19.2.2 GetChrstrRaisedDer()	49
6.19.2.3 GetCircularVelocityd()	49
6.19.2.4 GetFourVelocityd()	49
6.19.2.5 getFullDescriptionStr()	49
6.19.2.6 GetInsideISCOCircularVelocityd()	49
6.19.2.7 GetRadialVelocityd()	49
6.19.3 Member Data Documentation	50
6.19.3.1 m_betaPhi	50



6.19.3.2 m_betaR . . . . .	50
6.19.3.3 m_ISCOexists . . . . .	50
6.19.3.4 m_ISCOpphi . . . . .	50
6.19.3.5 m_ISCOpt . . . . .	50
6.19.3.6 m_ISCOr . . . . .	50
6.19.3.7 m_subKeplerParam . . . . .	50
6.20 GeneralSingularityTermination Class Reference . . . . .	50
6.20.1 Constructor & Destructor Documentation . . . . .	51
6.20.1.1 GeneralSingularityTermination() . . . . .	51
6.20.2 Member Function Documentation . . . . .	51
6.20.2.1 CheckTermination() . . . . .	51
6.20.2.2 getFullDescriptionStr() . . . . .	52
6.20.2.3 SingularityToString() . . . . .	52
6.20.3 Member Data Documentation . . . . .	52
6.20.3.1 TermOptions . . . . .	52
6.21 GeneralSingularityTermOptions Struct Reference . . . . .	52
6.21.1 Constructor & Destructor Documentation . . . . .	53
6.21.1.1 GeneralSingularityTermOptions() . . . . .	53
6.21.2 Member Data Documentation . . . . .	53
6.21.2.1 Epsilon . . . . .	53
6.21.2.2 OutputToConsole . . . . .	53
6.21.2.3 rLogScale . . . . .	53
6.21.2.4 Singularities . . . . .	53
6.22 Geodesic Class Reference . . . . .	54
6.22.1 Constructor & Destructor Documentation . . . . .	54
6.22.1.1 Geodesic() [1/3] . . . . .	54
6.22.1.2 Geodesic() [2/3] . . . . .	54
6.22.1.3 Geodesic() [3/3] . . . . .	55
6.22.2 Member Function Documentation . . . . .	55
6.22.2.1 getAllOutputStr() . . . . .	55
6.22.2.2 getCurrentLambda() . . . . .	55
6.22.2.3 getCurrentPos() . . . . .	55
6.22.2.4 getCurrentVel() . . . . .	55
6.22.2.5 getDiagnosticFinalValue() . . . . .	55
6.22.2.6 getScreenIndex() . . . . .	55
6.22.2.7 getTermCondition() . . . . .	55
6.22.2.8 operator=() . . . . .	55
6.22.2.9 Reset() . . . . .	56
6.22.2.10 Update() . . . . .	56
6.22.3 Member Data Documentation . . . . .	56
6.22.3.1 m_AllDiagnostics . . . . .	56
6.22.3.2 m_AllTerminations . . . . .	56

6.22.3.3 m_curLambda	56
6.22.3.4 m_CurrentPos	56
6.22.3.5 m_CurrentVel	56
6.22.3.6 m_ScreenIndex	56
6.22.3.7 m_TermCond	56
6.22.3.8 m_theIntegrator	57
6.22.3.9 m_theMetric	57
6.22.3.10 m_theSource	57
6.23 GeodesicOutputHandler Class Reference	57
6.23.1 Constructor & Destructor Documentation	58
6.23.1.1 GeodesicOutputHandler() [1/2]	58
6.23.1.2 GeodesicOutputHandler() [2/2]	58
6.23.2 Member Function Documentation	58
6.23.2.1 GetFileName()	58
6.23.2.2 getFullDescriptionStr()	58
6.23.2.3 NewGeodesicOutput()	59
6.23.2.4 OpenForFirstTime()	59
6.23.2.5 OutputFinished()	59
6.23.2.6 PrepareForOutput()	59
6.23.2.7 WriteCachedOutputToFile()	59
6.23.3 Member Data Documentation	59
6.23.3.1 m_AllCachedData	59
6.23.3.2 m_CurrentFullFiles	59
6.23.3.3 m_CurrentGeodesicsInFile	59
6.23.3.4 m_DiagNames	59
6.23.3.5 m_FileExtension	60
6.23.3.6 m_FilePrefix	60
6.23.3.7 m_FirstLineInfoString	60
6.23.3.8 m_nrGeodesicsPerFile	60
6.23.3.9 m_nrOutputsToCache	60
6.23.3.10 m_PrevCached	60
6.23.3.11 m_PrintFirstLineInfo	60
6.23.3.12 m_TimeStamp	60
6.23.3.13 m_WriteToConsole	60
6.24 GeodesicPositionDiagnostic Class Reference	61
6.24.1 Constructor & Destructor Documentation	62
6.24.1.1 GeodesicPositionDiagnostic()	62
6.24.2 Member Function Documentation	62
6.24.2.1 FinalDataValDistance()	62
6.24.2.2 getFinalDataVal()	62
6.24.2.3 getFullDataStr()	62
6.24.2.4 getFullDescriptionStr()	62

6.24.2.5 getNameStr()	62
6.24.2.6 Reset()	62
6.24.2.7 UpdateData()	63
6.24.3 Member Data Documentation	63
6.24.3.1 DiagOptions	63
6.24.3.2 m_AllSavedPoints	63
6.25 GeodesicPositionOptions Struct Reference	63
6.25.1 Constructor & Destructor Documentation	64
6.25.1.1 GeodesicPositionOptions()	64
6.25.2 Member Data Documentation	64
6.25.2.1 OutputNrSteps	64
6.26 GLMJohnsonSUEmission Struct Reference	64
6.26.1 Constructor & Destructor Documentation	65
6.26.1.1 GLMJohnsonSUEmission()	65
6.26.2 Member Function Documentation	65
6.26.2.1 GetEmission()	65
6.26.2.2 getFullDescriptionStr()	65
6.26.3 Member Data Documentation	65
6.26.3.1 m_gamma	65
6.26.3.2 m_mu	65
6.26.3.3 m_sigma	66
6.27 HorizonTermination Class Reference	66
6.27.1 Constructor & Destructor Documentation	67
6.27.1.1 HorizonTermination()	67
6.27.2 Member Function Documentation	67
6.27.2.1 CheckTermination()	67
6.27.2.2 getFullDescriptionStr()	67
6.27.3 Member Data Documentation	67
6.27.3.1 TermOptions	67
6.28 HorizonTermOptions Struct Reference	67
6.28.1 Constructor & Destructor Documentation	68
6.28.1.1 HorizonTermOptions()	68
6.28.2 Member Data Documentation	68
6.28.2.1 AtHorizonEps	68
6.28.2.2 HorizonRadius	68
6.28.2.3 rLogScale	68
6.29 InputCertainPixelsMesh Class Reference	69
6.29.1 Constructor & Destructor Documentation	69
6.29.1.1 InputCertainPixelsMesh() [1/3]	69
6.29.1.2 InputCertainPixelsMesh() [2/3]	70
6.29.1.3 InputCertainPixelsMesh() [3/3]	70
6.29.2 Member Function Documentation	70

6.29.2.1 EndCurrentLoop()	70
6.29.2.2 GeodesicFinished()	70
6.29.2.3 getCurNrGeodesics()	70
6.29.2.4 getFullDescriptionStr()	70
6.29.2.5 getNewInitConds()	71
6.29.2.6 IsFinished()	71
6.29.3 Member Data Documentation	71
6.29.3.1 m_Finished	71
6.29.3.2 m_PixelsToIntegrate	71
6.29.3.3 m_RowColumnSize	71
6.29.3.4 m_TotalPixels	71
6.30 JohannsenMetric Class Reference	71
6.30.1 Constructor & Destructor Documentation	73
6.30.1.1 JohannsenMetric() [1/2]	73
6.30.1.2 JohannsenMetric() [2/2]	73
6.30.2 Member Function Documentation	73
6.30.2.1 getFullDescriptionStr()	73
6.30.2.2 getMetric_dd()	73
6.30.2.3 getMetric_uu()	73
6.30.3 Member Data Documentation	73
6.30.3.1 m_alpha13Param	73
6.30.3.2 m_alpha22Param	74
6.30.3.3 m_alpha52Param	74
6.30.3.4 m_aParam	74
6.30.3.5 m_eps3Param	74
6.31 KerrMetric Class Reference	74
6.31.1 Constructor & Destructor Documentation	75
6.31.1.1 KerrMetric() [1/2]	75
6.31.1.2 KerrMetric() [2/2]	75
6.31.2 Member Function Documentation	76
6.31.2.1 getFullDescriptionStr()	76
6.31.2.2 getMetric_dd()	76
6.31.2.3 getMetric_uu()	76
6.31.3 Member Data Documentation	76
6.31.3.1 m_aParam	76
6.31.3.2 m_mParam	76
6.32 KerrSchildMetric Class Reference	76
6.32.1 Constructor & Destructor Documentation	77
6.32.1.1 KerrSchildMetric() [1/2]	77
6.32.1.2 KerrSchildMetric() [2/2]	78
6.32.2 Member Function Documentation	78
6.32.2.1 getFullDescriptionStr()	78

6.32.2.2 getMetric_dd()	78
6.32.2.3 getMetric_uu()	78
6.32.3 Member Data Documentation	78
6.32.3.1 m_aParam	78
6.33 MankoNovikovMetric Class Reference	79
6.33.1 Constructor & Destructor Documentation	80
6.33.1.1 MankoNovikovMetric() [1/2]	80
6.33.1.2 MankoNovikovMetric() [2/2]	80
6.33.2 Member Function Documentation	80
6.33.2.1 getFullDescriptionStr()	80
6.33.2.2 getMetric_dd()	80
6.33.2.3 getMetric_uu()	80
6.33.3 Member Data Documentation	81
6.33.3.1 m_alpha3Param	81
6.33.3.2 m_alphaParam	81
6.33.3.3 m_aParam	81
6.33.3.4 m_kParam	81
6.34 Mesh Class Reference	81
6.34.1 Constructor & Destructor Documentation	82
6.34.1.1 Mesh()	82
6.34.1.2 ~Mesh()	82
6.34.2 Member Function Documentation	82
6.34.2.1 EndCurrentLoop()	82
6.34.2.2 GeodesicFinished()	82
6.34.2.3 getCurNrGeodesics()	82
6.34.2.4 getFullDescriptionStr()	82
6.34.2.5 getNewInitConds()	83
6.34.2.6 IsFinished()	83
6.34.3 Member Data Documentation	83
6.34.3.1 m_DistanceDiagnostic	83
6.35 Metric Class Reference	83
6.35.1 Constructor & Destructor Documentation	84
6.35.1.1 ~Metric()	84
6.35.1.2 Metric()	84
6.35.2 Member Function Documentation	84
6.35.2.1 getChristoffel_udd()	84
6.35.2.2 getFullDescriptionStr()	84
6.35.2.3 getKretschmann()	84
6.35.2.4 getMetric_dd()	84
6.35.2.5 getMetric_uu()	85
6.35.2.6 getRiemann_uddd()	85
6.35.2.7 getrLogScale()	85

6.35.3 Member Data Documentation	85
6.35.3.1 m_rLogScale	85
6.35.3.2 m_Symmetries	85
6.36 NaNTermination Class Reference	85
6.36.1 Constructor & Destructor Documentation	86
6.36.1.1 NaNTermination()	86
6.36.2 Member Function Documentation	86
6.36.2.1 CheckTermination()	86
6.36.2.2 getFullDescriptionStr()	87
6.36.3 Member Data Documentation	87
6.36.3.1 TermOptions	87
6.37 NaNTermOptions Struct Reference	87
6.37.1 Constructor & Destructor Documentation	88
6.37.1.1 NaNTermOptions()	88
6.37.2 Member Data Documentation	88
6.37.2.1 OutputToConsole	88
6.38 NoSource Class Reference	88
6.38.1 Constructor & Destructor Documentation	89
6.38.1.1 NoSource()	89
6.38.2 Member Function Documentation	89
6.38.2.1 getFullDescriptionStr()	89
6.38.2.2 getSource()	89
6.39 SquareSubdivisionMesh::PixelInfo Struct Reference	89
6.39.1 Constructor & Destructor Documentation	90
6.39.1.1 PixelInfo()	90
6.39.2 Member Data Documentation	90
6.39.2.1 DiagValue	90
6.39.2.2 Index	90
6.39.2.3 LowerNbrIndex	90
6.39.2.4 RightNbrIndex	90
6.39.2.5 SubdivideLevel	90
6.39.2.6 Weight	90
6.40 SquareSubdivisionMeshV2::PixelInfo Struct Reference	90
6.40.1 Constructor & Destructor Documentation	91
6.40.1.1 PixelInfo()	91
6.40.2 Member Data Documentation	91
6.40.2.1 DiagValue	91
6.40.2.2 DownNbr	91
6.40.2.3 Index	91
6.40.2.4 LeftNbr	91
6.40.2.5 RightNbr	91
6.40.2.6 SEdiagNbr	91

6.40.2.7 SubdivideLevel	92
6.40.2.8 UpNbr	92
6.40.2.9 Weight	92
6.41 RasheedLarsenMetric Class Reference	92
6.41.1 Constructor & Destructor Documentation	93
6.41.1.1 RasheedLarsenMetric() [1/2]	93
6.41.1.2 RasheedLarsenMetric() [2/2]	93
6.41.2 Member Function Documentation	94
6.41.2.1 getFullDescriptionStr()	94
6.41.2.2 getMetric_dd()	94
6.41.2.3 getMetric_uu()	94
6.41.3 Member Data Documentation	94
6.41.3.1 m_aParam	94
6.41.3.2 m_mParam	94
6.41.3.3 m_pParam	94
6.41.3.4 m_qParam	94
6.42 SimpleSquareMesh Class Reference	95
6.42.1 Constructor & Destructor Documentation	95
6.42.1.1 SimpleSquareMesh() [1/2]	95
6.42.1.2 SimpleSquareMesh() [2/2]	96
6.42.2 Member Function Documentation	96
6.42.2.1 EndCurrentLoop()	96
6.42.2.2 GeodesicFinished()	96
6.42.2.3 getCurNrGeodesics()	96
6.42.2.4 getFullDescriptionStr()	96
6.42.2.5 getNewInitConds()	96
6.42.2.6 IsFinished()	97
6.42.3 Member Data Documentation	97
6.42.3.1 m_Finished	97
6.42.3.2 m_RowColumnSize	97
6.42.3.3 m_TotalPixels	97
6.43 SingularityMetric Class Reference	97
6.43.1 Constructor & Destructor Documentation	98
6.43.1.1 SingularityMetric()	98
6.43.2 Member Function Documentation	98
6.43.2.1 getSingularities()	98
6.43.3 Member Data Documentation	98
6.43.3.1 m_AllSingularities	98
6.44 Source Class Reference	99
6.44.1 Constructor & Destructor Documentation	99
6.44.1.1 Source()	99
6.44.1.2 ~Source()	99

6.44.2 Member Function Documentation	99
6.44.2.1 getFullDescriptionStr()	99
6.44.2.2 getSource()	100
6.44.3 Member Data Documentation	100
6.44.3.1 m_theMetric	100
6.45 SphericalHorizonMetric Class Reference	100
6.45.1 Constructor & Destructor Documentation	101
6.45.1.1 SphericalHorizonMetric() [1/2]	101
6.45.1.2 SphericalHorizonMetric() [2/2]	101
6.45.2 Member Function Documentation	101
6.45.2.1 getHorizonRadius()	101
6.45.3 Member Data Documentation	101
6.45.3.1 m_HorizonRadius	101
6.46 SquareSubdivisionMesh Class Reference	102
6.46.1 Constructor & Destructor Documentation	103
6.46.1.1 SquareSubdivisionMesh() [1/2]	103
6.46.1.2 SquareSubdivisionMesh() [2/2]	103
6.46.2 Member Function Documentation	103
6.46.2.1 EndCurrentLoop()	103
6.46.2.2 Explnt()	103
6.46.2.3 GeodesicFinished()	104
6.46.2.4 getCurNrGeodesics()	104
6.46.2.5 getFullDescriptionStr()	104
6.46.2.6 getNewInitConds()	104
6.46.2.7 InitializeFirstGrid()	104
6.46.2.8 IsFinished()	104
6.46.2.9 SubdivideAndQueue()	104
6.46.2.10 UpdateAllNeighbors()	105
6.46.2.11 UpdateAllWeights()	105
6.46.3 Member Data Documentation	105
6.46.3.1 m_AllPixels	105
6.46.3.2 m_CurrentPixelQueue	105
6.46.3.3 m_CurrentPixelQueueDone	105
6.46.3.4 m_InfinitePixels	105
6.46.3.5 m_InitialPixels	105
6.46.3.6 m_InitialSubDivideToFinal	105
6.46.3.7 m_IterationPixels	105
6.46.3.8 m_MaxPixels	105
6.46.3.9 m_MaxSubdivide	106
6.46.3.10 m_PixelsLeft	106
6.46.3.11 m_RowColumnSize	106
6.47 SquareSubdivisionMeshV2 Class Reference	106



6.47.1 Constructor & Destructor Documentation	107
6.47.1.1 SquareSubdivisionMeshV2() [1/2]	107
6.47.1.2 SquareSubdivisionMeshV2() [2/2]	108
6.47.2 Member Function Documentation	108
6.47.2.1 EndCurrentLoop()	108
6.47.2.2 Explnt()	108
6.47.2.3 GeodesicFinished()	108
6.47.2.4 getCurNrGeodesics()	108
6.47.2.5 GetDown()	108
6.47.2.6 getFullDescriptionStr()	109
6.47.2.7 GetLeft()	109
6.47.2.8 getNewInitConds()	109
6.47.2.9 GetRight()	109
6.47.2.10 GetUp()	109
6.47.2.11 InitializeFirstGrid()	109
6.47.2.12 IsFinished()	109
6.47.2.13 SubdivideAndQueue()	110
6.47.2.14 UpdateAllWeights()	110
6.47.3 Member Data Documentation	110
6.47.3.1 m_ActivePixels	110
6.47.3.2 m_AllPixels	110
6.47.3.3 m_CurrentPixelQueue	110
6.47.3.4 m_CurrentPixelQueueDone	110
6.47.3.5 m_CurrentPixelUpdating	110
6.47.3.6 m_InfinitePixels	110
6.47.3.7 m_InitialPixels	110
6.47.3.8 m_InitialSubDivideToFinal	110
6.47.3.9 m_IterationPixels	111
6.47.3.10 m_MaxPixels	111
6.47.3.11 m_MaxSubdivide	111
6.47.3.12 m_PixelsIntegrated	111
6.47.3.13 m_PixelsLeft	111
6.47.3.14 m_RowColumnSize	111
6.48 ST3CrMetric Class Reference	111
6.48.1 Constructor & Destructor Documentation	113
6.48.1.1 ST3CrMetric()	113
6.48.2 Member Function Documentation	113
6.48.2.1 f_om_phi()	113
6.48.2.2 f_phi()	113
6.48.2.3 get_omega()	113
6.48.2.4 getFullDescriptionStr()	113
6.48.2.5 getMetric_dd()	113

6.48.2.6 getMetric_uu()	114
6.48.3 Member Data Documentation	114
6.48.3.1 m_lambda	114
6.48.3.2 m_P	114
6.48.3.3 m_q0	114
6.49 Termination Class Reference	114
6.49.1 Constructor & Destructor Documentation	115
6.49.1.1 Termination() [1/2]	115
6.49.1.2 Termination() [2/2]	115
6.49.1.3 ~Termination()	115
6.49.2 Member Function Documentation	115
6.49.2.1 CheckTermination()	115
6.49.2.2 DecideUpdate()	115
6.49.2.3 getFullDescriptionStr()	115
6.49.2.4 Reset()	116
6.49.3 Member Data Documentation	116
6.49.3.1 m_OwnerGeodesic	116
6.49.3.2 m_StepsSinceUpdated	116
6.50 TerminationOptions Struct Reference	116
6.50.1 Constructor & Destructor Documentation	116
6.50.1.1 TerminationOptions()	116
6.50.1.2 ~TerminationOptions()	117
6.50.2 Member Data Documentation	117
6.50.2.1 UpdateEveryNSteps	117
6.51 ThetaSingularityTermination Class Reference	117
6.51.1 Constructor & Destructor Documentation	118
6.51.1.1 ThetaSingularityTermination()	118
6.51.2 Member Function Documentation	118
6.51.2.1 CheckTermination()	118
6.51.2.2 getFullDescriptionStr()	118
6.51.3 Member Data Documentation	118
6.51.3.1 TermOptions	118
6.52 ThetaSingularityTermOptions Struct Reference	119
6.52.1 Constructor & Destructor Documentation	119
6.52.1.1 ThetaSingularityTermOptions()	119
6.52.2 Member Data Documentation	119
6.52.2.1 ThetaSingEpsilon	119
6.53 TimeOutTermination Class Reference	120
6.53.1 Constructor & Destructor Documentation	121
6.53.1.1 TimeOutTermination()	121
6.53.2 Member Function Documentation	121
6.53.2.1 CheckTermination()	121

6.53.2.2 getFullDescriptionStr()	121
6.53.2.3 Reset()	121
6.53.3 Member Data Documentation	121
6.53.3.1 m_CurNrSteps	121
6.53.3.2 TermOptions	121
6.54 TimeOutTermOptions Struct Reference	122
6.54.1 Constructor & Destructor Documentation	122
6.54.1.1 TimeOutTermOptions()	122
6.54.2 Member Data Documentation	122
6.54.2.1 MaxSteps	122
6.55 Utilities::Timer Class Reference	123
6.55.1 Member Typedef Documentation	123
6.55.1.1 Clock	123
6.55.1.2 Second	123
6.55.2 Member Function Documentation	123
6.55.2.1 elapsed()	123
6.55.2.2 reset()	123
6.55.3 Member Data Documentation	124
6.55.3.1 m_beg	124
6.56 UpdateFrequency Struct Reference	124
6.56.1 Member Data Documentation	124
6.56.1.1 UpdateFinish	124
6.56.1.2 UpdateNSteps	124
6.56.1.3 UpdateStart	124
6.57 ViewScreen Class Reference	124
6.57.1 Constructor & Destructor Documentation	125
6.57.1.1 ViewScreen() [1/2]	125
6.57.1.2 ViewScreen() [2/2]	125
6.57.2 Member Function Documentation	126
6.57.2.1 ConstructVielbein()	126
6.57.2.2 EndCurrentLoop()	126
6.57.2.3 GeodesicFinished()	126
6.57.2.4 getCurNrGeodesics()	126
6.57.2.5 getFullDescriptionStr()	126
6.57.2.6 IsFinished()	126
6.57.2.7 SetNewInitialConditions()	126
6.57.3 Member Data Documentation	126
6.57.3.1 m_Direction	126
6.57.3.2 m_GeodType	127
6.57.3.3 m_Metric_dd	127
6.57.3.4 m_Pos	127
6.57.3.5 m_rLogScale	127

6.57.3.6 m_ScreenCenter . . . . .	127
6.57.3.7 m_ScreenSize . . . . .	127
6.57.3.8 m_theMesh . . . . .	127
6.57.3.9 m_theMetric . . . . .	127
6.57.3.10 m_Vielbein . . . . .	127
<b>7 File Documentation</b>	<b>129</b>
7.1 FOORT/src/Config.cpp File Reference . . . . .	129
7.2 FOORT/src/Config.h File Reference . . . . .	129
7.2.1 Macro Definition Documentation . . . . .	130
7.2.1.1 CONFIGURATION_MODE . . . . .	130
7.3 Config.h . . . . .	131
7.4 FOORT/src/ConfigReader.cpp File Reference . . . . .	132
7.5 FOORT/src/ConfigReader.h File Reference . . . . .	132
7.6 ConfigReader.h . . . . .	133
7.7 FOORT/src/Diagnostics.cpp File Reference . . . . .	135
7.7.1 Function Documentation . . . . .	135
7.7.1.1 CreateDiagnosticVector() . . . . .	135
7.8 FOORT/src/Diagnostics.h File Reference . . . . .	135
7.8.1 Typedef Documentation . . . . .	136
7.8.1.1 DiagBitflag . . . . .	136
7.8.1.2 DiagnosticUniqueVector . . . . .	136
7.8.2 Function Documentation . . . . .	137
7.8.2.1 CreateDiagnosticVector() . . . . .	137
7.8.3 Variable Documentation . . . . .	137
7.8.3.1 Diag_ClosestRadius . . . . .	137
7.8.3.2 Diag_EquatorialEmission . . . . .	137
7.8.3.3 Diag_EquatorialPasses . . . . .	137
7.8.3.4 Diag_FourColorScreen . . . . .	137
7.8.3.5 Diag_GeodesicPosition . . . . .	137
7.8.3.6 Diag_None . . . . .	137
7.9 Diagnostics.h . . . . .	138
7.10 FOORT/src/DiagnosticsEmission.cpp File Reference . . . . .	143
7.11 FOORT/src/DiagnosticsEmission.h File Reference . . . . .	144
7.12 DiagnosticsEmission.h . . . . .	144
7.13 FOORT/src/Geodesic.cpp File Reference . . . . .	146
7.14 FOORT/src/Geodesic.h File Reference . . . . .	146
7.15 Geodesic.h . . . . .	146
7.16 FOORT/src/Geometry.h File Reference . . . . .	148
7.16.1 Macro Definition Documentation . . . . .	149
7.16.1.1 LARGE_COUNTER_MAX . . . . .	149
7.16.1.2 PIXEL_MAX . . . . .	149

7.16.2 Typedef Documentation	149
7.16.2.1 FourIndex	149
7.16.2.2 largecounter	149
7.16.2.3 OneIndex	150
7.16.2.4 pixelcoord	150
7.16.2.5 Point	150
7.16.2.6 real	150
7.16.2.7 ScreenIndex	150
7.16.2.8 ScreenPoint	150
7.16.2.9 Singularity	150
7.16.2.10 SingularityCoord	150
7.16.2.11 ThreeIndex	150
7.16.2.12 TwoIndex	150
7.16.3 Function Documentation	151
7.16.3.1 operator*() [1/2]	151
7.16.3.2 operator*() [2/2]	151
7.16.3.3 operator+()	151
7.16.3.4 operator-()	151
7.16.3.5 operator/()	151
7.16.3.6 toString() [1/3]	151
7.16.3.7 toString() [2/3]	152
7.16.3.8 toString() [3/3]	152
7.16.4 Variable Documentation	152
7.16.4.1 dimension	152
7.16.4.2 pi	152
7.17 Geometry.h	152
7.18 FOORT/src/Header.h File Reference	154
7.19 Header.h	154
7.20 FOORT/src/InputOutput.cpp File Reference	154
7.20.1 Function Documentation	155
7.20.1.1 GetLoopMessageFrequency()	155
7.20.1.2 ScreenOutput()	155
7.20.1.3 SetLoopMessageFrequency()	155
7.20.1.4 SetOutputLevel()	155
7.21 FOORT/src/InputOutput.h File Reference	155
7.21.1 Enumeration Type Documentation	156
7.21.1.1 OutputLevel	156
7.21.2 Function Documentation	156
7.21.2.1 GetLoopMessageFrequency()	156
7.21.2.2 ScreenOutput()	156
7.21.2.3 SetLoopMessageFrequency()	156
7.21.2.4 SetOutputLevel()	157

7.22 InputOutput.h . . . . .	157
7.23 FOORT/src/Integrators.cpp File Reference . . . . .	158
7.24 FOORT/src/Integrators.h File Reference . . . . .	158
7.24.1 Typedef Documentation . . . . .	159
7.24.1.1 GeodesicIntegratorFunc . . . . .	159
7.25 Integrators.h . . . . .	160
7.26 FOORT/src/Main.cpp File Reference . . . . .	160
7.26.1 Function Documentation . . . . .	161
7.26.1.1 LoadPrecompiledOptions() . . . . .	161
7.26.1.2 main() . . . . .	161
7.27 FOORT/src/Mesh.cpp File Reference . . . . .	161
7.28 FOORT/src/Mesh.h File Reference . . . . .	162
7.29 Mesh.h . . . . .	162
7.30 FOORT/src/Metric.cpp File Reference . . . . .	167
7.31 FOORT/src/Metric.h File Reference . . . . .	167
7.32 Metric.h . . . . .	168
7.33 FOORT/src/Spline.h File Reference . . . . .	172
7.34 Spline.h . . . . .	172
7.35 FOORT/src/Terminations.cpp File Reference . . . . .	181
7.35.1 Function Documentation . . . . .	181
7.35.1.1 CreateTerminationVector() . . . . .	181
7.36 FOORT/src/Terminations.h File Reference . . . . .	181
7.36.1 Typedef Documentation . . . . .	182
7.36.1.1 TermBitflag . . . . .	182
7.36.1.2 TerminationUniqueVector . . . . .	182
7.36.2 Enumeration Type Documentation . . . . .	182
7.36.2.1 Term . . . . .	182
7.36.3 Function Documentation . . . . .	183
7.36.3.1 CreateTerminationVector() . . . . .	183
7.36.4 Variable Documentation . . . . .	183
7.36.4.1 Term_BoundarySphere . . . . .	183
7.36.4.2 Term_GeneralSingularity . . . . .	183
7.36.4.3 Term_Horizon . . . . .	183
7.36.4.4 Term_NaN . . . . .	183
7.36.4.5 Term_None . . . . .	183
7.36.4.6 Term_ThetaSingularity . . . . .	183
7.36.4.7 Term_TimeOut . . . . .	183
7.37 Terminations.h . . . . .	184
7.38 FOORT/src/Utilities.cpp File Reference . . . . .	188
7.39 FOORT/src/Utilities.h File Reference . . . . .	188
7.40 Utilities.h . . . . .	189
7.41 FOORT/src/ViewScreen.cpp File Reference . . . . .	190

---

7.42 FOORT/src/ViewScreen.h File Reference . . . . .	190
7.42.1 Enumeration Type Documentation . . . . .	190
7.42.1.1 GeodesicType . . . . .	190
7.43 ViewScreen.h . . . . .	191
<b>Index</b>	<b>193</b>





# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">Config</a>	9
<a href="#">ConfigReader</a>	12
<a href="#">Integrators</a>	12
<a href="#">tk</a>	14
<a href="#">tk::internal</a>	14
<a href="#">Utilities</a>	14



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

ConfigReader::ConfigCollection . . . . .	23
ConfigReader::ConfigCollection::ConfigSetting . . . . .	28
Diagnostic . . . . .	29
ClosestRadiusDiagnostic . . . . .	20
EquatorialPassesDiagnostic . . . . .	38
EquatorialEmissionDiagnostic . . . . .	33
FourColorScreenDiagnostic . . . . .	45
GeodesicPositionDiagnostic . . . . .	61
DiagnosticOptions . . . . .	31
ClosestRadiusOptions . . . . .	22
EquatorialPassesOptions . . . . .	41
EquatorialEmissionOptions . . . . .	36
GeodesicPositionOptions . . . . .	63
EmissionModel . . . . .	32
GLMJohnsonSUEmission . . . . .	64
FluidVelocityModel . . . . .	44
GeneralCircularRadialFluid . . . . .	47
Geodesic . . . . .	54
GeodesicOutputHandler . . . . .	57
Mesh . . . . .	81
InputCertainPixelsMesh . . . . .	69
SimpleSquareMesh . . . . .	95
SquareSubdivisionMesh . . . . .	102
SquareSubdivisionMeshV2 . . . . .	106
Metric . . . . .	83
BosonStarMetric . . . . .	15
FlatSpaceMetric . . . . .	42
SingularityMetric . . . . .	97
ST3CrMetric . . . . .	111
SphericalHorizonMetric . . . . .	100
JohannsenMetric . . . . .	71
KerrMetric . . . . .	74
KerrSchildMetric . . . . .	76

MankoNovikovMetric . . . . .	79
RasheedLarsenMetric . . . . .	92
SquareSubdivisionMesh::PixelInfo . . . . .	89
SquareSubdivisionMeshV2::PixelInfo . . . . .	90
std::runtime_error	
ConfigReader::ConfigReaderException . . . . .	27
Source . . . . .	99
NoSource . . . . .	88
Termination . . . . .	114
BoundarySphereTermination . . . . .	17
GeneralSingularityTermination . . . . .	50
HorizonTermination . . . . .	66
NaNTermination . . . . .	85
ThetaSingularityTermination . . . . .	117
TimeOutTermination . . . . .	120
TerminationOptions . . . . .	116
BoundarySphereTermOptions . . . . .	18
GeneralSingularityTermOptions . . . . .	52
HorizonTermOptions . . . . .	67
NaNTermOptions . . . . .	87
ThetaSingularityTermOptions . . . . .	119
TimeOutTermOptions . . . . .	122
Utilities::Timer . . . . .	123
UpdateFrequency . . . . .	124
ViewScreen . . . . .	124

# Chapter 3

## Class Index

### 3.1 Class List

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

BosonStarMetric	15
BoundarySphereTermination	17
BoundarySphereTermOptions	18
ClosestRadiusDiagnostic	20
ClosestRadiusOptions	22
ConfigReader::ConfigCollection	23
ConfigReader::ConfigReaderException	27
ConfigReader::ConfigCollection::ConfigSetting	28
Diagnostic	29
DiagnosticOptions	31
EmissionModel	32
EquatorialEmissionDiagnostic	33
EquatorialEmissionOptions	36
EquatorialPassesDiagnostic	38
EquatorialPassesOptions	41
FlatSpaceMetric	42
FluidVelocityModel	44
FourColorScreenDiagnostic	45
GeneralCircularRadialFluid	47
GeneralSingularityTermination	50
GeneralSingularityTermOptions	52
Geodesic	54
GeodesicOutputHandler	57
GeodesicPositionDiagnostic	61
GeodesicPositionOptions	63
GLMJohnsonSUEmission	64
HorizonTermination	66
HorizonTermOptions	67
InputCertainPixelsMesh	69
JohannsenMetric	71
KerrMetric	74
KerrSchildMetric	76
MankoNovikovMetric	79
Mesh	81
Metric	83

<a href="#">NaNTermination</a>	85
<a href="#">NaNTermOptions</a>	87
<a href="#">NoSource</a>	88
<a href="#">SquareSubdivisionMesh::PixelInfo</a>	89
<a href="#">SquareSubdivisionMeshV2::PixelInfo</a>	90
<a href="#">RasheedLarsenMetric</a>	92
<a href="#">SimpleSquareMesh</a>	95
<a href="#">SingularityMetric</a>	97
<a href="#">Source</a>	99
<a href="#">SphericalHorizonMetric</a>	100
<a href="#">SquareSubdivisionMesh</a>	102
<a href="#">SquareSubdivisionMeshV2</a>	106
<a href="#">ST3CrMetric</a>	111
<a href="#">Termination</a>	114
<a href="#">TerminationOptions</a>	116
<a href="#">ThetaSingularityTermination</a>	117
<a href="#">ThetaSingularityTermOptions</a>	119
<a href="#">TimeOutTermination</a>	120
<a href="#">TimeOutTermOptions</a>	122
<a href="#">Utilities::Timer</a>	123
<a href="#">UpdateFrequency</a>	124
<a href="#">ViewScreen</a>	124

# Chapter 4

## File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

FOORT/src/ <a href="#">Config.cpp</a>	129
FOORT/src/ <a href="#">Config.h</a>	129
FOORT/src/ <a href="#">ConfigReader.cpp</a>	132
FOORT/src/ <a href="#">ConfigReader.h</a>	132
FOORT/src/ <a href="#">Diagnostics.cpp</a>	135
FOORT/src/ <a href="#">Diagnostics.h</a>	135
FOORT/src/ <a href="#">DiagnosticsEmission.cpp</a>	143
FOORT/src/ <a href="#">DiagnosticsEmission.h</a>	144
FOORT/src/ <a href="#">Geodesic.cpp</a>	146
FOORT/src/ <a href="#">Geodesic.h</a>	146
FOORT/src/ <a href="#">Geometry.h</a>	148
FOORT/src/ <a href="#">Header.h</a>	154
FOORT/src/ <a href="#">InputOutput.cpp</a>	154
FOORT/src/ <a href="#">InputOutput.h</a>	155
FOORT/src/ <a href="#">Integrators.cpp</a>	158
FOORT/src/ <a href="#">Integrators.h</a>	158
FOORT/src/ <a href="#">Main.cpp</a>	160
FOORT/src/ <a href="#">Mesh.cpp</a>	161
FOORT/src/ <a href="#">Mesh.h</a>	162
FOORT/src/ <a href="#">Metric.cpp</a>	167
FOORT/src/ <a href="#">Metric.h</a>	167
FOORT/src/ <a href="#">Spline.h</a>	172
FOORT/src/ <a href="#">Terminations.cpp</a>	181
FOORT/src/ <a href="#">Terminations.h</a>	181
FOORT/src/ <a href="#">Utilities.cpp</a>	188
FOORT/src/ <a href="#">Utilities.h</a>	188
FOORT/src/ <a href="#">ViewScreen.cpp</a>	190
FOORT/src/ <a href="#">ViewScreen.h</a>	190





# Chapter 5

## Namespace Documentation

### 5.1 Config Namespace Reference

#### Typedefs

- using [ConfigCollection](#) = [ConfigReader::ConfigCollection](#)
- using [SettingError](#) = [std::invalid\\_argument](#)

#### Functions

- void [InitializeScreenOutput](#) (const [ConfigCollection](#) &theCfg)
- std::unique\_ptr< [Metric](#) > [GetMetric](#) (const [ConfigCollection](#) &theCfg)  
*[Config::GetMetric\(\)](#): Use configuration to create the correct [Metric](#) with specified parameters.*
- std::unique\_ptr< [Source](#) > [GetSource](#) (const [ConfigCollection](#) &theCfg, const [Metric](#) \*const theMetric)  
*[Config::GetSource\(\)](#): Use configuration to create the correct [Source](#) with specified parameters.*
- void [InitializeDiagnostics](#) (const [ConfigCollection](#) &theCfg, [DiagBitflag](#) &alldiags, [DiagBitflag](#) &valdiag, const [Metric](#) \*const theMetric)  
*[Config::InitializeDiagnostics\(\)](#): Use configuration to set the [Diagnostics](#) bitflag appropriately; initialize all [DiagnosticOptions](#) for all [Diagnostics](#) that are turned on; and set bitflag for diagnostic to be used for coarseness evaluating in [Mesh](#).*
- void [InitializeTerminations](#) (const [ConfigCollection](#) &theCfg, [TermBitflag](#) &allterms, const [Metric](#) \*const theMetric)  
*[Config::InitializeTerminations\(\)](#): Use configuration to set the [Termination](#) bitflag appropriately; and initialize all [TerminationOptions](#) for all [Terminations](#) that are turned on.*
- std::unique\_ptr< [ViewScreen](#) > [GetViewScreen](#) (const [ConfigCollection](#) &theCfg, [DiagBitflag](#) valdiag, const [Metric](#) \*const theMetric)  
*[Config::GetViewScreen\(\)](#): Use configuration to create the [ViewScreen](#) object; with options set according to the configuration.*
- std::unique\_ptr< [Mesh](#) > [GetMesh](#) (const [ConfigCollection](#) &theCfg, [DiagBitflag](#) valdiag)  
*[Config::GetMesh\(\)](#): Use configuration to create the [Mesh](#) object; with options set according to the configuration. [Config::GetViewScreen\(\)](#) calls this when creating the [ViewScreen](#) object.*
- [GeodesicIntegratorFunc](#) [GetGeodesicIntegrator](#) (const [ConfigCollection](#) &theCfg)  
*[Config::GetGeodesicIntegrator\(\)](#): Returns a pointer to the integrator function to be used as specified in the configuration file.*
- std::unique\_ptr< [GeodesicOutputHandler](#) > [GetOutputHandler](#) (const [ConfigCollection](#) &theCfg, [DiagBitflag](#) alldiags, [DiagBitflag](#) valdiag, std::string FirstLineInfo)  
*[Config::GetOutputHandler\(\)](#): Creates the [GeodesicOutputHandler](#) object with options specified according to the configuration file, for handling of geodesic outputs.*

## Variables

- constexpr auto [Output\\_Important\\_Default](#) = [OutputLevel::Level\\_0\\_WARNING](#)
- constexpr auto [Output\\_Other\\_Default](#) = [OutputLevel::Level\\_1\\_PROC](#)

## 5.1.1 Typedef Documentation

### 5.1.1.1 ConfigCollection

```
using Config::ConfigCollection = ConfigReader::ConfigCollection
```

### 5.1.1.2 SettingError

```
using Config::SettingError = std::invalid_argument
```

## 5.1.2 Function Documentation

### 5.1.2.1 GetGeodesicIntegrator()

```
GeodesicIntegratorFunc Config::GetGeodesicIntegrator (
    const ConfigCollection & theCfg)
```

[Config::GetGeodesicIntegrator\(\)](#): Returns a pointer to the integrator function to be used as specified in the configuration file.

### 5.1.2.2 GetMesh()

```
std::unique_ptr< Mesh > Config::GetMesh (
    const ConfigCollection & theCfg,
    DiagBitflag valdiag)
```

[Config::GetMesh\(\)](#): Use configuration to create the [Mesh](#) object; with options set according to the configuration. [Config::GetViewScreen\(\)](#) calls this when creating the [ViewScreen](#) object.

### 5.1.2.3 GetMetric()

```
std::unique_ptr< Metric > Config::GetMetric (
    const ConfigCollection & theCfg)
```

[Config::GetMetric\(\)](#): Use configuration to create the correct [Metric](#) with specified parameters.

### 5.1.2.4 GetOutputHandler()

```
std::unique_ptr< GeodesicOutputHandler > Config::GetOutputHandler (
    const ConfigCollection & theCfg,
    DiagBitflag alldiags,
    DiagBitflag valdiag,
    std::string FirstLineInfo)
```

[Config::GetOutputHandler\(\)](#): Creates the [GeodesicOutputHandler](#) object with options specified according to the configuration file, for handling of geodesic outputs.

### 5.1.2.5 GetSource()

```
std::unique_ptr< Source > Config::GetSource (
    const ConfigCollection & theCfg,
    const Metric *const theMetric)
```

[Config::GetSource\(\)](#): Use configuration to create the correct [Source](#) with specified parameters.

### 5.1.2.6 GetViewScreen()

```
std::unique_ptr< ViewScreen > Config::GetViewScreen (
    const ConfigCollection & theCfg,
    DiagBitflag valdiag,
    const Metric *const theMetric)
```

[Config::GetViewScreen\(\)](#): Use configuration to create the [ViewScreen](#) object; with options set according to the configuration.

### 5.1.2.7 InitializeDiagnostics()

```
void Config::InitializeDiagnostics (
    const ConfigCollection & theCfg,
    DiagBitflag & alldiags,
    DiagBitflag & valdiag,
    const Metric *const theMetric)
```

[Config::InitializeDiagnostics\(\)](#): Use configuration to set the Diagnostics bitflag appropriately; initialize all [DiagnosticOptions](#) for all Diagnostics that are turned on; and set bitflag for diagnostic to be used for coarseness evaluating in [Mesh](#).

### 5.1.2.8 InitializeScreenOutput()

```
void Config::InitializeScreenOutput (
    const ConfigCollection & theCfg)
```

### 5.1.2.9 InitializeTerminations()

```
void Config::InitializeTerminations (
    const ConfigCollection & theCfg,
    TermBitflag & allterms,
    const Metric *const theMetric)
```

[Config::InitializeTerminations\(\)](#): Use configuration to set the [Termination](#) bitflag appropriately; and initialize all [TerminationOptions](#) for all Terminations that are turned on.

## 5.1.3 Variable Documentation

### 5.1.3.1 Output\_Important\_Default

```
auto Config::Output_Important_Default = OutputLevel::Level_0_WARNING [constexpr]
```

### 5.1.3.2 Output\_Other\_Default

```
auto Config::Output_Other_Default = OutputLevel::Level_1_PROC [constexpr]
```

## 5.2 ConfigReader Namespace Reference

### Classes

- class [ConfigCollection](#)
- class [ConfigReaderException](#)

### Variables

- const long long [MaxStreamSize](#) {std::numeric\_limits<std::streamsize>::max() }

### 5.2.1 Variable Documentation

#### 5.2.1.1 MaxStreamSize

```
const long long ConfigReader::MaxStreamSize {std::numeric_limits<std::streamsize>::max() }
```

## 5.3 Integrators Namespace Reference

### Functions

- std::string [GetFullIntegratorDescription](#) ()
- [real](#) [GetAdaptiveStep](#) ([Point](#) curpos, [OneIndex](#) curvel)
- void [IntegrateGeodesicStep\\_RK4](#) ([Point](#) curpos, [OneIndex](#) curvel, [Point](#) &nextpos, [OneIndex](#) &nextvel, [real](#) &stepsize, const [Metric](#) \*theMetric, const [Source](#) \*theSource)
- void [IntegrateGeodesicStep\\_Verlet](#) ([Point](#) curpos, [OneIndex](#) curvel, [Point](#) &nextpos, [OneIndex](#) &nextvel, [real](#) &stepsize, const [Metric](#) \*theMetric, const [Source](#) \*theSource)

### Variables

- constexpr [real](#) [delta\\_nodiv0](#) = 1e-20
- [real](#) [Derivative\\_hval](#) {1e-7}
- std::string [IntegratorDescription](#) {"RK4"}
- [real](#) [epsilon](#) {0.03}
- [real](#) [SmallestPossibleStepsize](#) {1e-12}
- [real](#) [VerletVelocityTolerance](#) {0.001}

### 5.3.1 Function Documentation

#### 5.3.1.1 GetAdaptiveStep()

```
real Integrators::GetAdaptiveStep (
    Point curpos,
    OneIndex curvel)
```

### 5.3.1.2 GetFullIntegratorDescription()

```
std::string Integrators::GetFullIntegratorDescription ()
```

### 5.3.1.3 IntegrateGeodesicStep\_RK4()

```
void Integrators::IntegrateGeodesicStep_RK4 (
    Point curpos,
    OneIndex curvel,
    Point & nextpos,
    OneIndex & nextvel,
    real & stepsize,
    const Metric * theMetric,
    const Source * theSource)
```

### 5.3.1.4 IntegrateGeodesicStep\_Verlet()

```
void Integrators::IntegrateGeodesicStep_Verlet (
    Point curpos,
    OneIndex curvel,
    Point & nextpos,
    OneIndex & nextvel,
    real & stepsize,
    const Metric * theMetric,
    const Source * theSource)
```

## 5.3.2 Variable Documentation

### 5.3.2.1 delta\_nodiv0

```
real Integrators::delta_nodiv0 = 1e-20 [constexpr]
```

### 5.3.2.2 Derivative\_hval

```
real Integrators::Derivative_hval {1e-7} [inline]
```

### 5.3.2.3 epsilon

```
real Integrators::epsilon {0.03} [inline]
```

### 5.3.2.4 IntegratorDescription

```
std::string Integrators::IntegratorDescription {"RK4"} [inline]
```

### 5.3.2.5 SmallestPossibleStepsize

```
real Integrators::SmallestPossibleStepsize {1e-12} [inline]
```

### 5.3.2.6 VerletVelocityTolerance

```
real Integrators::VerletVelocityTolerance {0.001} [inline]
```

## 5.4 tk Namespace Reference

### Namespaces

- namespace [internal](#)

## 5.5 tk::internal Namespace Reference

## 5.6 Utilities Namespace Reference

### Classes

- class [Timer](#)

### Functions

- std::string [GetTimeStampString](#) ()  
Other functions in [Utilities](#).
- std::vector< std::string > [GetDiagNameStrings](#) ([DiagBitflag](#) alldiags, [DiagBitflag](#) valdiag)
- std::string [GetFirstLineInfoString](#) (const [Metric](#) \*theMetric, const [Source](#) \*theSource, [DiagBitflag](#) alldiags, [DiagBitflag](#) valdiag, [TermBitflag](#) allterms, const [ViewScreen](#) \*theView)

### 5.6.1 Function Documentation

#### 5.6.1.1 GetDiagNameStrings()

```
std::vector< std::string > Utilities::GetDiagNameStrings (
    DiagBitflag alldiags,
    DiagBitflag valdiag)
```

#### 5.6.1.2 GetFirstLineInfoString()

```
std::string Utilities::GetFirstLineInfoString (
    const Metric * theMetric,
    const Source * theSource,
    DiagBitflag alldiags,
    DiagBitflag valdiag,
    TermBitflag allterms,
    const ViewScreen * theView)
```

#### 5.6.1.3 GetTimeStampString()

```
std::string Utilities::GetTimeStampString ()
```

Other functions in [Utilities](#).

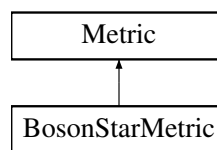
# Chapter 6

## Class Documentation

### 6.1 BosonStarMetric Class Reference

```
#include <Metric.h>
```

Inheritance diagram for BosonStarMetric:



#### Public Member Functions

- [BosonStarMetric](#) (bool rLogScale=false)  
*BosonStarMetric functions (implementation by Seppe Staelens)*
- [TwoIndex getMetric\\_dd](#) (const [Point](#) &p) const final
- [TwoIndex getMetric\\_uu](#) (const [Point](#) &p) const final
- [std::string getFullDescriptionStr](#) () const final

#### Public Member Functions inherited from [Metric](#)

- virtual [~Metric](#) ()=default
- [Metric](#) (bool rlogscale=false)  
*Metric (abstract base class) functions.*
- virtual [ThreeIndex getChristoffel\\_udd](#) (const [Point](#) &p) const
- virtual [FourIndex getRiemann\\_uddd](#) (const [Point](#) &p) const
- virtual [real getKretschmann](#) (const [Point](#) &p) const
- bool [getrLogScale](#) () const

#### Protected Attributes

- [tk::spline m\\_PhiSpline](#)
- [tk::spline m\\_mSpline](#)

## Protected Attributes inherited from [Metric](#)

- `std::vector< int > m\_Symmetries {}`
- `const bool m\_rLogScale`

## 6.1.1 Constructor & Destructor Documentation

### 6.1.1.1 [BosonStarMetric\(\)](#)

```
BosonStarMetric::BosonStarMetric (  
    bool rLogScale = false)
```

[BosonStarMetric](#) functions (implementation by Seppe Staelens)

## 6.1.2 Member Function Documentation

### 6.1.2.1 [getFullDescriptionStr\(\)](#)

```
std::string BosonStarMetric::getFullDescriptionStr () const [final], [virtual]
```

Reimplemented from [Metric](#).

### 6.1.2.2 [getMetric\\_dd\(\)](#)

```
TwoIndex BosonStarMetric::getMetric_dd (  
    const Point & p) const [final], [virtual]
```

Implements [Metric](#).

### 6.1.2.3 [getMetric\\_uu\(\)](#)

```
TwoIndex BosonStarMetric::getMetric_uu (  
    const Point & p) const [final], [virtual]
```

Implements [Metric](#).

## 6.1.3 Member Data Documentation

### 6.1.3.1 [m\\_mSpline](#)

```
tk::spline BosonStarMetric::m_mSpline [protected]
```



### 6.1.3.2 m\_PhiSpline

```
tk::spline BosonStarMetric::m_PhiSpline [protected]
```

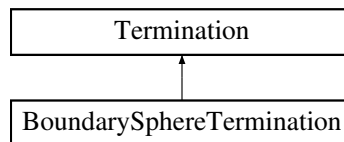
The documentation for this class was generated from the following files:

- FOORT/src/[Metric.h](#)
- FOORT/src/[Metric.cpp](#)

## 6.2 BoundarySphereTermination Class Reference

```
#include <Terminations.h>
```

Inheritance diagram for BoundarySphereTermination:



### Public Member Functions

- [BoundarySphereTermination](#) ([Geodesic](#) \*const theGeodesic)
- [Term CheckTermination](#) () final
- *[BoundarySphereTermination](#) functions.*
- `std::string` [getFullDescriptionStr](#) () const final

### Public Member Functions inherited from [Termination](#)

- [Termination](#) ()=delete
- [Termination](#) ([Geodesic](#) \*const theGeodesic)
- virtual void [Reset](#) ()
- *[Termination](#) (abstract base class) functions.*
- virtual [~Termination](#) ()=default

### Static Public Attributes

- static `std::unique_ptr`< [BoundarySphereTermOptions](#) > [TermOptions](#)

### Additional Inherited Members

### Protected Member Functions inherited from [Termination](#)

- bool [DecideUpdate](#) ([largecounter](#) UpdateNSteps)

## Protected Attributes inherited from [Termination](#)

- [Geodesic](#) \*const [m\\_OwnerGeodesic](#)
- [largecounter](#) [m\\_StepsSinceUpdated](#) {}

## 6.2.1 Constructor & Destructor Documentation

### 6.2.1.1 [BoundarySphereTermination](#)()

```
BoundarySphereTermination::BoundarySphereTermination (
    Geodesic *const theGeodesic) [inline]
```

## 6.2.2 Member Function Documentation

### 6.2.2.1 [CheckTermination](#)()

```
Term BoundarySphereTermination::CheckTermination () [final], [virtual]
```

[BoundarySphereTermination](#) functions.

Implements [Termination](#).

### 6.2.2.2 [getFullDescriptionStr](#)()

```
std::string BoundarySphereTermination::getFullDescriptionStr () const [final], [virtual]
```

Implements [Termination](#).

