FOORT

1.0

Generated by Doxygen 1.12.0

1 Namespace Index	1
1.1 Namespace List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Namespace Documentation	9
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	
5.2 ConfigReader Namespace Reference	
5.2.1 Variable Documentation	12
5.2.1.1 MaxStreamSize	12
5.3 Integrators Namespace Reference	12
5.3.1 Function Documentation	
5.3.1.1 GetAdaptiveStep()	
5.3.1.2 GetFullIntegratorDescription()	
5.3.1.3 IntegrateGeodesicStep_RK4()	
5.3.1.4 IntegrateGeodesicStep_Verlet()	
5.3.2 Variable Documentation	
5.3.2.1 delta_nodiv0	
5.3.2.2 Derivative_hval	
5.3.2.3 epsilon	_
5.3.2.4 IntegratorDescription	
Sieler integrators soonpation	

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
6 Class Documentation	15
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()	
6.6.2.24 ReadSettingValue()	
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
6.16.1 Constructor & Destructor Documentation
6.16.1.1 FlatSpaceMetric()
6.16.2 Member Function Documentation
6.16.2.1 getFullDescriptionStr()
6.