## 6.2.3 Member Data Documentation

### 6.2.3.1 [TermOptions](#)

```
std::unique_ptr< BoundarySphereTermOptions > BoundarySphereTermination::TermOptions [static]
```

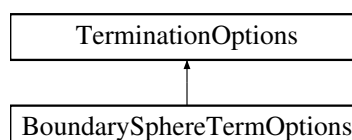
The documentation for this class was generated from the following files:

- FOORT/src/[Terminations.h](#)
- FOORT/src/[Config.cpp](#)
- FOORT/src/[Terminations.cpp](#)

## 6.3 [BoundarySphereTermOptions](#) Struct Reference

```
#include <Terminations.h>
```

Inheritance diagram for [BoundarySphereTermOptions](#):



### Public Member Functions

- [BoundarySphereTermOptions](#) ([real](#) theRadius, bool therLogScale, [largecounter](#) Nsteps)

### Public Member Functions inherited from [TerminationOptions](#)

- [TerminationOptions](#) ([largecounter](#) Nsteps)
- virtual [~TerminationOptions](#) ()=default

### Public Attributes

- const [real](#) SphereRadius
- const bool rLogScale

### Public Attributes inherited from [TerminationOptions](#)

- const [largecounter](#) UpdateEveryNSteps

## 6.3.1 Constructor & Destructor Documentation

### 6.3.1.1 BoundarySphereTermOptions()

```
BoundarySphereTermOptions::BoundarySphereTermOptions (  
    real theRadius,  
    bool therLogScale,  
    largecounter Nsteps) [inline]
```

## 6.3.2 Member Data Documentation

### 6.3.2.1 rLogScale

```
const bool BoundarySphereTermOptions::rLogScale
```

### 6.3.2.2 SphereRadius

```
const real BoundarySphereTermOptions::SphereRadius
```

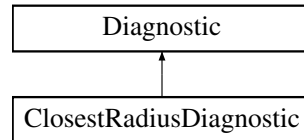
The documentation for this struct was generated from the following file:

- FOORT/src/[Terminations.h](#)

## 6.4 ClosestRadiusDiagnostic Class Reference

```
#include <Diagnostics.h>
```

Inheritance diagram for ClosestRadiusDiagnostic:



### Public Member Functions

- [ClosestRadiusDiagnostic](#) ([Geodesic](#) \*const theGeodesic)
- void [Reset](#) () final
- *[ClosestRadiusDiagnostic](#) functions.*
- void [UpdateData](#) () final
- std::string [getFullDataStr](#) () const final
- std::vector< [real](#) > [getFinalDataVal](#) () const final
- [real](#) [FinalDataValDistance](#) (const std::vector< [real](#) > &val1, const std::vector< [real](#) > &val2) const final
- std::string [getNameStr](#) () const final
- std::string [getFullDescriptionStr](#) () const final

### Public Member Functions inherited from [Diagnostic](#)

- [Diagnostic](#) ()=delete
- [Diagnostic](#) ([Geodesic](#) \*const theGeodesic)
- virtual [~Diagnostic](#) ()=default

### Static Public Attributes

- static std::unique\_ptr< [ClosestRadiusOptions](#) > [DiagOptions](#)

### Private Attributes

- [real](#) [m\\_ClosestRadius](#) {-1}

### Additional Inherited Members

### Protected Member Functions inherited from [Diagnostic](#)

- bool [DecideUpdate](#) (const [UpdateFrequency](#) &myUpdateFrequency)

### Protected Attributes inherited from [Diagnostic](#)

- [Geodesic](#) \*const [m\\_OwnerGeodesic](#)
- [largecounter](#) [m\\_StepsSinceUpdated](#) {}

## 6.4.1 Constructor & Destructor Documentation

### 6.4.1.1 ClosestRadiusDiagnostic()

```
ClosestRadiusDiagnostic::ClosestRadiusDiagnostic (  
    Geodesic *const theGeodesic) [inline]
```

## 6.4.2 Member Function Documentation

### 6.4.2.1 FinalDataValDistance()

```
real ClosestRadiusDiagnostic::FinalDataValDistance (  
    const std::vector< real > & val1,  
    const std::vector< real > & val2) const [final], [virtual]
```

Implements [Diagnostic](#).

### 6.4.2.2 getFinalDataVal()

```
std::vector< real > ClosestRadiusDiagnostic::getFinalDataVal () const [final], [virtual]
```

Implements [Diagnostic](#).

### 6.4.2.3 getFullDataStr()

```
std::string ClosestRadiusDiagnostic::getFullDataStr () const [final], [virtual]
```

Implements [Diagnostic](#).

### 6.4.2.4 getFullDescriptionStr()

```
std::string ClosestRadiusDiagnostic::getFullDescriptionStr () const [final], [virtual]
```

Reimplemented from [Diagnostic](#).

### 6.4.2.5 getNameStr()

```
std::string ClosestRadiusDiagnostic::getNameStr () const [final], [virtual]
```

Implements [Diagnostic](#).

### 6.4.2.6 Reset()

```
void ClosestRadiusDiagnostic::Reset () [final], [virtual]
```

[ClosestRadiusDiagnostic](#) functions.

Reimplemented from [Diagnostic](#).

### 6.4.2.7 UpdateData()

```
void ClosestRadiusDiagnostic::UpdateData () [final], [virtual]
```

Implements [Diagnostic](#).

## 6.4.3 Member Data Documentation

### 6.4.3.1 DiagOptions

```
std::unique_ptr< ClosestRadiusOptions > ClosestRadiusDiagnostic::DiagOptions [static]
```

### 6.4.3.2 m\_ClosestRadius

```
real ClosestRadiusDiagnostic::m_ClosestRadius {-1} [private]
```

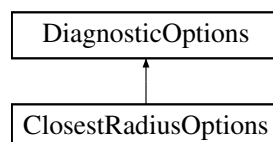
The documentation for this class was generated from the following files:

- FOORT/src/[Diagnostics.h](#)
- FOORT/src/[Config.cpp](#)
- FOORT/src/[Diagnostics.cpp](#)

## 6.5 ClosestRadiusOptions Struct Reference

```
#include <Diagnostics.h>
```

Inheritance diagram for ClosestRadiusOptions:



### Public Member Functions

- [ClosestRadiusOptions](#) (bool rlog, [UpdateFrequency](#) thefrequency)

### Public Member Functions inherited from [DiagnosticOptions](#)

- [DiagnosticOptions](#) ([UpdateFrequency](#) thefrequency)
- virtual [~DiagnosticOptions](#) ()=default

### Public Attributes

- const bool [RLogScale](#)

## Public Attributes inherited from [DiagnosticOptions](#)

- const [UpdateFrequency](#) *theUpdateFrequency*

### 6.5.1 Constructor & Destructor Documentation

#### 6.5.1.1 ClosestRadiusOptions()

```
ClosestRadiusOptions::ClosestRadiusOptions (
    bool rlog,
    UpdateFrequency thefrequency) [inline]
```

### 6.5.2 Member Data Documentation

#### 6.5.2.1 RLogScale

```
const bool ClosestRadiusOptions::RLogScale
```

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

- FOORT/src/[Diagnostics.h](#)

## 6.6 ConfigReader::ConfigCollection Class Reference

```
#include <ConfigReader.h>
```

### Classes

- struct [ConfigSetting](#)

### Public Member Functions

- bool [Exists](#) (std::string\_view SettingName) const
- int [NrSettings](#) () const
- bool [IsCollection](#) (std::string\_view SettingName) const
- bool [IsCollection](#) (int SettingIndex) const
- const [ConfigCollection](#) & [operator\[\]](#) (std::string\_view CollectionName) const
- const [ConfigCollection](#) & [operator\[\]](#) (int CollectionIndex) const
- template<class OutputType >  
bool [LookupValue](#) (std::string\_view SettingName, OutputType &theOutput) const
- template<class OutputType >  
bool [LookupValue](#) (int SettingIndex, OutputType &theOutput) const
- bool [LookupValueInteger](#) (std::string\_view SettingName, int &theOutput) const
- bool [LookupValueInteger](#) (std::string\_view SettingName, long &theOutput) const
- bool [LookupValueInteger](#) (std::string\_view SettingName, long long &theOutput) const
- bool [LookupValueInteger](#) (std::string\_view SettingName, unsigned int &theOutput) const
- bool [LookupValueInteger](#) (std::string\_view SettingName, unsigned long &theOutput) const
- bool [LookupValueInteger](#) (std::string\_view SettingName, unsigned long long &theOutput) const
- template<class OutputType >  
bool [LookupValueInteger](#) (int SettingIndex, OutputType &theOutput) const
- void [DisplayCollection](#) (std::ostream &OutputStream, int Indent=0) const
- bool [ReadFile](#) (const std::string &FileName)

## Private Types

- using [ConfigSettingValue](#)

## Private Member Functions

- int [GetSettingIndex](#) (std::string\_view SettingName) const
- void [DisplaySetting](#) (std::ostream &OutputStream, int SettingIndex, int Indent) const
- void [DisplayTabs](#) (std::ostream &OutputStream, int NrTabs) const
- void [ReadCollection](#) (std::ifstream &InputFile)
- void [ReadSettingName](#) (std::ifstream &InputFile, std::string &theName)
- void [ReadSettingSpecificChar](#) (std::ifstream &InputFile, char theChar) const
- void [ReadSettingValue](#) (std::ifstream &InputFile, [ConfigSettingValue](#) &theValue)

## Private Attributes

- std::vector< [ConfigSetting](#) > [m\\_Settings](#) {}

## 6.6.1 Member Typedef Documentation

### 6.6.1.1 ConfigSettingValue

using [ConfigReader::ConfigCollection::ConfigSettingValue](#) [private]

#### Initial value:

```
std::variant<bool,
              int, long, long long,
              double,
              std::string,
              std::unique_ptr<ConfigCollection>>
```

## 6.6.2 Member Function Documentation

### 6.6.2.1 DisplayCollection()

```
void ConfigCollection::DisplayCollection (
    std::ostream & OutputStream,
    int Indent = 0) const
```

### 6.6.2.2 DisplaySetting()

```
void ConfigCollection::DisplaySetting (
    std::ostream & OutputStream,
    int SettingIndex,
    int Indent) const [private]
```

### 6.6.2.3 DisplayTabs()

```
void ConfigCollection::DisplayTabs (
    std::ostream & OutputStream,
    int NrTabs) const [private]
```



#### 6.6.2.4 Exists()

```
bool ConfigCollection::Exists (
    std::string_view SettingName) const
```

#### 6.6.2.5 GetSettingIndex()

```
int ConfigCollection::GetSettingIndex (
    std::string_view SettingName) const [private]
```

#### 6.6.2.6 IsCollection() [1/2]

```
bool ConfigCollection::IsCollection (
    int SettingIndex) const
```

#### 6.6.2.7 IsCollection() [2/2]

```
bool ConfigCollection::IsCollection (
    std::string_view SettingName) const
```

#### 6.6.2.8 LookupValue() [1/2]

```
template<class OutputType >
bool ConfigReader::ConfigCollection::LookupValue (
    int SettingIndex,
    OutputType & theOutput) const
```

#### 6.6.2.9 LookupValue() [2/2]

```
template<class OutputType >
bool ConfigReader::ConfigCollection::LookupValue (
    std::string_view SettingName,
    OutputType & theOutput) const
```

#### 6.6.2.10 LookupValueInteger() [1/7]

```
template<class OutputType >
bool ConfigReader::ConfigCollection::LookupValueInteger (
    int SettingIndex,
    OutputType & theOutput) const
```

#### 6.6.2.11 LookupValueInteger() [2/7]

```
bool ConfigCollection::LookupValueInteger (
    std::string_view SettingName,
    int & theOutput) const
```

**6.6.2.12 LookupValueInteger() [3/7]**

```
bool ConfigCollection::LookupValueInteger (
    std::string_view SettingName,
    long & theOutput) const
```

**6.6.2.13 LookupValueInteger() [4/7]**

```
bool ConfigCollection::LookupValueInteger (
    std::string_view SettingName,
    long long & theOutput) const
```

**6.6.2.14 LookupValueInteger() [5/7]**

```
bool ConfigCollection::LookupValueInteger (
    std::string_view SettingName,
    unsigned int & theOutput) const
```

**6.6.2.15 LookupValueInteger() [6/7]**

```
bool ConfigCollection::LookupValueInteger (
    std::string_view SettingName,
    unsigned long & theOutput) const
```

**6.6.2.16 LookupValueInteger() [7/7]**

```
bool ConfigCollection::LookupValueInteger (
    std::string_view SettingName,
    unsigned long long & theOutput) const
```

**6.6.2.17 NrSettings()**

```
int ConfigCollection::NrSettings () const
```

**6.6.2.18 operator[]() [1/2]**

```
const ConfigCollection & ConfigCollection::operator[] (
    int CollectionIndex) const
```

**6.6.2.19 operator[]() [2/2]**

```
const ConfigCollection & ConfigCollection::operator[] (
    std::string_view CollectionName) const
```

#### 6.6.2.20 ReadCollection()

```
void ConfigCollection::ReadCollection (
    std::ifstream & InputFile) [private]
```

#### 6.6.2.21 ReadFile()

```
bool ConfigCollection::ReadFile (
    const std::string & FileName)
```

#### 6.6.2.22 ReadSettingName()

```
void ConfigCollection::ReadSettingName (
    std::ifstream & InputFile,
    std::string & theName) [private]
```

#### 6.6.2.23 ReadSettingSpecificChar()

```
void ConfigCollection::ReadSettingSpecificChar (
    std::ifstream & InputFile,
    char theChar) const [private]
```

#### 6.6.2.24 ReadSettingValue()

```
void ConfigCollection::ReadSettingValue (
    std::ifstream & InputFile,
    ConfigSettingValue & theValue) [private]
```

### 6.6.3 Member Data Documentation

#### 6.6.3.1 m\_Settings

```
std::vector<ConfigSetting> ConfigReader::ConfigCollection::m_Settings {} [private]
```

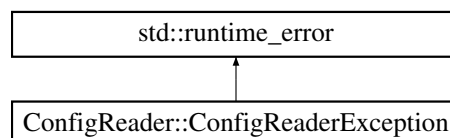
The documentation for this class was generated from the following files:

- FOORT/src/[ConfigReader.h](#)
- FOORT/src/[ConfigReader.cpp](#)

## 6.7 ConfigReader::ConfigReaderException Class Reference

```
#include <ConfigReader.h>
```

Inheritance diagram for ConfigReader::ConfigReaderException:



## Public Member Functions

- [ConfigReaderException](#) (const std::string &error, std::vector< int > settingtrace={})
- std::vector< int > [trace](#) () const

## Private Attributes

- std::vector< int > [m\\_settingtrace](#)

## 6.7.1 Constructor & Destructor Documentation

### 6.7.1.1 ConfigReaderException()

```
ConfigReader::ConfigReaderException::ConfigReaderException (
    const std::string & error,
    std::vector< int > settingtrace = {}) [inline]
```

## 6.7.2 Member Function Documentation

### 6.7.2.1 trace()

```
std::vector< int > ConfigReader::ConfigReaderException::trace () const [inline]
```

## 6.7.3 Member Data Documentation

### 6.7.3.1 m\_settingtrace

```
std::vector<int> ConfigReader::ConfigReaderException::m_settingtrace [private]
```

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

- FOORT/src/[ConfigReader.h](#)

## 6.8 ConfigReader::ConfigCollection::ConfigSetting Struct Reference

### Public Attributes

- std::string [SettingName](#)
- [ConfigSettingValue](#) [SettingValue](#)

## 6.8.1 Member Data Documentation

### 6.8.1.1 SettingName

```
std::string ConfigReader::ConfigCollection::ConfigSetting::SettingName
```

### 6.8.1.2 SettingValue

`ConfigSettingValue` `ConfigReader::ConfigCollection::ConfigSetting::SettingValue`

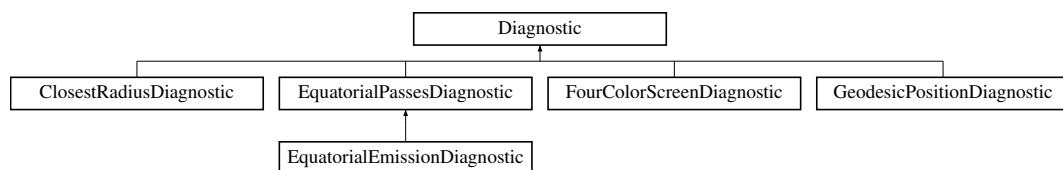
The documentation for this struct was generated from the following file:

- FOORT/src/[ConfigReader.h](#)

## 6.9 Diagnostic Class Reference

```
#include <Diagnostics.h>
```

Inheritance diagram for Diagnostic:



### Public Member Functions

- [Diagnostic](#) ()=delete
- [Diagnostic](#) ([Geodesic](#) \*const theGeodesic)
- virtual void [Reset](#) ()
- *[Diagnostic](#) (abstract base class) functions.*
- virtual [~Diagnostic](#) ()=default
- virtual void [UpdateData](#) ()=0
- virtual std::string [getFullDataStr](#) () const =0
- virtual std::vector< [real](#) > [getFinalDataVal](#) () const =0
- virtual [real](#) [FinalDataValDistance](#) (const std::vector< [real](#) > &val1, const std::vector< [real](#) > &val2) const =0
- virtual std::string [getNameStr](#) () const =0
- virtual std::string [getFullDescriptionStr](#) () const

### Protected Member Functions

- bool [DecideUpdate](#) (const [UpdateFrequency](#) &myUpdateFrequency)

### Protected Attributes

- [Geodesic](#) \*const [m\\_OwnerGeodesic](#)
- [largecounter](#) [m\\_StepsSinceUpdated](#) {}

## 6.9.1 Constructor & Destructor Documentation

### 6.9.1.1 Diagnostic() [1/2]

```
Diagnostic::Diagnostic () [delete]
```

### 6.9.1.2 Diagnostic() [2/2]

```
Diagnostic::Diagnostic (
    Geodesic *const theGeodesic) [inline]
```

### 6.9.1.3 ~Diagnostic()

```
virtual Diagnostic::~~Diagnostic () [virtual], [default]
```

## 6.9.2 Member Function Documentation

### 6.9.2.1 DecideUpdate()

```
bool Diagnostic::DecideUpdate (
    const UpdateFrequency & myUpdateFrequency) [protected]
```

### 6.9.2.2 FinalDataValDistance()

```
virtual real Diagnostic::FinalDataValDistance (
    const std::vector< real > & val1,
    const std::vector< real > & val2) const [pure virtual]
```

Implemented in [ClosestRadiusDiagnostic](#), [EquatorialEmissionDiagnostic](#), [EquatorialPassesDiagnostic](#), [FourColorScreenDiagnostic](#), and [GeodesicPositionDiagnostic](#).

### 6.9.2.3 getFinalDataVal()

```
virtual std::vector< real > Diagnostic::getFinalDataVal () const [pure virtual]
```

Implemented in [ClosestRadiusDiagnostic](#), [EquatorialEmissionDiagnostic](#), [EquatorialPassesDiagnostic](#), [FourColorScreenDiagnostic](#), and [GeodesicPositionDiagnostic](#).

### 6.9.2.4 getFullDataStr()

```
virtual std::string Diagnostic::getFullDataStr () const [pure virtual]
```

Implemented in [ClosestRadiusDiagnostic](#), [EquatorialEmissionDiagnostic](#), [EquatorialPassesDiagnostic](#), [FourColorScreenDiagnostic](#), and [GeodesicPositionDiagnostic](#).

### 6.9.2.5 getFullDescriptionStr()

```
std::string Diagnostic::getFullDescriptionStr () const [virtual]
```

Reimplemented in [ClosestRadiusDiagnostic](#), [EquatorialEmissionDiagnostic](#), [EquatorialPassesDiagnostic](#), [FourColorScreenDiagnostic](#), and [GeodesicPositionDiagnostic](#).

### 6.9.2.6 getNameStr()

```
virtual std::string Diagnostic::getNameStr () const [pure virtual]
```

Implemented in [ClosestRadiusDiagnostic](#), [EquatorialEmissionDiagnostic](#), [EquatorialPassesDiagnostic](#), [FourColorScreenDiagnostic](#), and [GeodesicPositionDiagnostic](#).

### 6.9.2.7 Reset()

```
void Diagnostic::Reset () [virtual]
```

[Diagnostic](#) (abstract base class) functions.

Reimplemented in [ClosestRadiusDiagnostic](#), [EquatorialEmissionDiagnostic](#), [EquatorialPassesDiagnostic](#), [FourColorScreenDiagnostic](#), and [GeodesicPositionDiagnostic](#).

### 6.9.2.8 UpdateData()

```
virtual void Diagnostic::UpdateData () [pure virtual]
```

Implemented in [ClosestRadiusDiagnostic](#), [EquatorialEmissionDiagnostic](#), [EquatorialPassesDiagnostic](#), [FourColorScreenDiagnostic](#), and [GeodesicPositionDiagnostic](#).

## 6.9.3 Member Data Documentation

### 6.9.3.1 m\_OwnerGeodesic

```
Geodesic* const Diagnostic::m_OwnerGeodesic [protected]
```

### 6.9.3.2 m\_StepsSinceUpdated

```
largecounter Diagnostic::m_StepsSinceUpdated {} [protected]
```

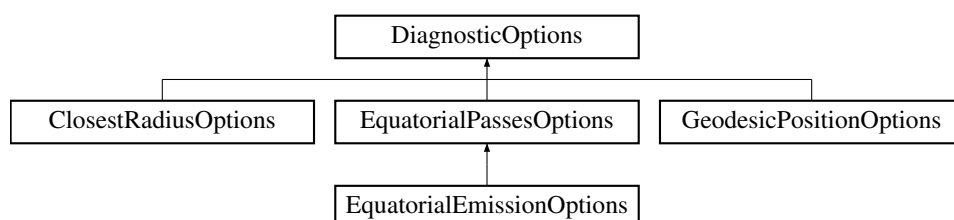
The documentation for this class was generated from the following files:

- FOORT/src/[Diagnostics.h](#)
- FOORT/src/[Diagnostics.cpp](#)

## 6.10 DiagnosticOptions Struct Reference

```
#include <Diagnostics.h>
```

Inheritance diagram for DiagnosticOptions:



## Public Member Functions

- [DiagnosticOptions](#) ([UpdateFrequency](#) thefrequency)
- virtual [~DiagnosticOptions](#) ()=default

## Public Attributes

- const [UpdateFrequency](#) theUpdateFrequency

## 6.10.1 Constructor & Destructor Documentation

### 6.10.1.1 DiagnosticOptions()

```
DiagnosticOptions::DiagnosticOptions (
    UpdateFrequency thefrequency) [inline]
```

### 6.10.1.2 ~DiagnosticOptions()

```
virtual DiagnosticOptions::~~DiagnosticOptions () [virtual], [default]
```

## 6.10.2 Member Data Documentation

### 6.10.2.1 theUpdateFrequency

```
const UpdateFrequency DiagnosticOptions::theUpdateFrequency
```

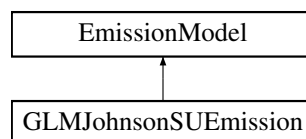
The documentation for this struct was generated from the following file:

- FOORT/src/[Diagnostics.h](#)

## 6.11 EmissionModel Struct Reference

```
#include <DiagnosticsEmission.h>
```

Inheritance diagram for EmissionModel:





## Public Member Functions

- virtual [~EmissionModel](#) ()=default
- virtual [real GetEmission](#) (const [Point](#) &p) const =0
- virtual std::string [getFullDescriptionStr](#) () const

*Emission model functions.*

## 6.11.1 Constructor & Destructor Documentation

### 6.11.1.1 ~EmissionModel()

```
virtual EmissionModel::~~EmissionModel () [virtual], [default]
```

## 6.11.2 Member Function Documentation

### 6.11.2.1 GetEmission()

```
virtual real EmissionModel::GetEmission (
    const Point & p) const [pure virtual]
```

Implemented in [GLMJohnsonSUEmission](#).

### 6.11.2.2 getFullDescriptionStr()

```
std::string EmissionModel::getFullDescriptionStr () const [virtual]
```

Emission model functions.

Reimplemented in [GLMJohnsonSUEmission](#).

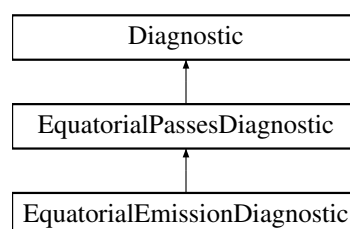
The documentation for this struct was generated from the following files:

- FOORT/src/[DiagnosticsEmission.h](#)
- FOORT/src/[DiagnosticsEmission.cpp](#)

## 6.12 EquatorialEmissionDiagnostic Class Reference

```
#include <Diagnostics.h>
```

Inheritance diagram for EquatorialEmissionDiagnostic:



## Public Member Functions

- [EquatorialEmissionDiagnostic](#) ([Geodesic](#) \*const theGeodesic)
- void [Reset](#) () final  
*[EquatorialEmissionDiagnostic](#) functions.*
- void [UpdateData](#) () final
- std::string [getFullDataStr](#) () const final
- std::vector< [real](#) > [getFinalDataVal](#) () const final
- [real](#) [FinalDataValDistance](#) (const std::vector< [real](#) > &val1, const std::vector< [real](#) > &val2) const final
- std::string [getNameStr](#) () const final
- std::string [getFullDescriptionStr](#) () const final

## Public Member Functions inherited from [EquatorialPassesDiagnostic](#)

- [EquatorialPassesDiagnostic](#) ([Geodesic](#) \*const theGeodesic)
- void [Reset](#) () override  
*[EquatorialPassesDiagnostic](#) functions.*
- void [UpdateData](#) () override
- std::string [getFullDataStr](#) () const override
- std::vector< [real](#) > [getFinalDataVal](#) () const override
- [real](#) [FinalDataValDistance](#) (const std::vector< [real](#) > &val1, const std::vector< [real](#) > &val2) const override
- std::string [getNameStr](#) () const override
- std::string [getFullDescriptionStr](#) () const override

## Public Member Functions inherited from [Diagnostic](#)

- [Diagnostic](#) ()=delete
- [Diagnostic](#) ([Geodesic](#) \*const theGeodesic)
- virtual [~Diagnostic](#) ()=default

## Static Public Attributes

- static std::unique\_ptr< [EquatorialEmissionOptions](#) > [DiagOptions](#)

## Static Public Attributes inherited from [EquatorialPassesDiagnostic](#)

- static std::unique\_ptr< [EquatorialPassesOptions](#) > [DiagOptions](#)

## Private Attributes

- [real](#) [m\\_Intensity](#) {0.0}

## Additional Inherited Members

## Protected Member Functions inherited from [Diagnostic](#)

- bool [DecideUpdate](#) (const [UpdateFrequency](#) &myUpdateFrequency)

## Protected Attributes inherited from [EquatorialPassesDiagnostic](#)

- `int m_EquatPasses {0}`

## Protected Attributes inherited from [Diagnostic](#)

- `Geodesic *const m_OwnerGeodesic`
- `largecounter m_StepsSinceUpdated {}`

## 6.12.1 Constructor & Destructor Documentation

### 6.12.1.1 EquatorialEmissionDiagnostic()

```
EquatorialEmissionDiagnostic::EquatorialEmissionDiagnostic (
    Geodesic *const theGeodesic) [inline]
```

## 6.12.2 Member Function Documentation

### 6.12.2.1 FinalDataValDistance()

```
real EquatorialEmissionDiagnostic::FinalDataValDistance (
    const std::vector< real > & val1,
    const std::vector< real > & val2) const [final], [virtual]
```

Implements [Diagnostic](#).

### 6.12.2.2 getFinalDataVal()

```
std::vector< real > EquatorialEmissionDiagnostic::getFinalDataVal () const [final], [virtual]
```

Implements [Diagnostic](#).

### 6.12.2.3 getFullDataStr()

```
std::string EquatorialEmissionDiagnostic::getFullDataStr () const [final], [virtual]
```

Implements [Diagnostic](#).

### 6.12.2.4 getFullDescriptionStr()

```
std::string EquatorialEmissionDiagnostic::getFullDescriptionStr () const [final], [virtual]
```

Reimplemented from [Diagnostic](#).

### 6.12.2.5 getNameStr()

```
std::string EquatorialEmissionDiagnostic::getNameStr () const [final], [virtual]
```

Implements [Diagnostic](#).

### 6.12.2.6 Reset()

```
void EquatorialEmissionDiagnostic::Reset () [final], [virtual]
```

[EquatorialEmissionDiagnostic](#) functions.

Reimplemented from [Diagnostic](#).

### 6.12.2.7 UpdateData()

```
void EquatorialEmissionDiagnostic::UpdateData () [final], [virtual]
```

Implements [Diagnostic](#).

## 6.12.3 Member Data Documentation

### 6.12.3.1 DiagOptions

```
std::unique_ptr< EquatorialEmissionOptions > EquatorialEmissionDiagnostic::DiagOptions [static]
```

### 6.12.3.2 m\_Intensity

```
real EquatorialEmissionDiagnostic::m_Intensity {0.0} [private]
```

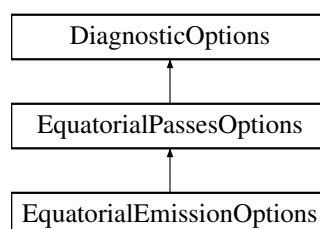
The documentation for this class was generated from the following files:

- FOORT/src/[Diagnostics.h](#)
- FOORT/src/[Config.cpp](#)
- FOORT/src/[Diagnostics.cpp](#)

## 6.13 EquatorialEmissionOptions Struct Reference

```
#include <Diagnostics.h>
```

Inheritance diagram for EquatorialEmissionOptions:



**Public Member Functions**

- [EquatorialEmissionOptions](#) ([real](#) thefudgefactor, int equatupper, std::unique\_ptr< [EmissionModel](#) > theemission, std::unique\_ptr< [FluidVelocityModel](#) > thefluidmodel, bool rlog, int theredshiftpower, [real](#) thethreshold, [UpdateFrequency](#) thefrequency)

**Public Member Functions inherited from [EquatorialPassesOptions](#)**

- [EquatorialPassesOptions](#) ([real](#) thethreshold, [UpdateFrequency](#) thefrequency)

**Public Member Functions inherited from [DiagnosticOptions](#)**

- [DiagnosticOptions](#) ([UpdateFrequency](#) thefrequency)
- virtual [~DiagnosticOptions](#) ()=default

**Public Attributes**

- const [real](#) GeometricFudgeFactor
- const int [EquatPassUpperBound](#)
- const bool [RLogScale](#)
- const int [RedShiftPower](#)
- const std::unique\_ptr< [EmissionModel](#) > [TheEmissionModel](#)
- const std::unique\_ptr< [FluidVelocityModel](#) > [TheFluidVelocityModel](#)

**Public Attributes inherited from [EquatorialPassesOptions](#)**

- const [real](#) [Threshold](#)

**Public Attributes inherited from [DiagnosticOptions](#)**

- const [UpdateFrequency](#) [theUpdateFrequency](#)

**6.13.1 Constructor & Destructor Documentation****6.13.1.1 [EquatorialEmissionOptions](#)()**

```
EquatorialEmissionOptions::EquatorialEmissionOptions (
    real thefudgefactor,
    int equatupper,
    std::unique_ptr< EmissionModel > theemission,
    std::unique_ptr< FluidVelocityModel > thefluidmodel,
    bool rlog,
    int theredshiftpower,
    real thethreshold,
    UpdateFrequency thefrequency) [inline]
```

## 6.13.2 Member Data Documentation

### 6.13.2.1 EquatPassUpperBound

```
const int EquatorialEmissionOptions::EquatPassUpperBound
```

### 6.13.2.2 GeometricFudgeFactor

```
const real EquatorialEmissionOptions::GeometricFudgeFactor
```

### 6.13.2.3 RedShiftPower

```
const int EquatorialEmissionOptions::RedShiftPower
```

### 6.13.2.4 RLogScale

```
const bool EquatorialEmissionOptions::RLogScale
```

### 6.13.2.5 TheEmissionModel

```
const std::unique_ptr<EmissionModel> EquatorialEmissionOptions::TheEmissionModel
```

### 6.13.2.6 TheFluidVelocityModel

```
const std::unique_ptr<FluidVelocityModel> EquatorialEmissionOptions::TheFluidVelocityModel
```

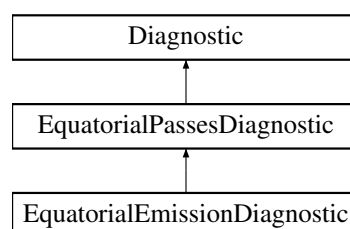
The documentation for this struct was generated from the following file:

- FOORT/src/[Diagnostics.h](#)

## 6.14 EquatorialPassesDiagnostic Class Reference

```
#include <Diagnostics.h>
```

Inheritance diagram for EquatorialPassesDiagnostic:



## Public Member Functions

- [EquatorialPassesDiagnostic](#) ([Geodesic](#) \*const theGeodesic)
- void [Reset](#) () override
- *[EquatorialPassesDiagnostic](#) functions.*
- void [UpdateData](#) () override
- std::string [getFullDataStr](#) () const override
- std::vector< [real](#) > [getFinalDataVal](#) () const override
- [real](#) [FinalDataValDistance](#) (const std::vector< [real](#) > &val1, const std::vector< [real](#) > &val2) const override
- std::string [getNameStr](#) () const override
- std::string [getFullDescriptionStr](#) () const override

## Public Member Functions inherited from [Diagnostic](#)

- [Diagnostic](#) ()=delete
- [Diagnostic](#) ([Geodesic](#) \*const theGeodesic)
- virtual [~Diagnostic](#) ()=default

## Static Public Attributes

- static std::unique\_ptr< [EquatorialPassesOptions](#) > [DiagOptions](#)

## Protected Attributes

- int [m\\_EquatPasses](#) {0}

## Protected Attributes inherited from [Diagnostic](#)

- [Geodesic](#) \*const [m\\_OwnerGeodesic](#)
- [largecounter](#) [m\\_StepsSinceUpdated](#) {}

## Private Attributes

- [real](#) [m\\_PrevTheta](#) {-1}

## Additional Inherited Members

## Protected Member Functions inherited from [Diagnostic](#)

- bool [DecideUpdate](#) (const [UpdateFrequency](#) &myUpdateFrequency)

## 6.14.1 Constructor & Destructor Documentation

### 6.14.1.1 [EquatorialPassesDiagnostic](#)()

```
EquatorialPassesDiagnostic::EquatorialPassesDiagnostic (
    Geodesic *const theGeodesic) [inline]
```

## 6.14.2 Member Function Documentation

### 6.14.2.1 FinalDataValDistance()

```
real EquatorialPassesDiagnostic::FinalDataValDistance (
    const std::vector< real > & val1,
    const std::vector< real > & val2) const [override], [virtual]
```

Implements [Diagnostic](#).

### 6.14.2.2 getFinalDataVal()

```
std::vector< real > EquatorialPassesDiagnostic::getFinalDataVal () const [override], [virtual]
```

Implements [Diagnostic](#).

### 6.14.2.3 getFullDataStr()

```
std::string EquatorialPassesDiagnostic::getFullDataStr () const [override], [virtual]
```

Implements [Diagnostic](#).

### 6.14.2.4 getFullDescriptionStr()

```
std::string EquatorialPassesDiagnostic::getFullDescriptionStr () const [override], [virtual]
```

Reimplemented from [Diagnostic](#).

### 6.14.2.5 getNameStr()

```
std::string EquatorialPassesDiagnostic::getNameStr () const [override], [virtual]
```

Implements [Diagnostic](#).

### 6.14.2.6 Reset()

```
void EquatorialPassesDiagnostic::Reset () [override], [virtual]
```

[EquatorialPassesDiagnostic](#) functions.

Reimplemented from [Diagnostic](#).

### 6.14.2.7 UpdateData()

```
void EquatorialPassesDiagnostic::UpdateData () [override], [virtual]
```

Implements [Diagnostic](#).



### 6.14.3 Member Data Documentation

#### 6.14.3.1 DiagOptions

```
std::unique_ptr< EquatorialPassesOptions > EquatorialPassesDiagnostic::DiagOptions [static]
```

#### 6.14.3.2 m\_EquatPasses

```
int EquatorialPassesDiagnostic::m_EquatPasses {0} [protected]
```

#### 6.14.3.3 m\_PrevTheta

```
real EquatorialPassesDiagnostic::m_PrevTheta {-1} [private]
```

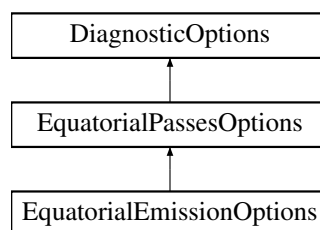
The documentation for this class was generated from the following files:

- FOORT/src/[Diagnostics.h](#)
- FOORT/src/[Config.cpp](#)
- FOORT/src/[Diagnostics.cpp](#)

## 6.15 EquatorialPassesOptions Struct Reference

```
#include <Diagnostics.h>
```

Inheritance diagram for EquatorialPassesOptions:



### Public Member Functions

- [EquatorialPassesOptions](#) ([real](#) thethreshold, [UpdateFrequency](#) thefrequency)

### Public Member Functions inherited from [DiagnosticOptions](#)

- [DiagnosticOptions](#) ([UpdateFrequency](#) thefrequency)
- virtual [~DiagnosticOptions](#) ()=default

### Public Attributes

- const [real](#) [Threshold](#)

## Public Attributes inherited from [DiagnosticOptions](#)

- const [UpdateFrequency](#) [theUpdateFrequency](#)

### 6.15.1 Constructor & Destructor Documentation

#### 6.15.1.1 EquatorialPassesOptions()

```
EquatorialPassesOptions::EquatorialPassesOptions (
    real thethreshold,
    UpdateFrequency thefrequency) [inline]
```

### 6.15.2 Member Data Documentation

#### 6.15.2.1 Threshold

```
const real EquatorialPassesOptions::Threshold
```

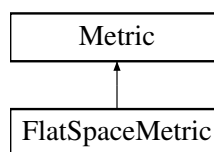
The documentation for this struct was generated from the following file:

- FOORT/src/[Diagnostics.h](#)

## 6.16 FlatSpaceMetric Class Reference

```
#include <Metric.h>
```

Inheritance diagram for FlatSpaceMetric:



### Public Member Functions

- [FlatSpaceMetric](#) (bool rlogscale=false)  
*FlatSpaceMetric functions.*
- [TwoIndex getMetric\\_dd](#) (const [Point](#) &p) const final
- [TwoIndex getMetric\\_uu](#) (const [Point](#) &p) const final
- std::string [getFullDescriptionStr](#) () const final

## Public Member Functions inherited from [Metric](#)

- virtual [~Metric](#) ()=default
- [Metric](#) (bool rlogscale=false)  
[Metric](#) (abstract base class) functions.
- virtual [ThreeIndex](#) [getChristoffel\\_udd](#) (const [Point](#) &p) const
- virtual [FourIndex](#) [getRiemann\\_uddd](#) (const [Point](#) &p) const
- virtual [real](#) [getKretschmann](#) (const [Point](#) &p) const
- bool [getrLogScale](#) () const

## Additional Inherited Members

## Protected Attributes inherited from [Metric](#)

- std::vector< int > [m\\_Symmetries](#) {}
- const bool [m\\_rLogScale](#)

## 6.16.1 Constructor & Destructor Documentation

### 6.16.1.1 FlatSpaceMetric()

```
FlatSpaceMetric::FlatSpaceMetric (
    bool rlogscale = false)
```

[FlatSpaceMetric](#) functions.

## 6.16.2 Member Function Documentation

### 6.16.2.1 getFullDescriptionStr()

```
std::string FlatSpaceMetric::getFullDescriptionStr () const [final], [virtual]
```

Reimplemented from [Metric](#).

### 6.16.2.2 getMetric\_dd()

```
TwoIndex FlatSpaceMetric::getMetric_dd (
    const Point & p) const [final], [virtual]
```

Implements [Metric](#).

### 6.16.2.3 getMetric\_uu()

```
TwoIndex FlatSpaceMetric::getMetric_uu (
    const Point & p) const [final], [virtual]
```

Implements [Metric](#).

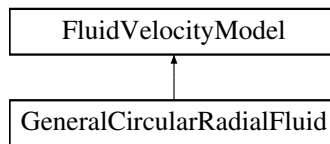
The documentation for this class was generated from the following files:

- FOORT/src/[Metric.h](#)
- FOORT/src/[Metric.cpp](#)

## 6.17 FluidVelocityModel Struct Reference

```
#include <DiagnosticsEmission.h>
```

Inheritance diagram for FluidVelocityModel:



### Public Member Functions

- [FluidVelocityModel](#) (const [Metric](#) \*const theMetric)
- virtual [~FluidVelocityModel](#) ()=default
- virtual [OneIndex GetFourVelocityd](#) (const [Point](#) &p) const =0
- virtual std::string [getFullDescriptionStr](#) () const

*[FluidVelocityModel](#) functions.*

### Protected Attributes

- const [Metric](#) \*const [m\\_theMetric](#)

### 6.17.1 Constructor & Destructor Documentation

#### 6.17.1.1 FluidVelocityModel()

```
FluidVelocityModel::FluidVelocityModel (
    const Metric *const theMetric) [inline]
```

#### 6.17.1.2 ~FluidVelocityModel()

```
virtual FluidVelocityModel::~~FluidVelocityModel () [virtual], [default]
```

### 6.17.2 Member Function Documentation

#### 6.17.2.1 GetFourVelocityd()

```
virtual OneIndex FluidVelocityModel::GetFourVelocityd (
    const Point & p) const [pure virtual]
```

Implemented in [GeneralCircularRadialFluid](#).

### 6.17.2.2 getFullDescriptionStr()

```
std::string FluidVelocityModel::getFullDescriptionStr () const [virtual]
```

[FluidVelocityModel](#) functions.

Reimplemented in [GeneralCircularRadialFluid](#).

## 6.17.3 Member Data Documentation

### 6.17.3.1 m\_theMetric

```
const Metric* const FluidVelocityModel::m_theMetric [protected]
```

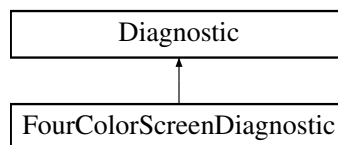
The documentation for this struct was generated from the following files:

- FOORT/src/[DiagnosticsEmission.h](#)
- FOORT/src/[DiagnosticsEmission.cpp](#)

## 6.18 FourColorScreenDiagnostic Class Reference

```
#include <Diagnostics.h>
```

Inheritance diagram for FourColorScreenDiagnostic:



### Public Member Functions

- [FourColorScreenDiagnostic](#) ([Geodesic](#) \*const theGeodesic)
- void [Reset](#) () final
- *FourColorScreen functions.*
- void [UpdateData](#) () override
- std::string [getFullDataStr](#) () const final
- std::vector< [real](#) > [getFinalDataVal](#) () const final
- [real](#) [FinalDataValDistance](#) (const std::vector< [real](#) > &val1, const std::vector< [real](#) > &val2) const final
- std::string [getNameStr](#) () const final
- std::string [getFullDescriptionStr](#) () const final

### Public Member Functions inherited from [Diagnostic](#)

- [Diagnostic](#) ()=delete
- [Diagnostic](#) ([Geodesic](#) \*const theGeodesic)
- virtual [~Diagnostic](#) ()=default

### Private Attributes

- int `m_quadrant` {0}

### Additional Inherited Members

### Protected Member Functions inherited from [Diagnostic](#)

- bool `DecideUpdate` (const [UpdateFrequency](#) &myUpdateFrequency)

### Protected Attributes inherited from [Diagnostic](#)

- [Geodesic](#) \*const `m_OwnerGeodesic`
- `largecounter` `m_StepsSinceUpdated` {}

## 6.18.1 Constructor & Destructor Documentation

### 6.18.1.1 FourColorScreenDiagnostic()

```
FourColorScreenDiagnostic::FourColorScreenDiagnostic (
    Geodesic *const theGeodesic) [inline]
```

## 6.18.2 Member Function Documentation

### 6.18.2.1 FinalDataValDistance()

```
real FourColorScreenDiagnostic::FinalDataValDistance (
    const std::vector< real > & val1,
    const std::vector< real > & val2) const [final], [virtual]
```

Implements [Diagnostic](#).

### 6.18.2.2 getFinalDataVal()

```
std::vector< real > FourColorScreenDiagnostic::getFinalDataVal () const [final], [virtual]
```

Implements [Diagnostic](#).

### 6.18.2.3 getFullDataStr()

```
std::string FourColorScreenDiagnostic::getFullDataStr () const [final], [virtual]
```

Implements [Diagnostic](#).

**6.18.2.4 getFullDescriptionStr()**

```
std::string FourColorScreenDiagnostic::getFullDescriptionStr () const [final], [virtual]
```

Reimplemented from [Diagnostic](#).

**6.18.2.5 getNameStr()**

```
std::string FourColorScreenDiagnostic::getNameStr () const [final], [virtual]
```

Implements [Diagnostic](#).

**6.18.2.6 Reset()**

```
void FourColorScreenDiagnostic::Reset () [final], [virtual]
```

FourColorScreen functions.

Reimplemented from [Diagnostic](#).

**6.18.2.7 UpdateData()**

```
void FourColorScreenDiagnostic::UpdateData () [override], [virtual]
```

Implements [Diagnostic](#).

**6.18.3 Member Data Documentation****6.18.3.1 m\_quadrant**

```
int FourColorScreenDiagnostic::m_quadrant {0} [private]
```

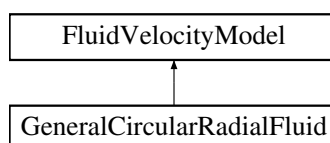
The documentation for this class was generated from the following files:

- FOORT/src/[Diagnostics.h](#)
- FOORT/src/[Diagnostics.cpp](#)

**6.19 GeneralCircularRadialFluid Struct Reference**

```
#include <DiagnosticsEmission.h>
```

Inheritance diagram for GeneralCircularRadialFluid:



## Public Member Functions

- [GeneralCircularRadialFluid](#) ([real](#) subKeplerParam, [real](#) betar, [real](#) betaphi, const [Metric](#) \*const theMetric)
  - [OneIndex GetFourVelocityd](#) (const [Point](#) &p) const final
  - std::string [getFullDescriptionStr](#) () const final
- [FluidVelocityModel](#) functions.*

## Public Member Functions inherited from [FluidVelocityModel](#)

- [FluidVelocityModel](#) (const [Metric](#) \*const theMetric)
- virtual [~FluidVelocityModel](#) ()=default

## Private Member Functions

- [OneIndex GetCircularVelocityd](#) (const [Point](#) &p, bool subKeplerianOn=true) const
- [OneIndex GetInsideISCOCircularVelocityd](#) (const [Point](#) &p) const
- [OneIndex GetRadialVelocityd](#) (const [Point](#) &p) const
- void [FindISCO](#) ()
- [TwoIndex GetChistrRaisedDer](#) ([real](#) r) const

## Private Attributes

- const [real](#) m\_subKeplerParam
- const [real](#) m\_betaR
- const [real](#) m\_betaPhi
- bool m\_ISCOexists {false}
- [real](#) m\_ISCO { -1.0 }
- [real](#) m\_ISCOpt {}
- [real](#) m\_ISCOpphi {}

## Additional Inherited Members

## Protected Attributes inherited from [FluidVelocityModel](#)

- const [Metric](#) \*const m\_theMetric

## 6.19.1 Constructor & Destructor Documentation

### 6.19.1.1 GeneralCircularRadialFluid()

```
GeneralCircularRadialFluid::GeneralCircularRadialFluid (
    real subKeplerParam,
    real betar,
    real betaphi,
    const Metric *const theMetric) [inline]
```



## 6.19.2 Member Function Documentation

### 6.19.2.1 FindISCO()

```
void GeneralCircularRadialFluid::FindISCO () [private]
```

### 6.19.2.2 GetChrstrRaisedDer()

```
TwoIndex GeneralCircularRadialFluid::GetChrstrRaisedDer (
    real r) const [private]
```

### 6.19.2.3 GetCircularVelocityd()

```
OneIndex GeneralCircularRadialFluid::GetCircularVelocityd (
    const Point & p,
    bool subKeplerianOn = true) const [private]
```

### 6.19.2.4 GetFourVelocityd()

```
OneIndex GeneralCircularRadialFluid::GetFourVelocityd (
    const Point & p) const [final], [virtual]
```

Implements [FluidVelocityModel](#).

### 6.19.2.5 getFullDescriptionStr()

```
std::string GeneralCircularRadialFluid::getFullDescriptionStr () const [final], [virtual]
```

[FluidVelocityModel](#) functions.

Reimplemented from [FluidVelocityModel](#).

### 6.19.2.6 GetInsideISCOCircularVelocityd()

```
OneIndex GeneralCircularRadialFluid::GetInsideISCOCircularVelocityd (
    const Point & p) const [private]
```

### 6.19.2.7 GetRadialVelocityd()

```
OneIndex GeneralCircularRadialFluid::GetRadialVelocityd (
    const Point & p) const [private]
```

### 6.19.3 Member Data Documentation

#### 6.19.3.1 m\_betaPhi

```
const real GeneralCircularRadialFluid::m_betaPhi [private]
```

#### 6.19.3.2 m\_betaR

```
const real GeneralCircularRadialFluid::m_betaR [private]
```

#### 6.19.3.3 m\_ISCOexists

```
bool GeneralCircularRadialFluid::m_ISCOexists {false} [private]
```

#### 6.19.3.4 m\_ISCOpphi

```
real GeneralCircularRadialFluid::m_ISCOpphi {} [private]
```

#### 6.19.3.5 m\_ISCOpt

```
real GeneralCircularRadialFluid::m_ISCOpt {} [private]
```

#### 6.19.3.6 m\_ISCOr

```
real GeneralCircularRadialFluid::m_ISCOr {-1.0} [private]
```

#### 6.19.3.7 m\_subKeplerParam

```
const real GeneralCircularRadialFluid::m_subKeplerParam [private]
```

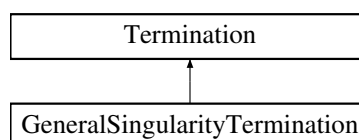
The documentation for this struct was generated from the following files:

- FOORT/src/[DiagnosticsEmission.h](#)
- FOORT/src/[DiagnosticsEmission.cpp](#)

## 6.20 GeneralSingularityTermination Class Reference

```
#include <Terminations.h>
```

Inheritance diagram for GeneralSingularityTermination:



**Public Member Functions**

- [GeneralSingularityTermination](#) ([Geodesic](#) \*const theGeodesic)
- [Term CheckTermination](#) () final  
*GeneralSingularityTermination functions.*
- std::string [getFullDescriptionStr](#) () const final

**Public Member Functions inherited from [Termination](#)**

- [Termination](#) ()=delete
- [Termination](#) ([Geodesic](#) \*const theGeodesic)
- virtual void [Reset](#) ()  
*Termination (abstract base class) functions.*
- virtual [~Termination](#) ()=default

**Static Public Attributes**

- static std::unique\_ptr< [GeneralSingularityTermOptions](#) > [TermOptions](#)

**Private Member Functions**

- std::string [SingularityToString](#) (int singnr) const

**Additional Inherited Members****Protected Member Functions inherited from [Termination](#)**

- bool [DecideUpdate](#) ([largecounter](#) UpdateNSteps)

**Protected Attributes inherited from [Termination](#)**

- [Geodesic](#) \*const [m\\_OwnerGeodesic](#)
- [largecounter](#) [m\\_StepsSinceUpdated](#) {}

**6.20.1 Constructor & Destructor Documentation****6.20.1.1 GeneralSingularityTermination()**

```
GeneralSingularityTermination::GeneralSingularityTermination (
    Geodesic *const theGeodesic) [inline]
```

**6.20.2 Member Function Documentation****6.20.2.1 CheckTermination()**

```
Term GeneralSingularityTermination::CheckTermination () [final], [virtual]
```

[GeneralSingularityTermination](#) functions.

Implements [Termination](#).

### 6.20.2.2 getFullDescriptionStr()

```
std::string GeneralSingularityTermination::getFullDescriptionStr () const [final], [virtual]
```

Implements [Termination](#).

### 6.20.2.3 SingularityToString()

```
std::string GeneralSingularityTermination::SingularityToString (
    int singnr) const [private]
```

## 6.20.3 Member Data Documentation

### 6.20.3.1 TermOptions

```
std::unique_ptr< GeneralSingularityTermOptions > GeneralSingularityTermination::TermOptions
[static]
```

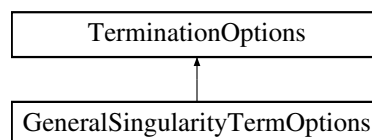
The documentation for this class was generated from the following files:

- FOORT/src/[Terminations.h](#)
- FOORT/src/[Config.cpp](#)
- FOORT/src/[Terminations.cpp](#)

## 6.21 GeneralSingularityTermOptions Struct Reference

```
#include <Terminations.h>
```

Inheritance diagram for GeneralSingularityTermOptions:



### Public Member Functions

- [GeneralSingularityTermOptions](#) (std::vector< [Singularity](#) > sings, [real](#) eps, bool consoleoutputon, bool therlogscale, [largecounter](#) Nsteps)

### Public Member Functions inherited from [TerminationOptions](#)

- [TerminationOptions](#) ([largecounter](#) Nsteps)
- virtual [~TerminationOptions](#) ()=default

## Public Attributes

- const std::vector< [Singularity](#) > [Singularities](#)
- const [real](#) [Epsilon](#)
- const bool [OutputToConsole](#)
- const bool [rLogScale](#)

## Public Attributes inherited from [TerminationOptions](#)

- const [largecounter](#) [UpdateEveryNSteps](#)

## 6.21.1 Constructor & Destructor Documentation

### 6.21.1.1 GeneralSingularityTermOptions()

```
GeneralSingularityTermOptions::GeneralSingularityTermOptions (
    std::vector< Singularity > sings,
    real eps,
    bool consoleoutputon,
    bool therlogscale,
    largecounter Nsteps) [inline]
```

## 6.21.2 Member Data Documentation

### 6.21.2.1 Epsilon

```
const real GeneralSingularityTermOptions::Epsilon
```

### 6.21.2.2 OutputToConsole

```
const bool GeneralSingularityTermOptions::OutputToConsole
```

### 6.21.2.3 rLogScale

```
const bool GeneralSingularityTermOptions::rLogScale
```

### 6.21.2.4 Singularities

```
const std::vector<Singularity> GeneralSingularityTermOptions::Singularities
```

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

- FOORT/src/[Terminations.h](#)

## 6.22 Geodesic Class Reference

```
#include <Geodesic.h>
```

### Public Member Functions

- [Geodesic](#) ()=delete
  - [Geodesic](#) (const [Geodesic](#) &)=delete
  - [Geodesic](#) & [operator=](#) (const [Geodesic](#) &)=delete
  - [Geodesic](#) (const [Metric](#) \*const theMetric, const [Source](#) \*const theSource, [DiagBitflag](#) diagbit, [DiagBitflag](#) valdiagbit, [TermBitflag](#) termbit, [GeodesicIntegratorFunc](#) theIntegrator)
  - void [Reset](#) ([ScreenIndex](#) scrindex, [Point](#) initpos, [OneIndex](#) initvel)
- [Geodesic](#) (and descendant classes) functions.*
- [Term Update](#) ()
  - [Term](#) [getTermCondition](#) () const
  - [Point](#) [getCurrentPos](#) () const
  - [OneIndex](#) [getCurrentVel](#) () const
  - [real](#) [getCurrentLambda](#) () const
  - [ScreenIndex](#) [getScreenIndex](#) () const
  - std::vector< std::string > [getAllOutputStr](#) () const
  - std::vector< [real](#) > [getDiagnosticFinalValue](#) () const

### Private Attributes

- [Term](#) [m\\_TermCond](#) {[Term::Uninitialized](#)}
- [Point](#) [m\\_CurrentPos](#) {}
- [OneIndex](#) [m\\_CurrentVel](#) {}
- [real](#) [m\\_curLambda](#) {0.0}
- [ScreenIndex](#) [m\\_ScreenIndex](#) {}
- const [Metric](#) \*const [m\\_theMetric](#)
- const [Source](#) \*const [m\\_theSource](#)
- const [DiagnosticUniqueVector](#) [m\\_AllDiagnostics](#)
- const [TerminationUniqueVector](#) [m\\_AllTerminations](#)
- const [GeodesicIntegratorFunc](#) [m\\_theIntegrator](#)

### 6.22.1 Constructor & Destructor Documentation

#### 6.22.1.1 [Geodesic](#)() [1/3]

```
Geodesic::Geodesic () [delete]
```

#### 6.22.1.2 [Geodesic](#)() [2/3]

```
Geodesic::Geodesic (
    const Geodesic & ) [delete]
```

### 6.22.1.3 Geodesic() [3/3]

```
Geodesic::Geodesic (
    const Metric *const theMetric,
    const Source *const theSource,
    DiagBitflag diagbit,
    DiagBitflag valdiagbit,
    TermBitflag termbit,
    GeodesicIntegratorFunc theIntegrator) [inline]
```

## 6.22.2 Member Function Documentation

### 6.22.2.1 getAllOutputStr()

```
std::vector< std::string > Geodesic::getAllOutputStr () const
```

### 6.22.2.2 getCurrentLambda()

```
real Geodesic::getCurrentLambda () const
```

### 6.22.2.3 getCurrentPos()

```
Point Geodesic::getCurrentPos () const
```

### 6.22.2.4 getCurrentVel()

```
OneIndex Geodesic::getCurrentVel () const
```

### 6.22.2.5 getDiagnosticFinalValue()

```
std::vector< real > Geodesic::getDiagnosticFinalValue () const
```

### 6.22.2.6 getScreenIndex()

```
ScreenIndex Geodesic::getScreenIndex () const
```

### 6.22.2.7 getTermCondition()

```
Term Geodesic::getTermCondition () const
```

### 6.22.2.8 operator=()

```
Geodesic & Geodesic::operator= (
    const Geodesic & ) [delete]
```

### 6.22.2.9 Reset()

```
void Geodesic::Reset (
    ScreenIndex scrindex,
    Point initpos,
    OneIndex initvel)
```

[Geodesic](#) (and descendant classes) functions.

### 6.22.2.10 Update()

```
Term Geodesic::Update ()
```

## 6.22.3 Member Data Documentation

### 6.22.3.1 m\_AllDiagnostics

```
const DiagnosticUniqueVector Geodesic::m_AllDiagnostics [private]
```

### 6.22.3.2 m\_AllTerminations

```
const TerminationUniqueVector Geodesic::m_AllTerminations [private]
```

### 6.22.3.3 m\_curLambda

```
real Geodesic::m_curLambda {0.0} [private]
```

### 6.22.3.4 m\_CurrentPos

```
Point Geodesic::m_CurrentPos {} [private]
```

### 6.22.3.5 m\_CurrentVel

```
OneIndex Geodesic::m_CurrentVel {} [private]
```

### 6.22.3.6 m\_ScreenIndex

```
ScreenIndex Geodesic::m_ScreenIndex {} [private]
```

### 6.22.3.7 m\_TermCond

```
Term Geodesic::m_TermCond {Term::Uninitialized} [private]
```



### 6.22.3.8 m\_theIntegrator

```
const GeodesicIntegratorFunc Geodesic::m_theIntegrator [private]
```

### 6.22.3.9 m\_theMetric

```
const Metric* const Geodesic::m_theMetric [private]
```

### 6.22.3.10 m\_theSource

```
const Source* const Geodesic::m_theSource [private]
```

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

- FOORT/src/[Geodesic.h](#)
- FOORT/src/[Geodesic.cpp](#)

## 6.23 GeodesicOutputHandler Class Reference

```
#include <InputOutput.h>
```

### Public Member Functions

- [GeodesicOutputHandler](#) ()=delete
- [GeodesicOutputHandler](#) (std::string FilePrefix, std::string TimeStamp, std::string FileExtension, std::vector< std::string > DiagNames, [largecounter](#) nroutputstocache=[LARGECOUNTER\\_MAX](#) - 1, [largecounter](#) geodperfile=[LARGECOUNTER\\_MAX](#), std::string firstlineinfo="")  
[GeodesicOutputHandler](#) functions.
- void [PrepareForOutput](#) ([largecounter](#) nrOutputToCome)
- void [NewGeodesicOutput](#) ([largecounter](#) index, std::vector< std::string > theOutput)
- void [OutputFinished](#) ()
- std::string [getFullDescriptionStr](#) () const

### Private Member Functions

- void [WriteCachedOutputToFile](#) ()
- void [OpenForFirstTime](#) (const std::string &filename)
- std::string [GetFileName](#) (int diagnr, unsigned short filenr) const

## Private Attributes

- const std::string [m\\_FilePrefix](#)
- const std::string [m\\_TimeStamp](#)
- const std::string [m\\_FileExtension](#)
- const std::vector< std::string > [m\\_DiagNames](#)
- const bool [m\\_PrintFirstLineInfo](#)
- const std::string [m\\_FirstLineInfoString](#)
- const [largecounter](#) [m\\_nrOutputsToCache](#) {}
- const [largecounter](#) [m\\_nrGeodesicsPerFile](#) {}
- bool [m\\_WriteToConsole](#) {false}
- [largecounter](#) [m\\_PrevCached](#) {0}
- [largecounter](#) [m\\_CurrentGeodesicsInFile](#) {0}
- unsigned short [m\\_CurrentFullFiles](#) {0}
- std::vector< std::vector< std::string > > [m\\_AllCachedData](#) {}

## 6.23.1 Constructor & Destructor Documentation

### 6.23.1.1 GeodesicOutputHandler() [1/2]

```
GeodesicOutputHandler::GeodesicOutputHandler () [delete]
```

### 6.23.1.2 GeodesicOutputHandler() [2/2]

```
GeodesicOutputHandler::GeodesicOutputHandler (
    std::string FilePrefix,
    std::string TimeStamp,
    std::string FileExtension,
    std::vector< std::string > DiagNames,
    largecounter nrouputstocache = LARGECOUNTER\_MAX - 1,
    largecounter geodperfile = LARGECOUNTER\_MAX,
    std::string firstlineinfo = "")
```

[GeodesicOutputHandler](#) functions.

## 6.23.2 Member Function Documentation

### 6.23.2.1 GetFileName()

```
std::string GeodesicOutputHandler::GetFileName (
    int diagnr,
    unsigned short filenr) const [private]
```

### 6.23.2.2 getFullDescriptionStr()

```
std::string GeodesicOutputHandler::getFullDescriptionStr () const
```

### 6.23.2.3 NewGeodesicOutput()

```
void GeodesicOutputHandler::NewGeodesicOutput (
    largecounter index,
    std::vector< std::string > theOutput)
```

### 6.23.2.4 OpenForFirstTime()

```
void GeodesicOutputHandler::OpenForFirstTime (
    const std::string & filename) [private]
```

### 6.23.2.5 OutputFinished()

```
void GeodesicOutputHandler::OutputFinished ()
```

### 6.23.2.6 PrepareForOutput()

```
void GeodesicOutputHandler::PrepareForOutput (
    largecounter nrOutputToCome)
```

### 6.23.2.7 WriteCachedOutputToFile()

```
void GeodesicOutputHandler::WriteCachedOutputToFile () [private]
```

## 6.23.3 Member Data Documentation

### 6.23.3.1 m\_AllCachedData

```
std::vector<std::vector<std::string> > GeodesicOutputHandler::m_AllCachedData {} [private]
```

### 6.23.3.2 m\_CurrentFullFiles

```
unsigned short GeodesicOutputHandler::m_CurrentFullFiles {0} [private]
```

### 6.23.3.3 m\_CurrentGeodesicsInFile

```
largecounter GeodesicOutputHandler::m_CurrentGeodesicsInFile {0} [private]
```

### 6.23.3.4 m\_DiagNames

```
const std::vector<std::string> GeodesicOutputHandler::m_DiagNames [private]
```

#### 6.23.3.5 m\_FileExtension

```
const std::string GeodesicOutputHandler::m_FileExtension [private]
```

#### 6.23.3.6 m\_FilePrefix

```
const std::string GeodesicOutputHandler::m_FilePrefix [private]
```

#### 6.23.3.7 m\_FirstLineInfoString

```
const std::string GeodesicOutputHandler::m_FirstLineInfoString [private]
```

#### 6.23.3.8 m\_nrGeodesicsPerFile

```
const largecounter GeodesicOutputHandler::m_nrGeodesicsPerFile {} [private]
```

#### 6.23.3.9 m\_nrOutputsToCache

```
const largecounter GeodesicOutputHandler::m_nrOutputsToCache {} [private]
```

#### 6.23.3.10 m\_PrevCached

```
largecounter GeodesicOutputHandler::m_PrevCached {0} [private]
```

#### 6.23.3.11 m\_PrintFirstLineInfo

```
const bool GeodesicOutputHandler::m_PrintFirstLineInfo [private]
```

#### 6.23.3.12 m\_TimeStamp

```
const std::string GeodesicOutputHandler::m_TimeStamp [private]
```

#### 6.23.3.13 m\_WriteToConsole

```
bool GeodesicOutputHandler::m_WriteToConsole {false} [private]
```

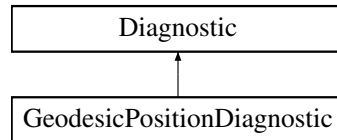
The documentation for this class was generated from the following files:

- [FOORT/src/InOutOutput.h](#)
- [FOORT/src/InOutOutput.cpp](#)

## 6.24 GeodesicPositionDiagnostic Class Reference

```
#include <Diagnostics.h>
```

Inheritance diagram for GeodesicPositionDiagnostic:



### Public Member Functions

- [GeodesicPositionDiagnostic](#) ([Geodesic](#) \*const theGeodesic)
- void [Reset](#) () final
- *[GeodesicPositionDiagnostic](#) functions.*
- void [UpdateData](#) () final
- std::string [getFullDataStr](#) () const final
- std::vector< [real](#) > [getFinalDataVal](#) () const final
- [real](#) [FinalDataValDistance](#) (const std::vector< [real](#) > &val1, const std::vector< [real](#) > &val2) const final
- std::string [getNameStr](#) () const final
- std::string [getFullDescriptionStr](#) () const final

### Public Member Functions inherited from [Diagnostic](#)

- [Diagnostic](#) ()=delete
- [Diagnostic](#) ([Geodesic](#) \*const theGeodesic)
- virtual [~Diagnostic](#) ()=default

### Static Public Attributes

- static std::unique\_ptr< [GeodesicPositionOptions](#) > [DiagOptions](#)

### Private Attributes

- std::vector< [Point](#) > [m\\_AllSavedPoints](#) {}

### Additional Inherited Members

### Protected Member Functions inherited from [Diagnostic](#)

- bool [DecideUpdate](#) (const [UpdateFrequency](#) &myUpdateFrequency)

### Protected Attributes inherited from [Diagnostic](#)

- [Geodesic](#) \*const [m\\_OwnerGeodesic](#)
- [largecounter](#) [m\\_StepsSinceUpdated](#) {}

## 6.24.1 Constructor & Destructor Documentation

### 6.24.1.1 GeodesicPositionDiagnostic()

```
GeodesicPositionDiagnostic::GeodesicPositionDiagnostic (  
    Geodesic *const theGeodesic) [inline]
```

## 6.24.2 Member Function Documentation

### 6.24.2.1 FinalDataValDistance()

```
real GeodesicPositionDiagnostic::FinalDataValDistance (  
    const std::vector< real > & val1,  
    const std::vector< real > & val2) const [final], [virtual]
```

Implements [Diagnostic](#).

### 6.24.2.2 getFinalDataVal()

```
std::vector< real > GeodesicPositionDiagnostic::getFinalDataVal () const [final], [virtual]
```

Implements [Diagnostic](#).

### 6.24.2.3 getFullDataStr()

```
std::string GeodesicPositionDiagnostic::getFullDataStr () const [final], [virtual]
```

Implements [Diagnostic](#).

### 6.24.2.4 getFullDescriptionStr()

```
std::string GeodesicPositionDiagnostic::getFullDescriptionStr () const [final], [virtual]
```

Reimplemented from [Diagnostic](#).

### 6.24.2.5 getNameStr()

```
std::string GeodesicPositionDiagnostic::getNameStr () const [final], [virtual]
```

Implements [Diagnostic](#).

### 6.24.2.6 Reset()

```
void GeodesicPositionDiagnostic::Reset () [final], [virtual]
```

[GeodesicPositionDiagnostic](#) functions.

Reimplemented from [Diagnostic](#).

### 6.24.2.7 UpdateData()

```
void GeodesicPositionDiagnostic::UpdateData () [final], [virtual]
```

Implements [Diagnostic](#).

## 6.24.3 Member Data Documentation

### 6.24.3.1 DiagOptions

```
std::unique_ptr< GeodesicPositionOptions > GeodesicPositionDiagnostic::DiagOptions [static]
```

### 6.24.3.2 m\_AllSavedPoints

```
std::vector<Point> GeodesicPositionDiagnostic::m_AllSavedPoints {} [private]
```

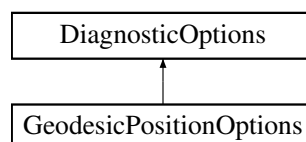
The documentation for this class was generated from the following files:

- FOORT/src/[Diagnostics.h](#)
- FOORT/src/[Config.cpp](#)
- FOORT/src/[Diagnostics.cpp](#)

## 6.25 GeodesicPositionOptions Struct Reference

```
#include <Diagnostics.h>
```

Inheritance diagram for GeodesicPositionOptions:



### Public Member Functions

- [GeodesicPositionOptions](#) ([largecounter](#) outputsteps, [UpdateFrequency](#) thefrequency)

### Public Member Functions inherited from [DiagnosticOptions](#)

- [DiagnosticOptions](#) ([UpdateFrequency](#) thefrequency)
- virtual [~DiagnosticOptions](#) ()=default

### Public Attributes

- const [largecounter](#) OutputNrSteps

## Public Attributes inherited from [DiagnosticOptions](#)

- const [UpdateFrequency](#) *theUpdateFrequency*

## 6.25.1 Constructor & Destructor Documentation

### 6.25.1.1 GeodesicPositionOptions()

```
GeodesicPositionOptions::GeodesicPositionOptions (
    largecounter outputsteps,
    UpdateFrequency thefrequency) [inline]
```

## 6.25.2 Member Data Documentation

### 6.25.2.1 OutputNrSteps

```
const largecounter GeodesicPositionOptions::OutputNrSteps
```

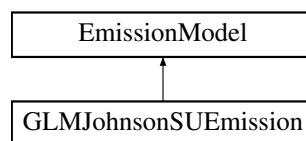
The documentation for this struct was generated from the following file:

- FOORT/src/[Diagnostics.h](#)

## 6.26 GLMJohnsonSUEmission Struct Reference

```
#include <DiagnosticsEmission.h>
```

Inheritance diagram for GLMJohnsonSUEmission:



## Public Member Functions

- [GLMJohnsonSUEmission](#) ([real](#) mu, [real](#) gamma, [real](#) sigma)
- [real](#) [GetEmission](#) (const [Point](#) &p) const final
- std::string [getFullDescriptionStr](#) () const final

*Emission model functions.*

## Public Member Functions inherited from [EmissionModel](#)

- virtual [~EmissionModel](#) ()=default



### Private Attributes

- const [real](#) `m_mu`
- const [real](#) `m_gamma`
- const [real](#) `m_sigma`

## 6.26.1 Constructor & Destructor Documentation

### 6.26.1.1 GLMJohnsonSUEmission()

```
GLMJohnsonSUEmission::GLMJohnsonSUEmission (  
    real mu,  
    real gamma,  
    real sigma) [inline]
```

## 6.26.2 Member Function Documentation

### 6.26.2.1 GetEmission()

```
real GLMJohnsonSUEmission::GetEmission (  
    const Point & p) const [final], [virtual]
```

Implements [EmissionModel](#).

### 6.26.2.2 getFullDescriptionStr()

```
std::string GLMJohnsonSUEmission::getFullDescriptionStr () const [final], [virtual]
```

Emission model functions.

Reimplemented from [EmissionModel](#).

## 6.26.3 Member Data Documentation

### 6.26.3.1 m\_gamma

```
const real GLMJohnsonSUEmission::m_gamma [private]
```

### 6.26.3.2 m\_mu

```
const real GLMJohnsonSUEmission::m_mu [private]
```

### 6.26.3.3 m\_sigma

```
const real GLMJohnsonSUEmission::m_sigma [private]
```

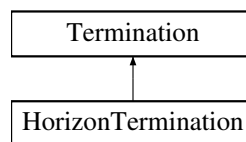
The documentation for this struct was generated from the following files:

- FOORT/src/[DiagnosticsEmission.h](#)
- FOORT/src/[DiagnosticsEmission.cpp](#)

## 6.27 HorizonTermination Class Reference

```
#include <Terminations.h>
```

Inheritance diagram for HorizonTermination:



### Public Member Functions

- [HorizonTermination](#) ([Geodesic](#) \*const theGeodesic)
- [Term CheckTermination](#) () final  
*[HorizonTermination](#) functions.*
- std::string [getFullDescriptionStr](#) () const final

### Public Member Functions inherited from [Termination](#)

- [Termination](#) ()=delete
- [Termination](#) ([Geodesic](#) \*const theGeodesic)
- virtual void [Reset](#) ()  
*[Termination](#) (abstract base class) functions.*
- virtual [~Termination](#) ()=default

### Static Public Attributes

- static std::unique\_ptr< [HorizonTermOptions](#) > [TermOptions](#)

### Additional Inherited Members

### Protected Member Functions inherited from [Termination](#)

- bool [DecideUpdate](#) ([largecounter](#) UpdateNSteps)

## Protected Attributes inherited from [Termination](#)

- [Geodesic](#) \*const [m\\_OwnerGeodesic](#)
- [largecounter](#) [m\\_StepsSinceUpdated](#) {}

## 6.27.1 Constructor & Destructor Documentation

### 6.27.1.1 [HorizonTermination](#)()

```
HorizonTermination::HorizonTermination (
    Geodesic *const theGeodesic) [inline]
```

## 6.27.2 Member Function Documentation

### 6.27.2.1 [CheckTermination](#)()

```
Term HorizonTermination::CheckTermination () [final], [virtual]
```

[HorizonTermination](#) functions.

Implements [Termination](#).

### 6.27.2.2 [getFullDescriptionStr](#)()

```
std::string HorizonTermination::getFullDescriptionStr () const [final], [virtual]
```

Implements [Termination](#).

## 6.27.3 Member Data Documentation

### 6.27.3.1 [TermOptions](#)

```
std::unique_ptr< HorizonTermOptions > HorizonTermination::TermOptions [static]
```

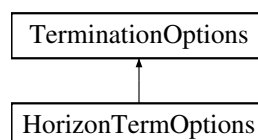
The documentation for this class was generated from the following files:

- FOORT/src/[Terminations.h](#)
- FOORT/src/[Config.cpp](#)
- FOORT/src/[Terminations.cpp](#)

## 6.28 [HorizonTermOptions](#) Struct Reference

```
#include <Terminations.h>
```

Inheritance diagram for [HorizonTermOptions](#):



## Public Member Functions

- [HorizonTermOptions](#) ([real](#) theHorizonRadius, [bool](#) therLogScale, [real](#) theAtHorizonEps, [largecounter](#) Nsteps)

## Public Member Functions inherited from [TerminationOptions](#)

- [TerminationOptions](#) ([largecounter](#) Nsteps)
- virtual [~TerminationOptions](#) ()=default

## Public Attributes

- const [real](#) [HorizonRadius](#)
- const [real](#) [AtHorizonEps](#)
- const [bool](#) [rLogScale](#)

## Public Attributes inherited from [TerminationOptions](#)

- const [largecounter](#) [UpdateEveryNSteps](#)

## 6.28.1 Constructor & Destructor Documentation

### 6.28.1.1 [HorizonTermOptions\(\)](#)

```
HorizonTermOptions::HorizonTermOptions (
    real theHorizonRadius,
    bool therLogScale,
    real theAtHorizonEps,
    largecounter Nsteps) [inline]
```

## 6.28.2 Member Data Documentation

### 6.28.2.1 [AtHorizonEps](#)

```
const real HorizonTermOptions::AtHorizonEps
```

### 6.28.2.2 [HorizonRadius](#)

```
const real HorizonTermOptions::HorizonRadius
```

### 6.28.2.3 [rLogScale](#)

```
const bool HorizonTermOptions::rLogScale
```

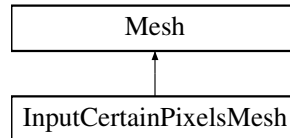
The documentation for this struct was generated from the following file:

- FOORT/src/[Terminations.h](#)

## 6.29 InputCertainPixelsMesh Class Reference

```
#include <Mesh.h>
```

Inheritance diagram for InputCertainPixelsMesh:



### Public Member Functions

- [InputCertainPixelsMesh](#) ()=delete
- [InputCertainPixelsMesh](#) (const [InputCertainPixelsMesh](#) &)=delete
- [InputCertainPixelsMesh](#) ([largecounter](#) totalPixels, [DiagBitflag](#) valdiag)
- *[InputCertainPixelsMesh](#) functions.*
- [largecounter](#) [getCurNrGeodesics](#) () const final
- void [getNewInitConds](#) ([largecounter](#) index, [ScreenPoint](#) &newunitpoint, [ScreenIndex](#) &newscreenindex) const final
- void [GeodesicFinished](#) ([largecounter](#) index, std::vector< [real](#) > finalValues) final
- void [EndCurrentLoop](#) () final
- bool [IsFinished](#) () const final
- std::string [getFullDescriptionStr](#) () const final
- *[Mesh](#) (abstract base class) functions.*

### Public Member Functions inherited from [Mesh](#)

- [Mesh](#) ([DiagBitflag](#) valdiag)
- virtual [~Mesh](#) ()=default

### Private Attributes

- const [pixelcoord](#) [m\\_RowColumnSize](#)
- [largecounter](#) [m\\_TotalPixels](#) {0}
- std::vector< [ScreenIndex](#) > [m\\_PixelsToIntegrate](#) {}
- bool [m\\_Finished](#) {false}

### Additional Inherited Members

### Protected Attributes inherited from [Mesh](#)

- const std::unique\_ptr< const [Diagnostic](#) > [m\\_DistanceDiagnostic](#)

## 6.29.1 Constructor & Destructor Documentation

### 6.29.1.1 InputCertainPixelsMesh() [1/3]

```
InputCertainPixelsMesh::InputCertainPixelsMesh () [delete]
```

### 6.29.1.2 InputCertainPixelsMesh() [2/3]

```
InputCertainPixelsMesh::InputCertainPixelsMesh (
    const InputCertainPixelsMesh & ) [delete]
```

### 6.29.1.3 InputCertainPixelsMesh() [3/3]

```
InputCertainPixelsMesh::InputCertainPixelsMesh (
    largecounter totalPixels,
    DiagBitflag valdiag)
```

[InputCertainPixelsMesh](#) functions.

## 6.29.2 Member Function Documentation

### 6.29.2.1 EndCurrentLoop()

```
void InputCertainPixelsMesh::EndCurrentLoop () [final], [virtual]
```

Implements [Mesh](#).

### 6.29.2.2 GeodesicFinished()

```
void InputCertainPixelsMesh::GeodesicFinished (
    largecounter index,
    std::vector< real > finalValues) [final], [virtual]
```

Implements [Mesh](#).

### 6.29.2.3 getCurNrGeodesics()

```
largecounter InputCertainPixelsMesh::getCurNrGeodesics () const [final], [virtual]
```

Implements [Mesh](#).

### 6.29.2.4 getFullDescriptionStr()

```
std::string InputCertainPixelsMesh::getFullDescriptionStr () const [final], [virtual]
```

[Mesh](#) (abstract base class) functions.

Reimplemented from [Mesh](#).

### 6.29.2.5 getNewInitConds()

```
void InputCertainPixelsMesh::getNewInitConds (
    largecounter index,
    ScreenPoint & newunitpoint,
    ScreenIndex & newscreenindex) const [final], [virtual]
```

Implements [Mesh](#).

### 6.29.2.6 IsFinished()

```
bool InputCertainPixelsMesh::IsFinished () const [final], [virtual]
```

Implements [Mesh](#).

## 6.29.3 Member Data Documentation

### 6.29.3.1 m\_Finished

```
bool InputCertainPixelsMesh::m_Finished {false} [private]
```

### 6.29.3.2 m\_PixelsToIntegrate

```
std::vector<ScreenIndex> InputCertainPixelsMesh::m_PixelsToIntegrate {} [private]
```

### 6.29.3.3 m\_RowColumnSize

```
const pixelcoord InputCertainPixelsMesh::m_RowColumnSize [private]
```

### 6.29.3.4 m\_TotalPixels

```
largecounter InputCertainPixelsMesh::m_TotalPixels {0} [private]
```

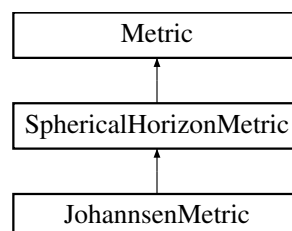
The documentation for this class was generated from the following files:

- FOORT/src/[Mesh.h](#)
- FOORT/src/[Mesh.cpp](#)

## 6.30 JohannsenMetric Class Reference

```
#include <Metric.h>
```

Inheritance diagram for JohannsenMetric:



### Public Member Functions

- [JohannsenMetric](#) ()=delete
- [JohannsenMetric](#) ([real](#) aParam, [real](#) alpha13Param, [real](#) alpha22Param, [real](#) alpha52Param, [real](#) eps3Param, [bool](#) rLogScale=false)
- *[JohannsenMetric](#) functions (implementation by Seppe Staelens)*
- [TwoIndex getMetric\\_dd](#) (const [Point](#) &p) const final
- [TwoIndex getMetric\\_uu](#) (const [Point](#) &p) const final
- [std::string getFullDescriptionStr](#) () const final

### Public Member Functions inherited from [SphericalHorizonMetric](#)

- [SphericalHorizonMetric](#) ()=delete
- [SphericalHorizonMetric](#) ([real](#) HorizonRadius, [bool](#) rLogScale)
- *[SphericalHorizonMetric](#) functions.*
- [real getHorizonRadius](#) () const

### Public Member Functions inherited from [Metric](#)

- virtual [~Metric](#) ()=default
- [Metric](#) ([bool](#) rlogscale=false)
- *[Metric](#) (abstract base class) functions.*
- virtual [ThreeIndex getChristoffel\\_udd](#) (const [Point](#) &p) const
- virtual [FourIndex getRiemann\\_uddd](#) (const [Point](#) &p) const
- virtual [real getKretschmann](#) (const [Point](#) &p) const
- [bool getrLogScale](#) () const

### Private Attributes

- const [real](#) m\_aParam
- const [real](#) m\_alpha13Param
- const [real](#) m\_alpha22Param
- const [real](#) m\_alpha52Param
- const [real](#) m\_eps3Param

### Additional Inherited Members

### Protected Attributes inherited from [SphericalHorizonMetric](#)

- const [real](#) m\_HorizonRadius

### Protected Attributes inherited from [Metric](#)

- [std::vector< int > m\\_Symmetries](#) {}
- const [bool](#) m\_rLogScale



## 6.30.1 Constructor & Destructor Documentation

### 6.30.1.1 JohanssenMetric() [1/2]

JohanssenMetric::JohanssenMetric () [delete]

### 6.30.1.2 JohanssenMetric() [2/2]

```
JohanssenMetric::JohanssenMetric (  
    real aParam,  
    real alpha13Param,  
    real alpha22Param,  
    real alpha52Param,  
    real eps3Param,  
    bool rLogScale = false)
```

[JohanssenMetric](#) functions (implementation by Seppe Staelens)

## 6.30.2 Member Function Documentation

### 6.30.2.1 getFullDescriptionStr()

```
std::string JohanssenMetric::getFullDescriptionStr () const [final], [virtual]
```

Reimplemented from [Metric](#).

### 6.30.2.2 getMetric\_dd()

```
TwoIndex JohanssenMetric::getMetric_dd (  
    const Point & p) const [final], [virtual]
```

Implements [Metric](#).

### 6.30.2.3 getMetric\_uu()

```
TwoIndex JohanssenMetric::getMetric_uu (  
    const Point & p) const [final], [virtual]
```

Implements [Metric](#).

## 6.30.3 Member Data Documentation

### 6.30.3.1 m\_alpha13Param

```
const real JohanssenMetric::m_alpha13Param [private]
```

### 6.30.3.2 m\_alpha22Param

```
const real JohannsenMetric::m_alpha22Param [private]
```

### 6.30.3.3 m\_alpha52Param

```
const real JohannsenMetric::m_alpha52Param [private]
```

### 6.30.3.4 m\_aParam

```
const real JohannsenMetric::m_aParam [private]
```

### 6.30.3.5 m\_eps3Param

```
const real JohannsenMetric::m_eps3Param [private]
```

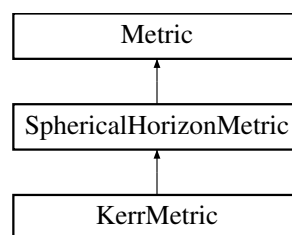
The documentation for this class was generated from the following files:

- [FOORT/src/Metric.h](#)
- [FOORT/src/Metric.cpp](#)

## 6.31 KerrMetric Class Reference

```
#include <Metric.h>
```

Inheritance diagram for KerrMetric:



### Public Member Functions

- [KerrMetric](#) ()=delete
- [KerrMetric](#) (real aParam, bool rLogScale=false, real mParam=1.)  
*KerrMetric* functions.
- [TwoIndex](#) getMetric\_dd (const [Point](#) &p) const final
- [TwoIndex](#) getMetric\_uu (const [Point](#) &p) const final
- std::string [getFullDescriptionStr](#) () const final

## Public Member Functions inherited from [SphericalHorizonMetric](#)

- [SphericalHorizonMetric](#) ()=delete
- [SphericalHorizonMetric](#) (real HorizonRadius, bool rLogScale)  
*SphericalHorizonMetric functions.*
- [real getHorizonRadius](#) () const

## Public Member Functions inherited from [Metric](#)

- virtual [~Metric](#) ()=default
- [Metric](#) (bool rlogscale=false)  
*Metric (abstract base class) functions.*
- virtual [ThreeIndex getChristoffel\\_udd](#) (const [Point](#) &p) const
- virtual [FourIndex getRiemann\\_uddd](#) (const [Point](#) &p) const
- virtual [real getKretschmann](#) (const [Point](#) &p) const
- bool [getrLogScale](#) () const

## Private Attributes

- const [real m\\_aParam](#)
- const [real m\\_mParam](#)

## Additional Inherited Members

## Protected Attributes inherited from [SphericalHorizonMetric](#)

- const [real m\\_HorizonRadius](#)

## Protected Attributes inherited from [Metric](#)

- `std::vector< int > m_Symmetries {}`
- const bool [m\\_rLogScale](#)

### 6.31.1 Constructor & Destructor Documentation

#### 6.31.1.1 [KerrMetric\(\)](#) [1/2]

```
KerrMetric::KerrMetric () [delete]
```

#### 6.31.1.2 [KerrMetric\(\)](#) [2/2]

```
KerrMetric::KerrMetric (
    real aParam,
    bool rLogScale = false,
    real mParam = 1.)
```

[KerrMetric](#) functions.

## 6.31.2 Member Function Documentation

### 6.31.2.1 getFullDescriptionStr()

```
std::string KerrMetric::getFullDescriptionStr () const [final], [virtual]
```

Reimplemented from [Metric](#).

### 6.31.2.2 getMetric\_dd()

```
TwoIndex KerrMetric::getMetric_dd (
    const Point & p) const [final], [virtual]
```

Implements [Metric](#).

### 6.31.2.3 getMetric\_uu()

```
TwoIndex KerrMetric::getMetric_uu (
    const Point & p) const [final], [virtual]
```

Implements [Metric](#).

## 6.31.3 Member Data Documentation

### 6.31.3.1 m\_aParam

```
const real KerrMetric::m_aParam [private]
```

### 6.31.3.2 m\_mParam

```
const real KerrMetric::m_mParam [private]
```

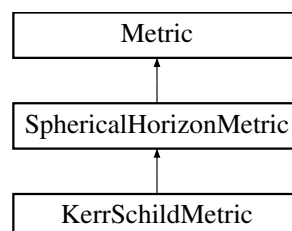
The documentation for this class was generated from the following files:

- FOORT/src/[Metric.h](#)
- FOORT/src/[Metric.cpp](#)

## 6.32 KerrSchildMetric Class Reference

```
#include <Metric.h>
```

Inheritance diagram for KerrSchildMetric:



### Public Member Functions

- [KerrSchildMetric](#) ()=delete
- [KerrSchildMetric](#) ([real](#) aParam, bool rLogScale=false)  
*KerrSchildMetric functions.*
- [TwoIndex getMetric\\_dd](#) (const [Point](#) &p) const final
- [TwoIndex getMetric\\_uu](#) (const [Point](#) &p) const final
- std::string [getFullDescriptionStr](#) () const final

### Public Member Functions inherited from [SphericalHorizonMetric](#)

- [SphericalHorizonMetric](#) ()=delete
- [SphericalHorizonMetric](#) ([real](#) HorizonRadius, bool rLogScale)  
*SphericalHorizonMetric functions.*
- [real getHorizonRadius](#) () const

### Public Member Functions inherited from [Metric](#)

- virtual [~Metric](#) ()=default
- [Metric](#) (bool rlogscale=false)  
*Metric (abstract base class) functions.*
- virtual [ThreeIndex getChristoffel\\_udd](#) (const [Point](#) &p) const
- virtual [FourIndex getRiemann\\_uddd](#) (const [Point](#) &p) const
- virtual [real getKretschmann](#) (const [Point](#) &p) const
- bool [getrLogScale](#) () const

### Private Attributes

- const [real m\\_aParam](#)

### Additional Inherited Members

### Protected Attributes inherited from [SphericalHorizonMetric](#)

- const [real m\\_HorizonRadius](#)

### Protected Attributes inherited from [Metric](#)

- std::vector< int > [m\\_Symmetries](#) {}
- const bool [m\\_rLogScale](#)

## 6.32.1 Constructor & Destructor Documentation

### 6.32.1.1 [KerrSchildMetric](#)() [1/2]

```
KerrSchildMetric::KerrSchildMetric () [delete]
```

### 6.32.1.2 KerrSchildMetric() [2/2]

```
KerrSchildMetric::KerrSchildMetric (  
    real aParam,  
    bool rLogScale = false)
```

[KerrSchildMetric](#) functions.

## 6.32.2 Member Function Documentation

### 6.32.2.1 getFullDescriptionStr()

```
std::string KerrSchildMetric::getFullDescriptionStr () const [final], [virtual]
```

Reimplemented from [Metric](#).

### 6.32.2.2 getMetric\_dd()

```
TwoIndex KerrSchildMetric::getMetric_dd (  
    const Point & p) const [final], [virtual]
```

Implements [Metric](#).

### 6.32.2.3 getMetric\_uu()

```
TwoIndex KerrSchildMetric::getMetric_uu (  
    const Point & p) const [final], [virtual]
```

Implements [Metric](#).

## 6.32.3 Member Data Documentation

### 6.32.3.1 m\_aParam

```
const real KerrSchildMetric::m_aParam [private]
```

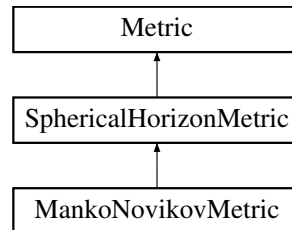
The documentation for this class was generated from the following files:

- FOORT/src/[Metric.h](#)
- FOORT/src/[Metric.cpp](#)

## 6.33 MankoNovikovMetric Class Reference

```
#include <Metric.h>
```

Inheritance diagram for MankoNovikovMetric:



### Public Member Functions

- [MankoNovikovMetric](#) ()=delete
- [MankoNovikovMetric](#) (real aParam, real alpha3Param, bool rLogScale=false)  
*MankoNovikovMetric functions (implementation by Seppe Staelens)*
- [TwoIndex getMetric\\_dd](#) (const [Point](#) &p) const final
- [TwoIndex getMetric\\_uu](#) (const [Point](#) &p) const final
- [std::string getFullDescriptionStr](#) () const final

### Public Member Functions inherited from [SphericalHorizonMetric](#)

- [SphericalHorizonMetric](#) ()=delete
- [SphericalHorizonMetric](#) (real HorizonRadius, bool rLogScale)  
*SphericalHorizonMetric functions.*
- [real getHorizonRadius](#) () const

### Public Member Functions inherited from [Metric](#)

- virtual [~Metric](#) ()=default
- [Metric](#) (bool rlogscale=false)  
*Metric (abstract base class) functions.*
- virtual [ThreeIndex getChristoffel\\_udd](#) (const [Point](#) &p) const
- virtual [FourIndex getRiemann\\_uddd](#) (const [Point](#) &p) const
- virtual [real getKretschmann](#) (const [Point](#) &p) const
- bool [getrLogScale](#) () const

### Private Attributes

- const [real m\\_aParam](#)
- const [real m\\_alpha3Param](#)
- const [real m\\_alphaParam](#)
- const [real m\\_kParam](#)

## Additional Inherited Members

### Protected Attributes inherited from [SphericalHorizonMetric](#)

- const [real](#) [m\\_HorizonRadius](#)

### Protected Attributes inherited from [Metric](#)

- `std::vector< int >` [m\\_Symmetries](#) {}
- const `bool` [m\\_rLogScale](#)

## 6.33.1 Constructor & Destructor Documentation

### 6.33.1.1 [MankoNovikovMetric\(\)](#) [1/2]

`MankoNovikovMetric::MankoNovikovMetric () [delete]`

### 6.33.1.2 [MankoNovikovMetric\(\)](#) [2/2]

```
MankoNovikovMetric::MankoNovikovMetric (
    real aParam,
    real alpha3Param,
    bool rLogScale = false)
```

[MankoNovikovMetric](#) functions (implementation by Seppe Staelens)

## 6.33.2 Member Function Documentation

### 6.33.2.1 [getFullDescriptionStr\(\)](#)

`std::string MankoNovikovMetric::getFullDescriptionStr () const [final], [virtual]`

Reimplemented from [Metric](#).

### 6.33.2.2 [getMetric\\_dd\(\)](#)

```
TwoIndex MankoNovikovMetric::getMetric_dd (
    const Point & p) const [final], [virtual]
```

Implements [Metric](#).

### 6.33.2.3 [getMetric\\_uu\(\)](#)

```
TwoIndex MankoNovikovMetric::getMetric_uu (
    const Point & p) const [final], [virtual]
```

Implements [Metric](#).



### 6.33.3 Member Data Documentation

#### 6.33.3.1 m\_alpha3Param

```
const real MankoNovikovMetric::m_alpha3Param [private]
```

#### 6.33.3.2 m\_alphaParam

```
const real MankoNovikovMetric::m_alphaParam [private]
```

#### 6.33.3.3 m\_aParam

```
const real MankoNovikovMetric::m_aParam [private]
```

#### 6.33.3.4 m\_kParam

```
const real MankoNovikovMetric::m_kParam [private]
```

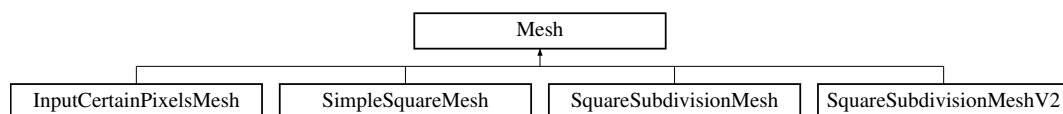
The documentation for this class was generated from the following files:

- FOORT/src/[Metric.h](#)
- FOORT/src/[Metric.cpp](#)

## 6.34 Mesh Class Reference

```
#include <Mesh.h>
```

Inheritance diagram for Mesh:



### Public Member Functions

- [Mesh](#) ([DiagBitflag](#) valdiag)
  - virtual [~Mesh](#) ()=default
  - virtual [largecounter](#) [getCurNrGeodesics](#) () const =0
  - virtual void [getNewInitConds](#) ([largecounter](#) index, [ScreenPoint](#) &newunitpoint, [ScreenIndex](#) &newscreenindex) const =0
  - virtual void [GeodesicFinished](#) ([largecounter](#) index, std::vector< [real](#) > finalValues)=0
  - virtual void [EndCurrentLoop](#) ()=0
  - virtual bool [IsFinished](#) () const =0
  - virtual std::string [getFullDescriptionStr](#) () const
- [Mesh](#) (abstract base class) functions.*

## Protected Attributes

- `const std::unique_ptr< const Diagnostic > m_DistanceDiagnostic`

## 6.34.1 Constructor & Destructor Documentation

### 6.34.1.1 Mesh()

```
Mesh::Mesh (
    DiagBitflag valdiag) [inline]
```

### 6.34.1.2 ~Mesh()

```
virtual Mesh::~Mesh () [virtual], [default]
```

## 6.34.2 Member Function Documentation

### 6.34.2.1 EndCurrentLoop()

```
virtual void Mesh::EndCurrentLoop () [pure virtual]
```

Implemented in [InputCertainPixelsMesh](#), [SimpleSquareMesh](#), [SquareSubdivisionMesh](#), and [SquareSubdivisionMeshV2](#).

### 6.34.2.2 GeodesicFinished()

```
virtual void Mesh::GeodesicFinished (
    largecounter index,
    std::vector< real > finalValues) [pure virtual]
```

Implemented in [InputCertainPixelsMesh](#), [SimpleSquareMesh](#), [SquareSubdivisionMesh](#), and [SquareSubdivisionMeshV2](#).

### 6.34.2.3 getCurNrGeodesics()

```
virtual largecounter Mesh::getCurNrGeodesics () const [pure virtual]
```

Implemented in [InputCertainPixelsMesh](#), [SimpleSquareMesh](#), [SquareSubdivisionMesh](#), and [SquareSubdivisionMeshV2](#).

### 6.34.2.4 getFullDescriptionStr()

```
std::string Mesh::getFullDescriptionStr () const [virtual]
```

[Mesh](#) (abstract base class) functions.

Reimplemented in [InputCertainPixelsMesh](#), [SimpleSquareMesh](#), [SquareSubdivisionMesh](#), and [SquareSubdivisionMeshV2](#).

### 6.34.2.5 getNewInitConds()

```
virtual void Mesh::getNewInitConds (
    largecounter index,
    ScreenPoint & newunitpoint,
    ScreenIndex & newscreenindex) const [pure virtual]
```

Implemented in [InputCertainPixelsMesh](#), [SimpleSquareMesh](#), [SquareSubdivisionMesh](#), and [SquareSubdivisionMeshV2](#).

### 6.34.2.6 IsFinished()

```
virtual bool Mesh::IsFinished () const [pure virtual]
```

Implemented in [InputCertainPixelsMesh](#), [SimpleSquareMesh](#), [SquareSubdivisionMesh](#), and [SquareSubdivisionMeshV2](#).

## 6.34.3 Member Data Documentation

### 6.34.3.1 m\_DistanceDiagnostic

```
const std::unique_ptr<const Diagnostic> Mesh::m_DistanceDiagnostic [protected]
```

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

- FOORT/src/[Mesh.h](#)
- FOORT/src/[Mesh.cpp](#)

## 6.35 Metric Class Reference

```
#include <Metric.h>
```

Inheritance diagram for Metric:



### Public Member Functions

- virtual [~Metric](#) ()=default
- [Metric](#) (bool rlogscale=false)
- [Metric](#) (abstract base class) functions.
- virtual [TwoIndex](#) [getMetric\\_dd](#) (const [Point](#) &p) const =0
- virtual [TwoIndex](#) [getMetric\\_uu](#) (const [Point](#) &p) const =0
- virtual [ThreeIndex](#) [getChristoffel\\_udd](#) (const [Point](#) &p) const
- virtual [FourIndex](#) [getRiemann\\_uddd](#) (const [Point](#) &p) const
- virtual [real](#) [getKretschmann](#) (const [Point](#) &p) const
- virtual std::string [getFullDescriptionStr](#) () const
- bool [getrLogScale](#) () const

## Protected Attributes

- `std::vector< int > m_Symmetries {}`
- `const bool m_rLogScale`

## 6.35.1 Constructor & Destructor Documentation

### 6.35.1.1 `~Metric()`

```
virtual Metric::~Metric () [virtual], [default]
```

### 6.35.1.2 `Metric()`

```
Metric::Metric (
    bool rlogscale = false)
```

[Metric](#) (abstract base class) functions.

## 6.35.2 Member Function Documentation

### 6.35.2.1 `getChristoffel_udd()`

```
ThreeIndex Metric::getChristoffel_udd (
    const Point & p) const [virtual]
```

### 6.35.2.2 `getFullDescriptionStr()`

```
std::string Metric::getFullDescriptionStr () const [virtual]
```

Reimplemented in [BosonStarMetric](#), [FlatSpaceMetric](#), [JohannsenMetric](#), [KerrMetric](#), [KerrSchildMetric](#), [MankoNovikovMetric](#), [RasheedLarsenMetric](#), and [ST3CrMetric](#).

### 6.35.2.3 `getKretschmann()`

```
real Metric::getKretschmann (
    const Point & p) const [virtual]
```

### 6.35.2.4 `getMetric_dd()`

```
virtual TwoIndex Metric::getMetric_dd (
    const Point & p) const [pure virtual]
```

Implemented in [BosonStarMetric](#), [FlatSpaceMetric](#), [JohannsenMetric](#), [KerrMetric](#), [KerrSchildMetric](#), [MankoNovikovMetric](#), [RasheedLarsenMetric](#), and [ST3CrMetric](#).

### 6.35.2.5 getMetric\_uu()

```
virtual TwoIndex Metric::getMetric_uu (
    const Point & p) const [pure virtual]
```

Implemented in [BosonStarMetric](#), [FlatSpaceMetric](#), [JohannsenMetric](#), [KerrMetric](#), [KerrSchildMetric](#), [MankoNovikovMetric](#), [RasheedLarsenMetric](#), and [ST3CrMetric](#).

### 6.35.2.6 getRiemann\_uddd()

```
FourIndex Metric::getRiemann_uddd (
    const Point & p) const [virtual]
```

### 6.35.2.7 getrLogScale()

```
bool Metric::getrLogScale () const
```

## 6.35.3 Member Data Documentation

### 6.35.3.1 m\_rLogScale

```
const bool Metric::m_rLogScale [protected]
```

### 6.35.3.2 m\_Symmetries

```
std::vector<int> Metric::m_Symmetries {} [protected]
```

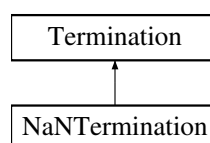
The documentation for this class was generated from the following files:

- [FOORT/src/Metric.h](#)
- [FOORT/src/Metric.cpp](#)

## 6.36 NaNTermination Class Reference

```
#include <Terminations.h>
```

Inheritance diagram for NaNTermination:



## Public Member Functions

- [NaNTermination](#) ([Geodesic](#) \*const theGeodesic)
- [Term CheckTermination](#) () final  
[NaNTermination](#) functions.
- std::string [getFullDescriptionStr](#) () const final

## Public Member Functions inherited from [Termination](#)

- [Termination](#) ()=delete
- [Termination](#) ([Geodesic](#) \*const theGeodesic)
- virtual void [Reset](#) ()  
[Termination](#) (abstract base class) functions.
- virtual [~Termination](#) ()=default

## Static Public Attributes

- static std::unique\_ptr< [NaNTermOptions](#) > [TermOptions](#)

## Additional Inherited Members

## Protected Member Functions inherited from [Termination](#)

- bool [DecideUpdate](#) ([largecounter](#) UpdateNSteps)

## Protected Attributes inherited from [Termination](#)

- [Geodesic](#) \*const [m\\_OwnerGeodesic](#)
- [largecounter](#) [m\\_StepsSinceUpdated](#) {}

## 6.36.1 Constructor & Destructor Documentation

### 6.36.1.1 NaNTermination()

```
NaNTermination::NaNTermination (
    Geodesic *const theGeodesic) [inline]
```

## 6.36.2 Member Function Documentation

### 6.36.2.1 CheckTermination()

```
Term NaNTermination::CheckTermination () [final], [virtual]
```

[NaNTermination](#) functions.

Implements [Termination](#).

### 6.36.2.2 getFullDescriptionStr()

```
std::string NaNTermination::getFullDescriptionStr () const [final], [virtual]
```

Implements [Termination](#).

## 6.36.3 Member Data Documentation

### 6.36.3.1 TermOptions

```
std::unique_ptr< NaNTermOptions > NaNTermination::TermOptions [static]
```

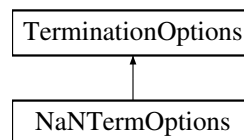
The documentation for this class was generated from the following files:

- FOORT/src/[Terminations.h](#)
- FOORT/src/[Config.cpp](#)
- FOORT/src/[Terminations.cpp](#)

## 6.37 NaNTermOptions Struct Reference

```
#include <Terminations.h>
```

Inheritance diagram for NaNTermOptions:



### Public Member Functions

- [NaNTermOptions](#) (bool consoleoutputon, [largecounter](#) Nsteps)

### Public Member Functions inherited from [TerminationOptions](#)

- [TerminationOptions](#) ([largecounter](#) Nsteps)
- virtual [~TerminationOptions](#) ()=default

### Public Attributes

- const bool [OutputToConsole](#)

### Public Attributes inherited from [TerminationOptions](#)

- const [largecounter](#) [UpdateEveryNSteps](#)

### 6.37.1 Constructor & Destructor Documentation

#### 6.37.1.1 NaNTermOptions()

```
NaNTermOptions::NaNTermOptions (
    bool consoleoutputon,
    largecounter Nsteps) [inline]
```

### 6.37.2 Member Data Documentation

#### 6.37.2.1 OutputToConsole

```
const bool NaNTermOptions::OutputToConsole
```

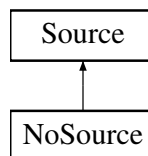
The documentation for this struct was generated from the following file:

- FOORT/src/[Terminations.h](#)

## 6.38 NoSource Class Reference

```
#include <Geodesic.h>
```

Inheritance diagram for NoSource:



#### Public Member Functions

- [NoSource](#) (const [Metric](#) \*const theMetric)
  - [OneIndex getSource](#) ([Point](#) pos, [OneIndex](#) vel) const final
  - std::string [getFullDescriptionStr](#) () const final
- [Source](#) (and descendant classes) functions.

#### Public Member Functions inherited from [Source](#)

- [Source](#) (const [Metric](#) \*const theMetric)
- virtual [~Source](#) ()=default

#### Additional Inherited Members

#### Protected Attributes inherited from [Source](#)

- const [Metric](#) \*const [m\\_theMetric](#)



## 6.38.1 Constructor & Destructor Documentation

### 6.38.1.1 NoSource()

```
NoSource::NoSource (
    const Metric *const theMetric) [inline]
```

## 6.38.2 Member Function Documentation

### 6.38.2.1 getFullDescriptionStr()

```
std::string NoSource::getFullDescriptionStr () const [final], [virtual]
```

[Source](#) (and descendant classes) functions.

Reimplemented from [Source](#).

### 6.38.2.2 getSource()

```
OneIndex NoSource::getSource (
    Point pos,
    OneIndex vel) const [final], [virtual]
```

Implements [Source](#).

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

- FOORT/src/[Geodesic.h](#)
- FOORT/src/[Geodesic.cpp](#)

## 6.39 SquareSubdivisionMesh::PixelInfo Struct Reference

### Public Member Functions

- [PixelInfo](#) ([ScreenIndex](#) ind, int subdiv)

### Public Attributes

- [ScreenIndex](#) [Index](#) {}
- int [SubdivideLevel](#) {}
- [real](#) [Weight](#) {-1}
- std::vector< [real](#) > [DiagValue](#) {}
- [largecounter](#) [LowerNbrIndex](#) {0}
- [largecounter](#) [RightNbrIndex](#) {0}

### 6.39.1 Constructor & Destructor Documentation

#### 6.39.1.1 PixelInfo()

```
SquareSubdivisionMesh::PixelInfo::PixelInfo (
    ScreenIndex ind,
    int subdiv) [inline]
```

### 6.39.2 Member Data Documentation

#### 6.39.2.1 DiagValue

```
std::vector<real> SquareSubdivisionMesh::PixelInfo::DiagValue {}
```

#### 6.39.2.2 Index

```
ScreenIndex SquareSubdivisionMesh::PixelInfo::Index {}
```

#### 6.39.2.3 LowerNbrIndex

```
largecounter SquareSubdivisionMesh::PixelInfo::LowerNbrIndex {0}
```

#### 6.39.2.4 RightNbrIndex

```
largecounter SquareSubdivisionMesh::PixelInfo::RightNbrIndex {0}
```

#### 6.39.2.5 SubdivideLevel

```
int SquareSubdivisionMesh::PixelInfo::SubdivideLevel {}
```

#### 6.39.2.6 Weight

```
real SquareSubdivisionMesh::PixelInfo::Weight {-1}
```

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

- FOORT/src/[Mesh.h](#)

## 6.40 SquareSubdivisionMeshV2::PixelInfo Struct Reference

### Public Member Functions

- [PixelInfo](#) ([ScreenIndex](#) ind, int subdiv)

## Public Attributes

- const [ScreenIndex](#) [Index](#) {}
- int [SubdivideLevel](#) {}
- [real](#) [Weight](#) {-1}
- std::vector< [real](#) > [DiagValue](#) {}
- [PixelInfo](#) \* [LeftNbr](#) {nullptr}
- [PixelInfo](#) \* [RightNbr](#) {nullptr}
- [PixelInfo](#) \* [UpNbr](#) {nullptr}
- [PixelInfo](#) \* [DownNbr](#) {nullptr}
- [PixelInfo](#) \* [SEdiagNbr](#) {nullptr}

## 6.40.1 Constructor & Destructor Documentation

### 6.40.1.1 PixelInfo()

```
SquareSubdivisionMeshV2::PixelInfo::PixelInfo (
    ScreenIndex ind,
    int subdiv) [inline]
```

## 6.40.2 Member Data Documentation

### 6.40.2.1 DiagValue

```
std::vector<real> SquareSubdivisionMeshV2::PixelInfo::DiagValue {}
```

### 6.40.2.2 DownNbr

```
PixelInfo* SquareSubdivisionMeshV2::PixelInfo::DownNbr {nullptr}
```

### 6.40.2.3 Index

```
const ScreenIndex SquareSubdivisionMeshV2::PixelInfo::Index {}
```

### 6.40.2.4 LeftNbr

```
PixelInfo* SquareSubdivisionMeshV2::PixelInfo::LeftNbr {nullptr}
```

### 6.40.2.5 RightNbr

```
PixelInfo* SquareSubdivisionMeshV2::PixelInfo::RightNbr {nullptr}
```

### 6.40.2.6 SEdiagNbr

```
PixelInfo* SquareSubdivisionMeshV2::PixelInfo::SEdiagNbr {nullptr}
```

### 6.40.2.7 SubdivideLevel

```
int SquareSubdivisionMeshV2::PixelInfo::SubdivideLevel {}
```

### 6.40.2.8 UpNbr

```
PixelInfo* SquareSubdivisionMeshV2::PixelInfo::UpNbr {nullptr}
```

### 6.40.2.9 Weight

```
real SquareSubdivisionMeshV2::PixelInfo::Weight {-1}
```

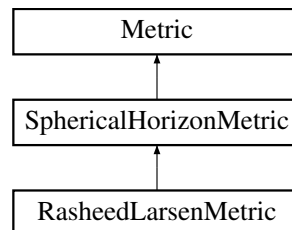
The documentation for this struct was generated from the following file:

- FOORT/src/[Mesh.h](#)

## 6.41 RasheedLarsenMetric Class Reference

```
#include <Metric.h>
```

Inheritance diagram for RasheedLarsenMetric:



### Public Member Functions

- [RasheedLarsenMetric](#) ()=delete
- [RasheedLarsenMetric](#) ([real](#) mParam, [real](#) aParam, [real](#) pParam, [real](#) qParam, bool rLogScale=false)
- *[RasheedLarsenMetric](#) functions.*
- [TwoIndex](#) [getMetric\\_dd](#) (const [Point](#) &p) const final
- [TwoIndex](#) [getMetric\\_uu](#) (const [Point](#) &p) const final
- std::string [getFullDescriptionStr](#) () const final

### Public Member Functions inherited from [SphericalHorizonMetric](#)

- [SphericalHorizonMetric](#) ()=delete
- [SphericalHorizonMetric](#) ([real](#) HorizonRadius, bool rLogScale)
- *[SphericalHorizonMetric](#) functions.*
- [real](#) [getHorizonRadius](#) () const

## Public Member Functions inherited from [Metric](#)

- virtual [~Metric](#) ()=default
- [Metric](#) (bool rlogscale=false)
- *[Metric](#) (abstract base class) functions.*
- virtual [ThreeIndex](#) [getChristoffel\\_udd](#) (const [Point](#) &p) const
- virtual [FourIndex](#) [getRiemann\\_uddd](#) (const [Point](#) &p) const
- virtual [real](#) [getKretschmann](#) (const [Point](#) &p) const
- bool [getrLogScale](#) () const

## Private Attributes

- const [real](#) [m\\_aParam](#)
- const [real](#) [m\\_mParam](#)
- const [real](#) [m\\_pParam](#)
- const [real](#) [m\\_qParam](#)

## Additional Inherited Members

## Protected Attributes inherited from [SphericalHorizonMetric](#)

- const [real](#) [m\\_HorizonRadius](#)

## Protected Attributes inherited from [Metric](#)

- std::vector< int > [m\\_Symmetries](#) {}
- const bool [m\\_rLogScale](#)

## 6.41.1 Constructor & Destructor Documentation

### 6.41.1.1 [RasheedLarsenMetric](#)() [1/2]

```
RasheedLarsenMetric::RasheedLarsenMetric () [delete]
```

### 6.41.1.2 [RasheedLarsenMetric](#)() [2/2]

```
RasheedLarsenMetric::RasheedLarsenMetric (
    real mParam,
    real aParam,
    real pParam,
    real qParam,
    bool rLogScale = false)
```

[RasheedLarsenMetric](#) functions.

## 6.41.2 Member Function Documentation

### 6.41.2.1 getFullDescriptionStr()

```
std::string RasheedLarsenMetric::getFullDescriptionStr () const [final], [virtual]
```

Reimplemented from [Metric](#).

### 6.41.2.2 getMetric\_dd()

```
TwoIndex RasheedLarsenMetric::getMetric_dd (  
    const Point & p) const [final], [virtual]
```

Implements [Metric](#).

### 6.41.2.3 getMetric\_uu()

```
TwoIndex RasheedLarsenMetric::getMetric_uu (  
    const Point & p) const [final], [virtual]
```

Implements [Metric](#).

## 6.41.3 Member Data Documentation

### 6.41.3.1 m\_aParam

```
const real RasheedLarsenMetric::m_aParam [private]
```

### 6.41.3.2 m\_mParam

```
const real RasheedLarsenMetric::m_mParam [private]
```

### 6.41.3.3 m\_pParam

```
const real RasheedLarsenMetric::m_pParam [private]
```

### 6.41.3.4 m\_qParam

```
const real RasheedLarsenMetric::m_qParam [private]
```

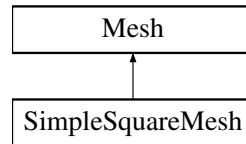
The documentation for this class was generated from the following files:

- FOORT/src/[Metric.h](#)
- FOORT/src/[Metric.cpp](#)

## 6.42 SimpleSquareMesh Class Reference

```
#include <Mesh.h>
```

Inheritance diagram for SimpleSquareMesh:



### Public Member Functions

- [SimpleSquareMesh](#) ()=delete
- [SimpleSquareMesh](#) ([largecounter](#) totalPixels, [DiagBitflag](#) valdiag)
- [largecounter](#) [getCurNrGeodesics](#) () const final
- void [getNewInitConds](#) ([largecounter](#) index, [ScreenPoint](#) &newunitpoint, [ScreenIndex](#) &newscreenindex) const final
- void [GeodesicFinished](#) ([largecounter](#) index, std::vector< [real](#) > finalValues) final
- void [EndCurrentLoop](#) () final
- bool [IsFinished](#) () const final
- *[SimpleSquareMesh](#) functions.*
- std::string [getFullDescriptionStr](#) () const final
- *[Mesh](#) (abstract base class) functions.*

### Public Member Functions inherited from [Mesh](#)

- [Mesh](#) ([DiagBitflag](#) valdiag)
- virtual [~Mesh](#) ()=default

### Private Attributes

- const [largecounter](#) [m\\_TotalPixels](#)
- const [pixelcoord](#) [m\\_RowColumnSize](#)
- bool [m\\_Finished](#) {false}

### Additional Inherited Members

### Protected Attributes inherited from [Mesh](#)

- const std::unique\_ptr< const [Diagnostic](#) > [m\\_DistanceDiagnostic](#)

## 6.42.1 Constructor & Destructor Documentation

### 6.42.1.1 SimpleSquareMesh() [1/2]

```
SimpleSquareMesh::SimpleSquareMesh () [delete]
```

### 6.42.1.2 SimpleSquareMesh() [2/2]

```
SimpleSquareMesh::SimpleSquareMesh (
    largecounter totalPixels,
    DiagBitflag valdiag) [inline]
```

## 6.42.2 Member Function Documentation

### 6.42.2.1 EndCurrentLoop()

```
void SimpleSquareMesh::EndCurrentLoop () [final], [virtual]
```

Implements [Mesh](#).

### 6.42.2.2 GeodesicFinished()

```
void SimpleSquareMesh::GeodesicFinished (
    largecounter index,
    std::vector< real > finalValues) [final], [virtual]
```

Implements [Mesh](#).

### 6.42.2.3 getCurNrGeodesics()

```
largecounter SimpleSquareMesh::getCurNrGeodesics () const [final], [virtual]
```

Implements [Mesh](#).

### 6.42.2.4 getFullDescriptionStr()

```
std::string SimpleSquareMesh::getFullDescriptionStr () const [final], [virtual]
```

[Mesh](#) (abstract base class) functions.

Reimplemented from [Mesh](#).

### 6.42.2.5 getNewInitConds()

```
void SimpleSquareMesh::getNewInitConds (
    largecounter index,
    ScreenPoint & newunitpoint,
    ScreenIndex & newscreenindex) const [final], [virtual]
```

Implements [Mesh](#).



### 6.42.2.6 IsFinished()

```
bool SimpleSquareMesh::IsFinished () const [final], [virtual]
```

[SimpleSquareMesh](#) functions.

Implements [Mesh](#).

## 6.42.3 Member Data Documentation

### 6.42.3.1 m\_Finished

```
bool SimpleSquareMesh::m_Finished {false} [private]
```

### 6.42.3.2 m\_RowColumnSize

```
const pixelcoord SimpleSquareMesh::m_RowColumnSize [private]
```

### 6.42.3.3 m\_TotalPixels

```
const largecounter SimpleSquareMesh::m_TotalPixels [private]
```

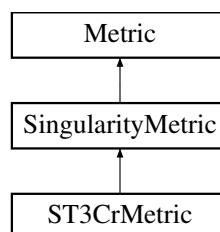
The documentation for this class was generated from the following files:

- FOORT/src/[Mesh.h](#)
- FOORT/src/[Mesh.cpp](#)

## 6.43 SingularityMetric Class Reference

```
#include <Metric.h>
```

Inheritance diagram for SingularityMetric:



### Public Member Functions

- [SingularityMetric](#) (std::vector< [Singularity](#) > thesings, bool rLogScale)  
[SingularityMetric](#) functions.
- std::vector< [Singularity](#) > [getSingularities](#) () const

## Public Member Functions inherited from [Metric](#)

- virtual [~Metric](#) ()=default
- [Metric](#) (bool rlogscale=false)  
[Metric](#) (*abstract base class*) functions.
- virtual [TwoIndex](#) [getMetric\\_dd](#) (const [Point](#) &p) const =0
- virtual [TwoIndex](#) [getMetric\\_uu](#) (const [Point](#) &p) const =0
- virtual [ThreeIndex](#) [getChristoffel\\_udd](#) (const [Point](#) &p) const
- virtual [FourIndex](#) [getRiemann\\_uddd](#) (const [Point](#) &p) const
- virtual [real](#) [getKretschmann](#) (const [Point](#) &p) const
- virtual std::string [getFullDescriptionStr](#) () const
- bool [getrLogScale](#) () const

## Protected Attributes

- const std::vector< [Singularity](#) > [m\\_AllSingularities](#)

## Protected Attributes inherited from [Metric](#)

- std::vector< int > [m\\_Symmetries](#) {}
- const bool [m\\_rLogScale](#)

## 6.43.1 Constructor & Destructor Documentation

### 6.43.1.1 SingularityMetric()

```
SingularityMetric::SingularityMetric (
    std::vector< Singularity > thesings,
    bool rLogScale)
```

[SingularityMetric](#) functions.

## 6.43.2 Member Function Documentation

### 6.43.2.1 getSingularities()

```
std::vector< Singularity > SingularityMetric::getSingularities () const
```

## 6.43.3 Member Data Documentation

### 6.43.3.1 m\_AllSingularities

```
const std::vector<Singularity> SingularityMetric::m_AllSingularities [protected]
```

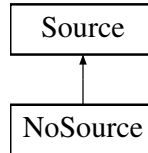
The documentation for this class was generated from the following files:

- FOORT/src/[Metric.h](#)
- FOORT/src/[Metric.cpp](#)

## 6.44 Source Class Reference

```
#include <Geodesic.h>
```

Inheritance diagram for Source:



### Public Member Functions

- [Source](#) (const [Metric](#) \*const theMetric)
  - virtual [~Source](#) ()=default
  - virtual [OneIndex](#) [getSource](#) ([Point](#) pos, [OneIndex](#) vel) const =0
  - virtual std::string [getFullDescriptionStr](#) () const
- [Source](#) (and descendant classes) functions.

### Protected Attributes

- const [Metric](#) \*const [m\\_theMetric](#)

## 6.44.1 Constructor & Destructor Documentation

### 6.44.1.1 Source()

```
Source::Source (
    const Metric *const theMetric) [inline]
```

### 6.44.1.2 ~Source()

```
virtual Source::~Source () [virtual], [default]
```

## 6.44.2 Member Function Documentation

### 6.44.2.1 getFullDescriptionStr()

```
std::string Source::getFullDescriptionStr () const [virtual]
```

[Source](#) (and descendant classes) functions.

Reimplemented in [NoSource](#).

### 6.44.2.2 getSource()

```
virtual OneIndex Source::getSource (
    Point pos,
    OneIndex vel) const [pure virtual]
```

Implemented in [NoSource](#).

## 6.44.3 Member Data Documentation

### 6.44.3.1 m\_theMetric

```
const Metric* const Source::m_theMetric [protected]
```

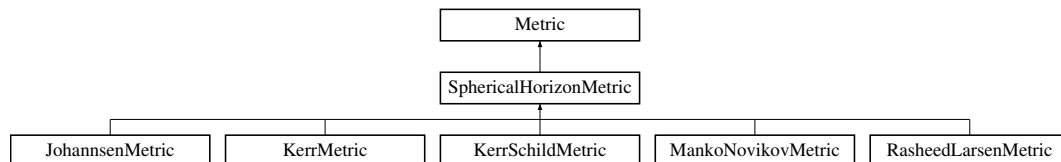
The documentation for this class was generated from the following files:

- [FOORT/src/Geodesic.h](#)
- [FOORT/src/Geodesic.cpp](#)

## 6.45 SphericalHorizonMetric Class Reference

```
#include <Metric.h>
```

Inheritance diagram for SphericalHorizonMetric:



### Public Member Functions

- [SphericalHorizonMetric](#) ()=delete
- [SphericalHorizonMetric](#) (real HorizonRadius, bool rLogScale)  
*SphericalHorizonMetric functions.*
- [real getHorizonRadius](#) () const

### Public Member Functions inherited from [Metric](#)

- virtual [~Metric](#) ()=default
- [Metric](#) (bool rlogscale=false)  
*Metric (abstract base class) functions.*
- virtual [TwoIndex getMetric\\_dd](#) (const [Point](#) &p) const =0
- virtual [TwoIndex getMetric\\_uu](#) (const [Point](#) &p) const =0
- virtual [ThreeIndex getChristoffel\\_udd](#) (const [Point](#) &p) const
- virtual [FourIndex getRiemann\\_uddd](#) (const [Point](#) &p) const
- virtual [real getKretschmann](#) (const [Point](#) &p) const
- virtual std::string [getFullDescriptionStr](#) () const
- bool [getrLogScale](#) () const

## Protected Attributes

- const [real m\\_HorizonRadius](#)

## Protected Attributes inherited from [Metric](#)

- `std::vector< int > m\_Symmetries {}`
- const bool [m\\_rLogScale](#)

## 6.45.1 Constructor & Destructor Documentation

### 6.45.1.1 SphericalHorizonMetric() [1/2]

```
SphericalHorizonMetric::SphericalHorizonMetric () [delete]
```

### 6.45.1.2 SphericalHorizonMetric() [2/2]

```
SphericalHorizonMetric::SphericalHorizonMetric (  
    real HorizonRadius,  
    bool rLogScale)
```

[SphericalHorizonMetric](#) functions.

## 6.45.2 Member Function Documentation

### 6.45.2.1 getHorizonRadius()

```
real SphericalHorizonMetric::getHorizonRadius () const
```

## 6.45.3 Member Data Documentation

### 6.45.3.1 m\_HorizonRadius

```
const real SphericalHorizonMetric::m_HorizonRadius [protected]
```

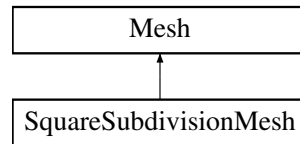
The documentation for this class was generated from the following files:

- FOORT/src/[Metric.h](#)
- FOORT/src/[Metric.cpp](#)

## 6.46 SquareSubdivisionMesh Class Reference

```
#include <Mesh.h>
```

Inheritance diagram for SquareSubdivisionMesh:



### Classes

- struct [PixelInfo](#)

### Public Member Functions

- [SquareSubdivisionMesh](#) ()=delete
- [SquareSubdivisionMesh](#) ([largecounter](#) maxPixels, [largecounter](#) initialPixels, int maxSubdivide, [largecounter](#) iterationPixels, bool initialSubToFinal, [DiagBitflag](#) valdiag)
- [largecounter](#) [getCurNrGeodesics](#) () const final  
*[SquareSubdivisionMesh](#) functions.*
- void [getNewInitConds](#) ([largecounter](#) index, [ScreenPoint](#) &newunitpoint, [ScreenIndex](#) &newscreenindex) const final
- void [GeodesicFinished](#) ([largecounter](#) index, std::vector< [real](#) > finalValues) final
- void [EndCurrentLoop](#) () final
- bool [IsFinished](#) () const final
- std::string [getFullDescriptionStr](#) () const final  
*[Mesh](#) (abstract base class) functions.*

### Public Member Functions inherited from [Mesh](#)

- [Mesh](#) ([DiagBitflag](#) valdiag)
- virtual [~Mesh](#) ()=default

### Private Member Functions

- void [InitializeFirstGrid](#) ()
- void [UpdateAllNeighbors](#) ()
- void [UpdateAllWeights](#) ()
- void [SubdivideAndQueue](#) ([largecounter](#) ind)
- [pixelcoord](#) [Explnt](#) (int base, int exp)

**Private Attributes**

- const [largecounter](#) `m_InitialPixels`
- const int `m_MaxSubdivide`
- const [pixelcoord](#) `m_RowColumnSize`
- const [largecounter](#) `m_IterationPixels`
- const [largecounter](#) `m_MaxPixels`
- const bool `m_InitialSubDivideToFinal`
- const bool `m_InfinitePixels`
- [largecounter](#) `m_PixelsLeft`
- `std::vector< PixelInfo > m_CurrentPixelQueue {}`
- `std::vector< bool > m_CurrentPixelQueueDone {}`
- `std::vector< PixelInfo > m_AllPixels {}`

**Additional Inherited Members****Protected Attributes inherited from [Mesh](#)**

- const `std::unique_ptr< const Diagnostic > m_DistanceDiagnostic`

**6.46.1 Constructor & Destructor Documentation****6.46.1.1 SquareSubdivisionMesh() [1/2]**

```
SquareSubdivisionMesh::SquareSubdivisionMesh () [delete]
```

**6.46.1.2 SquareSubdivisionMesh() [2/2]**

```
SquareSubdivisionMesh::SquareSubdivisionMesh (
    largecounter maxPixels,
    largecounter initialPixels,
    int maxSubdivide,
    largecounter iterationPixels,
    bool initialSubToFinal,
    DiagBitflag valdiag) [inline]
```

**6.46.2 Member Function Documentation****6.46.2.1 EndCurrentLoop()**

```
void SquareSubdivisionMesh::EndCurrentLoop () [final], [virtual]
```

Implements [Mesh](#).

**6.46.2.2 ExpInt()**

```
pixelcoord SquareSubdivisionMesh::ExpInt (
    int base,
    int exp) [private]
```

### 6.46.2.3 GeodesicFinished()

```
void SquareSubdivisionMesh::GeodesicFinished (
    largecounter index,
    std::vector< real > finalValues) [final], [virtual]
```

Implements [Mesh](#).

### 6.46.2.4 getCurNrGeodesics()

```
largecounter SquareSubdivisionMesh::getCurNrGeodesics () const [final], [virtual]
```

[SquareSubdivisionMesh](#) functions.

Implements [Mesh](#).

### 6.46.2.5 getFullDescriptionStr()

```
std::string SquareSubdivisionMesh::getFullDescriptionStr () const [final], [virtual]
```

[Mesh](#) (abstract base class) functions.

Reimplemented from [Mesh](#).

### 6.46.2.6 getNewInitConds()

```
void SquareSubdivisionMesh::getNewInitConds (
    largecounter index,
    ScreenPoint & newunitpoint,
    ScreenIndex & newscreenindex) const [final], [virtual]
```

Implements [Mesh](#).

### 6.46.2.7 InitializeFirstGrid()

```
void SquareSubdivisionMesh::InitializeFirstGrid () [private]
```

### 6.46.2.8 IsFinished()

```
bool SquareSubdivisionMesh::IsFinished () const [final], [virtual]
```

Implements [Mesh](#).

### 6.46.2.9 SubdivideAndQueue()

```
void SquareSubdivisionMesh::SubdivideAndQueue (
    largecounter ind) [private]
```



#### 6.46.2.10 UpdateAllNeighbors()

```
void SquareSubdivisionMesh::UpdateAllNeighbors () [private]
```

#### 6.46.2.11 UpdateAllWeights()

```
void SquareSubdivisionMesh::UpdateAllWeights () [private]
```

### 6.46.3 Member Data Documentation

#### 6.46.3.1 m\_AllPixels

```
std::vector<PixelInfo> SquareSubdivisionMesh::m_AllPixels {} [private]
```

#### 6.46.3.2 m\_CurrentPixelQueue

```
std::vector<PixelInfo> SquareSubdivisionMesh::m_CurrentPixelQueue {} [private]
```

#### 6.46.3.3 m\_CurrentPixelQueueDone

```
std::vector<bool> SquareSubdivisionMesh::m_CurrentPixelQueueDone {} [private]
```

#### 6.46.3.4 m\_InfinitePixels

```
const bool SquareSubdivisionMesh::m_InfinitePixels [private]
```

#### 6.46.3.5 m\_InitialPixels

```
const largecounter SquareSubdivisionMesh::m_InitialPixels [private]
```

#### 6.46.3.6 m\_InitialSubDividideToFinal

```
const bool SquareSubdivisionMesh::m_InitialSubDividideToFinal [private]
```

#### 6.46.3.7 m\_IterationPixels

```
const largecounter SquareSubdivisionMesh::m_IterationPixels [private]
```

#### 6.46.3.8 m\_MaxPixels

```
const largecounter SquareSubdivisionMesh::m_MaxPixels [private]
```

### 6.46.3.9 m\_MaxSubdivide

```
const int SquareSubdivisionMesh::m_MaxSubdivide [private]
```

### 6.46.3.10 m\_PixelsLeft

```
largecounter SquareSubdivisionMesh::m_PixelsLeft [private]
```

### 6.46.3.11 m\_RowColumnSize

```
const pixelcoord SquareSubdivisionMesh::m_RowColumnSize [private]
```

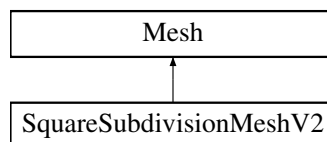
The documentation for this class was generated from the following files:

- FOORT/src/[Mesh.h](#)
- FOORT/src/[Mesh.cpp](#)

## 6.47 SquareSubdivisionMeshV2 Class Reference

```
#include <Mesh.h>
```

Inheritance diagram for SquareSubdivisionMeshV2:



### Classes

- struct [PixelInfo](#)

### Public Member Functions

- [SquareSubdivisionMeshV2](#) ()=delete
- [SquareSubdivisionMeshV2](#) ([largecounter](#) maxPixels, [largecounter](#) initialPixels, int maxSubdivide, [largecounter](#) iterationPixels, bool initialSubToFinal, [DiagBitflag](#) valdiag)
- [largecounter](#) getCurNrGeodesics () const final
- *SquareSubdivisionMeshV2 functions.*
- void [getNewInitConds](#) ([largecounter](#) index, [ScreenPoint](#) &newunitpoint, [ScreenIndex](#) &newscreenindex) const final
- void [GeodesicFinished](#) ([largecounter](#) index, std::vector< [real](#) > finalValues) final
- void [EndCurrentLoop](#) () final
- bool [IsFinished](#) () const final
- std::string [getFullDescriptionStr](#) () const final
- *Mesh (abstract base class) functions.*

## Public Member Functions inherited from [Mesh](#)

- [Mesh](#) ([DiagBitflag](#) valdiag)
- virtual [~Mesh](#) ()=default

## Private Member Functions

- void [InitializeFirstGrid](#) ()
- void [UpdateAllWeights](#) ()
- [PixelInfo](#) \* [GetUp](#) ([PixelInfo](#) \*p, int subdiv) const
- [PixelInfo](#) \* [GetDown](#) ([PixelInfo](#) \*p, int subdiv) const
- [PixelInfo](#) \* [GetRight](#) ([PixelInfo](#) \*p, int subdiv) const
- [PixelInfo](#) \* [GetLeft](#) ([PixelInfo](#) \*p, int subdiv) const
- void [SubdivideAndQueue](#) ([largecounter](#) ind)
- [pixelcoord](#) [Explnt](#) (int base, int exp) const

## Private Attributes

- const [largecounter](#) [m\\_InitialPixels](#)
- const int [m\\_MaxSubdivide](#)
- const [pixelcoord](#) [m\\_RowColumnSize](#)
- const [largecounter](#) [m\\_IterationPixels](#)
- const [largecounter](#) [m\\_MaxPixels](#)
- const bool [m\\_InitialSubDivideToFinal](#)
- const bool [m\\_InfinitePixels](#)
- [largecounter](#) [m\\_PixelsLeft](#)
- [largecounter](#) [m\\_PixelsIntegrated](#) {0}
- std::forward\_list< std::unique\_ptr< [PixelInfo](#) > > [m\\_AllPixels](#) {}
- std::vector< [PixelInfo](#) \* > [m\\_ActivePixels](#) {}
- std::vector< [PixelInfo](#) \* > [m\\_CurrentPixelQueue](#) {}
- std::vector< bool > [m\\_CurrentPixelQueueDone](#) {}
- std::vector< [PixelInfo](#) \* > [m\\_CurrentPixelUpdating](#) {}

## Additional Inherited Members

## Protected Attributes inherited from [Mesh](#)

- const std::unique\_ptr< const [Diagnostic](#) > [m\\_DistanceDiagnostic](#)

### 6.47.1 Constructor & Destructor Documentation

#### 6.47.1.1 SquareSubdivisionMeshV2() [1/2]

```
SquareSubdivisionMeshV2::SquareSubdivisionMeshV2 () [delete]
```

### 6.47.1.2 SquareSubdivisionMeshV2() [2/2]

```
SquareSubdivisionMeshV2::SquareSubdivisionMeshV2 (  
    largecounter maxPixels,  
    largecounter initialPixels,  
    int maxSubdivide,  
    largecounter iterationPixels,  
    bool initialSubToFinal,  
    DiagBitflag valdiag) [inline]
```

## 6.47.2 Member Function Documentation

### 6.47.2.1 EndCurrentLoop()

```
void SquareSubdivisionMeshV2::EndCurrentLoop () [final], [virtual]
```

Implements [Mesh](#).

### 6.47.2.2 ExpInt()

```
pixelcoord SquareSubdivisionMeshV2::ExpInt (  
    int base,  
    int exp) const [private]
```

### 6.47.2.3 GeodesicFinished()

```
void SquareSubdivisionMeshV2::GeodesicFinished (  
    largecounter index,  
    std::vector< real > finalValues) [final], [virtual]
```

Implements [Mesh](#).

### 6.47.2.4 getCurNrGeodesics()

```
largecounter SquareSubdivisionMeshV2::getCurNrGeodesics () const [final], [virtual]
```

[SquareSubdivisionMeshV2](#) functions.

Implements [Mesh](#).

### 6.47.2.5 GetDown()

```
SquareSubdivisionMeshV2::PixelInfo * SquareSubdivisionMeshV2::GetDown (  
    PixelInfo * p,  
    int subdiv) const [private]
```

### 6.47.2.6 getFullDescriptionStr()

```
std::string SquareSubdivisionMeshV2::getFullDescriptionStr () const [final], [virtual]
```

[Mesh](#) (abstract base class) functions.

Reimplemented from [Mesh](#).

### 6.47.2.7 GetLeft()

```
SquareSubdivisionMeshV2::PixelInfo * SquareSubdivisionMeshV2::GetLeft (
    PixelInfo * p,
    int subdiv) const [private]
```

### 6.47.2.8 getNewInitConds()

```
void SquareSubdivisionMeshV2::getNewInitConds (
    largecounter index,
    ScreenPoint & newunitpoint,
    ScreenIndex & newscreenindex) const [final], [virtual]
```

Implements [Mesh](#).

### 6.47.2.9 GetRight()

```
SquareSubdivisionMeshV2::PixelInfo * SquareSubdivisionMeshV2::GetRight (
    PixelInfo * p,
    int subdiv) const [private]
```

### 6.47.2.10 GetUp()

```
SquareSubdivisionMeshV2::PixelInfo * SquareSubdivisionMeshV2::GetUp (
    PixelInfo * p,
    int subdiv) const [private]
```

### 6.47.2.11 InitializeFirstGrid()

```
void SquareSubdivisionMeshV2::InitializeFirstGrid () [private]
```

### 6.47.2.12 IsFinished()

```
bool SquareSubdivisionMeshV2::IsFinished () const [final], [virtual]
```

Implements [Mesh](#).

#### 6.47.2.13 SubdivideAndQueue()

```
void SquareSubdivisionMeshV2::SubdivideAndQueue (  
    largecounter ind) [private]
```

#### 6.47.2.14 UpdateAllWeights()

```
void SquareSubdivisionMeshV2::UpdateAllWeights () [private]
```

### 6.47.3 Member Data Documentation

#### 6.47.3.1 m\_ActivePixels

```
std::vector<PixelInfo *> SquareSubdivisionMeshV2::m_ActivePixels {} [private]
```

#### 6.47.3.2 m\_AllPixels

```
std::forward_list<std::unique_ptr<PixelInfo> > SquareSubdivisionMeshV2::m_AllPixels {} [private]
```

#### 6.47.3.3 m\_CurrentPixelQueue

```
std::vector<PixelInfo *> SquareSubdivisionMeshV2::m_CurrentPixelQueue {} [private]
```

#### 6.47.3.4 m\_CurrentPixelQueueDone

```
std::vector<bool> SquareSubdivisionMeshV2::m_CurrentPixelQueueDone {} [private]
```

#### 6.47.3.5 m\_CurrentPixelUpdating

```
std::vector<PixelInfo *> SquareSubdivisionMeshV2::m_CurrentPixelUpdating {} [private]
```

#### 6.47.3.6 m\_InfinitePixels

```
const bool SquareSubdivisionMeshV2::m_InfinitePixels [private]
```

#### 6.47.3.7 m\_InitialPixels

```
const largecounter SquareSubdivisionMeshV2::m_InitialPixels [private]
```

#### 6.47.3.8 m\_InitialSubDividideToFinal

```
const bool SquareSubdivisionMeshV2::m_InitialSubDividideToFinal [private]
```

**6.47.3.9 m\_IterationPixels**

```
const largecounter SquareSubdivisionMeshV2::m_IterationPixels [private]
```

**6.47.3.10 m\_MaxPixels**

```
const largecounter SquareSubdivisionMeshV2::m_MaxPixels [private]
```

**6.47.3.11 m\_MaxSubdivide**

```
const int SquareSubdivisionMeshV2::m_MaxSubdivide [private]
```

**6.47.3.12 m\_PixelsIntegrated**

```
largecounter SquareSubdivisionMeshV2::m_PixelsIntegrated {0} [private]
```

**6.47.3.13 m\_PixelsLeft**

```
largecounter SquareSubdivisionMeshV2::m_PixelsLeft [private]
```

**6.47.3.14 m\_RowColumnSize**

```
const pixelcoord SquareSubdivisionMeshV2::m_RowColumnSize [private]
```

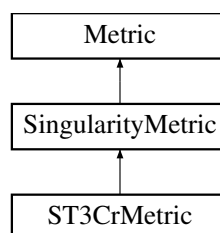
The documentation for this class was generated from the following files:

- [FOORT/src/Mesh.h](#)
- [FOORT/src/Mesh.cpp](#)

**6.48 ST3CrMetric Class Reference**

```
#include <Metric.h>
```

Inheritance diagram for ST3CrMetric:



### Public Member Functions

- [ST3CrMetric](#) ([real](#) P, [real](#) q0, [real](#) lambda, bool rlogscale=false)  
*ST3CrMetric functions.*
- [TwoIndex getMetric\\_dd](#) (const [Point](#) &p) const final
- [TwoIndex getMetric\\_uu](#) (const [Point](#) &p) const final
- std::string [getFullDescriptionStr](#) () const final

### Public Member Functions inherited from [SingularityMetric](#)

- [SingularityMetric](#) (std::vector< [Singularity](#) > thesings, bool rLogScale)  
*SingularityMetric functions.*
- std::vector< [Singularity](#) > [getSingularities](#) () const

### Public Member Functions inherited from [Metric](#)

- virtual [~Metric](#) ()=default
- [Metric](#) (bool rlogscale=false)  
*Metric (abstract base class) functions.*
- virtual [ThreeIndex getChristoffel\\_udd](#) (const [Point](#) &p) const
- virtual [FourIndex getRiemann\\_uddd](#) (const [Point](#) &p) const
- virtual [real getKretschmann](#) (const [Point](#) &p) const
- bool [getrLogScale](#) () const

### Private Member Functions

- [real get\\_omega](#) ([real](#) r, [real](#) theta, [real](#) l) const
- [real f\\_phi](#) ([real](#) phi, [real](#) r, [real](#) theta, [real](#) l, [real](#) R) const
- [real f\\_om\\_phi](#) ([real](#) phi, [real](#) r, [real](#) theta, [real](#) l, [real](#) R) const

### Private Attributes

- const [real](#) m\_P
- const [real](#) m\_q0
- const [real](#) m\_lambda

### Additional Inherited Members

### Protected Attributes inherited from [SingularityMetric](#)

- const std::vector< [Singularity](#) > m\_AllSingularities

### Protected Attributes inherited from [Metric](#)

- std::vector< int > m\_Symmetries {}
- const bool m\_rLogScale



## 6.48.1 Constructor & Destructor Documentation

### 6.48.1.1 ST3CrMetric()

```
ST3CrMetric::ST3CrMetric (
    real P,
    real q0,
    real lambda,
    bool rlogscale = false)
```

[ST3CrMetric](#) functions.

## 6.48.2 Member Function Documentation

### 6.48.2.1 f\_om\_phi()

```
real ST3CrMetric::f_om_phi (
    real phi,
    real r,
    real theta,
    real l,
    real R) const [private]
```

### 6.48.2.2 f\_phi()

```
real ST3CrMetric::f_phi (
    real phi,
    real r,
    real theta,
    real l,
    real R) const [private]
```

### 6.48.2.3 get\_omega()

```
real ST3CrMetric::get_omega (
    real r,
    real theta,
    real l) const [private]
```

### 6.48.2.4 getFullDescriptionStr()

```
std::string ST3CrMetric::getFullDescriptionStr () const [final], [virtual]
```

Reimplemented from [Metric](#).

### 6.48.2.5 getMetric\_dd()

```
TwoIndex ST3CrMetric::getMetric_dd (
    const Point & p) const [final], [virtual]
```

Implements [Metric](#).

### 6.48.2.6 getMetric\_uu()

```
TwoIndex ST3CrMetric::getMetric_uu (
    const Point & p) const [final], [virtual]
```

Implements [Metric](#).

## 6.48.3 Member Data Documentation

### 6.48.3.1 m\_lambda

```
const real ST3CrMetric::m_lambda [private]
```

### 6.48.3.2 m\_P

```
const real ST3CrMetric::m_P [private]
```

### 6.48.3.3 m\_q0

```
const real ST3CrMetric::m_q0 [private]
```

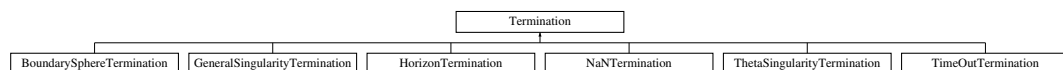
The documentation for this class was generated from the following files:

- FOORT/src/[Metric.h](#)
- FOORT/src/[Metric.cpp](#)

## 6.49 Termination Class Reference

```
#include <Terminations.h>
```

Inheritance diagram for Termination:



### Public Member Functions

- [Termination](#) ()=delete
- [Termination](#) ([Geodesic](#) \*const theGeodesic)
- virtual void [Reset](#) ()
- [Termination](#) (abstract base class) functions.
- virtual [~Termination](#) ()=default
- virtual [Term CheckTermination](#) ()=0
- virtual std::string [getFullDescriptionStr](#) () const =0

**Protected Member Functions**

- bool [DecideUpdate](#) ([largecounter](#) UpdateNSteps)

**Protected Attributes**

- [Geodesic](#) \*const [m\\_OwnerGeodesic](#)
- [largecounter](#) [m\\_StepsSinceUpdated](#) {}

**6.49.1 Constructor & Destructor Documentation****6.49.1.1 Termination() [1/2]**

```
Termination::Termination () [delete]
```

**6.49.1.2 Termination() [2/2]**

```
Termination::Termination (
    Geodesic *const theGeodesic) [inline]
```

**6.49.1.3 ~Termination()**

```
virtual Termination::~~Termination () [virtual], [default]
```

**6.49.2 Member Function Documentation****6.49.2.1 CheckTermination()**

```
virtual Term Termination::CheckTermination () [pure virtual]
```

Implemented in [BoundarySphereTermination](#), [GeneralSingularityTermination](#), [HorizonTermination](#), [NaNTermination](#), [ThetaSingularityTermination](#), and [TimeOutTermination](#).

**6.49.2.2 DecideUpdate()**

```
bool Termination::DecideUpdate (
    largecounter UpdateNSteps) [protected]
```

**6.49.2.3 getFullDescriptionStr()**

```
virtual std::string Termination::getFullDescriptionStr () const [pure virtual]
```

Implemented in [BoundarySphereTermination](#), [GeneralSingularityTermination](#), [HorizonTermination](#), [NaNTermination](#), [ThetaSingularityTermination](#), and [TimeOutTermination](#).

#### 6.49.2.4 Reset()

```
void Termination::Reset () [virtual]
```

[Termination](#) (abstract base class) functions.

Reimplemented in [TimeOutTermination](#).

### 6.49.3 Member Data Documentation

#### 6.49.3.1 m\_OwnerGeodesic

```
Geodesic* const Termination::m_OwnerGeodesic [protected]
```

#### 6.49.3.2 m\_StepsSinceUpdated

```
largecounter Termination::m_StepsSinceUpdated {} [protected]
```

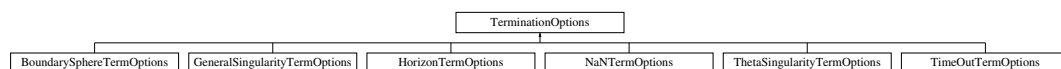
The documentation for this class was generated from the following files:

- FOORT/src/[Terminations.h](#)
- FOORT/src/[Terminations.cpp](#)

## 6.50 TerminationOptions Struct Reference

```
#include <Terminations.h>
```

Inheritance diagram for TerminationOptions:



### Public Member Functions

- [TerminationOptions](#) ([largecounter](#) Nsteps)
- virtual [~TerminationOptions](#) ()=default

### Public Attributes

- const [largecounter](#) UpdateEveryNSteps

### 6.50.1 Constructor & Destructor Documentation

#### 6.50.1.1 TerminationOptions()

```
TerminationOptions::TerminationOptions (
    largecounter Nsteps) [inline]
```

### 6.50.1.2 ~TerminationOptions()

```
virtual TerminationOptions::~~TerminationOptions () [virtual], [default]
```

## 6.50.2 Member Data Documentation

### 6.50.2.1 UpdateEveryNSteps

```
const largecounter TerminationOptions::UpdateEveryNSteps
```

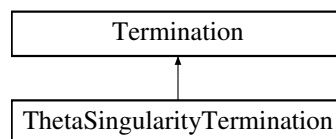
The documentation for this struct was generated from the following file:

- FOORT/src/[Terminations.h](#)

## 6.51 ThetaSingularityTermination Class Reference

```
#include <Terminations.h>
```

Inheritance diagram for ThetaSingularityTermination:



### Public Member Functions

- [ThetaSingularityTermination](#) ([Geodesic](#) \*const theGeodesic)
- [Term CheckTermination](#) () final  
*ThetaSingularityTermination functions.*
- std::string [getFullDescriptionStr](#) () const final

### Public Member Functions inherited from [Termination](#)

- [Termination](#) ()=delete
- [Termination](#) ([Geodesic](#) \*const theGeodesic)
- virtual void [Reset](#) ()  
*Termination (abstract base class) functions.*
- virtual [~Termination](#) ()=default

### Static Public Attributes

- static std::unique\_ptr< [ThetaSingularityTermOptions](#) > [TermOptions](#)

## Additional Inherited Members

## Protected Member Functions inherited from [Termination](#)

- bool [DecideUpdate](#) ([largecounter](#) UpdateNSteps)

## Protected Attributes inherited from [Termination](#)

- [Geodesic](#) \*const [m\\_OwnerGeodesic](#)
- [largecounter](#) [m\\_StepsSinceUpdated](#) {}

## 6.51.1 Constructor & Destructor Documentation

### 6.51.1.1 [ThetaSingularityTermination](#)()

```
ThetaSingularityTermination::ThetaSingularityTermination (
    Geodesic *const theGeodesic) [inline]
```

## 6.51.2 Member Function Documentation

### 6.51.2.1 [CheckTermination](#)()

```
Term ThetaSingularityTermination::CheckTermination () [final], [virtual]
```

[ThetaSingularityTermination](#) functions.

Implements [Termination](#).

### 6.51.2.2 [getFullDescriptionStr](#)()

```
std::string ThetaSingularityTermination::getFullDescriptionStr () const [final], [virtual]
```

Implements [Termination](#).

## 6.51.3 Member Data Documentation

### 6.51.3.1 [TermOptions](#)

```
std::unique_ptr< ThetaSingularityTermOptions > ThetaSingularityTermination::TermOptions [static]
```

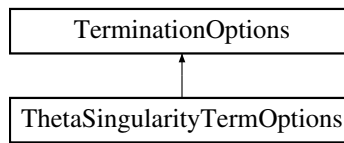
The documentation for this class was generated from the following files:

- FOORT/src/[Terminations.h](#)
- FOORT/src/[Config.cpp](#)
- FOORT/src/[Terminations.cpp](#)

## 6.52 ThetaSingularityTermOptions Struct Reference

```
#include <Terminations.h>
```

Inheritance diagram for ThetaSingularityTermOptions:



### Public Member Functions

- [ThetaSingularityTermOptions](#) ([real](#) epsilon, [largecounter](#) Nsteps)

### Public Member Functions inherited from [TerminationOptions](#)

- [TerminationOptions](#) ([largecounter](#) Nsteps)
- virtual [~TerminationOptions](#) ()=default

### Public Attributes

- const [real](#) [ThetaSingEpsilon](#)

### Public Attributes inherited from [TerminationOptions](#)

- const [largecounter](#) [UpdateEveryNSteps](#)

## 6.52.1 Constructor & Destructor Documentation

### 6.52.1.1 ThetaSingularityTermOptions()

```
ThetaSingularityTermOptions::ThetaSingularityTermOptions (
    real epsilon,
    largecounter Nsteps) [inline]
```

## 6.52.2 Member Data Documentation

### 6.52.2.1 ThetaSingEpsilon

```
const real ThetaSingularityTermOptions::ThetaSingEpsilon
```

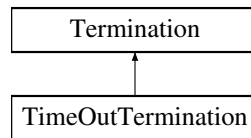
The documentation for this struct was generated from the following file:

- FOORT/src/[Terminations.h](#)

## 6.53 TimeOutTermination Class Reference

```
#include <Terminations.h>
```

Inheritance diagram for TimeOutTermination:



### Public Member Functions

- [TimeOutTermination](#) ([Geodesic](#) \*const theGeodesic)
- void [Reset](#) () final
- *[TimeOutTermination](#) functions.*
- [Term CheckTermination](#) () final
- std::string [getFullDescriptionStr](#) () const final

### Public Member Functions inherited from [Termination](#)

- [Termination](#) ()=delete
- [Termination](#) ([Geodesic](#) \*const theGeodesic)
- virtual [~Termination](#) ()=default

### Static Public Attributes

- static std::unique\_ptr< [TimeOutTermOptions](#) > [TermOptions](#)

### Private Attributes

- [largecounter](#) [m\\_CurNrSteps](#) {0}

### Additional Inherited Members

### Protected Member Functions inherited from [Termination](#)

- bool [DecideUpdate](#) ([largecounter](#) UpdateNSteps)

### Protected Attributes inherited from [Termination](#)

- [Geodesic](#) \*const [m\\_OwnerGeodesic](#)
- [largecounter](#) [m\\_StepsSinceUpdated](#) {}



## 6.53.1 Constructor & Destructor Documentation

### 6.53.1.1 TimeOutTermination()

```
TimeOutTermination::TimeOutTermination (  
    Geodesic *const theGeodesic) [inline]
```

## 6.53.2 Member Function Documentation

### 6.53.2.1 CheckTermination()

```
Term TimeOutTermination::CheckTermination () [final], [virtual]
```

Implements [Termination](#).

### 6.53.2.2 getFullDescriptionStr()

```
std::string TimeOutTermination::getFullDescriptionStr () const [final], [virtual]
```

Implements [Termination](#).

### 6.53.2.3 Reset()

```
void TimeOutTermination::Reset () [final], [virtual]
```

[TimeOutTermination](#) functions.

Reimplemented from [Termination](#).

## 6.53.3 Member Data Documentation

### 6.53.3.1 m\_CurNrSteps

```
largecounter TimeOutTermination::m_CurNrSteps {0} [private]
```

### 6.53.3.2 TermOptions

```
std::unique_ptr< TimeOutTermOptions > TimeOutTermination::TermOptions [static]
```

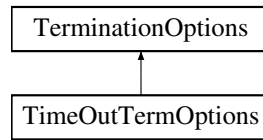
The documentation for this class was generated from the following files:

- FOORT/src/[Terminations.h](#)
- FOORT/src/[Config.cpp](#)
- FOORT/src/[Terminations.cpp](#)

## 6.54 TimeOutTermOptions Struct Reference

```
#include <Terminations.h>
```

Inheritance diagram for TimeOutTermOptions:



### Public Member Functions

- [TimeOutTermOptions](#) ([largecounter](#) MaxStepsAllowed, [largecounter](#) Nsteps)

### Public Member Functions inherited from [TerminationOptions](#)

- [TerminationOptions](#) ([largecounter](#) Nsteps)
- virtual [~TerminationOptions](#) ()=default

### Public Attributes

- const [largecounter](#) MaxSteps

### Public Attributes inherited from [TerminationOptions](#)

- const [largecounter](#) UpdateEveryNSteps

## 6.54.1 Constructor & Destructor Documentation

### 6.54.1.1 TimeOutTermOptions()

```
TimeOutTermOptions::TimeOutTermOptions (
    largecounter MaxStepsAllowed,
    largecounter Nsteps) [inline]
```

## 6.54.2 Member Data Documentation

### 6.54.2.1 MaxSteps

```
const largecounter TimeOutTermOptions::MaxSteps
```

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

- FOORT/src/[Terminations.h](#)

## 6.55 Utilities::Timer Class Reference

```
#include <Utilities.h>
```

### Public Member Functions

- void [reset](#) ()  
*Utilities::Timer functions.*
- double [elapsed](#) () const

### Private Types

- using [Clock](#) = std::chrono::steady\_clock
- using [Second](#) = std::chrono::duration<double, std::ratio<1>>

### Private Attributes

- std::chrono::time\_point< [Clock](#) > [m\\_beg](#) {Clock::now()}

### 6.55.1 Member Typedef Documentation

#### 6.55.1.1 Clock

```
using Utilities::Timer::Clock = std::chrono::steady_clock [private]
```

#### 6.55.1.2 Second

```
using Utilities::Timer::Second = std::chrono::duration<double, std::ratio<1>> [private]
```

### 6.55.2 Member Function Documentation

#### 6.55.2.1 elapsed()

```
double Utilities::Timer::elapsed () const
```

#### 6.55.2.2 reset()

```
void Utilities::Timer::reset ()
```

[Utilities::Timer](#) functions.

### 6.55.3 Member Data Documentation

#### 6.55.3.1 m\_beg

```
std::chrono::time_point<Clock> Utilities::Timer::m_beg {Clock::now()} [private]
```

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

- FOORT/src/[Utilities.h](#)
- FOORT/src/[Utilities.cpp](#)

## 6.56 UpdateFrequency Struct Reference

```
#include <Diagnostics.h>
```

### Public Attributes

- [largecounter](#) [UpdateNSteps](#) {0}
- bool [UpdateStart](#) {false}
- bool [UpdateFinish](#) {false}

### 6.56.1 Member Data Documentation

#### 6.56.1.1 UpdateFinish

```
bool UpdateFrequency::UpdateFinish {false}
```

#### 6.56.1.2 UpdateNSteps

```
largecounter UpdateFrequency::UpdateNSteps {0}
```

#### 6.56.1.3 UpdateStart

```
bool UpdateFrequency::UpdateStart {false}
```

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

- FOORT/src/[Diagnostics.h](#)

## 6.57 ViewScreen Class Reference

```
#include <ViewScreen.h>
```

**Public Member Functions**

- [ViewScreen](#) ()=delete
- [ViewScreen](#) ([Point](#) pos, [OneIndex](#) dir, [ScreenPoint](#) screensize, [ScreenPoint](#) screencenter, [std::unique\\_ptr](#)< [Mesh](#) > theMesh, const [Metric](#) \*const theMetric, [GeodesicType](#) thegeodtype=[GeodesicType::Null](#))
- void [SetNewInitialConditions](#) ([largecounter](#) index, [Point](#) &pos, [OneIndex](#) &vel, [ScreenIndex](#) &scrIndex) const
- bool [IsFinished](#) () const
- [largecounter](#) [getCurNrGeodesics](#) () const
- void [EndCurrentLoop](#) ()
- void [GeodesicFinished](#) ([largecounter](#) index, [std::vector](#)< [real](#) > finalValues)
- [std::string](#) [getFullDescriptionStr](#) () const

**Private Member Functions**

- void [ConstructVielbein](#) ()  
*[ViewScreen](#) functions.*

**Private Attributes**

- [TwoIndex](#) m\_Metric\_dd {}
- [TwoIndex](#) m\_Vielbein {}
- const [Point](#) m\_Pos
- const [OneIndex](#) m\_Direction
- const [ScreenPoint](#) m\_ScreenSize
- const [ScreenPoint](#) m\_ScreenCenter
- const bool m\_rLogScale
- const [Metric](#) \*const m\_theMetric
- const [GeodesicType](#) m\_GeodType {[GeodesicType::Null](#)}
- const [std::unique\\_ptr](#)< [Mesh](#) > m\_theMesh

**6.57.1 Constructor & Destructor Documentation****6.57.1.1 ViewScreen() [1/2]**

```
ViewScreen::ViewScreen () [delete]
```

**6.57.1.2 ViewScreen() [2/2]**

```
ViewScreen::ViewScreen (
    Point pos,
    OneIndex dir,
    ScreenPoint screensize,
    ScreenPoint screencenter,
    std::unique\_ptr< Mesh > theMesh,
    const Metric *const theMetric,
    GeodesicType thegeodtype = GeodesicType::Null) [inline]
```

## 6.57.2 Member Function Documentation

### 6.57.2.1 ConstructVielbein()

```
void ViewScreen::ConstructVielbein () [private]
```

[ViewScreen](#) functions.

### 6.57.2.2 EndCurrentLoop()

```
void ViewScreen::EndCurrentLoop ()
```

### 6.57.2.3 GeodesicFinished()

```
void ViewScreen::GeodesicFinished (  
    largecounter index,  
    std::vector< real > finalValues)
```

### 6.57.2.4 getCurNrGeodesics()

```
largecounter ViewScreen::getCurNrGeodesics () const
```

### 6.57.2.5 getFullDescriptionStr()

```
std::string ViewScreen::getFullDescriptionStr () const
```

### 6.57.2.6 IsFinished()

```
bool ViewScreen::IsFinished () const
```

### 6.57.2.7 SetNewInitialConditions()

```
void ViewScreen::SetNewInitialConditions (  
    largecounter index,  
    Point & pos,  
    OneIndex & vel,  
    ScreenIndex & scrIndex) const
```

## 6.57.3 Member Data Documentation

### 6.57.3.1 m\_Direction

```
const OneIndex ViewScreen::m_Direction [private]
```

### 6.57.3.2 m\_GeodType

```
const GeodesicType ViewScreen::m_GeodType {GeodesicType::Null} [private]
```

### 6.57.3.3 m\_Metric\_dd

```
TwoIndex ViewScreen::m_Metric_dd {} [private]
```

### 6.57.3.4 m\_Pos

```
const Point ViewScreen::m_Pos [private]
```

### 6.57.3.5 m\_rLogScale

```
const bool ViewScreen::m_rLogScale [private]
```

### 6.57.3.6 m\_ScreenCenter

```
const ScreenPoint ViewScreen::m_ScreenCenter [private]
```

### 6.57.3.7 m\_ScreenSize

```
const ScreenPoint ViewScreen::m_ScreenSize [private]
```

### 6.57.3.8 m\_theMesh

```
const std::unique_ptr<Mesh> ViewScreen::m_theMesh [private]
```

### 6.57.3.9 m\_theMetric

```
const Metric* const ViewScreen::m_theMetric [private]
```

### 6.57.3.10 m\_Vielbein

```
TwoIndex ViewScreen::m_Vielbein {} [private]
```

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

- FOORT/src/[ViewScreen.h](#)
- FOORT/src/[ViewScreen.cpp](#)





# Chapter 7

## File Documentation

### 7.1 FOORT/src/Config.cpp File Reference

```
#include "Config.h"  
#include "Utilities.h"  
#include <algorithm>  
#include <cctype>  
#include <utility>
```

### 7.2 FOORT/src/Config.h File Reference

```
#include "Geometry.h"  
#include "Metric.h"  
#include "Diagnostics.h"  
#include "Terminations.h"  
#include "ViewScreen.h"  
#include "Geodesic.h"  
#include "Integrators.h"  
#include <memory>  
#include <string>  
#include <exception>  
#include "ConfigReader.h"
```

#### Namespaces

- namespace [Config](#)

#### Macros

- #define [CONFIGURATION\\_MODE](#)

## Typedefs

- using `Config::ConfigCollection` = `ConfigReader::ConfigCollection`
- using `Config::SettingError` = `std::invalid_argument`

## Functions

- void `Config::InitializeScreenOutput` (const `ConfigCollection` &theCfg)
- std::unique\_ptr< `Metric` > `Config::GetMetric` (const `ConfigCollection` &theCfg)  
*`Config::GetMetric()`: Use configuration to create the correct `Metric` with specified parameters.*
- std::unique\_ptr< `Source` > `Config::GetSource` (const `ConfigCollection` &theCfg, const `Metric` \*const theMetric)  
*`Config::GetSource()`: Use configuration to create the correct `Source` with specified parameters.*
- void `Config::InitializeDiagnostics` (const `ConfigCollection` &theCfg, `DiagBitflag` &alldiags, `DiagBitflag` &valdiag, const `Metric` \*const theMetric)  
*`Config::InitializeDiagnostics()`: Use configuration to set the `Diagnostics` bitflag appropriately; initialize all `DiagnosticOptions` for all `Diagnostics` that are turned on; and set bitflag for diagnostic to be used for coarseness evaluating in `Mesh`.*
- void `Config::InitializeTerminations` (const `ConfigCollection` &theCfg, `TermBitflag` &allterms, const `Metric` \*const theMetric)  
*`Config::InitializeTerminations()`: Use configuration to set the `Termination` bitflag appropriately; and initialize all `TerminationOptions` for all `Terminations` that are turned on.*
- std::unique\_ptr< `ViewScreen` > `Config::GetViewScreen` (const `ConfigCollection` &theCfg, `DiagBitflag` valdiag, const `Metric` \*const theMetric)  
*`Config::GetViewScreen()`: Use configuration to create the `ViewScreen` object; with options set according to the configuration.*
- std::unique\_ptr< `Mesh` > `Config::GetMesh` (const `ConfigCollection` &theCfg, `DiagBitflag` valdiag)  
*`Config::GetMesh()`: Use configuration to create the `Mesh` object; with options set according to the configuration. `Config::GetViewScreen()` calls this when creating the `ViewScreen` object.*
- `GeodesicIntegratorFunc` `Config::GetGeodesicIntegrator` (const `ConfigCollection` &theCfg)  
*`Config::GetGeodesicIntegrator()`: Returns a pointer to the integrator function to be used as specified in the configuration file.*
- std::unique\_ptr< `GeodesicOutputHandler` > `Config::GetOutputHandler` (const `ConfigCollection` &theCfg, `DiagBitflag` alldiags, `DiagBitflag` valdiag, std::string FirstLineInfo)  
*`Config::GetOutputHandler()`: Creates the `GeodesicOutputHandler` object with options specified according to the configuration file, for handling of geodesic outputs.*

## Variables

- constexpr auto `Config::Output_Important_Default` = `OutputLevel::Level_0_WARNING`
- constexpr auto `Config::Output_Other_Default` = `OutputLevel::Level_1_PROC`

## 7.2.1 Macro Definition Documentation

### 7.2.1.1 CONFIGURATION\_MODE

```
#define CONFIGURATION_MODE
```

## 7.3 Config.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _FOORT_CONFIG_H
00002 #define _FOORT_CONFIG_H
00003
00010
00012 // COMMENT ONLY THIS LINE OUT TO BE IN PRECOMPILED OPTIONS MODE
00013 #define CONFIGURATION_MODE
00015
00016 // We need essentially all of the possible different objects as configuration functions initialize all
    of them
00017 #include "Geometry.h"
00018 #include "Metric.h"
00019 #include "Diagnostics.h"
00020 #include "Terminations.h"
00021 #include "ViewScreen.h"
00022 #include "Geodesic.h"
00023 #include "Integrators.h"
00024
00025 // The entire configuration namespace and its functions are only defined in CONFIGURATION MODE!
00026 #ifdef CONFIGURATION_MODE
00027
00028 #include <memory> // std::unique_ptr
00029 #include <string> // std::string
00030
00031 #include <exception> // needed to define our own configuration error
00032
00033 #include "ConfigReader.h"
00034
00035 // Namespace for all configuration functions that initialize objects based on configuration file
00036 namespace Config
00037 {
00038     // Output level for important missing information that will default
00039     // (e.g. no metric selected, no diagnostics selected, ...)
00040     constexpr auto Output_Important_Default = OutputLevel::Level_0_WARNING;
00041     // Output level for less important information that will default
00042     // (e.g. Kerr metric a parameter not specified)
00043     constexpr auto Output_Other_Default = OutputLevel::Level_1_PROC;
00044
00045     // Total configuration object, as loaded from configuration file
00046     using ConfigCollection = ConfigReader::ConfigCollection;
00047
00048     // An exception to throw whenever an important setting is not found
00049     // Always should be caught and then reverted to default settings
00050     using SettingError = std::invalid_argument;
00051
00052
00053     // Use configuration to initialize the screen output level
00054     void InitializeScreenOutput(const ConfigCollection &theCfg);
00055
00056     // Use configuration to create the correct Metric with specified parameters
00057     std::unique_ptr<Metric> GetMetric(const ConfigCollection &theCfg);
00058
00059     // Use configuration to create the correct Source with specified parameters
00060     std::unique_ptr<Source> GetSource(const ConfigCollection &theCfg, const Metric *const theMetric);
00061
00062     // Use configuration to set the Diagnostics bitflag appropriately;
00063     // initialize all DiagnosticOptions for all Diagnostics that are turned on;
00064     // and set bitflag for diagnostic to be used for coarseness evaluating in Mesh
00065     void InitializeDiagnostics(const ConfigCollection &theCfg, DiagBitflag &alldiags, DiagBitflag
&valdiag, const Metric *const theMetric);
00066
00067     // Use configuration to set the Terminations bitflag appropriately;
00068     // initialize all TerminationOptions for all Terminations that are turned on;
00069     void InitializeTerminations(const ConfigCollection &theCfg, TermBitflag &allterms, const Metric
*const theMetric);
00070
00071     // Use configuration to create ViewScreen appropriately
00072     std::unique_ptr<ViewScreen> GetViewScreen(const ConfigCollection &theCfg, DiagBitflag valdiag,
const Metric *const theMetric);
00073
00074     // GetViewScreen calls GetMesh to create the correct Mesh
00075     std::unique_ptr<Mesh> GetMesh(const ConfigCollection &theCfg, DiagBitflag valdiag);
00076
00077     // Use configuration to return a pointer to the correct integration function to use
00078     GeodesicIntegratorFunc GetGeodesicIntegrator(const ConfigCollection &theCfg);
00079
00080     // Use configuration to initialize the output handler
00081     std::unique_ptr<GeodesicOutputHandler> GetOutputHandler(const ConfigCollection &theCfg,
DiagBitflag alldiags, DiagBitflag valdiag,
std::string FirstLineInfo);
00082
00083
00084
00085 } // end namespace Config
00086
00087 #endif // CONFIGURATION_MODE

```

```
00088
00089 #endif
```

## 7.4 FOORT/src/ConfigReader.cpp File Reference

```
#include "ConfigReader.h"
#include <ios>
#include <type_traits>
```

## 7.5 FOORT/src/ConfigReader.h File Reference

```
#include <string_view>
#include <string>
#include <iostream>
#include <fstream>
#include <vector>
#include <variant>
#include <stdexcept>
#include <limits>
#include <memory>
```

### Classes

- class [ConfigReader::ConfigReaderException](#)
- class [ConfigReader::ConfigCollection](#)
- struct [ConfigReader::ConfigCollection::ConfigSetting](#)

### Namespaces

- namespace [ConfigReader](#)

### Variables

- const long long [ConfigReader::MaxStreamSize](#) {std::numeric\_limits<std::streamsize>::max() }

## 7.6 ConfigReader.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _CONFIGREADER_H
00002 #define _CONFIGREADER_H
00003
00004 #include <string_view> // std::string_view
00005 #include <string>       // std::string
00006 #include <iostream>     // needed for file and console output
00007 #include <fstream>      // needed for file input
00008 #include <vector>       // needed to create vectors of Settings
00009 #include <variant>      // needed for std::variant
00010 #include <stdexcept>    // needed for exceptions
00011 #include <limits>       // for std::numeric_limits
00012 #include <memory>       // for std::unique_ptr
00013
00014 // Namespace to put all ConfigReader objects
00015 namespace ConfigReader
00016 {
00017     // Shorthand for the maximum number of characters we can ignore in a stream
00018     const long long MaxStreamSize{std::numeric_limits<std::streamsize>::max()};
00019
00020     // Exception that is thrown if anything bad happens when reading in configuration file
00021     class ConfigReaderException : public std::runtime_error
00022     {
00023     public:
00024         ConfigReaderException(const std::string &error, std::vector<int> settingtrace = {})
00025             : std::runtime_error(error.c_str()), m_settingtrace{settingtrace}
00026         {
00027         }
00028         // no need to override what() since we can just use std::runtime_error::what()
00029
00030         std::vector<int> trace() const
00031         {
00032             return m_settingtrace;
00033         };
00034
00035     private:
00036         std::vector<int> m_settingtrace;
00037     };
00038
00039     // A collection of configuration settings
00040     class ConfigCollection
00041     {
00042     public:
00043         // Returns true if the collection contains a setting with the given name
00044         bool Exists(std::string_view SettingName) const;
00045         // Returns total number of settings in this collection
00046         int NrSettings() const;
00047
00048         // Returns true if the setting with the given name (resp. index) is a collection of
00049         // subsettings
00050         // (Returns false if this setting name does not exist, or index is out of bounds)
00051         bool IsCollection(std::string_view SettingName) const;
00052         bool IsCollection(int SettingIndex) const;
00053         // If CollectionName (resp. CollectionIndex) is the name (resp. index)
00054         // of a setting that is a subcollection, then this returns
00055         // a const reference to this subcollection; otherwise throws ConfigReaderException
00056         const ConfigCollection &operator[](std::string_view CollectionName) const;
00057         const ConfigCollection &operator[](int CollectionIndex) const;
00058
00059         // Generic functions that look up the value of a given setting
00060         // and put it in the output variable
00061         // Returns true if successful, false if unsuccessful
00062         // (either due to type mismatch of output and setting value, or if setting doesn't exist)
00063         // Templated functions implemented below in .h file!
00064         template <class OutputType>
00065         bool LookupValue(std::string_view SettingName, OutputType &theOutput) const;
00066         template <class OutputType>
00067         bool LookupValue(int SettingIndex, OutputType &theOutput) const;
00068
00069         // Specific functions for looking up signed integral types
00070         // These will first see if the setting value is of the given type, then
00071         // they will try smaller types and then recast to the larger type
00072         // (so e.g. for long long, first long long is tried, then long, then int)
00073         bool LookupValueInteger(std::string_view SettingName, int &theOutput) const;
00074         bool LookupValueInteger(std::string_view SettingName, long &theOutput) const;
00075         bool LookupValueInteger(std::string_view SettingName, long long &theOutput) const;
00076         // Specific functions for looking up unsigned integral types
00077         // These will first test if the setting value is of the signed case, if it is and it is >=0
00078         // then it always fits in
00079         // the unsigned version; then it will test larger (signed) types and see if the result still
00080         // fits in the unsigned version
00081         bool LookupValueInteger(std::string_view SettingName, unsigned int &theOutput) const;
00082         bool LookupValueInteger(std::string_view SettingName, unsigned long &theOutput) const;

```

```

00082     bool LookupValueInteger(std::string_view SettingName, unsigned long long &theOutput) const;
00083     // Templated functions implemented below in .h file!
00084     template <class OutputType>
00085     bool LookupValueInteger(int SettingIndex, OutputType &theOutput) const;
00086
00087     void DisplayCollection(std::ostream &OutputStream, int Indent = 0) const;
00088
00089     // this overwrites the current collection with the contents of the config file
00090     // Will throw ConfigReaderException if parse/syntax error in configuration file
00091     // Returns true if successful
00092     bool ReadFile(const std::string &FileName);
00093
00094 private:
00095     // A configuration setting value can be an integral type, bool, double, string, or a
00096     (sub)collection of settings
00097     // Note: when reading in a configuration file, an integral value will always be stored in the
00098     smallest possible type!
00099     using ConfigSettingValue = std::variant<bool,
00100                                         int, long, long long,
00101                                         double,
00102                                         std::string,
00103                                         std::unique_ptr<ConfigCollection>;
00104
00105     // A setting is a name and setting value
00106     struct ConfigSetting
00107     {
00108         std::string SettingName;
00109         ConfigSettingValue SettingValue;
00110     };
00111     // The vector of all settings in this collection
00112     std::vector<ConfigSetting> m_Settings{};
00113
00114     // Helper function that returns the index of the setting with the given name
00115     // returns -1 if setting not found
00116     int GetSettingIndex(std::string_view SettingName) const;
00117
00118     // Helper functions for DisplayCollection
00119     // DisplaySetting displays one setting; if it's a subcollection then it calls
00120     // DisplayCollection on the subcollection
00121     void DisplaySetting(std::ostream &OutputStream, int SettingIndex, int Indent) const;
00122     // Output given number of tabs
00123     void DisplayTabs(std::ostream &OutputStream, int NrTabs) const;
00124
00125     // Helper functions for ReadFile()
00126     // These all take an ifstream at a given location and will read in one specific object
00127     //
00128     // Read in entire collection (including subcollections)
00129     // Pre: stream is positioned right after { (or at beginning of file)
00130     // Post: stream is positioned right after } (or at EOF)
00131     void ReadCollection(std::ifstream &InputFile);
00132     // Read in a setting name
00133     // Pre: next non-whitespace character in stream is beginning of name
00134     // Post: stream has just read in all characters of name (but no more);
00135     // returns "" if there is no more setting to read in the current collection, in this case
00136     // '}' has been read in (for subcollections)
00137     void ReadSettingName(std::ifstream &InputFile, std::string &theName);
00138     // Read in one specific character only (not '!'!)
00139     // Pre: next non-whitespace character in stream is this character
00140     // Post: stream is just after character
00141     void ReadSettingSpecificChar(std::ifstream &InputFile, char theChar) const;
00142     // Read in setting value (could be subcollection)
00143     // Pre: next non-whitespace character in stream is beginning of value
00144     // (can be '{' for subcollection, digit for number, '"' for string, or 't'/'f' for boolean)
00145     // Post: Value has entirely been read in (i.e. next character should be ';')
00146     void ReadSettingValue(std::ifstream &InputFile, ConfigSettingValue &theValue);
00147
00148 };
00149 } // end namespace declarations
00150
00151 template <class OutputType>
00152 bool ConfigReader::ConfigCollection::LookupValue(int SettingIndex, OutputType &theOutput) const
00153 {
00154     // Get the setting value, if it is indeed of this type
00155     const OutputType *OutputPointer{std::get_if<OutputType>(&m_Settings[SettingIndex].SettingValue)};
00156     if (OutputPointer)
00157     {
00158         // Success, the setting is indeed of this type; set the output accordingly
00159         theOutput = *OutputPointer;
00160         return true;
00161     }
00162     return false;
00163 }
00164
00165 template <class OutputType>
00166 bool ConfigReader::ConfigCollection::LookupValue(std::string_view SettingName, OutputType &theOutput)
00167 const
00168 {
00169     {

```

```

00170 // Look up setting index and then defer to index-based implementation, if setting exists
00171 int SettingIndex{GetSettingIndex(SettingName)};
00172 if (SettingIndex >= 0)
00173 {
00174     return LookupValue(SettingIndex, theOutput);
00175 }
00176 return false;
00177 }
00178
00179 // This simply defers to the setting-name-based implementation of the function with given output type
00180 template <class OutputType>
00181 bool ConfigReader::ConfigCollection::LookupValueInteger(int SettingIndex, OutputType &theOutput) const
00182 {
00183     return LookupValueInteger(m_Settings[SettingIndex].SettingName, theOutput);
00184 }
00185
00186 #endif

```

## 7.7 FOORT/src/Diagnostics.cpp File Reference

```

#include "Diagnostics.h"
#include "DiagnosticsEmission.h"
#include "Geodesic.h"
#include "InputOutput.h"
#include "Integrators.h"
#include <algorithm>
#include <cmath>

```

### Functions

- [DiagnosticUniqueVector CreateDiagnosticVector](#) ([DiagBitflag](#) diagflags, [DiagBitflag](#) valdiag, [Geodesic](#) \*const theGeodesic)  
*Diagnostic helper function.*

### 7.7.1 Function Documentation

#### 7.7.1.1 CreateDiagnosticVector()

```

DiagnosticUniqueVector CreateDiagnosticVector (
    DiagBitflag diagflags,
    DiagBitflag valdiag,
    Geodesic *const theGeodesic)

```

[Diagnostic](#) helper function.

## 7.8 FOORT/src/Diagnostics.h File Reference

```

#include "Geometry.h"
#include "Metric.h"
#include "InputOutput.h"
#include "DiagnosticsEmission.h"
#include <cstdint>
#include <string>
#include <memory>
#include <vector>

```

## Classes

- struct [UpdateFrequency](#)
- class [Diagnostic](#)
- class [FourColorScreenDiagnostic](#)
- class [GeodesicPositionDiagnostic](#)
- class [EquatorialPassesDiagnostic](#)
- class [ClosestRadiusDiagnostic](#)
- class [EquatorialEmissionDiagnostic](#)
- struct [DiagnosticOptions](#)
- struct [GeodesicPositionOptions](#)
- struct [EquatorialPassesOptions](#)
- struct [ClosestRadiusOptions](#)
- struct [EquatorialEmissionOptions](#)

## Typedefs

- using [DiagBitflag](#) = std::uint16\_t
- using [DiagnosticUniqueVector](#) = std::vector<std::unique\_ptr<[Diagnostic](#)>>

## Functions

- [DiagnosticUniqueVector CreateDiagnosticVector](#) ([DiagBitflag](#) diagflags, [DiagBitflag](#) valdiag, [Geodesic](#) \*const theGeodesic)  
*[Diagnostic](#) helper function.*

## Variables

- constexpr [DiagBitflag](#) [Diag\\_None](#) {0b0000'0000'0000'0000}
- constexpr [DiagBitflag](#) [Diag\\_GeodesicPosition](#) {0b0000'0000'0000'0001}
- constexpr [DiagBitflag](#) [Diag\\_FourColorScreen](#) {0b0000'0000'0000'0010}
- constexpr [DiagBitflag](#) [Diag\\_EquatorialPasses](#) {0b0000'0000'0000'0100}
- constexpr [DiagBitflag](#) [Diag\\_ClosestRadius](#) {0b0000'0000'0000'1000}
- constexpr [DiagBitflag](#) [Diag\\_EquatorialEmission](#) {0b0000'0000'0001'0000}

## 7.8.1 Typedef Documentation

### 7.8.1.1 DiagBitflag

```
using DiagBitflag = std::uint16_t
```

### 7.8.1.2 DiagnosticUniqueVector

```
using DiagnosticUniqueVector = std::vector<std::unique_ptr<Diagnostic>>
```



## 7.8.2 Function Documentation

### 7.8.2.1 CreateDiagnosticVector()

```
DiagnosticUniqueVector CreateDiagnosticVector (  
    DiagBitflag diagflags,  
    DiagBitflag valdiag,  
    Geodesic *const theGeodesic)
```

`Diagnostic` helper function.

## 7.8.3 Variable Documentation

### 7.8.3.1 Diag\_ClosestRadius

```
DiagBitflag Diag_ClosestRadius {0b0000'0000'0000'1000} [constexpr]
```

### 7.8.3.2 Diag\_EquatorialEmission

```
DiagBitflag Diag_EquatorialEmission {0b0000'0000'0001'0000} [constexpr]
```

### 7.8.3.3 Diag\_EquatorialPasses

```
DiagBitflag Diag_EquatorialPasses {0b0000'0000'0000'0100} [constexpr]
```

### 7.8.3.4 Diag\_FourColorScreen

```
DiagBitflag Diag_FourColorScreen {0b0000'0000'0000'0010} [constexpr]
```

### 7.8.3.5 Diag\_GeodesicPosition

```
DiagBitflag Diag_GeodesicPosition {0b0000'0000'0000'0001} [constexpr]
```

### 7.8.3.6 Diag\_None

```
DiagBitflag Diag_None {0b0000'0000'0000'0000} [constexpr]
```

## 7.9 Diagnostics.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _FOORT_DIAGNOSTICS_H
00002 #define _FOORT_DIAGNOSTICS_H
00003
00009
00010 #include "Geometry.h" // for tensors
00011 #include "Metric.h"   // for metric
00012 #include "InputOutput.h" // for ScreenOutput
00013
00014 #include "DiagnosticsEmission.h" // Emission models
00015
00016 #include <stdint> // for std::uint16_t
00017 #include <string> // for strings
00018 #include <memory> // for std::unique_ptr
00019 #include <vector> // for std::vector
00020
00021 // Forward declaration of Geodesic class needed here, since Diagnostics are passed a pointer to their
00022 // owner Geodesic
00023 // (note "Geodesic.h" is NOT included to avoid header loop, and we do not need Geodesic member
00024 // functions here!)
00025 class Geodesic;
00026
00027
00028 // Diagnostic bitflags
00029 // Used for constructing vector of Diagnostics
00030 // Note that this means every diagnostic is either "on" or "off";
00031 // it is not possible to have a Diagnostic "on" more than once
00032 // Note: why not std::bitset<size>? Because in this way we can use expressions with DiagBitflag in
00033 // conditional expressions,
00034 // e.g. if ( mydiagbitflag & Diag_FourColorScreen)
00035 using DiagBitflag = std::uint16_t;
00036
00037 // Define a bitflag per existing diagnostic
00038 constexpr DiagBitflag Diag_None{0b0000'0000'0000'0000};
00039 constexpr DiagBitflag Diag_GeodesicPosition{0b0000'0000'0000'0001};
00040 constexpr DiagBitflag Diag_FourColorScreen{0b0000'0000'0000'0010};
00041 constexpr DiagBitflag Diag_EquatorialPasses{0b0000'0000'0000'0100};
00042 constexpr DiagBitflag Diag_ClosestRadius{0b0000'0000'0000'1000};
00043 constexpr DiagBitflag Diag_EquatorialEmission{0b0000'0000'0001'0000};
00044
00045 // Add a DiagBitflag for your new diagnostic. Make sure you use a bitflag that has not been used
00046 // before!
00047 // Sample code:
00048 /*
00049 constexpr DiagBitflag Diag_MyDiag { 0b0000'0000'0000'1000 };
00050 */
00051
00052 // This carries the information for a Diagnostic to update itself: if UpdateNsteps > 0, then every so
00053 // many steps.
00054 // If UpdateNSteps == 0, then it only updates at the start and/or finish of integration if the
00055 // appropriate bool is set.
00056 struct UpdateFrequency
00057 {
00058     largecounter UpdateNSteps{0};
00059     bool UpdateStart{false};
00060     bool UpdateFinish{false};
00061 };
00062
00063
00064 // Abstract base class for all diagnostics
00065 class Diagnostic
00066 {
00067 public:
00068     // Constructor must initialize the pointer to its owner Geodesic
00069     Diagnostic() = delete;
00070     Diagnostic(Geodesic *const theGeodesic) : m_OwnerGeodesic{theGeodesic}
00071     {
00072     }
00073
00074     // Resets Diagnostic object. This is called when the owner Geodesic is reset in order to start
00075     // integrating
00076     // a new geodesic.
00077     // The base class implementation only resets m_StepsSinceUpdated
00078     // Descendants can override this if they need to reset additional internal variables
00079     virtual void Reset();
00080
00081     // virtual destructor to ensure correct destruction of descendants
00082     virtual ~Diagnostic() = default;
00083
00084     // This is the heart of the Diagnostic. It calls the helper function DecideUpdate(), and if this
00085     // returns true, will update its internal status based on the current (new) state of its owner
00086     // geodesic.
00087     virtual void UpdateData() = 0;

```

```

00086
00087 // These functions are for use at the end of integration of a geodesic.
00088 // getFullData() returns all the data stored in the Diagnostic as a string (for output to file)
00089 virtual std::string getFullDataStr() const = 0;
00090 // getFinalDataVal() associates a value to the geodesic corresponding to the final value of this
diagnostic
00091 // (used for determining coarseness of nearby geodesics)
00092 virtual std::vector<real> getFinalDataVal() const = 0;
00093
00094 // Function used to determine distance between two values obtained from getFinalDataVal()
00095 // (for determining coarseness of nearby geodesics)
00096 // This should return a number >=0
00097 virtual real FinalDataValDistance(const std::vector<real> &vall, const std::vector<real> &val2)
const = 0;
00098
00099 // Getters for descriptions
00100 // This returns the name (only) of the Diagnostic, as a string without spaces that will be
appended
00101 // to an output file (e.g. prefix_DiagName.ext). Must be implemented!
00102 virtual std::string getNameStr() const = 0;
00103 // This returns the full description of the Diagnostic. Default implementation
00104 // returns GetDiagNameStr()
00105 virtual std::string getFullDescriptionStr() const;
00106
00107 protected:
00108 // The geodesic that owns the Diagnostic (a const pointer to the Geodesic)
00109 Geodesic *const m_OwnerGeodesic;
00110
00111 // Helper function to decide if the Diagnostic should indeed update its status, based on
00112 // its UpdateFrequency struct information (which will come from the Diagnostics's
DiagnosticOptions)
00113 bool DecideUpdate(const UpdateFrequency &myUpdateFrequency);
00114
00115 // The diagnostic is itself in charge of keeping track of how many steps it has been since it has
been updated
00116 // The Diagnostic's DiagnosticOptions struct tells it how many steps it needs to wait between
updates
00117 largecounter m_StepsSinceUpdated{};
00118 };
00119
00120 // Owner vector of derived Diagnostics classes
00121 using DiagnosticUniqueVector = std::vector<std::unique_ptr<Diagnostic>;
00122
00123 // Helper to create a new vector of Diagnostic options, based on the bitflag
00124 // The first diagnostic is the value diagnostic
00125 DiagnosticUniqueVector CreateDiagnosticVector(DiagBitflag diagflags, DiagBitflag valdiag, Geodesic
*const theGeodesic);
00126
00127
00128
00129
00130 // The four color screen: associates one of four colors based on the quadrant that the geodesic
finishes in
00131 // Will ONLY return a color if the geodesic indeed finishes because it passes through the boundary
sphere!
00132 class FourColorScreenDiagnostic final : public Diagnostic
00133 {
00134 public:
00135 // Basic constructor only passes on Geodesic pointer to base class constructor
00136 FourColorScreenDiagnostic(Geodesic *const theGeodesic) : Diagnostic(theGeodesic) {}
00137
00138 // Reset needs to be overridden in order to also reset m_quadrant
00139 void Reset() final;
00140
00141 // Sets the quadrant to the appropriate value, IF the boundary sphere has been reached
00142 void UpdateData() override;
00143
00144 // Both of these output functions simply returns the quadrant number associated with the
geodesic's end position
00145 std::string getFullDataStr() const final;
00146 std::vector<real> getFinalDataVal() const final;
00147
00148 // Discrete metric for distance: returns 0 if the quadrants are the same, 1 if they are not
00149 real FinalDataValDistance(const std::vector<real> &vall, const std::vector<real> &val2) const
final;
00150
00151 // Description string getters
00152 std::string getNameStr() const final;
00153 std::string getFullDescriptionStr() const final;
00154
00155 // FourColorScreen does not need any (static) options!
00156
00157 private:
00158 // Note initialization to 0; this means the default value returned will be 0
00159 // (e.g. if Term::BoundarySphere is not reached)
00160 int m_quadrant{0};
00161 };
00162
00163 // Forward declaration needed before Diagnostic

```

```

00164 struct GeodesicPositionOptions;
00165 // Geodesic position tracker
00166 class GeodesicPositionDiagnostic final : public Diagnostic
00167 {
00168 public:
00169     // Basic constructor only passes on Geodesic pointer to base class constructor
00170     GeodesicPositionDiagnostic(Geodesic *const theGeodesic) : Diagnostic(theGeodesic) {}
00171
00172     // Override Reset to also clear m_AllSavedPoints
00173     void Reset() final;
00174
00175     // Stores the current position of the geodesic
00176     void UpdateData() final;
00177
00178     // This returns as many stored positions as is specified in the options struct
00179     std::string getFullDataStr() const final;
00180     // This returns the final (theta,phi) value of the geodesic
00181     std::vector<real> getFinalDataVal() const final;
00182
00183     // This determines the angular distance between two geodesic
00184     // (based on their final angles on the boundary sphere (theta, phi))
00185     real FinalDataValDistance(const std::vector<real> &vall, const std::vector<real> &val2) const
00186     final;
00187
00188     // Description string getters
00189     std::string getNameStr() const final;
00190     std::string getFullDescriptionStr() const final;
00191
00192     // The options: specifies the update frequency but also how many points we want to output in the
00193     end
00194     static std::unique_ptr<GeodesicPositionOptions> DiagOptions;
00195 private:
00196     // Keeps track of the points that are saved
00197     std::vector<Point> m_AllSavedPoints{};
00198 };
00199 // Forward declaration needed before Diagnostic
00200 struct EquatorialPassesOptions;
00201 // Diagnostic for counting number of passes through equatorial plane
00202 class EquatorialPassesDiagnostic : public Diagnostic
00203 {
00204 public:
00205     // Basic constructor only passes on Geodesic pointer to base class constructor
00206     EquatorialPassesDiagnostic(Geodesic *const theGeodesic) : Diagnostic(theGeodesic) {}
00207
00208     // Override Reset to also reset m_EquatPasses and m_PrevTheta
00209     void Reset() override;
00210
00211     // Checks to see if we have a new cross over the equatorial plane
00212     void UpdateData() override;
00213
00214     // Returns the number of passes over the equatorial plane
00215     std::string getFullDataStr() const override;
00216     std::vector<real> getFinalDataVal() const override;
00217
00218     // Simple absolute value of difference of passes
00219     real FinalDataValDistance(const std::vector<real> &vall, const std::vector<real> &val2) const
00220     override;
00221
00222     // Description string getters
00223     std::string getNameStr() const override;
00224     std::string getFullDescriptionStr() const override;
00225
00226     // Needs the extra option of a threshold
00227     static std::unique_ptr<EquatorialPassesOptions> DiagOptions;
00228 protected:
00229     // Keeps track of how many passes have been made
00230     int m_EquatPasses{0};
00231
00232 private:
00233     // Keeps track of the previous theta angle, so that we can compare with current theta angle
00234     real m_PrevTheta{-1};
00235 };
00236 // Forward declaration needed before Diagnostic
00237 struct ClosestRadiusOptions;
00238 // Diagnostic keeps track of the closes radius that the geodesic passes through
00239 // Note: always keeps track of true radius, not log radius
00240 class ClosestRadiusDiagnostic final : public Diagnostic
00241 {
00242 public:
00243     // Basic constructor only passes on Geodesic pointer to base class constructor
00244     ClosestRadiusDiagnostic(Geodesic *const theGeodesic) : Diagnostic(theGeodesic) {}
00245
00246     // re-initialize closest radius
00247

```

```

00248     void Reset() final;
00249
00250     // Check to see if we have travelled closer
00251     void UpdateData() final;
00252
00253     // Return closest radius travelled
00254     std::string getFullDataStr() const final;
00255     std::vector<real> getFinalDataVal() const final;
00256
00257     // Returns the absolute value of the difference between closest radii
00258     real FinalDataValDistance(const std::vector<real> &vall, const std::vector<real> &val2) const
final;
00259
00260     // Description string getters
00261     std::string getNameStr() const final;
00262     std::string getFullDescriptionStr() const final;
00263
00264     // Option struct to keep track of RLogScale of Metric
00265     static std::unique_ptr<ClosestRadiusOptions> DiagOptions;
00266
00267 private:
00268     // Keeps track of closest radius travelled
00269     real m_ClosestRadius{-1};
00270 };
00271
00272 // Forward declaration needed before Diagnostic
00273 struct EquatorialEmissionOptions;
00274 // Diagnostic that calculates brightness intensity for the geodesic, based on
00275 // a specified equatorial disc emission model
00276 // (intensity profile and fluid velocity profile, both specified in the options struct)
00277 // Note that EquatorialEmissionDiagnostic inherits from EquatorialPassesDiagnostic,
00278 // since it needs to keep track of when it passes through the equatorial plane.
00279 // As a result it also keeps track of the number of equatorial passes
00280 class EquatorialEmissionDiagnostic final : public EquatorialPassesDiagnostic
00281 {
00282 public:
00283     // Basic constructor only passes on Geodesic pointer to parent class constructor
00284     EquatorialEmissionDiagnostic(Geodesic *const theGeodesic) :
EquatorialPassesDiagnostic(theGeodesic) {}
00285
00286     // Reset intensity back to 0.0, and call parent class Reset()
00287     void Reset() final;
00288
00289     // Use parent class UpdateData() to decide whether the update the intensity
00290     // using the specified emission model
00291     void UpdateData() final;
00292
00293     // Returns total intensity and total number of equatorial passes (i.e. normal output from
EquatorialPassesDiagnostic)
00294     std::string getFullDataStr() const final;
00295     std::vector<real> getFinalDataVal() const final;
00296
00297     // Distance is difference in intensities multiplied by a factor magnifying difference in
equatorial passes
00298     real FinalDataValDistance(const std::vector<real> &vall, const std::vector<real> &val2) const
final;
00299
00300     // Description string getters
00301     std::string getNameStr() const final;
00302     std::string getFullDescriptionStr() const final;
00303
00304     // Option struct to keep track emission model specifics
00305     static std::unique_ptr<EquatorialEmissionOptions> DiagOptions;
00306
00307 private:
00308     // Brightness intensity of the geodesic
00309     real m_Intensity{0.0};
00310 };
00311
00312 // Declare your Diagnostic class here, inheriting from Diagnostic.
00313 // Sample code:
00314 /*
00315 // Forward declaration needed before Diagnostic (if using base class DiagnosticOptions, this is
strictly speaking not necessary)
00316 struct DiagnosticOptions;
00317 // class definition
00318 class MyDiagnostic final : public Diagnostic // good practice to make the class final unless
descendant classes are possible
00319 {
00320 public:
00321     // constructor must at least take and pass along the const pointer to the owner Geodesic
00322     MyDiagnostic(Geodesic* const theGeodesic) : Diagnostic(theGeodesic) {}
00323
00324     // If you have MyDiagnostic-specific member variables, then they probably need to be reset at the
beginning
00325     // of integration for each new geodesic; in that case, you need to override Reset() to do so.
00326     // Note: make sure to call the base class implementation Diagnostic::Reset()
00327

```

```

00328     // from within your implementation of MyDiagnostic::Reset(), so that the base class internal
    variable is also reset!
00329     void Reset() final;
00330
00331     // This is the heart of the Diagnostic: here the Diagnostic updates its internal state according
    to
00332     // the owner Geodesic's current state
00333     void UpdateData() final;
00334
00335     // This should return the string that is to be outputted to the file as the final output of this
    Diagnostic
00336     // for its owner Geodesic
00337     std::string getFullDataStr() const final;
00338     // This should return a vector of real numbers that indicates the final "value" that should be
    associated to
00339     // the owner Geodesic --- this is then used in FinalDataValDistance() to find "distances" between
    geodesics.
00340     std::vector<real> getFinalDataVal() const final;
00341
00342     // This should return a (positive) distance of two values returned by getFinalDataVal(), indicated
    the
00343     // "distance" of two geodesics (this is used for Mesh refinement)
00344     real FinalDataValDistance(const std::vector<real>& val1, const std::vector<real>& val2) const
    final;
00345
00346     // Must implement getNameStr() (and recommended also to implement getFullDescriptionStr)
00347     // getNameStr() is a simple, short string (without spaces) that will be appended to the file name
00348     // where this Diagnostic's output is written. getFullDescriptionStr() is a descriptive string that
    should list
00349     // all relevant options set; this is outputted to e.g. the screen at runtime.
00350     std::string getNameStr() const final;
00351     std::string getFullDescriptionStr() const final;
00352
00353     // Diagnostic options: basic DiagnosticOptions only contains UpdateFrequency information,
00354     // if needed, can use a descendant class with more options (see e.g. GeodesicPositionDiagnostic)
00355     static std::unique_ptr<DiagnosticOptions> DiagOptions;
00356
00357 private:
00358     // will probably want some private member variable(s) to keep track of whatever geodesic property
    is desired
00359 };
00360 */
00362
00365
00366 // Base class for DiagnosticOptions. Other Diagnostics can inherit from here if they require more
    options.
00367 struct DiagnosticOptions
00368 {
00369 public:
00370     // Basic constructor only sets the number of steps between updates
00371     DiagnosticOptions(UpdateFrequency thefrequency) : theUpdateFrequency{thefrequency}
00372     {
00373     }
00374
00375     virtual ~DiagnosticOptions() = default;
00376
00377     const UpdateFrequency theUpdateFrequency;
00378 };
00379
00380 // GeodesicPositionDiagnostic needs to keep track of number of steps to output
00381 struct GeodesicPositionOptions : public DiagnosticOptions
00382 {
00383 public:
00384     GeodesicPositionOptions(largecounter outputsteps, UpdateFrequency thefrequency) :
    OutputNrSteps{outputsteps},
00385     DiagnosticOptions(thefrequency)
00386     {
00387     }
00388
00389     const largecounter OutputNrSteps;
00390 };
00391
00392 // EquatorialPassesDiagnostic needs to keep track of the threshold
00393 struct EquatorialPassesOptions : public DiagnosticOptions
00394 {
00395 public:
00396     EquatorialPassesOptions(real thethreshold, UpdateFrequency thefrequency) :
    Threshold{thethreshold},
00397     DiagnosticOptions(thefrequency)
00398     {
00399     }
00400
00401     const real Threshold;
00402 };
00403

```

```

00404 // ClosestRadiusOptions needs to keep track of whether we are using r or log(r) radial coordinate
00405 struct ClosestRadiusOptions : public DiagnosticOptions
00406 {
00407 public:
00408     ClosestRadiusOptions(bool rlog, UpdateFrequency thefrequency) : RLogScale{rlog},
00409                                                                    DiagnosticOptions(thefrequency)
00410     {
00411     }
00412
00413     const bool RLogScale;
00414 };
00415
00416 // Forward declarations of abstract EmissionModel and FluidVelocityModel classes.
00417 // See Diagnostics_Emission.h and .cpp for declarations and definitions of these and their descendant
00418 // classes!
00419 struct EmissionModel;
00420 struct FluidVelocityModel;
00421 // EquatorialEmissionDiagnostic needs to keep track of all tuneable parameters of the emission
00422 // Note that this inherits from EquatorialPassesOptions
00423 struct EquatorialEmissionOptions : public EquatorialPassesOptions
00424 {
00425 public:
00426     EquatorialEmissionOptions(real thefudgefactor, int equatupper,
00427                               std::unique_ptr<EmissionModel> theemission,
00428                               std::unique_ptr<FluidVelocityModel> thefluidmodel,
00429                               bool rlog, int theredshiftpower, real thethreshold, UpdateFrequency
00430                               thefrequency) : RedShiftPower{theredshiftpower}, RLogScale{rlog},
00431                                             GeometricFudgeFactor{thefudgefactor},
00432                                             EquatPassUpperBound{equatupper},
00433                                             TheEmissionModel{std::move(theemission)}, TheFluidVelocityModel{std::move(thefluidmodel)},
00434                                             EquatorialPassesOptions(thethreshold, thefrequency) // constructor for parent class
00435     {
00436     }
00437
00438     // Geometric fudge factor for n>0 passes
00439     const real GeometricFudgeFactor;
00440     // Upper bound allowed for contribution to intensity
00441     const int EquatPassUpperBound;
00442
00443     // Logarithmic radius scale used or not
00444     const bool RLogScale;
00445
00446     // Power of redshift in intensity contribution (should be 3 or 4)
00447     const int RedShiftPower;
00448
00449     // The emission model used (see Diagnostics_Emission.h and .cpp for specific models)
00450     const std::unique_ptr<EmissionModel> TheEmissionModel;
00451     // The fluid velocity model used (see Diagnostics_Emission.h and .cpp for specific models)
00452     const std::unique_ptr<FluidVelocityModel> TheFluidVelocityModel;
00453 };
00454
00455 // if necessary, define your new DiagnosticOptions class here
00456 // Sample code:
00457 /*
00458 struct MyDiagnosticOptions : public DiagnosticOptions
00459 {
00460 public:
00461     // Constructor should pass along UpdateFrequency information to base class
00462     MyDiagnosticOptions(UpdateFrequency thefrequency) : DiagnosticOptions(thefrequency) //, (...)
00463     other initializations
00464     {}
00465
00466     // other member variables here - make them const!
00467     // ...
00468 };
00469 */
00470 #endif

```

## 7.10 FOORT/src/DiagnosticsEmission.cpp File Reference

```

#include "DiagnosticsEmission.h"
#include "Integrators.h"
#include <cmath>

```

## 7.11 FOORT/src/DiagnosticsEmission.h File Reference

```
#include "Geometry.h"
#include "Metric.h"
#include "InputOutput.h"
#include <string>
#include <cmath>
```

### Classes

- struct [EmissionModel](#)
- struct [GLMJohnsonSUEmission](#)
- struct [FluidVelocityModel](#)
- struct [GeneralCircularRadialFluid](#)

## 7.12 DiagnosticsEmission.h

[Go to the documentation of this file.](#)

```
00001 #ifndef _FOORT_DIAGNOSTICS_EMISSION_H
00002 #define _FOORT_DIAGNOSTICS_EMISSION_H
00003
00004 #include "Geometry.h" // Tensor objects
00005 #include "Metric.h" // for Metric functions
00006 #include "InputOutput.h" // for ScreenOutput
00007
00008 #include <string> // for strings
00009 #include <cmath> // for fmax, fmin
00010
00011 // Here we declare the emission and fluid velocity models used for equatorial disc emission
00012
00013 // Emission model abstract base class
00014 struct EmissionModel
00015 {
00016 public:
00017     // Virtual destructor to ensure correct destruction
00018     virtual ~EmissionModel() = default;
00019
00020     // Function which returns the emitted brightness intensity at Point p
00021     // Note: always pass the true radius (not log(r)) to EmissionModel!
00022     virtual real GetEmission(const Point &p) const = 0;
00023
00024     // Description string getter
00025     virtual std::string getFullDescriptionStr() const;
00026 };
00027
00028 // The Johnson SU emission model used in GLM
00029 struct GLMJohnsonSUEmission final : public EmissionModel
00030 {
00031 public:
00032     // Constructor
00033     GLMJohnsonSUEmission(real mu, real gamma, real sigma) : m_mu{mu}, m_gamma{gamma}, m_sigma{sigma}
00034     {
00035     }
00036
00037     // Emitted brightness (only depends on radius)
00038     real GetEmission(const Point &p) const final;
00039
00040     // Description string getter
00041     std::string getFullDescriptionStr() const final;
00042
00043 private:
00044     // mu, gamma, sigma are the three parameters of the model
00045     const real m_mu;
00046     const real m_gamma;
00047     const real m_sigma;
00048 };
00049
00050
00051
00052 // Fluid velocity abstract base class
00053 struct FluidVelocityModel
```



```

00054 {
00055 public:
00056     // Constructor is passed Metric pointer
00057     FluidVelocityModel(const Metric *const theMetric) : m_theMetric{theMetric} {}
00058
00059     // Virtual destructor to ensure correct destruction
00060     virtual ~FluidVelocityModel() = default;
00061
00062     // Get the local four-velocity (with index down!) of the fluid at Point p
00063     virtual OneIndex GetFourVelocityd(const Point &p) const = 0;
00064
00065     // Description string getter
00066     virtual std::string getFullDescriptionStr() const;
00067
00068 protected:
00069     // Metric pointer, used for e.g. calculating geodesic orbits
00070     const Metric *const m_theMetric;
00071 };
00072
00073 // This fluid velocity model has three tuneable parameters and represents fluid travelling at a mix of
00074 // (sub)Keplerian circular orbits and radially infalling orbits in the equatorial plane
00075 struct GeneralCircularRadialFluid final : public FluidVelocityModel
00076 {
00077     // Constructor with three parameters and Metric pointer (which is passed to base class
00078     // constructor)
00079     GeneralCircularRadialFluid(real subKeplerParam, real betar, real betaphi, const Metric *const
00080 theMetric) : m_subKeplerParam{fmin(fmax(subKeplerParam, 0.0), 1.0)}, m_betaR{fmin(fmax(betar, 0.0),
00081 1.0)},
00082 m_betaPhi{fmin(fmax(betaphi, 0.0), 1.0)}, FluidVelocityModel(theMetric)
00083 {
00084     // Do some checks on three params, which must lie between 0.0 and 1.0 (note that they are
00085     // adjusted as such in
00086     // initializer above)
00087     if (subKeplerParam < 0.0)
00088         ScreenOutput("Sub-Keplerian parameter must be between 0 and 1; adjusting to 0",
00089 OutputLevel::Level_0_WARNING);
00090     if (subKeplerParam > 1.0)
00091         ScreenOutput("Sub-Keplerian parameter must be between 0 and 1; adjusting to 1",
00092 OutputLevel::Level_0_WARNING);
00093     if (betar < 0.0)
00094         ScreenOutput("beta_r parameter must be between 0 and 1; adjusting to 0",
00095 OutputLevel::Level_0_WARNING);
00096     if (betar > 1.0)
00097         ScreenOutput("beta_r parameter must be between 0 and 1; adjusting to 1",
00098 OutputLevel::Level_0_WARNING);
00099     if (betaphi < 0.0)
00100         ScreenOutput("beta_phi parameter must be between 0 and 1; adjusting to 0",
00101 OutputLevel::Level_0_WARNING);
00102     if (betaphi > 1.0)
00103         ScreenOutput("beta_phi parameter must be between 0 and 1; adjusting to 1",
00104 OutputLevel::Level_0_WARNING);
00105
00106     // Find the (equatorial) ISCO for this Metric
00107     FindISCO();
00108 }
00109
00110 // Get the local four-velocity of the fluid according to this model
00111 // Note: will always calculate with Point p exactly on the equator theta=pi/2, despite what p[2]
00112 // may be passed
00113 OneIndex GetFourVelocityd(const Point &p) const final;
00114
00115 // Description string getter
00116 std::string getFullDescriptionStr() const final;
00117
00118 private:
00119     // Three parameters determining the flow
00120     const real m_subKeplerParam;
00121     const real m_betaR;
00122     const real m_betaPhi;
00123
00124     // Helper function to get circular (sub)Keplerian velocity outside the ISCO
00125     OneIndex GetCircularVelocityd(const Point &p, bool subKeplerianOn = true) const;
00126     // Helper function to get circular and infalling (sub)Keplerian velocity inside ISCO
00127     // (according to prescription of Cunningham)
00128     OneIndex GetInsideISCOCircularVelocityd(const Point &p) const;
00129
00130     // Helper function to get radial infalling velocity
00131     OneIndex GetRadialVelocityd(const Point &p) const;
00132
00133     // Helper function to find the ISCO (called in Constructor)
00134     void FindISCO();
00135     // ISCO radius and momentum (index down) components
00136     bool m_ISCOexists{false};
00137     real m_ISCO{ -1.0 };

```

```

00129     real m_ISCOpt{};
00130     real m_ISCOpphi{};
00131
00132     // Helper function which returns \partial_r(g^{ab})\Gamma^r_{bc}g^{cd}),
00133     // whose sign (after contracted with p_a p_d of the corresponding circular orbit)
00134     // tells us if the circular orbit considered is stable or not
00135     TwoIndex GetChrstrRaisedDer(real r) const;
00136 };
00137
00138 #endif

```

## 7.13 FOORT/src/Geodesic.cpp File Reference

```

#include "Geodesic.h"
#include "InputOutput.h"

```

## 7.14 FOORT/src/Geodesic.h File Reference

```

#include "Geometry.h"
#include "Metric.h"
#include "Diagnostics.h"
#include "Terminations.h"
#include "Integrators.h"
#include <string>
#include <vector>

```

### Classes

- class [Source](#)
- class [NoSource](#)
- class [Geodesic](#)

## 7.15 Geodesic.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _FOORT_GEODESIC_H
00002 #define _FOORT_GEODESIC_H
00003
00010
00011 #include "Geometry.h" // for tensor objects
00012 #include "Metric.h" // for the metric
00013 #include "Diagnostics.h" // Geodesics own Diagnostics
00014 #include "Terminations.h" // Geodesics own Terminations
00015 #include "Integrators.h" // Geodesics use an GeodesicIntegratorFunc to integrate itself
00016
00017 #include <string> // for strings
00018 #include <vector> // for std::vector
00019
00022
00023 // Abstract base class
00024 class Source
00025 {
00026 public:
00027     // Constructor initializes Metric
00028     Source(const Metric *const theMetric) : m_theMetric{theMetric} {}
00029
00030     // Virtual destructor to ensure correct descendant destruction
00031     virtual ~Source() = default;

```

```

00032
00033 // Get the source for the current geodesic position and velocity
00034 virtual OneIndex getSource(Point pos, OneIndex vel) const = 0;
00035
00036 // Full description string (space allowed), to be outputted to file
00037 virtual std::string getFullDescriptionStr() const;
00038
00039 protected:
00040 // A const pointer to a const metric
00041 const Metric *const m_theMetric;
00042 };
00043
00044 // NoSource: there is no source, i.e. the geodesic is indeed a geodesic (and feels no force)
00045 class NoSource final : public Source
00046 {
00047 public:
00048 // Simple constructor passes on the Metric pointer to the base constructor
00049 NoSource(const Metric *const theMetric) : Source(theMetric) {}
00050
00051 // Returns zero source
00052 OneIndex getSource(Point pos, OneIndex vel) const final;
00053
00054 // Description string getter
00055 std::string getFullDescriptionStr() const final;
00056 };
00057
00060
00061 // Geodesic class: an instance of this class is created for each Geodesic that is integrated.
00062 // The Geodesic is in charge of integrating itself until termination, updating its Diagnostics
// accordingly,
00063 // and (after termination) returning the appropriate output.
00064 class Geodesic
00065 {
00066 public:
00067 // Default constructor not allowed
00068 Geodesic() = delete;
00069 // Copy constructor or copy assignment not allowed
00070 Geodesic(const Geodesic &) = delete;
00071 Geodesic &operator=(const Geodesic &) = delete;
00072
00073 // Constructor which creates the Geodesic object
00074 // Takes the following arguments which initialize the private member variables that remain the
// same over all geodesics
00075 // - Metric (pointer)
00076 // - Source (pointer)
00077 // - Diagnostic bitflag (& value Diagnostic bitflag) (used to create a vector of new instances of
// Diagnostics)
00078 // - Termination bitflag (used to create a vector of new instances of Terminations)
00079 // - Geodesic integrator function to use for integrating geodesic equation
00080 Geodesic(const Metric *const theMetric, const Source *const theSource,
00081         DiagBitflag diagbit, DiagBitflag valdiagbit,
00082         TermBitflag termbit, GeodesicIntegratorFunc theIntegrator) : m_theMetric(theMetric),
// m_theSource{theSource},
00083
// m_AllDiagnostics{CreateDiagnosticVector(diagbit, valdiagbit, this)},
00084
// m_AllTerminations{CreateTerminationVector(termbit, this)},
00085
// m_theIntegrator{theIntegrator}
00086 {
00087 }
00088
00089 // This initializes/resets the geodesic with a given ScreenIndex, initial position, and initial
// velocity
00090 // Also resets all Diagnostics and Terminations, resets the TermCondition to Term::Continue,
00091 // and puts the Geodesic back to lambda = 0.0.
00092 void Reset(ScreenIndex scrindex, Point initpos, OneIndex initvel);
00093
00094 // This makes the Geodesic integrate itself one step; then the Geodesic loops through all
// Terminations and Diagnostics to update
00095 Term Update();
00096
00097 // Getters for properties of its internal state
00098 Term getTermCondition() const; // Current termination condition (Term::Continue if not done
// integrating)
00099 Point getCurrentPos() const; // Current position
00100 OneIndex getCurrentVel() const; // Current velocity
00101 real getCurrentLambda() const; // Current value of affine parameter
00102 ScreenIndex getScreenIndex() const; // screen index
00103
00104 // Output getters, to be called after the Geodesic terminates
00105 // This gets the complete output that should be written to the output files;
00106 // there is one string more than the count of Diagnostics: one string per Diagnostic,
00107 // PLUS the first string is the screen index.
00108 std::vector<std::string> getAllOutputStr() const;
00109 // This returns the "value" (from the Diagnostic that was set to the value Diagnostic) that is
// associated

```

```

00110    // to the Geodesic. Will be used to determine "distance" between Geodesics which is used in Mesh
    refinement.
00111    std::vector<real> getDiagnosticFinalValue() const;
00112
00113 private:
00114    // These variables define its internal state
00115    Term m_TermCond{Term::Uninitialized}; // As long as this is Term::Continue, not done integrating
    yet
00116    Point m_CurrentPos{}; // Current position
00117    OneIndex m_CurrentVel{}; // Current proper velocity
00118    real m_curLambda{0.0}; // Current value of affine parameter (starts at 0.0)
00119
00120    // The Geodesic keeps track of what index it has been assigned;
00121    // it outputs this information in its final output string
00122    ScreenIndex m_ScreenIndex{};
00123
00124    // These are const pointers (or const vectors of pointers) that contain all the information the
    Geodesic needs
00125    const Metric *const m_theMetric; // Metric is needed to evaluate the geodesic equation
00126    const Source *const m_theSource; // Source for the rhs of the geodesic equation
00127    // An instance of each Diagnostic and Termination is created for the Geodesic (in its
    constructor);
00128    // so the Geodesic is the owner of these objects.
00129    const DiagnosticUniqueVector m_AllDiagnostics;
00130    const TerminationUniqueVector m_AllTerminations;
00131    const GeodesicIntegratorFunc m_theIntegrator; // This is the function that will integrate the
    geodesic equation one step
00132 };
00133
00134 #endif

```

## 7.16 FOORT/src/Geometry.h File Reference

```

#include <limits>
#include <string>
#include <array>
#include <utility>
#include <vector>

```

### Macros

- #define `LARGECOUNTER_MAX` `std::numeric_limits<largecounter>::max()`
- #define `PIXEL_MAX` `std::numeric_limits<pixelcoord>::max()`

### Typedefs

- using `real` = double
  - using `largecounter` = unsigned long
  - using `pixelcoord` = `largecounter`
  - using `Point` = `std::array<real, dimension>`
- TENSOR DEFINITIONS.*
- using `ScreenPoint` = `std::array<real, dimension - 2>`
  - using `ScreenIndex` = `std::array<pixelcoord, dimension - 2>`
  - using `OneIndex` = `Point`
  - using `TwoIndex` = `std::array<OneIndex, dimension>`
  - using `ThreeIndex` = `std::array<TwoIndex, dimension>`
  - using `FourIndex` = `std::array<ThreeIndex, dimension>`
  - using `SingularityCoord` = `std::pair<int, real>`
  - using `Singularity` = `std::vector<SingularityCoord>`

## Functions

- template<size\_t TensorDim>  
std::string [toString](#) (const std::array< [largecounter](#), TensorDim > &theTensor)  
*PRINTING TENSORS TO STRING.*
- template<size\_t TensorDim>  
std::string [toString](#) (const std::array< [real](#), TensorDim > &theTensor)
- template<typename Tensor, size\_t TensorDim>  
std::string [toString](#) (const std::array< Tensor, TensorDim > &theTensor)
- template<typename t, size\_t TensorDim>  
std::array< t, TensorDim > [operator+](#) (const std::array< t, TensorDim > &a1, const std::array< t, TensorDim > &a2)  
*TENSOR ARITHMETIC: addition/subtraction of tensors, scalar multiplication/division.*
- template<typename t, size\_t TensorDim>  
std::array< t, TensorDim > [operator-](#) (const std::array< t, TensorDim > &a1, const std::array< t, TensorDim > &a2)
- template<typename t, size\_t TensorDim>  
std::array< t, TensorDim > [operator\\*](#) (const std::array< t, TensorDim > &t1, [real](#) lambda)
- template<typename t, size\_t TensorDim>  
std::array< t, TensorDim > [operator\\*](#) ([real](#) lambda, const std::array< t, TensorDim > &t1)
- template<typename t, size\_t TensorDim>  
std::array< t, TensorDim > [operator/](#) (const std::array< t, TensorDim > &t1, [real](#) lambda)

## Variables

- constexpr [real](#) [pi](#) {3.1415926535}  
*CONSTANTS.*
- constexpr int [dimension](#) {4}

## 7.16.1 Macro Definition Documentation

### 7.16.1.1 LARGE\_COUNTER\_MAX

```
#define LARGE_COUNTER_MAX std::numeric_limits<largecounter>::max()
```

### 7.16.1.2 PIXEL\_MAX

```
#define PIXEL_MAX std::numeric_limits<pixelcoord>::max()
```

## 7.16.2 Typedef Documentation

### 7.16.2.1 FourIndex

```
using FourIndex = std::array<ThreeIndex, dimension>
```

### 7.16.2.2 largecounter

```
using largecounter = unsigned long
```

### 7.16.2.3 OneIndex

```
using OneIndex = Point
```

### 7.16.2.4 pixelcoord

```
using pixelcoord = largecounter
```

### 7.16.2.5 Point

```
using Point = std::array<real, dimension>
```

TENSOR DEFINITIONS.

### 7.16.2.6 real

```
using real = double
```

### 7.16.2.7 ScreenIndex

```
using ScreenIndex = std::array<pixelcoord, dimension - 2>
```

### 7.16.2.8 ScreenPoint

```
using ScreenPoint = std::array<real, dimension - 2>
```

### 7.16.2.9 Singularity

```
using Singularity = std::vector<SingularityCoord>
```

### 7.16.2.10 SingularityCoord

```
using SingularityCoord = std::pair<int, real>
```

### 7.16.2.11 ThreeIndex

```
using ThreeIndex = std::array<TwoIndex, dimension>
```

### 7.16.2.12 TwoIndex

```
using TwoIndex = std::array<OneIndex, dimension>
```

### 7.16.3 Function Documentation

#### 7.16.3.1 operator\*() [1/2]

```
template<typename t , size_t TensorDim>
std::array< t, TensorDim > operator* (
    const std::array< t, TensorDim > & t1,
    real lambda)
```

#### 7.16.3.2 operator\*() [2/2]

```
template<typename t , size_t TensorDim>
std::array< t, TensorDim > operator* (
    real lambda,
    const std::array< t, TensorDim > & t1)
```

#### 7.16.3.3 operator+()

```
template<typename t , size_t TensorDim>
std::array< t, TensorDim > operator+ (
    const std::array< t, TensorDim > & a1,
    const std::array< t, TensorDim > & a2)
```

TENSOR ARITHMETIC: addition/subtraction of tensors, scalar multiplication/division.

#### 7.16.3.4 operator-()

```
template<typename t , size_t TensorDim>
std::array< t, TensorDim > operator- (
    const std::array< t, TensorDim > & a1,
    const std::array< t, TensorDim > & a2)
```

#### 7.16.3.5 operator/()

```
template<typename t , size_t TensorDim>
std::array< t, TensorDim > operator/ (
    const std::array< t, TensorDim > & t1,
    real lambda)
```

#### 7.16.3.6 toString() [1/3]

```
template<size_t TensorDim>
std::string toString (
    const std::array< largecounter, TensorDim > & theTensor)
```

PRINTING TENSORS TO STRING.

### 7.16.3.7 toString() [2/3]

```
template<size_t TensorDim>
std::string toString (
    const std::array< real, TensorDim > & theTensor)
```

### 7.16.3.8 toString() [3/3]

```
template<typename Tensor , size_t TensorDim>
std::string toString (
    const std::array< Tensor, TensorDim > & theTensor)
```

## 7.16.4 Variable Documentation

### 7.16.4.1 dimension

```
int dimension {4} [inline], [constexpr]
```

### 7.16.4.2 pi

```
real pi {3.1415926535} [inline], [constexpr]
```

CONSTANTS.

## 7.17 Geometry.h

[Go to the documentation of this file.](#)

```
00001 #ifndef _FOORT_GEOMETRY_H
00002 #define _FOORT_GEOMETRY_H
00003
00011
00012 #include <limits> // for std::numeric_limits
00013 #include <string> // needed for toString(...) to convert tensors to strings
00014 #include <array> // needed to define tensors as fixed-size arrays of real or pixelcoord
00015 #include <utility> // needed for std::pair
00016 #include <vector> // for std::vector
00017
00018 // A real number.
00019 // (Could be changed to use arbitrary precision in the future.)
00020 using real = double;
00021
00022 // Note: An unsigned long is guaranteed to be able to hold at least 4 294 967 295 (4.10^10).
00023 // An unsigned int is only guaranteed to be able to hold 65 535, although
00024 // many modern-day implementations will actually make the int 32-bit and so much larger
00025 //
00026 // This type is used to count geodesics integrated
00027 using largecounter = unsigned long;
00028 // A pixel coordinate: always >=0 and integer; we use the largecounter type
00029 using pixelcoord = largecounter;
00030
00031 // Macro definition of maximum value that can be held in this large counter
00032 #ifndef LARGE_COUNTER_MAX
00033 #define LARGE_COUNTER_MAX std::numeric_limits<largecounter>::max()
00034 #endif
00035 #ifndef PIXEL_MAX
00036 #define PIXEL_MAX std::numeric_limits<pixelcoord>::max()
00037 #endif
00038
00042
00043 inline constexpr real pi{3.1415926535};
```



```

00044
00045 // The spacetime dimension
00046 inline constexpr int dimension{4};
00047
00051
00052 // A point in spacetime
00053 // Note that coordinates are always assumed to be (t, r, theta, phi)
00054 using Point = std::array<real, dimension>;
00055
00056 // A point on the ViewScreen; this does not have a time or radial extent
00057 using ScreenPoint = std::array<real, dimension - 2>;
00058
00059 // An index on the ViewScreen (row, column)
00060 using ScreenIndex = std::array<pixelcoord, dimension - 2>;
00061
00062 // Object with one index has the same structure as a Point
00063 using OneIndex = Point;
00064 // Object with two indices is an array of OneIndex objects
00065 using TwoIndex = std::array<OneIndex, dimension>;
00066 // Object with three indices is an array of TwoIndex objects
00067 using ThreeIndex = std::array<TwoIndex, dimension>;
00068 // Object with four indices is an array of ThreeIndex objects
00069 using FourIndex = std::array<ThreeIndex, dimension>;
00070
00071 // Definition used to define singularity of arbitrary codimension
00072 // SingularityCoord: pair of (coordinate number, coordinate value)
00073 using SingularityCoord = std::pair<int, real>;
00074 // Singularity: a number of SingularityCoords together that define a arbitrary codimension
// surface/line/point
00075 using Singularity = std::vector<SingularityCoord>;
00076
00080
00081 // Base case for single index tensor of unsigned integers (ScreenIndex).
00082 // We do not want toString(ScreenIndex) to convert its entries to reals and use the
00083 // implementation for a single index tensor of reals, because we do not want decimal points in our
00084 // string for the ints!
00085 template <size_t TensorDim>
00086 std::string toString(const std::array<largecounter, TensorDim> &theTensor)
00087 {
00088     std::string theStr{"("}; // no spaces for the innermost brackets
00089     for (int i = 0; i < TensorDim - 1; ++i)
00090     {
00091         // Here we use the std::to_string to convert the real to string
00092         theStr += std::to_string(theTensor[i]);
00093         theStr += ", ";
00094     }
00095     theStr += std::to_string(theTensor[TensorDim - 1]);
00096     theStr += ")"; // no spaces for the innermost brackets
00097
00098     return theStr;
00099 }
00100
00101 // Base case for single index tensor of reals (Point, OneIndex, ScreenPoint)
00102 template <size_t TensorDim>
00103 std::string toString(const std::array<real, TensorDim> &theTensor)
00104 {
00105     std::string theStr{"("}; // no spaces for the innermost brackets
00106
00107     for (int i = 0; i < TensorDim - 1; ++i)
00108     {
00109         // Here we use the std::to_string to convert the real to string
00110         theStr += std::to_string(theTensor[i]);
00111         theStr += ", ";
00112     }
00113     theStr += std::to_string(theTensor[TensorDim - 1]);
00114     theStr += ")"; // no spaces for the innermost brackets
00115
00116     return theStr;
00117 }
00118
00119 // General printing function for a tensor (TwoIndex, ThreeIndex, FourIndex);
00120 // recursively calls the lower rank tensor to print itself
00121 template <typename Tensor, size_t TensorDim>
00122 std::string toString(const std::array<Tensor, TensorDim> &theTensor)
00123 {
00124     std::string theStr{" ("}; // All but the innermost brackets have an extra space padding the
// bracket
00125
00126     for (int i = 0; i < TensorDim - 1; ++i)
00127     {
00128         theStr += toString(theTensor[i]);
00129         theStr += ", ";
00130     }
00131     theStr += toString(theTensor[TensorDim - 1]); // the last element doesn't have a comma after it
00132
00133     theStr += " )"; // All but the innermost brackets have an extra space padding the bracket
00134

```

```

00135     return theStr;
00136 }
00137
00141
00142 // Function to recursively call + on the lower rank tensor (OR the underlying reals/ints, if the
    tensor is rank 1)
00143 template <typename t, size_t TensorDim>
00144 std::array<t, TensorDim> operator+(const std::array<t, TensorDim> &a1, const std::array<t, TensorDim>
    &a2)
00145 {
00146     std::array<t, TensorDim> temp{a1};
00147     for (int i = 0; i < TensorDim; ++i)
00148         temp[i] = temp[i] + a2[i];
00149
00150     return temp;
00151 }
00152
00153 // Function to recursively call - on the lower rank tensor (OR the underlying reals/ints, if the
    tensor is rank 1)
00154 template <typename t, size_t TensorDim>
00155 std::array<t, TensorDim> operator-(const std::array<t, TensorDim> &a1, const std::array<t, TensorDim>
    &a2)
00156 {
00157     std::array<t, TensorDim> temp{a1};
00158     for (int i = 0; i < TensorDim; ++i)
00159         temp[i] = temp[i] - a2[i];
00160
00161     return temp;
00162 }
00163
00164 // Function to recursively scalar multiply the lower-rank tensors (OR the underlying reals/ints for
    the rank-1 tensor)
00165 template <typename t, size_t TensorDim>
00166 std::array<t, TensorDim> operator*(const std::array<t, TensorDim> &t1, real lambda)
00167 {
00168     std::array<t, TensorDim> temp{t1};
00169     for (int i = 0; i < TensorDim; ++i)
00170         temp[i] = static_cast<t>(temp[i] * lambda); // static_cast necessary if t is integral type!
00171     return temp;
00172 }
00173
00174 // For multiplication with a scalar on the left, call the multiplication on the right defined above
00175 template <typename t, size_t TensorDim>
00176 std::array<t, TensorDim> operator*(real lambda, const std::array<t, TensorDim> &t1)
00177 {
00178     return t1 * lambda;
00179 }
00180
00181 // For division with a scalar, call the multiplication defined above
00182 template <typename t, size_t TensorDim>
00183 std::array<t, TensorDim> operator/(const std::array<t, TensorDim> &t1, real lambda)
00184 {
00185     return t1 * (1 / lambda);
00186 }
00187
00188 #endif

```

## 7.18 FOORT/src/Header.h File Reference

## 7.19 Header.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
```

## 7.20 FOORT/src/InputOutput.cpp File Reference

```

#include "InputOutput.h"
#include <algorithm>
#include <filesystem>

```

## Functions

- void [SetOutputLevel](#) ([OutputLevel](#) theLvl)
- void [SetLoopMessageFrequency](#) ([largecounter](#) thefreq)
- [largecounter](#) [GetLoopMessageFrequency](#) ()
- void [ScreenOutput](#) (std::string\_view theOutput, [OutputLevel](#) lvl, bool newLine)

## 7.20.1 Function Documentation

### 7.20.1.1 GetLoopMessageFrequency()

```
largecounter GetLoopMessageFrequency ()
```

### 7.20.1.2 ScreenOutput()

```
void ScreenOutput (  
    std::string_view theOutput,  
    OutputLevel lvl,  
    bool newLine)
```

### 7.20.1.3 SetLoopMessageFrequency()

```
void SetLoopMessageFrequency (  
    largecounter thefreq)
```

### 7.20.1.4 SetOutputLevel()

```
void SetOutputLevel (  
    OutputLevel theLvl)
```

## 7.21 FOORT/src/InOutOutput.h File Reference

```
#include "Geometry.h"  
#include <string_view>  
#include <iostream>  
#include <fstream>  
#include <string>  
#include <vector>
```

## Classes

- class [GeodesicOutputHandler](#)

## Enumerations

- enum class `OutputLevel` {  
`Level_0_WARNING` = 0 , `Level_1_PROC` = 1 , `Level_2_SUBPROC` = 2 , `Level_3_ALLDetail` = 3 ,  
`Level_4_DEBUG` = 4 , `MaxLevel` }

## Functions

- void `SetOutputLevel` (`OutputLevel` theLvl)
- void `ScreenOutput` (std::string\_view theOutput, `OutputLevel` lvl=`OutputLevel::Level_3_ALLDetail`, bool newLine=true)
- void `SetLoopMessageFrequency` (`largecounter` thefreq)
- `largecounter` `GetLoopMessageFrequency` ()

## 7.21.1 Enumeration Type Documentation

### 7.21.1.1 OutputLevel

```
enum class OutputLevel [strong]
```

#### Enumerator

Level_0_WARNING	
Level_1_PROC	
Level_2_SUBPROC	
Level_3_ALLDetail	
Level_4_DEBUG	
MaxLevel	

## 7.21.2 Function Documentation

### 7.21.2.1 GetLoopMessageFrequency()

```
largecounter GetLoopMessageFrequency ()
```

### 7.21.2.2 ScreenOutput()

```
void ScreenOutput (
    std::string_view theOutput,
    OutputLevel lvl = OutputLevel::Level_3_ALLDetail,
    bool newLine = true)
```

### 7.21.2.3 SetLoopMessageFrequency()

```
void SetLoopMessageFrequency (
    largecounter thefreq)
```

## 7.21.2.4 SetOutputLevel()

```
void SetOutputLevel (
    OutputLevel theLvl)
```

## 7.22 InputOutput.h

[Go to the documentation of this file.](#)

```
00001 #ifndef _FOORT_INPUTOUTPUT_H
00002 #define _FOORT_INPUTOUTPUT_H
00003
00009
00010 #include "Geometry.h" // for basic tensor objects
00011
00012 #include <string_view> // std::string_view
00013 #include <iostream>     // needed for file and console output
00014 #include <fstream>      // needed for file output
00015 #include <string>        // std::string used in various places
00016 #include <vector>        // needed to create vectors of strings
00017
00020
00021 // Determines at what priority level the output is generated to the console.
00022 enum class OutputLevel
00023 {
00024     Level_0_WARNING = 0, // Only warnings are outputted
00025     Level_1_PROC = 1,    // Coarsest level output; only the major procedures produce output
00026     Level_2_SUBPROC = 2, // Subprocedures can also produce output
00027     Level_3_ALLDetail = 3, // Finest level output; all details are shown
00028     Level_4_DEBUG = 4,    // Finest level output AND debug messages as well
00029
00030     MaxLevel // (unused)
00031 };
00032
00033 // Set the output level used; note: the output level itself is a static variable in InputOutput.cpp
00034 void SetOutputLevel(OutputLevel theLvl);
00035
00036 // Outputs line to screen console, contingent on it being allowed by the set outputlevel
00037 void ScreenOutput(std::string_view theOutput, OutputLevel lvl = OutputLevel::Level_3_ALLDetail, bool
    newline = true);
00038
00039 // Set and Get for the loop message frequency (messages indicating progress during each integration
    loop)
00040 void SetLoopMessageFrequency(largecounter thefreq);
00041 largecounter GetLoopMessageFrequency();
00042
00045 // GeodesicOutputHandler declaration
00046
00047 // GeodesicOutputHandler handles all of the output to file.
00048 // It gets passed all of the output strings for every Geodesic, it then
00049 // stores this data until it eventually writes all data to the appropriate files
00050 class GeodesicOutputHandler
00051 {
00052 public:
00053     // No default constructor possible
00054     GeodesicOutputHandler() = delete;
00055     // Constructor must pass the following strings that are used to construct the file names of the
    output file:
00056     // FilePrefix, TimeStamp, FileExtension, and a vector of strings DiagNames (the names of each of
    the Diagnostics that will
00057     // be outputting). It must also specify how many outputs to cache before outputting to a file, and
00058     // how many geodesics are allowed per file created
00059     GeodesicOutputHandler(std::string FilePrefix, std::string TimeStamp, std::string FileExtension,
        std::vector<std::string> DiagNames,
00060         largecounter nroputstocache = LARGE_COUNTER_MAX - 1, // note -1,
00061         // since we will
    actually cache one more than this number before outputting everything
00063         largecounter geodperfile = LARGE_COUNTER_MAX,
        std::string firstlineinfo = "");
00064
00065
00066     // This tells the OutputHandler to prepare for this many geodesic outputs to arrive;
00067     // the internal state needs to be prepared such that they can come in without providing a data
    race
00068     void PrepareForOutput(largecounter nrOutputToCome);
00069
00070     // A new vector of output strings from a (single) Geodesic;
00071     // the length of the vector should be m_DiagNames.size()+1, since the first entry
00072     // is the screen index
00073     // NOTE: this procedure needs to be thread-safe!
00074     void NewGeodesicOutput(largecounter index, std::vector<std::string> theOutput);
```

```

00075
00076 // Calling this indicates that there is no further output to be expected;
00077 // this means we will write all remaining cached output to file
00078 void OutputFinished();
00079
00080 // Returns full description string of output handler
00081 std::string getFullDescriptionStr() const;
00082
00083 private:
00084 // Helper function: write everything that is cached to file now (clear the cache)
00085 void WriteCachedOutputToFile();
00086
00087 // Helper function: open the file with name filename for the first time, preparing it to write
00088 // This will effectively clear this file of any pre-existing content.
00089 void OpenForFirstTime(const std::string &filename);
00090
00091 // Helper function: return the full file name for the n-th output file
00092 // for the Diagnostic diagnr (this is an entry in m_DiagNames)
00093 std::string GetFileName(int diagnr, unsigned short filenr) const;
00094
00095 // The const strings that are used to construct the output file names
00096 const std::string m_FilePrefix;
00097 const std::string m_TimeStamp;
00098 const std::string m_FileExtension;
00099 const std::vector<std::string> m_DiagNames;
00100
00101 // const variables that control whether we write a descriptive first line in
00102 // every output file or not, and what that first line is
00103 const bool m_PrintFirstLineInfo;
00104 const std::string m_FirstLineInfoString;
00105
00106 // consts setting the maximum number of outputs that can be cached before writing output to file,
00107 // and the max number of geodesics to store in a file
00108 const largecounter m_nrOutputsToCache{};
00109 const largecounter m_nrGeodesicsPerFile{};
00110
00111 // If this is false, then we are writing to file(s). At any time, if a file I/O error occurs,
00112 // the output handler switches to outputting everything to the console
00113 bool m_WriteToConsole{false};
00114
00115 // How many outputs are already cached in m_nrOutputsToCache before the current iteration of
output
00116 largecounter m_PrevCached{0};
00117
00118 // The number of geodesics already written to the current file
00119 // (once this hits m_nrGeodesicsPerFile, this file is full)
00120 largecounter m_CurrentGeodesicsInFile{0};
00121
00122 // The current counter of completely full files (this is kept track of
00123 // so that it knows what the next file to write output to is)
00124 // (We had better not have more than 60k files!)
00125 unsigned short m_CurrentFullFiles{0};
00126
00127 // Cached data that has not been written to a file yet
00128 // (once this hits a size of > m_nrOutputsToCache,
00129 // this must be written to file(s))
00130 std::vector<std::vector<std::string> m_AllCachedData{};
00131 };
00132
00133 #endif

```

## 7.23 FOORT/src/Integrators.cpp File Reference

```

#include "Integrators.h"
#include "Geodesic.h"
#include <algorithm>
#include <cmath>
#include <sstream>
#include <iostream>

```

## 7.24 FOORT/src/Integrators.h File Reference

```

#include "Geometry.h"
#include "Metric.h"

```

```
#include <string>
```

## Namespaces

- namespace [Integrators](#)

## Typedefs

- using [GeodesicIntegratorFunc](#) = void (\*)([Point](#), [OneIndex](#), [Point](#) &, [OneIndex](#) &, [real](#) &, const [Metric](#) \*, const [Source](#) \*)

## Functions

- std::string [Integrators::GetFullIntegratorDescription](#) ()
- [real](#) [Integrators::GetAdaptiveStep](#) ([Point](#) curpos, [OneIndex](#) curvel)
- void [Integrators::IntegrateGeodesicStep\\_RK4](#) ([Point](#) curpos, [OneIndex](#) curvel, [Point](#) &nextpos, [OneIndex](#) &nextvel, [real](#) &stepsize, const [Metric](#) \*theMetric, const [Source](#) \*theSource)
- void [Integrators::IntegrateGeodesicStep\\_Verlet](#) ([Point](#) curpos, [OneIndex](#) curvel, [Point](#) &nextpos, [OneIndex](#) &nextvel, [real](#) &stepsize, const [Metric](#) \*theMetric, const [Source](#) \*theSource)

## Variables

- constexpr [real](#) [Integrators::delta\\_nodiv0](#) = 1e-20
- [real](#) [Integrators::Derivative\\_hval](#) {1e-7}
- std::string [Integrators::IntegratorDescription](#) {"RK4"}
- [real](#) [Integrators::epsilon](#) {0.03}
- [real](#) [Integrators::SmallestPossibleStepsize](#) {1e-12}
- [real](#) [Integrators::VerletVelocityTolerance](#) {0.001}

## 7.24.1 Typedef Documentation

### 7.24.1.1 GeodesicIntegratorFunc

```
using GeodesicIntegratorFunc = void (*)(Point, OneIndex, Point &, OneIndex &, real &, const Metric *, const Source *)
```

## 7.25 Integrators.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _FOORT_INTEGRATORS_H
00002 #define _FOORT_INTEGRATORS_H
00003
00009
00010 #include "Geometry.h" // for basic tensor objects
00011 #include "Metric.h"   // for the metric
00012
00013 #include <string> // std::string
00014
00015 // Forward declaration needed of Source
00016 // (Source is declared in Geodesic.h, but we want to avoid a header loop!)
00017 class Source;
00018
00019 // This is the structure of an function that integrates the geodesic equation one step
00020 // It takes as arguments:
00021 // - current position, current velocity
00022 // - references to: next position, next velocity, affine parameter step size (which the functions
    sets)
00023 // - pointers to: the Metric, the Source that are used to evaluate the geodesic equation
00024 using GeodesicIntegratorFunc = void (*)(Point, OneIndex, Point &, OneIndex &, real &, const Metric *,
    const Source *);
00025
00026 // Namespace for integrator constants and functions
00027 namespace Integrators
00028 {
00029     // This is used to avoid dividing by zero
00030     constexpr real delta_nodiv0 = 1e-20;
00031
00032     // The amount of any coordinate that we shift to calculate derivatives (using central difference)
00033     inline real Derivative_hval{1e-7};
00034
00035     // The name of the integrator selected
00036     inline std::string IntegratorDescription{"RK4"};
00037
00038     // Full descriptive string of integrator and all integrator options
00039     std::string GetFullIntegratorDescription();
00040
00041     // This is the base step size to be taken (the integrator will adapt this if necessary)
00042     inline real epsilon{0.03};
00043
00044     // The affine parameter must always go forward by at least this amount
00045     inline real SmallestPossibleStepsize{1e-12};
00046
00047     // Function to get (adaptive) step size
00048     real GetAdaptiveStep(Point curpos, OneIndex curvel);
00049
00050     // This is a GeodesicIntegratorFunc
00051     // Using the Runge-Kutta-4 algorithm to integrate the geodesic equation
00052     void IntegrateGeodesicStep_RK4(Point curpos, OneIndex curvel,
        Point &nextpos, OneIndex &nextvel, real &stepsize, const Metric
        *theMetric, const Source *theSource);
00054
00055     inline real VerletVelocityTolerance{0.001};
00056
00057     // This is a GeodesicIntegratorFunc
00058     // Using the velocity Verlet algorithm to integrate the geodesic equation
00059     void IntegrateGeodesicStep_Verlet(Point curpos, OneIndex curvel,
        Point &nextpos, OneIndex &nextvel, real &stepsize, const Metric
        *theMetric, const Source *theSource);
00061 }
00062
00063 #endif

```

## 7.26 FOORT/src/Main.cpp File Reference

```

#include "Geometry.h"
#include "Metric.h"
#include "Diagnostics.h"
#include "Terminations.h"
#include "Geodesic.h"
#include "ViewScreen.h"
#include "Integrators.h"
#include "InputOutput.h"

```



```
#include "Utilities.h"
#include <omp.h>
#include <iostream>
#include "Config.h"
```

## Functions

- void [LoadPrecompiledOptions](#) (std::unique\_ptr< [Metric](#) > &theM, std::unique\_ptr< [Source](#) > &theS, [DiagBitflag](#) &AllDiags, [DiagBitflag](#) &ValDiag, [TermBitflag](#) &AllTerms, std::unique\_ptr< [ViewScreen](#) > &theView, [GeodesicIntegratorFunc](#) &theIntegrator, std::unique\_ptr< [GeodesicOutputHandler](#) > &theOutputHandler)
- int [main](#) (int argc, char \*argv[])

## 7.26.1 Function Documentation

### 7.26.1.1 LoadPrecompiledOptions()

```
void LoadPrecompiledOptions (
    std::unique_ptr< Metric > & theM,
    std::unique_ptr< Source > & theS,
    DiagBitflag & AllDiags,
    DiagBitflag & ValDiag,
    TermBitflag & AllTerms,
    std::unique_ptr< ViewScreen > & theView,
    GeodesicIntegratorFunc & theIntegrator,
    std::unique_ptr< GeodesicOutputHandler > & theOutputHandler)
```

### 7.26.1.2 main()

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

## 7.27 FOORT/src/Mesh.cpp File Reference

```
#include "Mesh.h"
#include "Utilities.h"
#include <algorithm>
#include <limits>
#include <iostream>
```

## 7.28 FOORT/src/Mesh.h File Reference

```
#include "Geometry.h"
#include "Diagnostics.h"
#include "InputOutput.h"
#include <cmath>
#include <utility>
#include <forward_list>
#include <memory>
#include <vector>
#include <array>
#include <string>
```

### Classes

- class [Mesh](#)
- class [SimpleSquareMesh](#)
- class [InputCertainPixelsMesh](#)
- class [SquareSubdivisionMesh](#)
- struct [SquareSubdivisionMesh::PixelInfo](#)
- class [SquareSubdivisionMeshV2](#)
- struct [SquareSubdivisionMeshV2::PixelInfo](#)

## 7.29 Mesh.h

[Go to the documentation of this file.](#)

```
00001 #ifndef _FOORT_MESH_H
00002 #define _FOORT_MESH_H
00003
00009
00010 #include "Geometry.h" // needed for basic tensor objects
00011 #include "Diagnostics.h" // needed for Diagnostic "value" and "distance" functions
00012 #include "InputOutput.h" // needed for ScreenOutput
00013
00014 #include <cmath> // needed for sqrt (only on Linux)
00015 #include <utility> // std::move
00016 #include <forward_list> // std::forward_list
00017 #include <memory> // std::unique_ptr
00018 #include <vector> // std::vector
00019 #include <array> // std::array
00020 #include <string> // for strings
00021
00022 // Abstract Mesh base class
00023 class Mesh
00024 {
00025 public:
00026     // Basic constructor constructs Diagnostic that is used for determinines "values" and "distances"
    // between values
00027     Mesh(DiagBitflag valdiag)
00028     {
00029         // Calling CreateDiagnosticVector in this way will create a vector with exactly one element in
    // it,
00030         // i.e. the Diagnostic we need!
00031         : m_DistanceDiagnostic{std::move((CreateDiagnosticVector(valdiag, valdiag, nullptr))[0])}
00032     {
00033     }
00034     // virtual destructor to ensure correct destruction of descendants
00035     virtual ~Mesh() = default;
00036
00037     // Getter for how many geodesics the Mesh currently wants to integrate in this iteration
00038     virtual largecounter getCurNrGeodesics() const = 0;
00039
00040     // This sets a new initial conditions (in the form of a ScreenPoint and ScreenIndex)
00041     // for a next pixel to be integrated in the current iteration
```

```

00042     virtual void getNewInitConds(largecounter index, ScreenPoint &newunitpoint, ScreenIndex
&newscreenindex) const = 0;
00043
00044     // When a geodesic is finished integrating, it tells the Mesh and passes on its final "value"
00045     // NOTE: despite being a non-const member function, this must be designed to be thread-safe!
00046     virtual void GeodesicFinished(largecounter index, std::vector<real> finalValues) = 0;
00047
00048     // This is called when the current iteration is finished. The Mesh can now evaluate whether to
continue or not
00049     virtual void EndCurrentLoop() = 0;
00050
00051     // Returns false if the Mesh wants another iteration of pixels to integrate
00052     virtual bool IsFinished() const = 0;
00053
00054     // Returns a string description of the Mesh (spaces allowed), describing its options
00055     virtual std::string getFullDescriptionStr() const;
00056
00057 protected:
00058     // The Diagnostic (a const pointer to a const Diagnostic object) that is used to calculate
00059     // distances (using FinalDataValDistance()) between the "values" that are assigned to Geodesics
00060     const std::unique_ptr<const Diagnostic> m_DistanceDiagnostic;
00061 };
00062
00063 // A simple square mesh that will integrate a square of evenly spaced pixels
00064 class SimpleSquareMesh final : public Mesh
00065 {
00066 public:
00067     // Default constructor not possible
00068     SimpleSquareMesh() = delete;
00069     // Constructor initializes total number of pixels and passes valdiag to base constructor
00070     // Note that we static_cast the sqrt() to round off the row/column size to an integer number
00071     SimpleSquareMesh(largecounter totalPixels, DiagBitflag valdiag)
00072         : m_TotalPixels{static_cast<pixelcoord>(sqrt(totalPixels)) *
static_cast<pixelcoord>(sqrt(totalPixels))},
m_RowColumnSize{static_cast<pixelcoord>(sqrt(totalPixels))},
Mesh(valdiag)
00073     {
00074         if constexpr (dimension != 4)
00075             ScreenOutput("SimpleSquareMesh only defined in 4D!", OutputLevel::Level_0_WARNING);
00076     }
00077
00078     // Declarations of overriding virtual functions
00079
00080     largecounter getCurNrGeodesics() const final;
00081
00082     void getNewInitConds(largecounter index, ScreenPoint &newunitpoint, ScreenIndex &newscreenindex)
const final;
00083
00084     void GeodesicFinished(largecounter index, std::vector<real> finalValues) final;
00085
00086     void EndCurrentLoop() final;
00087
00088     bool IsFinished() const final;
00089
00090     // Description string getter
00091     std::string getFullDescriptionStr() const final;
00092
00093 private:
00094     // Total amount of pixels in grid (is the square of m_RowColumnSize)
00095     const largecounter m_TotalPixels;
00096     // Amount of pixels per row or column (square grid)
00097     const pixelcoord m_RowColumnSize;
00098     // Are we done integrating or not?
00099     bool m_Finished{false};
00100 };
00101
00102 // Mesh which integrates only certain user-inputted pixels
00103 class InputCertainPixelsMesh : public Mesh
00104 {
00105 public:
00106     // Default constructor not possible
00107     InputCertainPixelsMesh() = delete;
00108     // Copy constructor not possible
00109     InputCertainPixelsMesh(const InputCertainPixelsMesh &) = delete;
00110     // Constructor given in Mesh.cpp file; constructor asks for input of pixels
00111     InputCertainPixelsMesh(largecounter totalPixels, DiagBitflag valdiag);
00112
00113     // Declarations of overriding virtual functions
00114
00115     largecounter getCurNrGeodesics() const final;
00116
00117     void getNewInitConds(largecounter index, ScreenPoint &newunitpoint, ScreenIndex &newscreenindex)
const final;
00118
00119     void GeodesicFinished(largecounter index, std::vector<real> finalValues) final;
00120
00121     void EndCurrentLoop() final;

```

```

00124
00125     bool IsFinished() const final;
00126
00127     // Description string getter
00128     std::string getFullDescriptionStr() const final;
00129
00130 private:
00131     // Total pixel size of the screen (square grid)
00132     const pixelcoord m_RowColumnSize;
00133
00134     // How many pixels have been inputted in total, i.e. need integrating
00135     largecounter m_TotalPixels{0};
00136     // All pixels' location
00137     std::vector<ScreenIndex> m_PixelsToIntegrate{};
00138     // Are we finished integrating?
00139     bool m_Finished{false};
00140 };
00141
00142 // Adaptive subdivision Mesh: starts with evenly spaced, square Mesh,
00143 // then decides to subdivide certain squares of pixels into smaller squares,
00144 // based on which pixels have a bigger "weight", which is defined as the maximum
00145 // "distance" (using the Diagnostic value distance) between the upper-left
00146 // vertex of the square with the other three vertices of the square.
00147 class SquareSubdivisionMesh : public Mesh
00148 {
00149 public:
00150     // default constructor not possible
00151     SquareSubdivisionMesh() = delete;
00152     // Constructor must be called with arguments:
00153     // - maxPixels: max. nr of pixels that can be integrated in TOTAL, over all iterations (if 0, then
    this is infinite,
00154     // i.e. we keep integrating until all squares are maximally subdivided or have weight 0)
00155     // - initialPixels: initial number of pixels to integrate (spaced equally over the screen)
00156     // - maxSubdivide: maximum number of times that we can subdivide squares (note: 1 denotes the
00157     // initial grid, so 2 would be subdividing the squares once)
00158     // - iterationPixels: maximum number of pixels to subdivide in each integration iteration
00159     // (max number of pixels that will be integrates is then 5*iterationPixels)
00160     // - initialSubToFinal: once we decide to subdivide a square, do we automatically keep subdividing
    it
00161     // until we reach maxSubdivision?
00162     // - valdiag: the "value" and "distance" Diagnostic to use
00163     SquareSubdivisionMesh(largecounter maxPixels, largecounter initialPixels, int maxSubdivide,
    largecounter iterationPixels, bool initialSubToFinal,
00164                          DiagBitflag valdiag)
00165     : m_InitialPixels{static_cast<pixelcoord>(sqrt(initialPixels)) *
    static_cast<pixelcoord>(sqrt(initialPixels))},
00166       m_MaxSubdivide{maxSubdivide},
00167       m_RowColumnSize{static_cast<pixelcoord>((sqrt(initialPixels) - 1) * ExpInt(2, maxSubdivide -
    1) + 1)},
00168       m_PixelsLeft{maxPixels}, m_MaxPixels{maxPixels}, m_InfinitePixels{maxPixels == 0},
    m_IterationPixels{iterationPixels},
00169       m_InitialSubDivideToFinal{initialSubToFinal}, Mesh(valdiag)
00170     {
00171         if constexpr (dimension != 4)
00172             ScreenOutput("SquareSubdivisionMesh only defined in 4D!", OutputLevel::Level_0_WARNING);
00173
00174         // DEBUG message for constructor (can delete)
00175         ScreenOutput("SquareSubdivisionMesh constructed: maxPixels: " + (m_InfinitePixels ? "infinite"
    : std::to_string(maxPixels)) + "; m_InitialPixels: " + std::to_string(m_InitialPixels) + ";
    m_RowColumnSize: " + std::to_string(m_RowColumnSize), OutputLevel::Level_4_DEBUG);
00176
00177         // Initialize the initial square, equally spaced grid
00178         InitializeFirstGrid();
00179     }
00180
00181     // Declarations of overriding virtual functions
00182
00183     largecounter getCurNrGeodesics() const final;
00184
00185     void getNewInitConds(largecounter index, ScreenPoint &newunitpoint, ScreenIndex &newscreenindex)
    const final;
00186
00187     void GeodesicFinished(largecounter index, std::vector<real> finalValues) final;
00188
00189     void EndCurrentLoop() final;
00190
00191     bool IsFinished() const final;
00192
00193     // Description string getter
00194     std::string getFullDescriptionStr() const final;
00195
00196 private:
00197     // How many initial pixels (spread uniformly over the grid) do we integrate?
00198     const largecounter m_InitialPixels;
00199     // How many times are we allowed to subdivide a square? Note: the initial grid is already at 1
00200     const int m_MaxSubdivide;
00201     // The total size in pixels of a row or column (square grid)

```

```

00202     const pixelcoord m_RowColumnSize;
00203     // How many pixels per iteration can we subdivide?
00204     const largecounter m_IterationPixels;
00205     // How many pixels can we integrate in total over all iterations?
00206     const largecounter m_MaxPixels;
00207     // If we decide to subdivide a square, do we automatically subdivide it further to the max level?
00208     const bool m_InitialSubDivideToFinal;
00209     // Are we allowed to integrate as many pixels as we want? (m_MaxPixels == 0)
00210     const bool m_InfinitePixels;
00211
00212     // How many pixels are we still allowed to integrate (if !m_InfinitePixels)?
00213     largecounter m_PixelsLeft;
00214
00215     // A struct the Mesh uses to keep all information about a given pixel
00216     struct PixelInfo
00217     {
00218         // Constructor with its ScreenIndex and current subdivision level
00219         PixelInfo(ScreenIndex ind, int subdiv) : Index{ind}, SubdivideLevel{subdiv} {}
00220
00221         // The pixel's screenindex
00222         ScreenIndex Index{};
00223
00224         // The level at which the pixel has been subdivided
00225         // Note: initial grid pixels are at 1; pixels at 0 are pixels that cannot be subdivided
00226         // (for example, at the right or lower edges)
00227         int SubdivideLevel{};
00228
00229         // Weight of the pixel: if negative, this signifies that it needs to be updated/calculated!
00230         // The weight is determined as the max of the distance (as calculated by the value Diagnostic)
00231         // between its values and those of its right, lower, and right-lower neighbors.
00232         real Weight{-1};
00233
00234         // The values associated to this pixel (as calculated by the value Diagnostic)
00235         std::vector<real> DiagValue{};
00236
00237         // Where its lower and right neighbors are located in m_AllPixels
00238         // Note: the pixel with index 0 is (0,0) and can never be the lower or right neighbor of any
00239         // other pixel!
00240         largecounter LowerNbrIndex{0};
00241         largecounter RightNbrIndex{0};
00242     };
00243     // The current queue of pixels to be integrated
00244     std::vector<PixelInfo> m_CurrentPixelQueue{};
00245     // A bool for every pixel in the current queue: gets set to true when the pixel is done
00246     // integrating and gets its values returned
00247     std::vector<bool> m_CurrentPixelQueueDone{};
00248     // All pixels that have been integrated already (so does not include the pixels in the current
00249     // queue)
00250     std::vector<PixelInfo> m_AllPixels{};
00251
00252     // Initializes the first nxn screen in m_CurrentPixelQueue
00253     void InitializeFirstGrid();
00254
00255     // Updates the neighbors of all pixels (that should have neighbors and don't yet) in m_AllPixels
00256     void UpdateAllNeighbors();
00257
00258     // Updates all weights of the pixels in m_AllPixels with weight < 0 and subdiv > 0 and subdiv <
00259     // m_MaxSubdivide
00260     // Assumes all squares have neighbors assigned correctly
00261     void UpdateAllWeights();
00262
00263     // This will take the pixel m_AllPixels[ind] and subdivide it,
00264     // adding up to <=5 pixels to the CurrentPixelQueue
00265     void SubdivideAndQueue(largecounter ind);
00266
00267     // Helper function to exponentiate an int to an int
00268     // Note: the result can be larger than fits in an int, but the base is always 2 and the exp is
00269     // always
00270     // a number <=m_MaxSubdivide (which is an int)
00271     pixelcoord ExpInt(int base, int exp);
00272 };
00273
00274 // Adaptive subdivision Mesh: starts with evenly spaced, square Mesh,
00275 // then decides to subdivide certain squares of pixels into smaller squares,
00276 // based on which pixels have a bigger "weight", which is defined as the maximum
00277 // "distance" (using the Diagnostic value distance) between the upper-left
00278 // vertex of the square with the other three vertices of the square.
00279 // V2: new way of dealing with neighbors and looping over pixels
00280 class SquareSubdivisionMeshV2 : public Mesh
00281 {
00282 public:
00283     // default constructor not possible
00284     SquareSubdivisionMeshV2() = delete;
00285     // Constructor must be called with arguments:
00286     // - maxPixels: max. nr of pixels that can be integrated in TOTAL, over all iterations (if 0, then
00287     //   this is infinite,
00288     // - i.e. we keep integrating until all squares are maximally subdivided or have weight 0)

```

```

00283 // - initialPixels: initial number of pixels to integrate (spaced equally over the screen)
00284 // - maxSubdivide: maximum number of times that we can subdivide squares (note: 1 denotes the
00285 // initial grid, so 2 would be subdividing the squares once)
00286 // - iterationPixels: maximum number of pixels to subdivide in each integration iteration
00287 // (max number of pixels that will be integrates is then 5*iterationPixels)
00288 // - initialSubToFinal: once we decide to subdivide a square, do we automatically keep subdividing
it
00289 // until we reach maxSubdivision?
00290 // - valdiag: the "value" and "distance" Diagnostic to use
00291 SquareSubdivisionMeshV2(largecounter maxPixels, largecounter initialPixels, int maxSubdivide,
largecounter iterationPixels, bool initialSubToFinal,
00292                         DiagBitflag valdiag)
00293 : m_InitialPixels(static_cast<pixelcoord>(sqrt(initialPixels)) *
static_cast<pixelcoord>(sqrt(initialPixels))),
00294   m_MaxSubdivide{maxSubdivide},
00295   m_RowColumnSize{static_cast<pixelcoord>((sqrt(initialPixels) - 1) * ExpInt(2, maxSubdivide -
1) + 1)},
00296   m_PixelsLeft{maxPixels}, m_MaxPixels{maxPixels}, m_InfinitePixels{maxPixels == 0},
m_IterationPixels{iterationPixels},
00297   m_InitialSubDivideToFinal{initialSubToFinal}, Mesh(valdiag)
00298 {
00299     if constexpr (dimension != 4)
00300         ScreenOutput("SquareSubdivisionMeshV2 only defined in 4D!", OutputLevel::Level_0_WARNING);
00301
00302     // Initialize the initial square, equally spaced grid
00303     InitializeFirstGrid();
00304 }
00305
00306 // Declarations of overriding virtual functions
00307
00308 largecounter getCurNrGeodesics() const final;
00309
00310 void getNewInitConds(largecounter index, ScreenPoint &newunitpoint, ScreenIndex &newscreenindex)
const final;
00311
00312 void GeodesicFinished(largecounter index, std::vector<real> finalValues) final;
00313
00314 void EndCurrentLoop() final;
00315
00316 bool IsFinished() const final;
00317
00318 // Description string getter
00319 std::string getFullDescriptionStr() const final;
00320
00321 private:
00322 // How many initial pixels (spread uniformly over the grid) do we integrate?
00323 const largecounter m_InitialPixels;
00324 // How many times are we allowed to subdivide a square? Note: the initial grid is already at 1
00325 const int m_MaxSubdivide;
00326 // The total size in pixels of a row or column (square grid)
00327 const pixelcoord m_RowColumnSize;
00328 // How many pixels per iteration can we subdivide?
00329 const largecounter m_IterationPixels;
00330 // How many pixels can we integrate in total over all iterations?
00331 const largecounter m_MaxPixels;
00332 // If we decide to subdivide a square, do we automatically subdivide it further to the max level?
00333 const bool m_InitialSubDivideToFinal;
00334 // Are we allowed to integrate as many pixels as we want? (m_MaxPixels == 0)
00335 const bool m_InfinitePixels;
00336
00337 // How many pixels are we still allowed to integrate (if !m_InfinitePixels)?
00338 largecounter m_PixelsLeft;
00339
00340 // How many pixels we have integrated so far
00341 largecounter m_PixelsIntegrated{0};
00342
00343 // A struct the Mesh uses to keep all information about a given pixel
00344 struct PixelInfo
00345 {
00346     // Constructor with its ScreenIndex and current subdivision level
00347     PixelInfo(ScreenIndex ind, int subdiv) : Index{ind}, SubdivideLevel{subdiv} {}
00348
00349     // The pixel's screenindex: this gets set by the constructor and cannot change anymore
00350     const ScreenIndex Index{};
00351
00352     // The level at which the pixel has been subdivided
00353     // Note: initial grid pixels are at 1; pixels at 0 are pixels that cannot be subdivided
00354     // (for example, at the right or lower edges)
00355     int SubdivideLevel{};
00356
00357     // Weight of the pixel: if negative, this signifies that it needs to be updated/calculated!
00358     // The weight is determined as the max of the distance (as calculated by the value Diagnostic)
00359     // between its values and those of its right, lower, and right-lower neighbors.
00360     real Weight{-1};
00361
00362     // The values associated to this pixel (as calculated by the value Diagnostic)
00363     std::vector<real> DiagValue{};

```

```

00364
00365     // Pointers to its neighbors
00366     PixelInfo *LeftNbr{nullptr};
00367     PixelInfo *RightNbr{nullptr};
00368     PixelInfo *UpNbr{nullptr};
00369     PixelInfo *DownNbr{nullptr};
00370     PixelInfo *SEdiagNbr{nullptr};
00371 };
00372
00373     // Master list of all pixels
00374     // std::forward_list is more space-efficient than std::list, bidirectional iteration is not
needed, no random access supported
00375     // This list is only used to store the owner pointers (and thus the objects) of all pixels.
00376     // The other pixel vectors are used to iterate through (and need random access)
00377     std::forward_list<std::unique_ptr<PixelInfo>> m_AllPixels{};
00378
00379     // List of active pixels, i.e. those that can be subdivided and have non-zero weight
00380     std::vector<PixelInfo *> m_ActivePixels{};
00381     // List of current queue of pixels to be sent to be integrated
00382     std::vector<PixelInfo *> m_CurrentPixelQueue{};
00383     // A bool for every pixel in the current queue: gets set to true when the pixel is done
integrating and gets its values returned
00384     std::vector<bool> m_CurrentPixelQueueDone{};
00385     // List of pixels that are already integrated but need updating weights after current queue is all
integrated
00386     std::vector<PixelInfo *> m_CurrentPixelUpdating{};
00387
00388     // Initializes the first nxn screen and puts them in m_CurrentPixelQueue
00389     void InitializeFirstGrid();
00390
00391     // Updates all weights of the pixels in m_CurrentPixelUpdating;
00392     // these should have subdiv > 0 and subdiv < m_MaxSubdivide, and all their neighbors assigned
correctly
00393     // All pixels with weight > 0 will be added to m_ActivePixels
00394     void UpdateAllWeights();
00395
00396     // Helper functions that return the appropriate neighbor of p, ONLY if this neighbor exists at the
subdivision level specified
00397     // Returns nullptr otherwise; also return nullptr if p==nullptr
00398     PixelInfo *GetUp(PixelInfo *p, int subdiv) const;
00399     PixelInfo *GetDown(PixelInfo *p, int subdiv) const;
00400     PixelInfo *GetRight(PixelInfo *p, int subdiv) const;
00401     PixelInfo *GetLeft(PixelInfo *p, int subdiv) const;
00402
00403     // This will take the pixel m_AllPixels[ind] and subdivide it,
00404     // adding up to <=5 pixels to the CurrentPixelQueue
00405     void SubdivideAndQueue(largecounter ind);
00406
00407     // Helper function to exponentiate an int to an int
00408     // Note: the result can be larger than fits in an int, but the base is always 2 and the exp is
always
00409     // a number <=m_MaxSubdivide (which is an int)
00410     pixelcoord ExpInt(int base, int exp) const;
00411 };
00412
00413 #endif

```

## 7.30 FOORT/src/Metric.cpp File Reference

```

#include "Metric.h"
#include "InputOutput.h"
#include "Integrators.h"
#include <cmath>
#include <algorithm>
#include "Spline.h"
#include <sstream>

```

## 7.31 FOORT/src/Metric.h File Reference

```

#include "Geometry.h"
#include "Spline.h"

```

```
#include <string>
#include <vector>
```

## Classes

- class [Metric](#)
- class [SphericalHorizonMetric](#)
- class [KerrMetric](#)
- class [FlatSpaceMetric](#)
- class [RasheedLarsenMetric](#)
- class [JohannsenMetric](#)
- class [MankoNovikovMetric](#)
- class [KerrSchildMetric](#)
- class [SingularityMetric](#)
- class [ST3CrMetric](#)
- class [BosonStarMetric](#)

## 7.32 Metric.h

[Go to the documentation of this file.](#)

```
00001 #ifndef _FOORT_METRIC_H
00002 #define _FOORT_METRIC_H
00003
00004 #include "Geometry.h" // Needed for basic tensor objects etc.
00005
00006 #include "Spline.h"
00007 #include <string> // for strings
00008 #include <vector> // needed for the (non-fixed size) vector of symmetries in the metric
00009
00010
00011 // The abstract base class for all Metrics.
00012 class Metric
00013 {
00014 public:
00015     // Virtual destructor to ensure correct destruction of descendants
00016     virtual ~Metric() = default;
00017
00018     Metric(bool rlogscale = false);
00019
00020     // Basic functions that return the metric with indices down or up:
00021     // pure virtual as they must be defined in the descendant classes.
00022     //
00023     // Get the metric at Point p, indices down
00024     virtual TwoIndex getMetric_dd(const Point &p) const = 0;
00025     // Get the metric at Point p, indices up
00026     virtual TwoIndex getMetric_uu(const Point &p) const = 0;
00027
00028     // The following functions return the Christoffel and other derivative quantities of the metric.
00029     // They are implemented for this base class, BUT are left as virtual functions to allow for
00030     // other metrics to implement their own (more efficient)
00031     // way of calculating them, if so desired.
00032     //
00033     // Get the Christoffel symbol, indices up-down-down
00034     virtual ThreeIndex getChristoffel_udd(const Point &p) const;
00035     // Get the Riemann tensor, indices up-down-down-down
00036     virtual FourIndex getRiemann_uddd(const Point &p) const;
00037     // Get the Kretschmann scalar
00038     virtual real getKretschmann(const Point &p) const;
00039
00040     // Function to get the description of the metric
00041     // (used for outputting to the screen while running and possibly to the output files)
00042     // There is a base class implementation of this function returning an un-descriptive string
00043     virtual std::string getFullDescriptionStr() const;
00044
00045     bool getLogScale() const;
00046
00047 protected:
00048     // The symmetries (coordinate Killing vectors) of the metric. Should be set by descendant
00049     // constructor.
```



```

00054     std::vector<int> m_Symmetries{};
00055     // Are we using a logarithmic r coordinate?
00056     const bool m_rLogScale;
00057 };
00058
00059 // Abstract base class for a metric that has a spherical horizon (i.e. horizon at constant radius r)
00060 class SphericalHorizonMetric : public Metric
00061 {
00062 public:
00063     // No default construction allowed, must specify horizon radius
00064     SphericalHorizonMetric() = delete;
00065     // Constructor that initializes horizon radius
00066     SphericalHorizonMetric(real HorizonRadius, bool rLogScale);
00067
00068     // Getter functions for the two member variables
00069     real getHorizonRadius() const;
00070
00071 protected:
00072     // Radius of the horizon
00073     const real m_HorizonRadius;
00074 };
00075
00076 // The Kerr metric (normalized so that M = 1)
00077 class KerrMetric final : public SphericalHorizonMetric
00078 {
00079 private:
00080     // Mass-rescaled rotation parameter for Kerr
00081     // Note that this should be between -1 and 1.
00082     const real m_aParam;
00083
00084     // Mass parameter for Kerr. Default is 1.
00085     const real m_mParam;
00086
00087 public:
00088     // No default constructor allowed, must specify a
00089     KerrMetric() = delete;
00090
00091     // Constructor setting parameter a
00092     KerrMetric(real aParam, bool rLogScale = false, real mParam = 1.);
00093
00094     // The override of the basic metric getter functions
00095     TwoIndex getMetric_dd(const Point &p) const final;
00096     TwoIndex getMetric_uu(const Point &p) const final;
00097
00098     // The override of the description string getter
00099     std::string getFullDescriptionStr() const final;
00100 };
00101
00102 // Flat space (4D)
00103 class FlatSpaceMetric final : public Metric
00104 {
00105 public:
00106     // Simple (default) constructor is all that is needed
00107     FlatSpaceMetric(bool rlogscale = false);
00108
00109     // The override of the basic metric getter functions
00110     TwoIndex getMetric_dd(const Point &p) const final;
00111     TwoIndex getMetric_uu(const Point &p) const final;
00112
00113     // The override of the description string getter
00114     std::string getFullDescriptionStr() const final;
00115 };
00116
00117 // Rasheed-Larsen black hole
00118 class RasheedLarsenMetric final : public SphericalHorizonMetric
00119 {
00120 private:
00121     // Rasheed-Larsen is specified by four parameters
00122     const real m_aParam;
00123     const real m_mParam;
00124     const real m_pParam;
00125     const real m_qParam;
00126
00127 public:
00128     // No default constructor allowed, must specify parameters
00129     RasheedLarsenMetric() = delete;
00130
00131     // Constructor setting parameter a
00132     RasheedLarsenMetric(real mParam, real aParam, real pParam, real qParam, bool rLogScale = false);
00133
00134     // The override of the basic metric getter functions
00135     TwoIndex getMetric_dd(const Point &p) const final;
00136     TwoIndex getMetric_uu(const Point &p) const final;
00137
00138     // The override of the description string getter
00139     std::string getFullDescriptionStr() const final;
00140 };

```

```

00141
00142 // Johanssen black hole metric (implementation by Seppe Staelens)
00143 class JohanssenMetric final : public SphericalHorizonMetric
00144 {
00145 private:
00146     // Johanssen up to first order in deviation function is specified by five parameters (if M=1)
00147     const real m_aParam;
00148     const real m_alpha13Param;
00149     const real m_alpha22Param;
00150     const real m_alpha52Param;
00151     const real m_eps3Param;
00152
00153 public:
00154     // No default constructor allowed, must specify parameters
00155     JohanssenMetric() = delete;
00156
00157     // Constructor setting parameter a
00158     JohanssenMetric(real aParam, real alpha13Param, real alpha22Param, real alpha52Param, real
eps3Param, bool rLogScale = false);
00159
00160     // The override of the basic metric getter functions
00161     TwoIndex getMetric_dd(const Point &p) const final;
00162     TwoIndex getMetric_uu(const Point &p) const final;
00163
00164     // The override of the description string getter
00165     std::string getFullDescriptionStr() const final;
00166 };
00167
00168 // Manko-Novikov metric (with angular momentum and M3 parameter turned on) (implementation by Seppe
Staelens)
00169 class MankoNovikovMetric final : public SphericalHorizonMetric
00170 {
00171 private:
00172     // Manko-Novikov metric with only alpha3 as symmetry breaking parameter
00173     const real m_aParam;
00174     const real m_alpha3Param;
00175
00176     // These are convenient derived quantities from a
00177     const real m_alphaParam;
00178     const real m_kParam;
00179
00180 public:
00181     // No default constructor allowed, must specify parameters
00182     MankoNovikovMetric() = delete;
00183
00184     // Constructor setting parameter a and alpha3
00185     MankoNovikovMetric(real aParam, real alpha3Param, bool rLogScale = false);
00186
00187     // The override of the basic metric getter functions
00188     TwoIndex getMetric_dd(const Point &p) const final;
00189     TwoIndex getMetric_uu(const Point &p) const final;
00190
00191     // The override of the description string getter
00192     std::string getFullDescriptionStr() const final;
00193 };
00194
00195 // The Kerr metric in Kerr-Schild coordinates (normalized so that M = 1)
00196 class KerrSchildMetric final : public SphericalHorizonMetric
00197 {
00198 private:
00199     // Rotation parameter for Kerr
00200     // Note that this should be between -1 and 1 since M=1
00201     const real m_aParam;
00202
00203 public:
00204     // No default constructor allowed, must specify a
00205     KerrSchildMetric() = delete;
00206
00207     // Constructor setting parameter a
00208     KerrSchildMetric(real aParam, bool rLogScale = false);
00209
00210     // The override of the basic metric getter functions
00211     TwoIndex getMetric_dd(const Point &p) const final;
00212     TwoIndex getMetric_uu(const Point &p) const final;
00213
00214     // The override of the description string getter
00215     std::string getFullDescriptionStr() const final;
00216 };
00217
00218 // Abstract base class for a metric with an arbitrary number of singularities (of arbitrary
codimension)
00219 class SingularityMetric : public Metric
00220 {
00221 public:
00222     // Constructor that initializes singularities
00223     SingularityMetric(std::vector<Singularity> thesings, bool rLogScale);
00224

```

```

00225 // Getter functions for the two member variables
00226 std::vector<Singularity> getSinguliarities() const;
00227
00228 protected:
00229 // All singularities of the metric
00230 const std::vector<Singularity> m_AllSinguliarities;
00231 };
00232
00233 // Ring fuzzball (implementation Lies Van Dael)
00234 class ST3CrMetric final : public SingularityMetric
00235 {
00236 public:
00237 // Constructor which will be called to initialize all parameters of the metric
00238 ST3CrMetric(real P, real q0, real lambda, bool rlogscale = false);
00239
00240 // The basic getter functions
00241 TwoIndex getMetric_dd(const Point &p) const final;
00242 TwoIndex getMetric_uu(const Point &p) const final;
00243
00244 // The description string getter
00245 std::string getFullDescriptionStr() const final;
00246
00247 private:
00248 const real m_P;
00249 const real m_q0;
00250 const real m_lambda;
00251
00252 real get_omega(real r, real theta, real l) const;
00253 real f_phi(real phi, real r, real theta, real l, real R) const;
00254 real f_om_phi(real phi, real r, real theta, real l, real R) const;
00255 };
00256
00257 // Boson star with solitonic potential (sigma = 0.06, phi_c = 0.044) (implementation by Seppe
    Staelens)
00258 class BosonStarMetric final : public Metric
00259 {
00260 public:
00261 // Simple (default) constructor is all that is needed
00262 BosonStarMetric(bool rLogScale = false);
00263 // The override of the basic metric getter functions
00264 TwoIndex getMetric_dd(const Point &p) const final;
00265 TwoIndex getMetric_uu(const Point &p) const final;
00266 // The override of the description string getter
00267 std::string getFullDescriptionStr() const final;
00268
00269 protected:
00270 // The spline interpolator for Phi
00271 tk::spline m_PhiSpline;
00272 // The spline interpolator for m
00273 tk::spline m_mSpline;
00274 };
00275
00277 // Declare your new Metric class here, publically inheriting from the base class Metric
00278 // (or SphericalHorizonMetric if your Metric has a horizon, or SingularityMetric if your Metric has
    other, arbitrary singularities)
00279 // Give definitions (implementation) of these functions in Metric.cpp (or other source code file)
00280 // Don't forget to set m_Symmetries appropriately (in the constructor),
00281 // if your metric has any symmetry (e.g. stationarity, axisymmetry)!
00282 // Sample code:
00283 /*
00284 class MyMetric final : public Metric // good practice to make the class final unless descendant
    classes are possible
00285 {
00286 public:
00287 // Constructor which will be called to initialize all parameters of the metric
00288 MyMetric(args...);
00289
00290 // The basic getter functions
00291 // These MUST be implemented
00292 TwoIndex getMetric_dd(const Point& p) const final;
00293 TwoIndex getMetric_uu(const Point& p) const final;
00294
00295 // The description string getter
00296 // This is optional (but recommended) to implement; if not implemented,
00297 // the base class Metric::getFullDescriptionStr() will be called instead
00298 std::string getFullDescriptionStr() const final;
00299
00300 private:
00301 // good practice to have all const params (initialized in the constructor)
00302 // since the metric cannot change after initialization
00303 // const params...;
00304
00305 };
00306 */
00308
00309 #endif

```

## 7.33 FOORT/src/Spline.h File Reference

```
#include <cstdio>
#include <cassert>
#include <cmath>
#include <vector>
#include <algorithm>
```

### Namespaces

- namespace `tk`
- namespace `tk::internal`

## 7.34 Spline.h

[Go to the documentation of this file.](#)

```
00001 /*
00002  * spline.h
00003  *
00004  * simple cubic spline interpolation library without external
00005  * dependencies
00006  *
00007  * -----
00008  * Copyright (C) 2011, 2014, 2016, 2021 Tino Kluge (ttk448 at gmail.com)
00009  *
00010  * This program is free software; you can redistribute it and/or
00011  * modify it under the terms of the GNU General Public License
00012  * as published by the Free Software Foundation; either version 2
00013  * of the license, or (at your option) any later version.
00014  *
00015  * This program is distributed in the hope that it will be useful,
00016  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00017  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00018  * GNU General Public License for more details.
00019  *
00020  * You should have received a copy of the GNU General Public License
00021  * along with this program. If not, see <http://www.gnu.org/licenses/>.
00022  * -----
00023  *
00024  */
00025
00026 #ifndef TK_SPLINE_H
00027 #define TK_SPLINE_H
00028
00029 #include <cstdio>
00030 #include <cassert>
00031 #include <cmath>
00032 #include <vector>
00033 #include <algorithm>
00034 #ifdef HAVE_SSTREAM
00035 #include <sstream>
00036 #include <string>
00037 #endif // HAVE_SSTREAM
00038
00039 // not ideal but disable unused-function warnings
00040 // (we get them because we have implementations in the header file,
00041 // and this is because we want to be able to quickly separate them
00042 // into a cpp file if necessary)
00043 #pragma GCC diagnostic push
00044 #pragma GCC diagnostic ignored "-Wunused-function"
00045
00046 // unnamed namespace only because the implementation is in this
00047 // header file and we don't want to export symbols to the obj files
00048 namespace
00049 {
00050     namespace tk
00051     {
00052         // spline interpolation
00053         class spline
```

```

00056     {
00057     public:
00058         // spline types
00059         enum spline_type
00060         {
00061             linear = 10,          // linear interpolation
00062             cspline = 30,         // cubic splines (classical C^2)
00063             cspline_hermite = 31 // cubic hermite splines (local, only C^1)
00064         };
00065
00066         // boundary condition type for the spline end-points
00067         enum bd_type
00068         {
00069             first_deriv = 1,
00070             second_deriv = 2
00071         };
00072
00073     protected:
00074         std::vector<double> m_x, m_y; // x,y coordinates of points
00075         // interpolation parameters
00076         // f(x) = a_i + b_i*(x-x_i) + c_i*(x-x_i)^2 + d_i*(x-x_i)^3
00077         // where a_i = y_i, or else it won't go through grid points
00078         std::vector<double> m_b, m_c, m_d; // spline coefficients
00079         double m_c0;                     // for left extrapolation
00080         spline_type m_type;
00081         bd_type m_left, m_right;
00082         double m_left_value, m_right_value;
00083         bool m_made_monotonic;
00084         void set_coeffs_from_b();          // calculate c_i, d_i from b_i
00085         size_t find_closest(double x) const; // closest idx so that m_x[idx]<=x
00086
00087     public:
00088         // default constructor: set boundary condition to be zero curvature
00089         // at both ends, i.e. natural splines
00090         spline() : m_type(cspline),
00091                 m_left(second_deriv), m_right(second_deriv),
00092                 m_left_value(0.0), m_right_value(0.0), m_made_monotonic(false)
00093         {
00094             ;
00095         }
00096         spline(const std::vector<double> &X, const std::vector<double> &Y,
00097               spline_type type = cspline,
00098               bool make_monotonic = false,
00099               bd_type left = second_deriv, double left_value = 0.0,
00100               bd_type right = second_deriv, double right_value = 0.0) : m_type(type),
00101                               m_left(left),
00102                               m_left_value(left_value),
00103                               m_right(right),
00104                               m_right_value(right_value),
00105                               m_made_monotonic(false)
00106         {
00107             // false correct here: make_monotonic() sets it
00108             {
00109                 this->set_points(X, Y, m_type);
00110                 if (make_monotonic)
00111                 {
00112                     this->make_monotonic();
00113                 }
00114             }
00115
00116             // modify boundary conditions: if called it must be before set_points()
00117             void set_boundary(bd_type left, double left_value,
00118                             bd_type right, double right_value);
00119
00120             // set all data points (cubic_spline=false means linear interpolation)
00121             void set_points(const std::vector<double> &x,
00122                             const std::vector<double> &y,
00123                             spline_type type = cspline);
00124
00125             // adjust coefficients so that the spline becomes piecewise monotonic
00126             // where possible
00127             // this is done by adjusting slopes at grid points by a non-negative
00128             // factor and this will break C^2
00129             // this can also break boundary conditions if adjustments need to
00130             // be made at the boundary points
00131             // returns false if no adjustments have been made, true otherwise
00132             bool make_monotonic();
00133
00134             // evaluates the spline at point x
00135             double operator()(double x) const;
00136             double deriv(int order, double x) const;
00137
00138             // returns the input data points
00139             std::vector<double> get_x() const { return m_x; }
00140             std::vector<double> get_y() const { return m_y; }
00141             double get_x_min() const
00142             {
00143                 assert(!m_x.empty());

```

```

00140         return m_x.front();
00141     }
00142     double get_x_max() const
00143     {
00144         assert(!m_x.empty());
00145         return m_x.back();
00146     }
00147
00148 #ifndef HAVE_SSTREAM
00149     // spline info string, i.e. spline type, boundary conditions etc.
00150     std::string info() const;
00151 #endif // HAVE_SSTREAM
00152 };
00153
00154 namespace internal
00155 {
00156
00157     // band matrix solver
00158     class band_matrix
00159     {
00160     private:
00161         std::vector<std::vector<double>> m_upper; // upper band
00162         std::vector<std::vector<double>> m_lower; // lower band
00163     public:
00164         band_matrix() {}; // constructor
00165         band_matrix(int dim, int n_u, int n_l); // constructor
00166         ~band_matrix() {}; // destructor
00167         void resize(int dim, int n_u, int n_l); // init with dim,n_u,n_l
00168         int dim() const; // matrix dimension
00169         int num_upper() const
00170         {
00171             return (int)m_upper.size() - 1;
00172         }
00173         int num_lower() const
00174         {
00175             return (int)m_lower.size() - 1;
00176         }
00177         // access operator
00178         double &operator()(int i, int j); // write
00179         double operator()(int i, int j) const; // read
00180         // we can store an additional diagonal (in m_lower)
00181         double &saved_diag(int i);
00182         double saved_diag(int i) const;
00183         void lu_decompose();
00184         std::vector<double> r_solve(const std::vector<double> &b) const;
00185         std::vector<double> l_solve(const std::vector<double> &b) const;
00186         std::vector<double> lu_solve(const std::vector<double> &b,
00187                                     bool is_lu_decomposed = false);
00188     };
00189
00190 } // namespace internal
00191
00192 // -----
00193 // implementation part, which could be separated into a cpp file
00194 // -----
00195
00196 // spline implementation
00197 // -----
00198
00199 void spline::set_boundary(spline::bd_type left, double left_value,
00200                          spline::bd_type right, double right_value)
00201 {
00202     assert(m_x.size() == 0); // set_points() must not have happened yet
00203     m_left = left;
00204     m_right = right;
00205     m_left_value = left_value;
00206     m_right_value = right_value;
00207 }
00208
00209 void spline::set_coeffs_from_b()
00210 {
00211     assert(m_x.size() == m_y.size());
00212     assert(m_x.size() == m_b.size());
00213     assert(m_x.size() > 2);
00214     size_t n = m_b.size();
00215     if (m_c.size() != n)
00216         m_c.resize(n);
00217     if (m_d.size() != n)
00218         m_d.resize(n);
00219
00220     for (size_t i = 0; i < n - 1; i++)
00221     {
00222         const double h = m_x[i + 1] - m_x[i];
00223         // from continuity and differentiability condition
00224         m_c[i] = (3.0 * (m_y[i + 1] - m_y[i]) / h - (2.0 * m_b[i] + m_b[i + 1])) / h;
00225         // from differentiability condition
00226         m_d[i] = ((m_b[i + 1] - m_b[i]) / (3.0 * h) - 2.0 / 3.0 * m_c[i]) / h;

```

```

00227     }
00228
00229     // for left extrapolation coefficients
00230     m_c0 = (m_left == first_deriv) ? 0.0 : m_c[0];
00231 }
00232
00233 void spline::set_points(const std::vector<double> &x,
00234                       const std::vector<double> &y,
00235                       spline_type type)
00236 {
00237     assert(x.size() == y.size());
00238     assert(x.size() > 2);
00239     m_type = type;
00240     m_made_monotonic = false;
00241     m_x = x;
00242     m_y = y;
00243     int n = (int)x.size();
00244     // check strict monotonicity of input vector x
00245     for (int i = 0; i < n - 1; i++)
00246     {
00247         assert(m_x[i] < m_x[i + 1]);
00248     }
00249
00250     if (type == linear)
00251     {
00252         // linear interpolation
00253         m_d.resize(n);
00254         m_c.resize(n);
00255         m_b.resize(n);
00256         for (int i = 0; i < n - 1; i++)
00257         {
00258             m_d[i] = 0.0;
00259             m_c[i] = 0.0;
00260             m_b[i] = (m_y[i + 1] - m_y[i]) / (m_x[i + 1] - m_x[i]);
00261         }
00262         // ignore boundary conditions, set slope equal to the last segment
00263         m_b[n - 1] = m_b[n - 2];
00264         m_c[n - 1] = 0.0;
00265         m_d[n - 1] = 0.0;
00266     }
00267     else if (type == cspline)
00268     {
00269         // classical cubic splines which are C^2 (twice cont differentiable)
00270         // this requires solving an equation system
00271
00272         // setting up the matrix and right hand side of the equation system
00273         // for the parameters b[]
00274         internal::band_matrix A(n, 1, 1);
00275         std::vector<double> rhs(n);
00276         for (int i = 1; i < n - 1; i++)
00277         {
00278             A(i, i - 1) = 1.0 / 3.0 * (x[i] - x[i - 1]);
00279             A(i, i) = 2.0 / 3.0 * (x[i + 1] - x[i - 1]);
00280             A(i, i + 1) = 1.0 / 3.0 * (x[i + 1] - x[i]);
00281             rhs[i] = (y[i + 1] - y[i]) / (x[i + 1] - x[i]) - (y[i] - y[i - 1]) / (x[i] - x[i -
00282 1]);
00283         }
00284         // boundary conditions
00285         if (m_left == spline::second_deriv)
00286         {
00287             // 2*c[0] = f''
00288             A(0, 0) = 2.0;
00289             A(0, 1) = 0.0;
00290             rhs[0] = m_left_value;
00291         }
00292         else if (m_left == spline::first_deriv)
00293         {
00294             // b[0] = f', needs to be re-expressed in terms of c:
00295             // (2c[0]+c[1]) (x[1]-x[0]) = 3 ((y[1]-y[0])/(x[1]-x[0]) - f')
00296             A(0, 0) = 2.0 * (x[1] - x[0]);
00297             A(0, 1) = 1.0 * (x[1] - x[0]);
00298             rhs[0] = 3.0 * ((y[1] - y[0]) / (x[1] - x[0]) - m_left_value);
00299         }
00300         else
00301         {
00302             assert(false);
00303         }
00304         if (m_right == spline::second_deriv)
00305         {
00306             // 2*c[n-1] = f''
00307             A(n - 1, n - 1) = 2.0;
00308             A(n - 1, n - 2) = 0.0;
00309             rhs[n - 1] = m_right_value;
00310         }
00311         else if (m_right == spline::first_deriv)
00312         {
00313             // b[n-1] = f', needs to be re-expressed in terms of c:

```

```

00313         // (c[n-2]+2c[n-1])(x[n-1]-x[n-2])
00314         // = 3 (f' - (y[n-1]-y[n-2])/(x[n-1]-x[n-2]))
00315         A(n-1, n-1) = 2.0 * (x[n-1] - x[n-2]);
00316         A(n-1, n-2) = 1.0 * (x[n-1] - x[n-2]);
00317         rhs[n-1] = 3.0 * (m_right_value - (y[n-1] - y[n-2]) / (x[n-1] - x[n-
2]));
00318     }
00319     else
00320     {
00321         assert(false);
00322     }
00323
00324     // solve the equation system to obtain the parameters c[]
00325     m_c = A.lu_solve(rhs);
00326
00327     // calculate parameters b[] and d[] based on c[]
00328     m_d.resize(n);
00329     m_b.resize(n);
00330     for (int i = 0; i < n-1; i++)
00331     {
00332         m_d[i] = 1.0 / 3.0 * (m_c[i+1] - m_c[i]) / (x[i+1] - x[i]);
00333         m_b[i] = (y[i+1] - y[i]) / (x[i+1] - x[i]) - 1.0 / 3.0 * (2.0 * m_c[i] + m_c[i
+ 1]) * (x[i+1] - x[i]);
00334     }
00335     // for the right extrapolation coefficients (zero cubic term)
00336     // f_{n-1}(x) = y_{n-1} + b*(x-x_{n-1}) + c*(x-x_{n-1})^2
00337     double h = x[n-1] - x[n-2];
00338     // m_c[n-1] is determined by the boundary condition
00339     m_d[n-1] = 0.0;
00340     m_b[n-1] = 3.0 * m_d[n-2] * h * h + 2.0 * m_c[n-2] * h + m_b[n-2]; // =
f'_{n-2}(x_{n-1})
00341     if (m_right == first_deriv)
00342         m_c[n-1] = 0.0; // force linear extrapolation
00343     }
00344     else if (type == cspline_hermite)
00345     {
00346         // hermite cubic splines which are C^1 (cont. differentiable)
00347         // and derivatives are specified on each grid point
00348         // (here we use 3-point finite differences)
00349         m_b.resize(n);
00350         m_c.resize(n);
00351         m_d.resize(n);
00352         // set b to match 1st order derivative finite difference
00353         for (int i = 1; i < n-1; i++)
00354         {
00355             const double h = m_x[i+1] - m_x[i];
00356             const double hl = m_x[i] - m_x[i-1];
00357             m_b[i] = -h / (hl * (hl + h)) * m_y[i-1] + (h - hl) / (hl * h) * m_y[i] + hl /
(h * (hl + h)) * m_y[i+1];
00358         }
00359         // boundary conditions determine b[0] and b[n-1]
00360         if (m_left == first_deriv)
00361         {
00362             m_b[0] = m_left_value;
00363         }
00364         else if (m_left == second_deriv)
00365         {
00366             const double h = m_x[1] - m_x[0];
00367             m_b[0] = 0.5 * (-m_b[1] - 0.5 * m_left_value * h + 3.0 * (m_y[1] - m_y[0]) / h);
00368         }
00369         else
00370         {
00371             assert(false);
00372         }
00373         if (m_right == first_deriv)
00374         {
00375             m_b[n-1] = m_right_value;
00376             m_c[n-1] = 0.0;
00377         }
00378         else if (m_right == second_deriv)
00379         {
00380             const double h = m_x[n-1] - m_x[n-2];
00381             m_b[n-1] = 0.5 * (-m_b[n-2] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] -
m_y[n-2]) / h);
00382             m_c[n-1] = 0.5 * m_right_value;
00383         }
00384         else
00385         {
00386             assert(false);
00387         }
00388         m_d[n-1] = 0.0;
00389
00390         // parameters c and d are determined by continuity and differentiability
00391         set_coeffs_from_b();
00392     }
00393     else
00394     {

```



```

00395         assert(false);
00396     }
00397
00398     // for left extrapolation coefficients
00399     m_c0 = (m_left == first_deriv) ? 0.0 : m_c[0];
00400 }
00401
00402 bool spline::make_monotonic()
00403 {
00404     assert(m_x.size() == m_y.size());
00405     assert(m_x.size() == m_b.size());
00406     assert(m_x.size() > 2);
00407     bool modified = false;
00408     const int n = (int)m_x.size();
00409     // make sure: input data monotonic increasing --> b_i >= 0
00410     //               input data monotonic decreasing --> b_i <= 0
00411     for (int i = 0; i < n; i++)
00412     {
00413         int ipl = std::max(i - 1, 0);
00414         int ipl = std::min(i + 1, n - 1);
00415         if ((m_y[ipl] <= m_y[i]) && (m_y[i] <= m_y[ipl]) && m_b[i] < 0.0) ||
00416             ((m_y[ipl] >= m_y[i]) && (m_y[i] >= m_y[ipl]) && m_b[i] > 0.0))
00417         {
00418             modified = true;
00419             m_b[i] = 0.0;
00420         }
00421     }
00422     // if input data is monotonic (b[i], b[i+1], avg have all the same sign)
00423     // ensure a sufficient criteria for monotonicity is satisfied:
00424     //      sqrt(b[i]^2+b[i+1]^2) <= 3 |avg|, with avg=(y[i+1]-y[i])/h,
00425     for (int i = 0; i < n - 1; i++)
00426     {
00427         double h = m_x[i + 1] - m_x[i];
00428         double avg = (m_y[i + 1] - m_y[i]) / h;
00429         if (avg == 0.0 && (m_b[i] != 0.0 || m_b[i + 1] != 0.0))
00430         {
00431             modified = true;
00432             m_b[i] = 0.0;
00433             m_b[i + 1] = 0.0;
00434         }
00435         else if ((m_b[i] >= 0.0 && m_b[i + 1] >= 0.0 && avg > 0.0) ||
00436                 (m_b[i] <= 0.0 && m_b[i + 1] <= 0.0 && avg < 0.0))
00437         {
00438             // input data is monotonic
00439             double r = sqrt(m_b[i] * m_b[i] + m_b[i + 1] * m_b[i + 1]) / std::fabs(avg);
00440             if (r > 3.0)
00441             {
00442                 // sufficient criteria for monotonicity: r <= 3
00443                 // adjust b[i] and b[i+1]
00444                 modified = true;
00445                 m_b[i] *= (3.0 / r);
00446                 m_b[i + 1] *= (3.0 / r);
00447             }
00448         }
00449     }
00450
00451     if (modified == true)
00452     {
00453         set_coeffs_from_b();
00454         m_made_monotonic = true;
00455     }
00456
00457     return modified;
00458 }
00459
00460 // return the closest idx so that m_x[idx] <= x (return 0 if x < m_x[0])
00461 size_t spline::find_closest(double x) const
00462 {
00463     std::vector<double>::const_iterator it;
00464     it = std::upper_bound(m_x.begin(), m_x.end(), x); // *it > x
00465     size_t idx = std::max(int(it - m_x.begin()) - 1, 0); // m_x[idx] <= x
00466     return idx;
00467 }
00468
00469 double spline::operator()(double x) const
00470 {
00471     // polynomial evaluation using Horner's scheme
00472     // TODO: consider more numerically accurate algorithms, e.g.:
00473     //   - Clenshaw
00474     //   - Even-Odd method by A.C.R. Newbery
00475     //   - Compensated Horner Scheme
00476     size_t n = m_x.size();
00477     size_t idx = find_closest(x);
00478
00479     double h = x - m_x[idx];
00480     double interpol;
00481     if (x < m_x[0])

```

```

00482     {
00483         // extrapolation to the left
00484         interpol = (m_c0 * h + m_b[0]) * h + m_y[0];
00485     }
00486     else if (x > m_x[n - 1])
00487     {
00488         // extrapolation to the right
00489         interpol = (m_c[n - 1] * h + m_b[n - 1]) * h + m_y[n - 1];
00490     }
00491     else
00492     {
00493         // interpolation
00494         interpol = ((m_d[idx] * h + m_c[idx]) * h + m_b[idx]) * h + m_y[idx];
00495     }
00496     return interpol;
00497 }
00498
00499 double spline::deriv(int order, double x) const
00500 {
00501     assert(order > 0);
00502     size_t n = m_x.size();
00503     size_t idx = find_closest(x);
00504
00505     double h = x - m_x[idx];
00506     double interpol;
00507     if (x < m_x[0])
00508     {
00509         // extrapolation to the left
00510         switch (order)
00511         {
00512             case 1:
00513                 interpol = 2.0 * m_c0 * h + m_b[0];
00514                 break;
00515             case 2:
00516                 interpol = 2.0 * m_c0;
00517                 break;
00518             default:
00519                 interpol = 0.0;
00520                 break;
00521         }
00522     }
00523     else if (x > m_x[n - 1])
00524     {
00525         // extrapolation to the right
00526         switch (order)
00527         {
00528             case 1:
00529                 interpol = 2.0 * m_c[n - 1] * h + m_b[n - 1];
00530                 break;
00531             case 2:
00532                 interpol = 2.0 * m_c[n - 1];
00533                 break;
00534             default:
00535                 interpol = 0.0;
00536                 break;
00537         }
00538     }
00539     else
00540     {
00541         // interpolation
00542         switch (order)
00543         {
00544             case 1:
00545                 interpol = (3.0 * m_d[idx] * h + 2.0 * m_c[idx]) * h + m_b[idx];
00546                 break;
00547             case 2:
00548                 interpol = 6.0 * m_d[idx] * h + 2.0 * m_c[idx];
00549                 break;
00550             case 3:
00551                 interpol = 6.0 * m_d[idx];
00552                 break;
00553             default:
00554                 interpol = 0.0;
00555                 break;
00556         }
00557     }
00558     return interpol;
00559 }
00560
00561 #ifndef HAVE_SSTREAM
00562 std::string spline::info() const
00563 {
00564     std::stringstream ss;
00565     ss << "type " << m_type << ", left boundary deriv " << m_left << " = ";
00566     ss << m_left_value << ", right boundary deriv " << m_right << " = ";
00567     ss << m_right_value << std::endl;
00568     if (m_made_monotonic)

```

```

00569         {
00570             ss << "(spline has been adjusted for piece-wise monotonicity)";
00571         }
00572         return ss.str();
00573     }
00574 #endif // HAVE_SSTREAM
00575
00576     namespace internal
00577     {
00578
00579         // band_matrix implementation
00580         // -----
00581
00582         band_matrix::band_matrix(int dim, int n_u, int n_l)
00583         {
00584             resize(dim, n_u, n_l);
00585         }
00586         void band_matrix::resize(int dim, int n_u, int n_l)
00587         {
00588             assert(dim > 0);
00589             assert(n_u >= 0);
00590             assert(n_l >= 0);
00591             m_upper.resize(n_u + 1);
00592             m_lower.resize(n_l + 1);
00593             for (size_t i = 0; i < m_upper.size(); i++)
00594             {
00595                 m_upper[i].resize(dim);
00596             }
00597             for (size_t i = 0; i < m_lower.size(); i++)
00598             {
00599                 m_lower[i].resize(dim);
00600             }
00601         }
00602         int band_matrix::dim() const
00603         {
00604             if (m_upper.size() > 0)
00605             {
00606                 return m_upper[0].size();
00607             }
00608             else
00609             {
00610                 return 0;
00611             }
00612         }
00613
00614         // defines the new operator (), so that we can access the elements
00615         // by A(i,j), index going from i=0,...,dim()-1
00616         double &band_matrix::operator()(int i, int j)
00617         {
00618             int k = j - i; // what band is the entry
00619             assert((i >= 0) && (i < dim()) && (j >= 0) && (j < dim()));
00620             assert((-num_lower() <= k) && (k <= num_upper()));
00621             // k=0 -> diagonal, k<0 lower left part, k>0 upper right part
00622             if (k >= 0)
00623                 return m_upper[k][i];
00624             else
00625                 return m_lower[-k][i];
00626         }
00627         double band_matrix::operator()(int i, int j) const
00628         {
00629             int k = j - i; // what band is the entry
00630             assert((i >= 0) && (i < dim()) && (j >= 0) && (j < dim()));
00631             assert((-num_lower() <= k) && (k <= num_upper()));
00632             // k=0 -> diagonal, k<0 lower left part, k>0 upper right part
00633             if (k >= 0)
00634                 return m_upper[k][i];
00635             else
00636                 return m_lower[-k][i];
00637         }
00638         // second diag (used in LU decomposition), saved in m_lower
00639         double band_matrix::saved_diag(int i) const
00640         {
00641             assert((i >= 0) && (i < dim()));
00642             return m_lower[0][i];
00643         }
00644         double &band_matrix::saved_diag(int i)
00645         {
00646             assert((i >= 0) && (i < dim()));
00647             return m_lower[0][i];
00648         }
00649
00650         // LR-Decomposition of a band matrix
00651         void band_matrix::lu_decompose()
00652         {
00653             int i_max, j_max;
00654             int j_min;
00655             double x;

```

```

00656
00657 // preconditioning
00658 // normalize column i so that a_ii=1
00659 for (int i = 0; i < this->dim(); i++)
00660 {
00661     assert(this->operator()(i, i) != 0.0);
00662     this->saved_diag(i) = 1.0 / this->operator()(i, i);
00663     j_min = std::max(0, i - this->num_lower());
00664     j_max = std::min(this->dim() - 1, i + this->num_upper());
00665     for (int j = j_min; j <= j_max; j++)
00666     {
00667         this->operator()(i, j) *= this->saved_diag(i);
00668     }
00669     this->operator()(i, i) = 1.0; // prevents rounding errors
00670 }
00671
00672 // Gauss LR-Decomposition
00673 for (int k = 0; k < this->dim(); k++)
00674 {
00675     i_max = std::min(this->dim() - 1, k + this->num_lower()); // num_lower not a
mistake!
00676     for (int i = k + 1; i <= i_max; i++)
00677     {
00678         assert(this->operator()(k, k) != 0.0);
00679         x = -this->operator()(i, k) / this->operator()(k, k);
00680         this->operator()(i, k) = -x; // assembly part of L
00681         j_max = std::min(this->dim() - 1, k + this->num_upper());
00682         for (int j = k + 1; j <= j_max; j++)
00683         {
00684             // assembly part of R
00685             this->operator()(i, j) = this->operator()(i, j) + x * this->operator()(k,
j);
00686         }
00687     }
00688 }
00689 // solves Ly=b
00690 std::vector<double> band_matrix::l_solve(const std::vector<double> &b) const
00691 {
00692     assert(this->dim() == (int)b.size());
00693     std::vector<double> x(this->dim());
00694     int j_start;
00695     double sum;
00696     for (int i = 0; i < this->dim(); i++)
00697     {
00698         sum = 0;
00699         j_start = std::max(0, i - this->num_lower());
00700         for (int j = j_start; j < i; j++)
00701             sum += this->operator()(i, j) * x[j];
00702         x[i] = (b[i] * this->saved_diag(i)) - sum;
00703     }
00704     return x;
00705 }
00706 // solves Rx=y
00707 std::vector<double> band_matrix::r_solve(const std::vector<double> &b) const
00708 {
00709     assert(this->dim() == (int)b.size());
00710     std::vector<double> x(this->dim());
00711     int j_stop;
00712     double sum;
00713     for (int i = this->dim() - 1; i >= 0; i--)
00714     {
00715         sum = 0;
00716         j_stop = std::min(this->dim() - 1, i + this->num_upper());
00717         for (int j = i + 1; j <= j_stop; j++)
00718             sum += this->operator()(i, j) * x[j];
00719         x[i] = (b[i] - sum) / this->operator()(i, i);
00720     }
00721     return x;
00722 }
00723
00724 std::vector<double> band_matrix::lu_solve(const std::vector<double> &b,
00725 bool is_lu_decomposed)
00726 {
00727     assert(this->dim() == (int)b.size());
00728     std::vector<double> x, y;
00729     if (is_lu_decomposed == false)
00730     {
00731         this->lu_decompose();
00732     }
00733     y = this->l_solve(b);
00734     x = this->r_solve(y);
00735     return x;
00736 }
00737
00738 } // namespace internal
00739
00740

```

```

00741     } // namespace tk
00742
00743 } // namespace
00744
00745 #pragma GCC diagnostic pop
00746
00747 #endif /* TK_SPLINE_H */

```

## 7.35 FOORT/src/Terminations.cpp File Reference

```

#include "Terminations.h"
#include "Geodesic.h"
#include "InputOutput.h"
#include <cmath>

```

### Functions

- [TerminationUniqueVector CreateTerminationVector](#) ([TermBitflag](#) termflags, [Geodesic](#) \*const theGeodesic)  
*Termination helper function.*

### 7.35.1 Function Documentation

#### 7.35.1.1 CreateTerminationVector()

```

TerminationUniqueVector CreateTerminationVector (
    TermBitflag termflags,
    Geodesic *const theGeodesic)

```

[Termination](#) helper function.

## 7.36 FOORT/src/Terminations.h File Reference

```

#include "Geometry.h"
#include <stdint>
#include <memory>
#include <vector>
#include <utility>

```

### Classes

- class [Termination](#)
- class [HorizonTermination](#)
- class [BoundarySphereTermination](#)
- class [TimeOutTermination](#)
- class [ThetaSingularityTermination](#)
- class [NaNTermination](#)
- class [GeneralSingularityTermination](#)
- struct [TerminationOptions](#)
- struct [HorizonTermOptions](#)
- struct [BoundarySphereTermOptions](#)
- struct [TimeOutTermOptions](#)
- struct [ThetaSingularityTermOptions](#)
- struct [NaNTermOptions](#)
- struct [GeneralSingularityTermOptions](#)

## Typedefs

- using [TermBitflag](#) = std::uint16\_t
- using [TerminationUniqueVector](#) = std::vector<std::unique\_ptr<[Termination](#)>>>

## Enumerations

- enum class [Term](#) {  
[Uninitialized](#) = -1 , [Continue](#) = 0 , [Horizon](#) , [BoundarySphere](#) ,  
[TimeOut](#) , [ThetaSingularity](#) , [NaN](#) , [GeneralSingularity](#) ,  
[Maxterms](#) }

## Functions

- [TerminationUniqueVector CreateTerminationVector](#) ([TermBitflag](#) termflags, [Geodesic](#) \*const theGeodesic)  
[Termination](#) helper function.

## Variables

- constexpr [TermBitflag Term\\_None](#) {0b0000'0000'0000'0000}
- constexpr [TermBitflag Term\\_BoundarySphere](#) {0b0000'0000'0000'0001}
- constexpr [TermBitflag Term\\_TimeOut](#) {0b0000'0000'0000'0010}
- constexpr [TermBitflag Term\\_Horizon](#) {0b0000'0000'0000'0100}
- constexpr [TermBitflag Term\\_ThetaSingularity](#) {0b0000'0000'0000'1000}
- constexpr [TermBitflag Term\\_NaN](#) {0b0000'0000'0001'0000}
- constexpr [TermBitflag Term\\_GeneralSingularity](#) {0b0000'0000'0010'0000}

## 7.36.1 Typedef Documentation

### 7.36.1.1 TermBitflag

```
using TermBitflag = std::uint16_t
```

### 7.36.1.2 TerminationUniqueVector

```
using TerminationUniqueVector = std::vector<std::unique_ptr<Termination>>>
```

## 7.36.2 Enumeration Type Documentation

### 7.36.2.1 Term

```
enum class Term [strong]
```

#### Enumerator

Uninitialized	
Continue	
Horizon	
BoundarySphere	
TimeOut	
ThetaSingularity	
NaN	
GeneralSingularity	
Maxterms	

## 7.36.3 Function Documentation

### 7.36.3.1 CreateTerminationVector()

```
TerminationUniqueVector CreateTerminationVector (  
    TermBitflag termflags,  
    Geodesic *const theGeodesic)
```

Termination helper function.

## 7.36.4 Variable Documentation

### 7.36.4.1 Term\_BoundarySphere

```
TermBitflag Term_BoundarySphere {0b0000'0000'0000'0001} [constexpr]
```

### 7.36.4.2 Term\_GeneralSingularity

```
TermBitflag Term_GeneralSingularity {0b0000'0000'0010'0000} [constexpr]
```

### 7.36.4.3 Term\_Horizon

```
TermBitflag Term_Horizon {0b0000'0000'0000'0100} [constexpr]
```

### 7.36.4.4 Term\_NaN

```
TermBitflag Term_NaN {0b0000'0000'0001'0000} [constexpr]
```

### 7.36.4.5 Term\_None

```
TermBitflag Term_None {0b0000'0000'0000'0000} [constexpr]
```

### 7.36.4.6 Term\_ThetaSingularity

```
TermBitflag Term_ThetaSingularity {0b0000'0000'0000'1000} [constexpr]
```

### 7.36.4.7 Term\_TimeOut

```
TermBitflag Term_TimeOut {0b0000'0000'0000'0010} [constexpr]
```

## 7.37 Terminations.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _FOORT_TERMINATIONS_H
00002 #define _FOORT_TERMINATIONS_H
00003
00009
00010 #include "Geometry.h" // for basic tensor objects
00011
00012 #include <cstdint> // for std::uint16_t
00013 #include <memory> // for std::unique_ptr
00014 #include <vector> // for std::vector
00015 #include <utility> // for std::pair
00016
00017 // Forward declaration of Geodesic class needed here, since Diagnostics are passed a pointer to their
owner Geodesic
00018 // (note "Geodesic.h" is NOT included to avoid header loop, and we do not need Geodesic member
functions here!)
00019 class Geodesic;
00020
00023 // Used for constructing vector of Terminations
00024 // Note that this means every Termination is either "on" or "off";
00025 // it is not possible to have a Termination "on" more than once
00026 using TermBitflag = std::uint16_t;
00027
00028 // Define a bitflag per existing Termination
00029 constexpr TermBitflag Term_None{0b0000'0000'0000'0000};
00030 constexpr TermBitflag Term_BoundarySphere{0b0000'0000'0000'0001};
00031 constexpr TermBitflag Term_TimeOut{0b0000'0000'0000'0010};
00032 constexpr TermBitflag Term_Horizon{0b0000'0000'0000'0100};
00033 constexpr TermBitflag Term_ThetaSingularity{0b0000'0000'0000'1000};
00034 constexpr TermBitflag Term_NaN{0b0000'0000'0001'0000};
00035 constexpr TermBitflag Term_GeneralSingularity{0b0000'0000'0010'0000};
00036
00038 // Add a TermBitflag for your new Termination. Make sure you use a bitflag that has not been used
before!
00039 // Sample code:
00040 /*
00041 constexpr TermBitflag Term_MyTerm { 0b0000'0000'0000'1000 };
00042 */
00044
00047
00048 // Possible termination conditions that can be set by Terminations
00049 enum class Term
00050 {
00051     Uninitialized = -1, // Geodesic has not been properly initialized yet with initial
position/velocity
00052     Continue = 0, // All is right, continue integrating geodesic
00053     Horizon, // STOP, encountered horizon (set by HorizonTermination)
00054     BoundarySphere, // STOP, encountered boundary sphere (set by BoundarySphereTermination)
00055     TimeOut, // STOP, taken too many steps (set by TimeOutTermination)
00056     ThetaSingularity, // STOP, too close to polar coordinate singularity (theta = 0 or theta = pi/2)
00057     NaN, // STOP, NaN encountered in geodesic position or velocity
00058     GeneralSingularity, // STOP, singularity encountered (of any codimension)
00059
00061 // Add a new Termination condition that your new Termination can set
00062 // Sample code:
00063 /*
00064 MyTermCond, // STOP, encountered (...)
00065 */
00067
00068 Maxterms // Number of termination conditions that exist
00069 };
00070
00073
00074 // Abstract base class for all Terminations
00075 class Termination
00076 {
00077 public:
00078     // Constructor must initialize the pointer to its owner Geodesic
00079     Termination() = delete;
00080     Termination(Geodesic *const theGeodesic) : m_OwnerGeodesic{theGeodesic}
00081     {
00082     }
00083
00084     // Resets Termination object. This is called when the owner Geodesic is reset in order to start
integrating
00085     // a new geodesic.
00086     // The base class implementation only resets m_StepsSinceUpdated
00087     // Descendants can override this if they need to reset additional internal variables
00088     virtual void Reset();
00089
00090     // virtual destructor to ensure correct destruction of descendants
00091     virtual ~Termination() = default;
00092

```



```

00093 // Function that is called to determine whether Termination wants to
00094 // terminate the Geodesic. Returns Term::Continue if no termination wanted,
00095 // otherwise it returns the appropriate Term condition
00096 virtual Term CheckTermination() = 0;
00097
00098 // This returns the full description of the Termination
00099 virtual std::string getFullDescriptionStr() const = 0;
00100
00101 protected:
00102 // The geodesic that owns the Termination (a const pointer to the Geodesic)
00103 Geodesic *const m_OwnerGeodesic;
00104
00105 // Helper function to decide if the Termination should indeed update its status, based on
00106 // UpdateNSteps (which is set to 0 if we always update)
00107 bool DecideUpdate(largecounter UpdateNSteps);
00108
00109 // The termination is itself in charge of keeping track of how many steps it has been since it has
00110 // been updated
00111 // The Termination's TerminationOptions struct tells it how many steps it needs to wait between
00112 // updates
00113 largecounter m_StepsSinceUpdated{};
00114 };
00115
00116 // The owner vector of derived Termination classes
00117 using TerminationUniqueVector = std::vector<std::unique_ptr<Termination>>;
00118
00119 // Helper to create a new vector of Termination options, based on the bitflag
00120 TerminationUniqueVector CreateTerminationVector(TermBitflag termflags, Geodesic *const theGeodesic);
00121
00122 // Forward declaration needed before Termination
00123 struct HorizonTermOptions;
00124 // Horizon termination: terminate geodesics if they get too close to the horizon (returns
00125 // Term::Horizon)
00126 class HorizonTermination final : public Termination
00127 {
00128 public:
00129 // Basic constructor only passes on Geodesic pointer to base class constructor
00130 HorizonTermination(Geodesic *const theGeodesic) : Termination(theGeodesic) {}
00131
00132 // Check if we are too close to the horizon
00133 Term CheckTermination() final;
00134
00135 // Description string
00136 std::string getFullDescriptionStr() const final;
00137
00138 // Options (contains horizon radius, if we are using logarithmic r coordinate, and distance
00139 // allowed from the horizon)
00140 static std::unique_ptr<HorizonTermOptions> TermOptions;
00141 };
00142
00143 // Forward declaration needed before Termination
00144 struct BoundarySphereTermOptions;
00145 // The Boundary Sphere: this terminates the geodesic (and returns Term::BoundarySphere) if
00146 // the geodesic reaches outside of the boundary sphere
00147 class BoundarySphereTermination final : public Termination
00148 {
00149 public:
00150 // Basic constructor only passes on Geodesic pointer to base class constructor
00151 BoundarySphereTermination(Geodesic *const theGeodesic) : Termination(theGeodesic) {}
00152
00153 // Check if we have passed the boundary sphere
00154 Term CheckTermination() final;
00155
00156 // Description string
00157 std::string getFullDescriptionStr() const final;
00158
00159 // The options that the BoundarySphereTermination keeps (contains the radius of the boundary
00160 // sphere)
00161 static std::unique_ptr<BoundarySphereTermOptions> TermOptions;
00162 };
00163
00164 // Forward declaration needed before Termination
00165 struct TimeOutTermOptions;
00166 // The Time Out: this terminates the geodesic if too many steps have been
00167 // taken in its integration (and returns Term::TimeOut)
00168 class TimeOutTermination final : public Termination
00169 {
00170 public:
00171 // Basic constructor only passes on Geodesic pointer to base class constructor
00172 TimeOutTermination(Geodesic *const theGeodesic) : Termination(theGeodesic) {}
00173
00174 // This descendant needs to override Reset in order to also reset m_CurNrSteps
00175 void Reset() final;
00176
00177 // Check if we have already taken too many steps
00178 Term CheckTermination() final;

```

```

00177
00178 // Description string
00179 std::string getFullDescriptionStr() const final;
00180
00181 // The options that the TimeOutTermination keeps (contains max number of steps allowed)
00182 static std::unique_ptr<TimeOutTermOptions> TermOptions;
00183
00184 private:
00185 // Keep track of the number of steps that the geodesic has taken so far
00186 largecounter m_CurNrSteps{0};
00187 };
00188
00189 // Forward declaration needed before Termination
00190 struct ThetaSingularityTermOptions;
00191 class ThetaSingularityTermination final : public Termination
00192 {
00193 public:
00194     ThetaSingularityTermination(Geodesic *const theGeodesic) : Termination(theGeodesic) {}
00195
00196     // No override of Reset() necessary
00197
00198     // Check the specific termination condition
00199     Term CheckTermination() final;
00200
00201     // Description string
00202     std::string getFullDescriptionStr() const final;
00203
00204     // The options that the Termination keeps (will probably be a descendant struct instead, which
00205     // specifies any additional options the Termination needs)
00206     static std::unique_ptr<ThetaSingularityTermOptions> TermOptions;
00207 };
00208
00209 // Forward declaration needed before Termination
00210 struct NaNTermOptions;
00211 // NaN termination: terminate geodesics if position or velocity contains a nan
00212 class NaNTermination final : public Termination
00213 {
00214 public:
00215     // Basic constructor only passes on Geodesic pointer to base class constructor
00216     NaNTermination(Geodesic *const theGeodesic) : Termination(theGeodesic) {}
00217
00218     // Check if we are too close to the horizon
00219     Term CheckTermination() final;
00220
00221     // Description string
00222     std::string getFullDescriptionStr() const final;
00223
00224     // Options (contains horizon radius, if we are using logarithmic r coordinate, and distance
00225     // allowed from the horizon)
00226     static std::unique_ptr<NaNTermOptions> TermOptions;
00227 };
00228
00229 // Forward declaration needed before Termination
00230 struct GeneralSingularityTermOptions;
00231 // General singularity: terminate geodesic if it comes too close to one of a given number of
00232 // (arbitrary codimension) singularities
00233 class GeneralSingularityTermination final : public Termination
00234 {
00235 public:
00236     // Basic constructor only passes on Geodesic pointer to base class constructor
00237     GeneralSingularityTermination(Geodesic *const theGeodesic) : Termination(theGeodesic) {}
00238
00239     // Check if we are too close to the horizon
00240     Term CheckTermination() final;
00241
00242     // Description string
00243     std::string getFullDescriptionStr() const final;
00244
00245     // Options (contains horizon radius, if we are using logarithmic r coordinate, and distance
00246     // allowed from the horizon)
00247     static std::unique_ptr<GeneralSingularityTermOptions> TermOptions;
00248
00249 private:
00250     std::string SingularityToString(int singnr) const;
00251 };
00252
00253 // Declare your Termination class here, inheriting from Termination.
00254 // Sample code:
00255 /*
00256 // Forward declaration needed before Termination
00257 struct TerminationOptions; // possibly will instead need to declare descendant options struct
00258 class MyTermination final : public Termination // good practice to make the class final unless
00259     descendant classes are possible
00260 {
00261 public:
00262     // Constructor must at least pass on Geodesic pointer to base class constructor

```

```

00260     MyTermination(Geodesic* const theGeodesic) : Termination(theGeodesic) {}
00261
00262     // Do you need to reset any internal variables specific to MyTermination? If so, override Reset()
    (This is not mandatory)
00263     // Note: make sure to call the base class implementation Termination::Reset()
00264     // from within your implementation of MyTermination::Reset(), so that the base class internal
    variable is also reset!
00265     void Reset() final;
00266
00267     // Check the specific termination condition
00268     Term CheckTermination() final;
00269
00270     // Description string
00271     std::string getFullDescriptionStr() const final;
00272
00273     // The options that the Termination keeps (will probably be a descendant struct instead, which
    specifies
00274     // any additional options the Termination needs)
00275     static std::unique_ptr<TerminationOptions> TermOptions;
00276
00277 private:
00278     // any private member variables that are needed to keep track of things
00279 };
00280 */
00281
00282
00283
00284 // Base class for TerminationOptions. Other Terminations can inherit from here if they require more
    options.
00285 struct TerminationOptions
00286 {
00287 public:
00288     // Basic constructor only sets the number of steps between updates
00289     TerminationOptions(largecounter Nsteps) : UpdateEveryNSteps{Nsteps}
00290     {
00291     }
00292
00293     // virtual destructor to ensure correct destruction of descendants
00294     virtual ~TerminationOptions() = default;
00295
00296     const largecounter UpdateEveryNSteps;
00297 };
00298
00299 // Options class for HorizonTermination; keeps track of location of horizon radius and the epsilon to
    terminate away from the horizon
00300 struct HorizonTermOptions : public TerminationOptions
00301 {
00302 public:
00303     HorizonTermOptions(real theHorizonRadius, bool therLogScale, real theAtHorizonEps, largecounter
    Nsteps) : HorizonRadius{theHorizonRadius}, AtHorizonEps{theAtHorizonEps}, rLogScale{therLogScale},
    TerminationOptions(Nsteps)
00304     {
00305     }
00306
00307     const real HorizonRadius;
00308     const real AtHorizonEps;
00309     const bool rLogScale;
00310 };
00311
00312 // Options class for BoundarySphere; has to keep track of the BoundarySphere's radius
00313 struct BoundarySphereTermOptions : public TerminationOptions
00314 {
00315 public:
00316     BoundarySphereTermOptions(real theRadius, bool therLogScale, largecounter Nsteps) :
    SphereRadius{theRadius}, rLogScale{therLogScale},
00317     TerminationOptions(Nsteps)
00318     {
00319     }
00320
00321     const real SphereRadius;
00322     const bool rLogScale;
00323 };
00324
00325 // Options class for TimeOut; has to keep track of the max. number of integration steps allowed
00326 struct TimeOutTermOptions : public TerminationOptions
00327 {
00328 public:
00329     TimeOutTermOptions(largecounter MaxStepsAllowed, largecounter Nsteps) : MaxSteps{MaxStepsAllowed},
    TerminationOptions(Nsteps)
00330     {
00331     }
00332
00333     const largecounter MaxSteps;
00334 };
00335
00336 // Options class for ThetaSingularityTermination
00337 struct ThetaSingularityTermOptions : public TerminationOptions

```

```

00340 {
00341 public:
00342     ThetaSingularityTermOptions(real epsilon, largecounter Nsteps) : ThetaSingEpsilon(epsilon),
TerminationOptions(Nsteps)
00343     {
00344     }
00345
00346     const real ThetaSingEpsilon;
00347 };
00348
00349 // Options class for TimeOut; has to keep track of the max. number of integration steps allowed
00350 struct NaNTermOptions : public TerminationOptions
00351 {
00352 public:
00353     NaNTermOptions(bool consoleoutputon, largecounter Nsteps) : OutputToConsole{consoleoutputon},
TerminationOptions(Nsteps)
00354     {
00355     }
00356
00357     const bool OutputToConsole;
00358 };
00359
00360 struct GeneralSingularityTermOptions : public TerminationOptions
00361 {
00362 public:
00363     GeneralSingularityTermOptions(std::vector<Singularity> sings,
00364                                     real eps, bool consoleoutputon, bool therlogscale, largecounter
Nsteps)
00365         : Singularities{std::move(sings)}, Epsilon{eps}, OutputToConsole{consoleoutputon},
rLogScale{therlogscale},
00366         TerminationOptions(Nsteps) {}
00367
00368     const std::vector<Singularity> Singularities;
00369     const real Epsilon;
00370     const bool OutputToConsole;
00371     const bool rLogScale;
00372 };
00373 };
00374
00376 // Add your new TerminationOptions struct here, inheriting from TerminationOptions (if needed)
00377 // Sample code:
00378 /*
00379 struct MyTermOptions : public TerminationOptions
00380 {
00381 public:
00382     MyTermOptions(..., largecounter Nsteps) : TerminationOptions(Nsteps) //, other initialization
00383     {}
00384
00385     // member variables (const!) here
00386 };
00387 */
00389
00390 #endif

```

## 7.38 FOORT/src/Utilities.cpp File Reference

```

#include "Utilities.h"
#include <sstream>
#include <iomanip>

```

## 7.39 FOORT/src/Utilities.h File Reference

```

#include "Metric.h"
#include "Diagnostics.h"
#include "Terminations.h"
#include "Geodesic.h"
#include "ViewScreen.h"
#include "Integrators.h"
#include <chrono>
#include <string>
#include <vector>

```

**Classes**

- class [Utilities::Timer](#)

**Namespaces**

- namespace [Utilities](#)

**Functions**

- `std::string Utilities::GetTimeStampString ()`  
Other functions in [Utilities](#).
- `std::vector< std::string > Utilities::GetDiagNameStrings (DiagBitflag alldiags, DiagBitflag valdiag)`
- `std::string Utilities::GetFirstLineInfoString (const Metric *theMetric, const Source *theSource, DiagBitflag alldiags, DiagBitflag valdiag, TermBitflag allterms, const ViewScreen *theView)`

## 7.40 Utilities.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _FOORT_UTILITIES_H
00002 #define _FOORT_UTILITIES_H
00003
00009
00010 // We use Metric, Diagnostic, Termination, Geodesic, ViewScreen, and Integrator declarations here
00011 #include "Metric.h"
00012 #include "Diagnostics.h"
00013 #include "Terminations.h"
00014 #include "Geodesic.h"
00015 #include "ViewScreen.h"
00016 #include "Integrators.h"
00017
00018 #include <chrono> // for timer functionality
00019 #include <string> // for strings
00020 #include <vector> // for std::vector
00021
00022 // Namespace that contains our utility functions
00023 namespace Utilities
00024 {
00025     // Timer class to keep track of elapsed time
00026     class Timer
00027     {
00028     private:
00029         // Type aliases to make accessing nested type easier
00030         using Clock = std::chrono::steady_clock;
00031         using Second = std::chrono::duration<double, std::ratio<1>;
00032
00033         // begin time
00034         std::chrono::time_point<Clock> m_beg{Clock::now()};
00035
00036     public:
00037         // reset begin time
00038         void reset();
00039
00040         // returns time elapsed since begin time
00041         double elapsed() const;
00042     };
00043
00044     // Returns a string of the current time (in a format that can be used to append to file names)
00045     std::string GetTimeStampString();
00046
00047     // Helper function to get all Diagnostic Names (for outputting to files)
00048     std::vector<std::string> GetDiagNameStrings(DiagBitflag alldiags, DiagBitflag valdiag);
00049
00050     // This returns the full string to be written to every output file as its first line
00051     // It contains information about all the settings used to produce the output
00052     std::string GetFirstLineInfoString(const Metric *theMetric, const Source *theSource,
00053                                       DiagBitflag alldiags, DiagBitflag valdiag, TermBitflag
allterms, const ViewScreen *theView);
00054 }
00055
00056 #endif

```

## 7.41 FOORT/src/ViewScreen.cpp File Reference

```
#include "ViewScreen.h"
#include <algorithm>
```

## 7.42 FOORT/src/ViewScreen.h File Reference

```
#include "Geometry.h"
#include "Metric.h"
#include "Mesh.h"
#include <memory>
#include <utility>
#include <array>
#include <string>
```

### Classes

- class [ViewScreen](#)

### Enumerations

- enum class [GeodesicType](#) { [Null](#) = 0 , [Timelike](#) = -1 , [Spacelike](#) = 1 }

### 7.42.1 Enumeration Type Documentation

#### 7.42.1.1 GeodesicType

```
enum class GeodesicType [strong]
```

#### Enumerator

<a href="#">Null</a>	
<a href="#">Timelike</a>	
<a href="#">Spacelike</a>	

## 7.43 ViewScreen.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _FOORT_VIEWSCREEN_H
00002 #define _FOORT_VIEWSCREEN_H
00003
00011
00012 #include "Geometry.h" // For basic tensor objects
00013 #include "Metric.h"   // For the Metric object
00014 #include "Mesh.h"     // For the Mesh object
00015
00016 #include <memory>      // std::unique_ptr
00017 #include <utility>      // std::move
00018 #include <array>        // std::array
00019 #include <string>       // strings
00020
00021 // Type of geodesic being integrated. NOTE: only Null supported/implemented at the moment!
00022 enum class GeodesicType
00023 {
00024     Null = 0,
00025     Timelike = -1,
00026     Spacelike = 1,
00027 };
00028
00029 // ViewScreen class: this class is in charge of converting a pixel on the screen (which the Mesh wants
// to integrate)
00030 // to physical initial conditions for the position and velocity of a geodesic. It owns a Mesh
// instance, which will tell it
00031 // which pixels to integrate etc.
00032 class ViewScreen
00033 {
00034 public:
00035     // No default constructor possible
00036     ViewScreen() = delete;
00037     // constructor must pass following arguments along:
00038     // - physical position and looking direction;
00039     // - screen dimensions (in dimensions of length)
00040     // - the Mesh used (ViewScreen must become a owner of this object!)
00041     // - the Metric used (ViewScreen is NOT the owner of the Metric)
00042     // - the geodesic type to be integrated (null, timelike, spacelike)
00043     ViewScreen(Point pos, OneIndex dir, ScreenPoint screensize, ScreenPoint screencenter,
00044         std::unique_ptr<Mesh> theMesh, const Metric *const theMetric, GeodesicType thegeodtype
= GeodesicType::Null)
00045     : m_Pos{pos}, m_Direction{dir}, m_ScreenSize{screensize}, m_ScreenCenter{screencenter},
00046       m_theMesh{std::move(theMesh)},
00047       m_theMetric{theMetric}, m_GeodType{thegeodtype},
00048       m_rLogScale{theMetric->getrLogScale()}
00049     {
00050         // At the moment, we don't even use the direction; we are always pointed towards the origin
00051         if (m_Direction != Point{0, -1, 0, 0})
00052         {
00053             ScreenOutput("ViewScreen is only supported pointing inwards at the moment; Direction = {0,
-1, 0, 0} will be used",
00054                 OutputLevel::Level_0_WARNING);
00055         }
00056         // At the moment, we are only integrating null geodesics
00057         if (m_GeodType != GeodesicType::Null)
00058         {
00059             ScreenOutput("ViewScreen only supports null geodesics at the moment; geodesics integrated
will be null.",
00060                 OutputLevel::Level_0_WARNING);
00061         }
00062
00063         // Construct the vielbein now
00064         ConstructVielbein();
00065     }
00066
00067     // Heart of the ViewScreen: here, the ViewScreen is asked to provide initial conditions
00068     // for the geodesic nr index of the current iteration; based on the screen index
00069     // that the Mesh gives, it sets up these physical initial conditions.
00070     void SetNewInitialConditions(largecounter index, Point &pos, OneIndex &vel, ScreenIndex &scrIndex)
const;
00071
00072     // These member functions essentially pass on information to/from the Mesh
00073     bool IsFinished() const; // Does the ViewScreen (i.e. the Mesh) want to integrate
// more geodesics or not?
00074     largecounter getCurNrGeodesics() const; // Current number of geodesics in this iteration
00075     void EndCurrentLoop(); // The current iteration of geodesics is finished;
// prepare the next one
00076     // NOTE: despite not being const, this function has been designed to be threadsafe!
00077     void GeodesicFinished(largecounter index, std::vector<real> finalValues); // This geodesic has
// been integrated, returning its final "values"
00078
00079     // Description string getter (spaces allowed), also will contain information about the Mesh
00080     std::string getFullDescriptionStr() const;

```

```
00081
00082 private:
00083     // The metric at the position of the viewscreen (we only need indices down)
00084     TwoIndex m_Metric_dd{};
00085     // The vielbein used to transform from the curved spacetime at the viewscreen to a locally flat
    frame
00086     TwoIndex m_Vielbein{};
00087
00088     // Helper function to construct the vielbein given the metric
00089     void ConstructVielbein();
00090
00091     // The position and looking direction of the camera
00092     const Point m_Pos;
00093     const OneIndex m_Direction;
00094     // The screensize (in physical units of length)
00095     const ScreenPoint m_ScreenSize;
00096     // The screen center
00097     const ScreenPoint m_ScreenCenter;
00098
00099     // Whether the metric uses a logarithmic r coordinate or not
00100     const bool m_rLogScale;
00101
00102     // const pointer to const Metric
00103     const Metric *const m_theMetric;
00104
00105     // The geodesic type to be integrated
00106     const GeodesicType m_GeodType{GeodesicType::Null};
00107
00108     // The const pointer to the Mesh we are using to determine pixels to be integrated
00109     const std::unique_ptr<Mesh> m_theMesh;
00110 };
00111
00112 #endif
```



# Index

- ~Diagnostic
  - Diagnostic, [30](#)
- ~DiagnosticOptions
  - DiagnosticOptions, [32](#)
- ~EmissionModel
  - EmissionModel, [33](#)
- ~FluidVelocityModel
  - FluidVelocityModel, [44](#)
- ~Mesh
  - Mesh, [82](#)
- ~Metric
  - Metric, [84](#)
- ~Source
  - Source, [99](#)
- ~Termination
  - Termination, [115](#)
- ~TerminationOptions
  - TerminationOptions, [116](#)
- AtHorizonEps
  - HorizonTermOptions, [68](#)
- BosonStarMetric, [15](#)
  - BosonStarMetric, [16](#)
  - getFullDescriptionStr, [16](#)
  - getMetric\_dd, [16](#)
  - getMetric\_uu, [16](#)
  - m\_mSpline, [16](#)
  - m\_PhiSpline, [16](#)
- BoundarySphere
  - Terminations.h, [182](#)
- BoundarySphereTermination, [17](#)
  - BoundarySphereTermination, [18](#)
  - CheckTermination, [18](#)
  - getFullDescriptionStr, [18](#)
  - TermOptions, [18](#)
- BoundarySphereTermOptions, [18](#)
  - BoundarySphereTermOptions, [19](#)
  - rLogScale, [19](#)
  - SphereRadius, [19](#)
- CheckTermination
  - BoundarySphereTermination, [18](#)
  - GeneralSingularityTermination, [51](#)
  - HorizonTermination, [67](#)
  - NaNTermination, [86](#)
  - Termination, [115](#)
  - ThetaSingularityTermination, [118](#)
  - TimeOutTermination, [121](#)
- Clock
  - Utilities::Timer, [123](#)
- ClosestRadiusDiagnostic, [20](#)
  - ClosestRadiusDiagnostic, [21](#)
  - DiagOptions, [22](#)
  - FinalDataValDistance, [21](#)
  - getFinalDataVal, [21](#)
  - getFullDataStr, [21](#)
  - getFullDescriptionStr, [21](#)
  - getNameStr, [21](#)
  - m\_ClosestRadius, [22](#)
  - Reset, [21](#)
  - UpdateData, [21](#)
- ClosestRadiusOptions, [22](#)
  - ClosestRadiusOptions, [23](#)
  - RLogScale, [23](#)
- Config, [9](#)
  - ConfigCollection, [10](#)
  - GetGeodesicIntegrator, [10](#)
  - GetMesh, [10](#)
  - GetMetric, [10](#)
  - GetOutputHandler, [10](#)
  - GetSource, [10](#)
  - GetViewScreen, [11](#)
  - InitializeDiagnostics, [11](#)
  - InitializeScreenOutput, [11](#)
  - InitializeTerminations, [11](#)
  - Output\_Important\_Default, [11](#)
  - Output\_Other\_Default, [11](#)
  - SettingError, [10](#)
- Config.h
  - CONFIGURATION\_MODE, [130](#)
- ConfigCollection
  - Config, [10](#)
- ConfigReader, [12](#)
  - MaxStreamSize, [12](#)
- ConfigReader::ConfigCollection, [23](#)
  - ConfigSettingValue, [24](#)
  - DisplayCollection, [24](#)
  - DisplaySetting, [24](#)
  - DisplayTabs, [24](#)
  - Exists, [24](#)
  - GetSettingIndex, [25](#)
  - IsCollection, [25](#)
  - LookupValue, [25](#)
  - LookupValueInteger, [25](#), [26](#)
  - m\_Settings, [27](#)
  - NrSettings, [26](#)
  - operator[], [26](#)
  - ReadCollection, [26](#)

- ReadFile, 27
- ReadSettingName, 27
- ReadSettingSpecificChar, 27
- ReadSettingValue, 27
- ConfigReader::ConfigCollection::ConfigSetting, 28
  - SettingName, 28
  - SettingValue, 28
- ConfigReader::ConfigReaderException, 27
  - ConfigReaderException, 28
  - m\_settingtrace, 28
  - trace, 28
- ConfigReaderException
  - ConfigReader::ConfigReaderException, 28
- ConfigSettingValue
  - ConfigReader::ConfigCollection, 24
- CONFIGURATION\_MODE
  - Config.h, 130
- ConstructVielbein
  - ViewScreen, 126
- Continue
  - Terminations.h, 182
- CreateDiagnosticVector
  - Diagnostics.cpp, 135
  - Diagnostics.h, 137
- CreateTerminationVector
  - Terminations.cpp, 181
  - Terminations.h, 183
- DecideUpdate
  - Diagnostic, 30
  - Termination, 115
- delta\_nodiv0
  - Integrators, 13
- Derivative\_hval
  - Integrators, 13
- Diag\_ClosestRadius
  - Diagnostics.h, 137
- Diag\_EquatorialEmission
  - Diagnostics.h, 137
- Diag\_EquatorialPasses
  - Diagnostics.h, 137
- Diag\_FourColorScreen
  - Diagnostics.h, 137
- Diag\_GeodesicPosition
  - Diagnostics.h, 137
- Diag\_None
  - Diagnostics.h, 137
- DiagBitflag
  - Diagnostics.h, 136
- Diagnostic, 29
  - ~Diagnostic, 30
  - DecideUpdate, 30
  - Diagnostic, 29
  - FinalDataValDistance, 30
  - getFinalDataVal, 30
  - getFullDataStr, 30
  - getFullDescriptionStr, 30
  - getNameStr, 30
  - m\_OwnerGeodesic, 31
  - m\_StepsSinceUpdated, 31
  - Reset, 31
  - UpdateData, 31
- DiagnosticOptions, 31
  - ~DiagnosticOptions, 32
  - DiagnosticOptions, 32
  - theUpdateFrequency, 32
- Diagnostics.cpp
  - CreateDiagnosticVector, 135
- Diagnostics.h
  - CreateDiagnosticVector, 137
  - Diag\_ClosestRadius, 137
  - Diag\_EquatorialEmission, 137
  - Diag\_EquatorialPasses, 137
  - Diag\_FourColorScreen, 137
  - Diag\_GeodesicPosition, 137
  - Diag\_None, 137
  - DiagBitflag, 136
  - DiagnosticUniqueVector, 136
- DiagnosticUniqueVector
  - Diagnostics.h, 136
- DiagOptions
  - ClosestRadiusDiagnostic, 22
  - EquatorialEmissionDiagnostic, 36
  - EquatorialPassesDiagnostic, 41
  - GeodesicPositionDiagnostic, 63
- DiagValue
  - SquareSubdivisionMesh::PixelInfo, 90
  - SquareSubdivisionMeshV2::PixelInfo, 91
- dimension
  - Geometry.h, 152
- DisplayCollection
  - ConfigReader::ConfigCollection, 24
- DisplaySetting
  - ConfigReader::ConfigCollection, 24
- DisplayTabs
  - ConfigReader::ConfigCollection, 24
- DownNbr
  - SquareSubdivisionMeshV2::PixelInfo, 91
- elapsed
  - Utilities::Timer, 123
- EmissionModel, 32
  - ~EmissionModel, 33
  - GetEmission, 33
  - getFullDescriptionStr, 33
- EndCurrentLoop
  - InputCertainPixelsMesh, 70
  - Mesh, 82
  - SimpleSquareMesh, 96
  - SquareSubdivisionMesh, 103
  - SquareSubdivisionMeshV2, 108
  - ViewScreen, 126
- Epsilon
  - GeneralSingularityTermOptions, 53
- epsilon
  - Integrators, 13
- EquatorialEmissionDiagnostic, 36
  - DiagOptions, 36

- EquatorialEmissionDiagnostic, 35
- FinalDataValDistance, 35
- getFinalDataVal, 35
- getFullDataStr, 35
- getFullDescriptionStr, 35
- getNameStr, 35
- m\_Intensity, 36
- Reset, 36
- UpdateData, 36
- EquatorialEmissionOptions, 36
  - EquatorialEmissionOptions, 37
  - EquatPassUpperBound, 38
  - GeometricFudgeFactor, 38
  - RedShiftPower, 38
  - RLogScale, 38
  - TheEmissionModel, 38
  - TheFluidVelocityModel, 38
- EquatorialPassesDiagnostic, 38
  - DiagOptions, 41
  - EquatorialPassesDiagnostic, 39
  - FinalDataValDistance, 40
  - getFinalDataVal, 40
  - getFullDataStr, 40
  - getFullDescriptionStr, 40
  - getNameStr, 40
  - m\_EquatPasses, 41
  - m\_PrevTheta, 41
  - Reset, 40
  - UpdateData, 40
- EquatorialPassesOptions, 41
  - EquatorialPassesOptions, 42
  - Threshold, 42
- EquatPassUpperBound
  - EquatorialEmissionOptions, 38
- Exists
  - ConfigReader::ConfigCollection, 24
- Explnt
  - SquareSubdivisionMesh, 103
  - SquareSubdivisionMeshV2, 108
- f\_om\_phi
  - ST3CrMetric, 113
- f\_phi
  - ST3CrMetric, 113
- FinalDataValDistance
  - ClosestRadiusDiagnostic, 21
  - Dagnostic, 30
  - EquatorialEmissionDiagnostic, 35
  - EquatorialPassesDiagnostic, 40
  - FourColorScreenDiagnostic, 46
  - GeodesicPositionDiagnostic, 62
- FindISCO
  - GeneralCircularRadialFluid, 49
- FlatSpaceMetric, 42
  - FlatSpaceMetric, 43
  - getFullDescriptionStr, 43
  - getMetric\_dd, 43
  - getMetric\_uu, 43
- FluidVelocityModel, 44
  - ~FluidVelocityModel, 44
  - FluidVelocityModel, 44
  - GetFourVelocityd, 44
  - getFullDescriptionStr, 44
  - m\_theMetric, 45
- FOORT/src/Config.cpp, 129
- FOORT/src/Config.h, 129, 131
- FOORT/src/ConfigReader.cpp, 132
- FOORT/src/ConfigReader.h, 132, 133
- FOORT/src/Diagnostics.cpp, 135
- FOORT/src/Diagnostics.h, 135, 138
- FOORT/src/DiagnosticsEmission.cpp, 143
- FOORT/src/DiagnosticsEmission.h, 144
- FOORT/src/Geodesic.cpp, 146
- FOORT/src/Geodesic.h, 146
- FOORT/src/Geometry.h, 148, 152
- FOORT/src/Header.h, 154
- FOORT/src/InputOutput.cpp, 154
- FOORT/src/InputOutput.h, 155, 157
- FOORT/src/Integrators.cpp, 158
- FOORT/src/Integrators.h, 158, 160
- FOORT/src/Main.cpp, 160
- FOORT/src/Mesh.cpp, 161
- FOORT/src/Mesh.h, 162
- FOORT/src/Metric.cpp, 167
- FOORT/src/Metric.h, 167, 168
- FOORT/src/Spline.h, 172
- FOORT/src/Terminations.cpp, 181
- FOORT/src/Terminations.h, 181, 184
- FOORT/src/Utilities.cpp, 188
- FOORT/src/Utilities.h, 188, 189
- FOORT/src/ViewScreen.cpp, 190
- FOORT/src/ViewScreen.h, 190, 191
- FourColorScreenDiagnostic, 45
  - FinalDataValDistance, 46
  - FourColorScreenDiagnostic, 46
  - getFinalDataVal, 46
  - getFullDataStr, 46
  - getFullDescriptionStr, 46
  - getNameStr, 47
  - m\_quadrant, 47
  - Reset, 47
  - UpdateData, 47
- FourIndex
  - Geometry.h, 149
- GeneralCircularRadialFluid, 47
  - FindISCO, 49
  - GeneralCircularRadialFluid, 48
  - GetChrstrRaisedDer, 49
  - GetCircularVelocityd, 49
  - GetFourVelocityd, 49
  - getFullDescriptionStr, 49
  - GetInsideISCOCircularVelocityd, 49
  - GetRadialVelocityd, 49
  - m\_betaPhi, 50
  - m\_betaR, 50
  - m\_ISCOexists, 50
  - m\_ISCOpphi, 50

- m\_ISCOpt, 50
  - m\_ISCO, 50
  - m\_subKeplerParam, 50
- GeneralSingularity
  - Terminations.h, 182
- GeneralSingularityTermination, 50
  - CheckTermination, 51
  - GeneralSingularityTermination, 51
  - getFullDescriptionStr, 51
  - SingularityToString, 52
  - TermOptions, 52
- GeneralSingularityTermOptions, 52
  - Epsilon, 53
  - GeneralSingularityTermOptions, 53
  - OutputToConsole, 53
  - rLogScale, 53
  - Singularities, 53
- Geodesic, 54
  - Geodesic, 54
  - getAllOutputStr, 55
  - getCurrentLambda, 55
  - getCurrentPos, 55
  - getCurrentVel, 55
  - getDiagnosticFinalValue, 55
  - getScreenIndex, 55
  - getTermCondition, 55
  - m\_AllDiagnostics, 56
  - m\_AllTerminations, 56
  - m\_curLambda, 56
  - m\_CurrentPos, 56
  - m\_CurrentVel, 56
  - m\_ScreenIndex, 56
  - m\_TermCond, 56
  - m\_theIntegrator, 56
  - m\_theMetric, 57
  - m\_theSource, 57
  - operator=, 55
  - Reset, 55
  - Update, 56
- GeodesicFinished
  - InputCertainPixelsMesh, 70
  - Mesh, 82
  - SimpleSquareMesh, 96
  - SquareSubdivisionMesh, 103
  - SquareSubdivisionMeshV2, 108
  - ViewScreen, 126
- GeodesicIntegratorFunc
  - Integrators.h, 159
- GeodesicOutputHandler, 57
  - GeodesicOutputHandler, 58
  - GetFileName, 58
  - getFullDescriptionStr, 58
  - m\_AllCachedData, 59
  - m\_CurrentFullFiles, 59
  - m\_CurrentGeodesicsInFile, 59
  - m\_DiagNames, 59
  - m\_FileExtension, 59
  - m\_FilePrefix, 60
- m\_FirstLineInfoString, 60
  - m\_nrGeodesicsPerFile, 60
  - m\_nrOutputsToCache, 60
  - m\_PrevCached, 60
  - m\_PrintFirstLineInfo, 60
  - m\_TimeStamp, 60
  - m\_WriteToConsole, 60
  - NewGeodesicOutput, 58
  - OpenForFirstTime, 59
  - OutputFinished, 59
  - PrepareForOutput, 59
  - WriteCachedOutputToFile, 59
- GeodesicPositionDiagnostic, 61
  - DiagOptions, 63
  - FinalDataValDistance, 62
  - GeodesicPositionDiagnostic, 62
  - getFinalDataVal, 62
  - getFullDataStr, 62
  - getFullDescriptionStr, 62
  - getNameStr, 62
  - m\_AllSavedPoints, 63
  - Reset, 62
  - UpdateData, 62
- GeodesicPositionOptions, 63
  - GeodesicPositionOptions, 64
  - OutputNrSteps, 64
- GeodesicType
  - ViewScreen.h, 190
- GeometricFudgeFactor
  - EquatorialEmissionOptions, 38
- Geometry.h
  - dimension, 152
  - FourIndex, 149
  - largecounter, 149
  - LARGECOUNTER\_MAX, 149
  - OneIndex, 149
  - operator+, 151
  - operator-, 151
  - operator/, 151
  - operator\*, 151
  - pi, 152
  - PIXEL\_MAX, 149
  - pixelcoord, 150
  - Point, 150
  - real, 150
  - ScreenIndex, 150
  - ScreenPoint, 150
  - Singularity, 150
  - SingularityCoord, 150
  - ThreeIndex, 150
  - toString, 151, 152
  - TwoIndex, 150
- get\_omega
  - ST3CrMetric, 113
- GetAdaptiveStep
  - Integrators, 12
- getAllOutputStr
  - Geodesic, 55

- getChristoffel\_udd
  - Metric, [84](#)
- GetChrstrRaisedDer
  - GeneralCircularRadialFluid, [49](#)
- GetCircularVelocityd
  - GeneralCircularRadialFluid, [49](#)
- getCurNrGeodesics
  - InputCertainPixelsMesh, [70](#)
  - Mesh, [82](#)
  - SimpleSquareMesh, [96](#)
  - SquareSubdivisionMesh, [104](#)
  - SquareSubdivisionMeshV2, [108](#)
  - ViewScreen, [126](#)
- getCurrentLambda
  - Geodesic, [55](#)
- getCurrentPos
  - Geodesic, [55](#)
- getCurrentVel
  - Geodesic, [55](#)
- GetDiagNameStrings
  - Utilities, [14](#)
- getDiagnosticFinalValue
  - Geodesic, [55](#)
- GetDown
  - SquareSubdivisionMeshV2, [108](#)
- GetEmission
  - EmissionModel, [33](#)
  - GLMJohnsonSUEmission, [65](#)
- GetFileName
  - GeodesicOutputHandler, [58](#)
- getFinalDataVal
  - ClosestRadiusDiagnostic, [21](#)
  - Dagnostic, [30](#)
  - EquatorialEmissionDiagnostic, [35](#)
  - EquatorialPassesDiagnostic, [40](#)
  - FourColorScreenDiagnostic, [46](#)
  - GeodesicPositionDiagnostic, [62](#)
- GetFirstLineInfoString
  - Utilities, [14](#)
- GetFourVelocityd
  - FluidVelocityModel, [44](#)
  - GeneralCircularRadialFluid, [49](#)
- getFullDataStr
  - ClosestRadiusDiagnostic, [21](#)
  - Dagnostic, [30](#)
  - EquatorialEmissionDiagnostic, [35](#)
  - EquatorialPassesDiagnostic, [40](#)
  - FourColorScreenDiagnostic, [46](#)
  - GeodesicPositionDiagnostic, [62](#)
- getFullDescriptionStr
  - BosonStarMetric, [16](#)
  - BoundarySphereTermination, [18](#)
  - ClosestRadiusDiagnostic, [21](#)
  - Dagnostic, [30](#)
  - EmissionModel, [33](#)
  - EquatorialEmissionDiagnostic, [35](#)
  - EquatorialPassesDiagnostic, [40](#)
  - FlatSpaceMetric, [43](#)
  - FluidVelocityModel, [44](#)
  - FourColorScreenDiagnostic, [46](#)
  - GeneralCircularRadialFluid, [49](#)
  - GeneralSingularityTermination, [51](#)
  - GeodesicOutputHandler, [58](#)
  - GeodesicPositionDiagnostic, [62](#)
  - GLMJohnsonSUEmission, [65](#)
  - HorizonTermination, [67](#)
  - InputCertainPixelsMesh, [70](#)
  - JohannsenMetric, [73](#)
  - KerrMetric, [76](#)
  - KerrSchildMetric, [78](#)
  - MankoNovikovMetric, [80](#)
  - Mesh, [82](#)
  - Metric, [84](#)
  - NaNTermination, [86](#)
  - NoSource, [89](#)
  - RasheedLarsenMetric, [94](#)
  - SimpleSquareMesh, [96](#)
  - Source, [99](#)
  - SquareSubdivisionMesh, [104](#)
  - SquareSubdivisionMeshV2, [108](#)
  - ST3CrMetric, [113](#)
  - Termination, [115](#)
  - ThetaSingularityTermination, [118](#)
  - TimeOutTermination, [121](#)
  - ViewScreen, [126](#)
- GetFullIntegratorDescription
  - Integrators, [12](#)
- GetGeodesicIntegrator
  - Config, [10](#)
- getHorizonRadius
  - SphericalHorizonMetric, [101](#)
- GetInsideISCOCircularVelocityd
  - GeneralCircularRadialFluid, [49](#)
- getKretschmann
  - Metric, [84](#)
- GetLeft
  - SquareSubdivisionMeshV2, [109](#)
- GetLoopMessageFrequency
  - InputOutput.cpp, [155](#)
  - InputOutput.h, [156](#)
- GetMesh
  - Config, [10](#)
- GetMetric
  - Config, [10](#)
- getMetric\_dd
  - BosonStarMetric, [16](#)
  - FlatSpaceMetric, [43](#)
  - JohannsenMetric, [73](#)
  - KerrMetric, [76](#)
  - KerrSchildMetric, [78](#)
  - MankoNovikovMetric, [80](#)
  - Metric, [84](#)
  - RasheedLarsenMetric, [94](#)
  - ST3CrMetric, [113](#)
- getMetric\_uu
  - BosonStarMetric, [16](#)

- FlatSpaceMetric, [43](#)
- JohannsenMetric, [73](#)
- KerrMetric, [76](#)
- KerrSchildMetric, [78](#)
- MankoNovikovMetric, [80](#)
- Metric, [84](#)
- RasheedLarsenMetric, [94](#)
- ST3CrMetric, [113](#)
- getNameStr
  - ClosestRadiusDiagnostic, [21](#)
  - Diagnostic, [30](#)
  - EquatorialEmissionDiagnostic, [35](#)
  - EquatorialPassesDiagnostic, [40](#)
  - FourColorScreenDiagnostic, [47](#)
  - GeodesicPositionDiagnostic, [62](#)
- getNewInitConds
  - InputCertainPixelsMesh, [70](#)
  - Mesh, [82](#)
  - SimpleSquareMesh, [96](#)
  - SquareSubdivisionMesh, [104](#)
  - SquareSubdivisionMeshV2, [109](#)
- GetOutputHandler
  - Config, [10](#)
- GetRadialVelocityd
  - GeneralCircularRadialFluid, [49](#)
- getRiemann\_uddd
  - Metric, [85](#)
- GetRight
  - SquareSubdivisionMeshV2, [109](#)
- getrLogScale
  - Metric, [85](#)
- getScreenIndex
  - Geodesic, [55](#)
- GetSettingIndex
  - ConfigReader::ConfigCollection, [25](#)
- getSingularities
  - SingularityMetric, [98](#)
- GetSource
  - Config, [10](#)
- getSource
  - NoSource, [89](#)
  - Source, [99](#)
- getTermCondition
  - Geodesic, [55](#)
- GetTimeStampString
  - Utilities, [14](#)
- GetUp
  - SquareSubdivisionMeshV2, [109](#)
- GetViewScreen
  - Config, [11](#)
- GLMJohnsonSUEmission, [64](#)
  - GetEmission, [65](#)
  - getFullDescriptionStr, [65](#)
  - GLMJohnsonSUEmission, [65](#)
  - m\_gamma, [65](#)
  - m\_mu, [65](#)
  - m\_sigma, [65](#)
- Horizon
  - Terminations.h, [182](#)
- HorizonRadius
  - HorizonTermOptions, [68](#)
- HorizonTermination, [66](#)
  - CheckTermination, [67](#)
  - getFullDescriptionStr, [67](#)
  - HorizonTermination, [67](#)
  - TermOptions, [67](#)
- HorizonTermOptions, [67](#)
  - AtHorizonEps, [68](#)
  - HorizonRadius, [68](#)
  - HorizonTermOptions, [68](#)
  - rLogScale, [68](#)
- Index
  - SquareSubdivisionMesh::PixelInfo, [90](#)
  - SquareSubdivisionMeshV2::PixelInfo, [91](#)
- InitializeDiagnostics
  - Config, [11](#)
- InitializeFirstGrid
  - SquareSubdivisionMesh, [104](#)
  - SquareSubdivisionMeshV2, [109](#)
- InitializeScreenOutput
  - Config, [11](#)
- InitializeTerminations
  - Config, [11](#)
- InputCertainPixelsMesh, [69](#)
  - EndCurrentLoop, [70](#)
  - GeodesicFinished, [70](#)
  - getCurNrGeodesics, [70](#)
  - getFullDescriptionStr, [70](#)
  - getNewInitConds, [70](#)
  - InputCertainPixelsMesh, [69, 70](#)
  - IsFinished, [71](#)
  - m\_Finished, [71](#)
  - m\_PixelsToIntegrate, [71](#)
  - m\_RowColumnSize, [71](#)
  - m\_TotalPixels, [71](#)
- InputOutput.cpp
  - GetLoopMessageFrequency, [155](#)
  - ScreenOutput, [155](#)
  - SetLoopMessageFrequency, [155](#)
  - SetOutputLevel, [155](#)
- InputOutput.h
  - GetLoopMessageFrequency, [156](#)
  - Level\_0\_WARNING, [156](#)
  - Level\_1\_PROC, [156](#)
  - Level\_2\_SUBPROC, [156](#)
  - Level\_3\_ALLDDETAIL, [156](#)
  - Level\_4\_DEBUG, [156](#)
  - MaxLevel, [156](#)
  - OutputLevel, [156](#)
  - ScreenOutput, [156](#)
  - SetLoopMessageFrequency, [156](#)
  - SetOutputLevel, [156](#)
- IntegrateGeodesicStep\_RK4
  - Integrators, [13](#)
- IntegrateGeodesicStep\_Verlet
  - Integrators, [13](#)

- IntegratorDescription
  - Integrators, [13](#)
- Integrators, [12](#)
  - delta\_nodiv0, [13](#)
  - Derivative\_hval, [13](#)
  - epsilon, [13](#)
  - GetAdaptiveStep, [12](#)
  - GetFullIntegratorDescription, [12](#)
  - IntegrateGeodesicStep\_RK4, [13](#)
  - IntegrateGeodesicStep\_Verlet, [13](#)
  - IntegratorDescription, [13](#)
  - SmallestPossibleStepsize, [13](#)
  - VerletVelocityTolerance, [14](#)
- Integrators.h
  - GeodesicIntegratorFunc, [159](#)
- IsCollection
  - ConfigReader::ConfigCollection, [25](#)
- IsFinished
  - InputCertainPixelsMesh, [71](#)
  - Mesh, [83](#)
  - SimpleSquareMesh, [96](#)
  - SquareSubdivisionMesh, [104](#)
  - SquareSubdivisionMeshV2, [109](#)
  - ViewScreen, [126](#)
- JohannsenMetric, [71](#)
  - getFullDescriptionStr, [73](#)
  - getMetric\_dd, [73](#)
  - getMetric\_uu, [73](#)
  - JohannsenMetric, [73](#)
  - m\_alpha13Param, [73](#)
  - m\_alpha22Param, [73](#)
  - m\_alpha52Param, [74](#)
  - m\_aParam, [74](#)
  - m\_eps3Param, [74](#)
- KerrMetric, [74](#)
  - getFullDescriptionStr, [76](#)
  - getMetric\_dd, [76](#)
  - getMetric\_uu, [76](#)
  - KerrMetric, [75](#)
  - m\_aParam, [76](#)
  - m\_mParam, [76](#)
- KerrSchildMetric, [76](#)
  - getFullDescriptionStr, [78](#)
  - getMetric\_dd, [78](#)
  - getMetric\_uu, [78](#)
  - KerrSchildMetric, [77](#)
  - m\_aParam, [78](#)
- largecounter
  - Geometry.h, [149](#)
- LARGECOUNTER\_MAX
  - Geometry.h, [149](#)
- LeftNbr
  - SquareSubdivisionMeshV2::PixelInfo, [91](#)
- Level\_0\_WARNING
  - InputOutput.h, [156](#)
- Level\_1\_PROC
  - InputOutput.h, [156](#)
- Level\_2\_SUBPROC
  - InputOutput.h, [156](#)
- Level\_3\_ALLDDETAIL
  - InputOutput.h, [156](#)
- Level\_4\_DEBUG
  - InputOutput.h, [156](#)
- LoadPrecompiledOptions
  - Main.cpp, [161](#)
- LookupValue
  - ConfigReader::ConfigCollection, [25](#)
- LookupValueInteger
  - ConfigReader::ConfigCollection, [25, 26](#)
- LowerNbrIndex
  - SquareSubdivisionMesh::PixelInfo, [90](#)
- m\_ActivePixels
  - SquareSubdivisionMeshV2, [110](#)
- m\_AllCachedData
  - GeodesicOutputHandler, [59](#)
- m\_AllDiagnostics
  - Geodesic, [56](#)
- m\_AllPixels
  - SquareSubdivisionMesh, [105](#)
  - SquareSubdivisionMeshV2, [110](#)
- m\_AllSavedPoints
  - GeodesicPositionDiagnostic, [63](#)
- m\_AllSingularities
  - SingularityMetric, [98](#)
- m\_AllTerminations
  - Geodesic, [56](#)
- m\_alpha13Param
  - JohannsenMetric, [73](#)
- m\_alpha22Param
  - JohannsenMetric, [73](#)
- m\_alpha3Param
  - MankoNovikovMetric, [81](#)
- m\_alpha52Param
  - JohannsenMetric, [74](#)
- m\_alphaParam
  - MankoNovikovMetric, [81](#)
- m\_aParam
  - JohannsenMetric, [74](#)
  - KerrMetric, [76](#)
  - KerrSchildMetric, [78](#)
  - MankoNovikovMetric, [81](#)
  - RasheedLarsenMetric, [94](#)
- m\_beg
  - Utilities::Timer, [124](#)
- m\_betaPhi
  - GeneralCircularRadialFluid, [50](#)
- m\_betaR
  - GeneralCircularRadialFluid, [50](#)
- m\_ClosestRadius
  - ClosestRadiusDiagnostic, [22](#)
- m\_curLambda
  - Geodesic, [56](#)
- m\_CurNrSteps
  - TimeOutTermination, [121](#)



- m\_CurrentFullFiles
  - GeodesicOutputHandler, [59](#)
- m\_CurrentGeodesicsInFile
  - GeodesicOutputHandler, [59](#)
- m\_CurrentPixelQueue
  - SquareSubdivisionMesh, [105](#)
  - SquareSubdivisionMeshV2, [110](#)
- m\_CurrentPixelQueueDone
  - SquareSubdivisionMesh, [105](#)
  - SquareSubdivisionMeshV2, [110](#)
- m\_CurrentPixelUpdating
  - SquareSubdivisionMeshV2, [110](#)
- m\_CurrentPos
  - Geodesic, [56](#)
- m\_CurrentVel
  - Geodesic, [56](#)
- m\_DiagNames
  - GeodesicOutputHandler, [59](#)
- m\_Direction
  - ViewScreen, [126](#)
- m\_DistanceDiagnostic
  - Mesh, [83](#)
- m\_eps3Param
  - JohannsenMetric, [74](#)
- m\_EquatPasses
  - EquatorialPassesDiagnostic, [41](#)
- m\_FileExtension
  - GeodesicOutputHandler, [59](#)
- m\_FilePrefix
  - GeodesicOutputHandler, [60](#)
- m\_Finished
  - InputCertainPixelsMesh, [71](#)
  - SimpleSquareMesh, [97](#)
- m\_FirstLineInfoString
  - GeodesicOutputHandler, [60](#)
- m\_gamma
  - GLMJohnsonSUEmission, [65](#)
- m\_GeodType
  - ViewScreen, [126](#)
- m\_HorizonRadius
  - SphericalHorizonMetric, [101](#)
- m\_InfinitePixels
  - SquareSubdivisionMesh, [105](#)
  - SquareSubdivisionMeshV2, [110](#)
- m\_InitialPixels
  - SquareSubdivisionMesh, [105](#)
  - SquareSubdivisionMeshV2, [110](#)
- m\_InitialSubDivideToFinal
  - SquareSubdivisionMesh, [105](#)
  - SquareSubdivisionMeshV2, [110](#)
- m\_Intensity
  - EquatorialEmissionDiagnostic, [36](#)
- m\_ISCOexists
  - GeneralCircularRadialFluid, [50](#)
- m\_ISCOpphi
  - GeneralCircularRadialFluid, [50](#)
- m\_ISCOpt
  - GeneralCircularRadialFluid, [50](#)
- m\_ISCOr
  - GeneralCircularRadialFluid, [50](#)
- m\_IterationPixels
  - SquareSubdivisionMesh, [105](#)
  - SquareSubdivisionMeshV2, [110](#)
- m\_kParam
  - MankoNovikovMetric, [81](#)
- m\_lambda
  - ST3CrMetric, [114](#)
- m\_MaxPixels
  - SquareSubdivisionMesh, [105](#)
  - SquareSubdivisionMeshV2, [111](#)
- m\_MaxSubdivide
  - SquareSubdivisionMesh, [105](#)
  - SquareSubdivisionMeshV2, [111](#)
- m\_Metric\_dd
  - ViewScreen, [127](#)
- m\_mParam
  - KerrMetric, [76](#)
  - RasheedLarsenMetric, [94](#)
- m\_mSpline
  - BosonStarMetric, [16](#)
- m\_mu
  - GLMJohnsonSUEmission, [65](#)
- m\_nrGeodesicsPerFile
  - GeodesicOutputHandler, [60](#)
- m\_nrOutputsToCache
  - GeodesicOutputHandler, [60](#)
- m\_OwnerGeodesic
  - Diagnostic, [31](#)
  - Termination, [116](#)
- m\_P
  - ST3CrMetric, [114](#)
- m\_PhiSpline
  - BosonStarMetric, [16](#)
- m\_PixelsIntegrated
  - SquareSubdivisionMeshV2, [111](#)
- m\_PixelsLeft
  - SquareSubdivisionMesh, [106](#)
  - SquareSubdivisionMeshV2, [111](#)
- m\_PixelsToIntegrate
  - InputCertainPixelsMesh, [71](#)
- m\_Pos
  - ViewScreen, [127](#)
- m\_pParam
  - RasheedLarsenMetric, [94](#)
- m\_PrevCached
  - GeodesicOutputHandler, [60](#)
- m\_PrevTheta
  - EquatorialPassesDiagnostic, [41](#)
- m\_PrintFirstLineInfo
  - GeodesicOutputHandler, [60](#)
- m\_q0
  - ST3CrMetric, [114](#)
- m\_qParam
  - RasheedLarsenMetric, [94](#)
- m\_quadrant
  - FourColorScreenDiagnostic, [47](#)



- m\_rLogScale
  - Metric, [85](#)
  - ViewScreen, [127](#)
- m\_RowColumnSize
  - InputCertainPixelsMesh, [71](#)
  - SimpleSquareMesh, [97](#)
  - SquareSubdivisionMesh, [106](#)
  - SquareSubdivisionMeshV2, [111](#)
- m\_ScreenCenter
  - ViewScreen, [127](#)
- m\_ScreenIndex
  - Geodesic, [56](#)
- m\_ScreenSize
  - ViewScreen, [127](#)
- m\_Settings
  - ConfigReader::ConfigCollection, [27](#)
- m\_settingtrace
  - ConfigReader::ConfigReaderException, [28](#)
- m\_sigma
  - GLMJohnsonSUEmission, [65](#)
- m\_StepsSinceUpdated
  - Diagnostic, [31](#)
  - Termination, [116](#)
- m\_subKeplerParam
  - GeneralCircularRadialFluid, [50](#)
- m\_Symmetries
  - Metric, [85](#)
- m\_TermCond
  - Geodesic, [56](#)
- m\_theIntegrator
  - Geodesic, [56](#)
- m\_theMesh
  - ViewScreen, [127](#)
- m\_theMetric
  - FluidVelocityModel, [45](#)
  - Geodesic, [57](#)
  - Source, [100](#)
  - ViewScreen, [127](#)
- m\_theSource
  - Geodesic, [57](#)
- m\_TimeStamp
  - GeodesicOutputHandler, [60](#)
- m\_TotalPixels
  - InputCertainPixelsMesh, [71](#)
  - SimpleSquareMesh, [97](#)
- m\_Vielbein
  - ViewScreen, [127](#)
- m\_WriteToConsole
  - GeodesicOutputHandler, [60](#)
- main
  - Main.cpp, [161](#)
- Main.cpp
  - LoadPrecompiledOptions, [161](#)
  - main, [161](#)
- MankoNovikovMetric, [79](#)
  - getFullDescriptionStr, [80](#)
  - getMetric\_dd, [80](#)
  - getMetric\_uu, [80](#)
- m\_alpha3Param, [81](#)
- m\_alphaParam, [81](#)
- m\_aParam, [81](#)
- m\_kParam, [81](#)
- MankoNovikovMetric, [80](#)
- MaxLevel
  - InputOutput.h, [156](#)
- MaxSteps
  - TimeOutTermOptions, [122](#)
- MaxStreamSize
  - ConfigReader, [12](#)
- Maxterms
  - Terminations.h, [182](#)
- Mesh, [81](#)
  - ~Mesh, [82](#)
  - EndCurrentLoop, [82](#)
  - GeodesicFinished, [82](#)
  - getCurNrGeodesics, [82](#)
  - getFullDescriptionStr, [82](#)
  - getNewInitConds, [82](#)
  - IsFinished, [83](#)
  - m\_DistanceDiagnostic, [83](#)
  - Mesh, [82](#)
- Metric, [83](#)
  - ~Metric, [84](#)
  - getChristoffel\_udd, [84](#)
  - getFullDescriptionStr, [84](#)
  - getKretschmann, [84](#)
  - getMetric\_dd, [84](#)
  - getMetric\_uu, [84](#)
  - getRiemann\_uddd, [85](#)
  - getrLogScale, [85](#)
  - m\_rLogScale, [85](#)
  - m\_Symmetries, [85](#)
  - Metric, [84](#)
- NaN
  - Terminations.h, [182](#)
- NaNTermination, [85](#)
  - CheckTermination, [86](#)
  - getFullDescriptionStr, [86](#)
  - NaNTermination, [86](#)
  - TermOptions, [87](#)
- NaNTermOptions, [87](#)
  - NaNTermOptions, [88](#)
  - OutputToConsole, [88](#)
- NewGeodesicOutput
  - GeodesicOutputHandler, [58](#)
- NoSource, [88](#)
  - getFullDescriptionStr, [89](#)
  - getSource, [89](#)
  - NoSource, [89](#)
- NrSettings
  - ConfigReader::ConfigCollection, [26](#)
- Null
  - ViewScreen.h, [190](#)
- OneIndex
  - Geometry.h, [149](#)

- OpenForFirstTime
  - GeodesicOutputHandler, 59
- operator+
  - Geometry.h, 151
- operator-
  - Geometry.h, 151
- operator/
  - Geometry.h, 151
- operator=
  - Geodesic, 55
- operator[]
  - ConfigReader::ConfigCollection, 26
- operator\*
  - Geometry.h, 151
- Output\_Important\_Default
  - Config, 11
- Output\_Other\_Default
  - Config, 11
- OutputFinished
  - GeodesicOutputHandler, 59
- OutputLevel
  - InputOutput.h, 156
- OutputNrSteps
  - GeodesicPositionOptions, 64
- OutputToConsole
  - GeneralSingularityTermOptions, 53
  - NaNTermOptions, 88
- pi
  - Geometry.h, 152
- PIXEL\_MAX
  - Geometry.h, 149
- pixelcoord
  - Geometry.h, 150
- PixellInfo
  - SquareSubdivisionMesh::PixellInfo, 90
  - SquareSubdivisionMeshV2::PixellInfo, 91
- Point
  - Geometry.h, 150
- PrepareForOutput
  - GeodesicOutputHandler, 59
- RasheedLarsenMetric, 92
  - getFullDescriptionStr, 94
  - getMetric\_dd, 94
  - getMetric\_uu, 94
  - m\_aParam, 94
  - m\_mParam, 94
  - m\_pParam, 94
  - m\_qParam, 94
  - RasheedLarsenMetric, 93
- ReadCollection
  - ConfigReader::ConfigCollection, 26
- ReadFile
  - ConfigReader::ConfigCollection, 27
- ReadSettingName
  - ConfigReader::ConfigCollection, 27
- ReadSettingSpecificChar
  - ConfigReader::ConfigCollection, 27
- ReadSettingValue
  - ConfigReader::ConfigCollection, 27
- real
  - Geometry.h, 150
- RedShiftPower
  - EquatorialEmissionOptions, 38
- Reset
  - ClosestRadiusDiagnostic, 21
  - Diagnostic, 31
  - EquatorialEmissionDiagnostic, 36
  - EquatorialPassesDiagnostic, 40
  - FourColorScreenDiagnostic, 47
  - Geodesic, 55
  - GeodesicPositionDiagnostic, 62
  - Termination, 115
  - TimeOutTermination, 121
- reset
  - Utilities::Timer, 123
- RightNbr
  - SquareSubdivisionMeshV2::PixellInfo, 91
- RightNbrIndex
  - SquareSubdivisionMesh::PixellInfo, 90
- RLogScale
  - ClosestRadiusOptions, 23
  - EquatorialEmissionOptions, 38
- rLogScale
  - BoundarySphereTermOptions, 19
  - GeneralSingularityTermOptions, 53
  - HorizonTermOptions, 68
- ScreenIndex
  - Geometry.h, 150
- ScreenOutput
  - InputOutput.cpp, 155
  - InputOutput.h, 156
- ScreenPoint
  - Geometry.h, 150
- Second
  - Utilities::Timer, 123
- SEdiagNbr
  - SquareSubdivisionMeshV2::PixellInfo, 91
- SetLoopMessageFrequency
  - InputOutput.cpp, 155
  - InputOutput.h, 156
- SetNewInitialConditions
  - ViewScreen, 126
- SetOutputLevel
  - InputOutput.cpp, 155
  - InputOutput.h, 156
- SettingError
  - Config, 10
- SettingName
  - ConfigReader::ConfigCollection::ConfigSetting, 28
- SettingValue
  - ConfigReader::ConfigCollection::ConfigSetting, 28
- SimpleSquareMesh, 95
  - EndCurrentLoop, 96
  - GeodesicFinished, 96
  - getCurNrGeodesics, 96

- getFullDescriptionStr, 96
- getNewInitConds, 96
- IsFinished, 96
- m\_Finished, 97
- m\_RowColumnSize, 97
- m\_TotalPixels, 97
- SimpleSquareMesh, 95
- Singularities
  - GeneralSingularityTermOptions, 53
- Singularity
  - Geometry.h, 150
- SingularityCoord
  - Geometry.h, 150
- SingularityMetric, 97
  - getSingularities, 98
  - m\_AllSingularities, 98
  - SingularityMetric, 98
- SingularityToString
  - GeneralSingularityTermination, 52
- SmallestPossibleStepsize
  - Integrators, 13
- Source, 99
  - ~Source, 99
  - getFullDescriptionStr, 99
  - getSource, 99
  - m\_theMetric, 100
  - Source, 99
- Spacelike
  - ViewScreen.h, 190
- SphereRadius
  - BoundarySphereTermOptions, 19
- SphericalHorizonMetric, 100
  - getHorizonRadius, 101
  - m\_HorizonRadius, 101
  - SphericalHorizonMetric, 101
- SquareSubdivisionMesh, 102
  - EndCurrentLoop, 103
  - Explnt, 103
  - GeodesicFinished, 103
  - getCurNrGeodesics, 104
  - getFullDescriptionStr, 104
  - getNewInitConds, 104
  - InitializeFirstGrid, 104
  - IsFinished, 104
  - m\_AllPixels, 105
  - m\_CurrentPixelQueue, 105
  - m\_CurrentPixelQueueDone, 105
  - m\_InfinitePixels, 105
  - m\_InitialPixels, 105
  - m\_InitialSubDivideToFinal, 105
  - m\_IterationPixels, 105
  - m\_MaxPixels, 105
  - m\_MaxSubdivide, 105
  - m\_PixelsLeft, 106
  - m\_RowColumnSize, 106
  - SquareSubdivisionMesh, 103
  - SubdivideAndQueue, 104
  - UpdateAllNeighbors, 104
  - UpdateAllWeights, 105
- SquareSubdivisionMesh::PixelInfo, 89
  - DiagValue, 90
  - Index, 90
  - LowerNbrIndex, 90
  - PixelInfo, 90
  - RightNbrIndex, 90
  - SubdivideLevel, 90
  - Weight, 90
- SquareSubdivisionMeshV2, 106
  - EndCurrentLoop, 108
  - Explnt, 108
  - GeodesicFinished, 108
  - getCurNrGeodesics, 108
  - GetDown, 108
  - getFullDescriptionStr, 108
  - GetLeft, 109
  - getNewInitConds, 109
  - GetRight, 109
  - GetUp, 109
  - InitializeFirstGrid, 109
  - IsFinished, 109
  - m\_ActivePixels, 110
  - m\_AllPixels, 110
  - m\_CurrentPixelQueue, 110
  - m\_CurrentPixelQueueDone, 110
  - m\_CurrentPixelUpdating, 110
  - m\_InfinitePixels, 110
  - m\_InitialPixels, 110
  - m\_InitialSubDivideToFinal, 110
  - m\_IterationPixels, 110
  - m\_MaxPixels, 111
  - m\_MaxSubdivide, 111
  - m\_PixelsIntegrated, 111
  - m\_PixelsLeft, 111
  - m\_RowColumnSize, 111
  - SquareSubdivisionMeshV2, 107
  - SubdivideAndQueue, 109
  - UpdateAllWeights, 110
- SquareSubdivisionMeshV2::PixelInfo, 90
  - DiagValue, 91
  - DownNbr, 91
  - Index, 91
  - LeftNbr, 91
  - PixelInfo, 91
  - RightNbr, 91
  - SEdiagNbr, 91
  - SubdivideLevel, 91
  - UpNbr, 92
  - Weight, 92
- ST3CrMetric, 111
  - f\_om\_phi, 113
  - f\_phi, 113
  - get\_omega, 113
  - getFullDescriptionStr, 113
  - getMetric\_dd, 113
  - getMetric\_uu, 113
  - m\_lambda, 114

- m\_P, 114
  - m\_q0, 114
  - ST3CrMetric, 113
- SubdivideAndQueue
  - SquareSubdivisionMesh, 104
  - SquareSubdivisionMeshV2, 109
- SubdivideLevel
  - SquareSubdivisionMesh::PixelInfo, 90
  - SquareSubdivisionMeshV2::PixelInfo, 91
- Term
  - Terminations.h, 182
- Term\_BoundarySphere
  - Terminations.h, 183
- Term\_GeneralSingularity
  - Terminations.h, 183
- Term\_Horizon
  - Terminations.h, 183
- Term\_NaN
  - Terminations.h, 183
- Term\_None
  - Terminations.h, 183
- Term\_ThetaSingularity
  - Terminations.h, 183
- Term\_TimeOut
  - Terminations.h, 183
- TermBitflag
  - Terminations.h, 182
- Termination, 114
  - ~Termination, 115
  - CheckTermination, 115
  - DecideUpdate, 115
  - getFullDescriptionStr, 115
  - m\_OwnerGeodesic, 116
  - m\_StepsSinceUpdated, 116
  - Reset, 115
  - Termination, 115
- TerminationOptions, 116
  - ~TerminationOptions, 116
  - TerminationOptions, 116
  - UpdateEveryNSteps, 117
- Terminations.cpp
  - CreateTerminationVector, 181
- Terminations.h
  - BoundarySphere, 182
  - Continue, 182
  - CreateTerminationVector, 183
  - GeneralSingularity, 182
  - Horizon, 182
  - Maxterms, 182
  - NaN, 182
  - Term, 182
  - Term\_BoundarySphere, 183
  - Term\_GeneralSingularity, 183
  - Term\_Horizon, 183
  - Term\_NaN, 183
  - Term\_None, 183
  - Term\_ThetaSingularity, 183
  - Term\_TimeOut, 183
- TermBitflag, 182
- TerminationUniqueVector, 182
- ThetaSingularity, 182
- TimeOut, 182
- Uninitialized, 182
- TerminationUniqueVector
  - Terminations.h, 182
- TermOptions
  - BoundarySphereTermination, 18
  - GeneralSingularityTermination, 52
  - HorizonTermination, 67
  - NaNTermination, 87
  - ThetaSingularityTermination, 118
  - TimeOutTermination, 121
- TheEmissionModel
  - EquatorialEmissionOptions, 38
- TheFluidVelocityModel
  - EquatorialEmissionOptions, 38
- ThetaSingEpsilon
  - ThetaSingularityTermOptions, 119
- ThetaSingularity
  - Terminations.h, 182
- ThetaSingularityTermination, 117
  - CheckTermination, 118
  - getFullDescriptionStr, 118
  - TermOptions, 118
  - ThetaSingularityTermination, 118
- ThetaSingularityTermOptions, 119
  - ThetaSingEpsilon, 119
  - ThetaSingularityTermOptions, 119
- theUpdateFrequency
  - DiagnosticOptions, 32
- ThreeIndex
  - Geometry.h, 150
- Threshold
  - EquatorialPassesOptions, 42
- Timelike
  - ViewScreen.h, 190
- TimeOut
  - Terminations.h, 182
- TimeOutTermination, 120
  - CheckTermination, 121
  - getFullDescriptionStr, 121
  - m\_CurNrSteps, 121
  - Reset, 121
  - TermOptions, 121
  - TimeOutTermination, 121
- TimeOutTermOptions, 122
  - MaxSteps, 122
  - TimeOutTermOptions, 122
- tk, 14
- tk::internal, 14
- toString
  - Geometry.h, 151, 152
- trace
  - ConfigReader::ConfigReaderException, 28
- TwoIndex
  - Geometry.h, 150

- Uninitialized
  - Terminations.h, [182](#)
- Update
  - Geodesic, [56](#)
- UpdateAllNeighbors
  - SquareSubdivisionMesh, [104](#)
- UpdateAllWeights
  - SquareSubdivisionMesh, [105](#)
  - SquareSubdivisionMeshV2, [110](#)
- UpdateData
  - ClosestRadiusDiagnostic, [21](#)
  - Diagnostic, [31](#)
  - EquatorialEmissionDiagnostic, [36](#)
  - EquatorialPassesDiagnostic, [40](#)
  - FourColorScreenDiagnostic, [47](#)
  - GeodesicPositionDiagnostic, [62](#)
- UpdateEveryNSteps
  - TerminationOptions, [117](#)
- UpdateFinish
  - UpdateFrequency, [124](#)
- UpdateFrequency, [124](#)
  - UpdateFinish, [124](#)
  - UpdateNSteps, [124](#)
  - UpdateStart, [124](#)
- UpdateNSteps
  - UpdateFrequency, [124](#)
- UpdateStart
  - UpdateFrequency, [124](#)
- UpNbr
  - SquareSubdivisionMeshV2::PixelInfo, [92](#)
- Utilities, [14](#)
  - GetDiagNameStrings, [14](#)
  - GetFirstLineInfoString, [14](#)
  - GetTimeStampString, [14](#)
- Utilities::Timer, [123](#)
  - Clock, [123](#)
  - elapsed, [123](#)
  - m\_beg, [124](#)
  - reset, [123](#)
  - Second, [123](#)
- VerletVelocityTolerance
  - Integrators, [14](#)
- ViewScreen, [124](#)
  - ConstructVielbein, [126](#)
  - EndCurrentLoop, [126](#)
  - GeodesicFinished, [126](#)
  - getCurNrGeodesics, [126](#)
  - getFullDescriptionStr, [126](#)
  - IsFinished, [126](#)
  - m\_Direction, [126](#)
  - m\_GeodType, [126](#)
  - m\_Metric\_dd, [127](#)
  - m\_Pos, [127](#)
  - m\_rLogScale, [127](#)
  - m\_ScreenCenter, [127](#)
  - m\_ScreenSize, [127](#)
  - m\_theMesh, [127](#)
  - m\_theMetric, [127](#)
  - m\_Vielbein, [127](#)
  - SetNewInitialConditions, [126](#)
  - ViewScreen, [125](#)
- ViewScreen.h
  - GeodesicType, [190](#)
  - Null, [190](#)
  - Spacelike, [190](#)
  - Timelike, [190](#)
- Weight
  - SquareSubdivisionMesh::PixelInfo, [90](#)
  - SquareSubdivisionMeshV2::PixelInfo, [92](#)
- WriteCachedOutputToFile
  - GeodesicOutputHandler, [59](#)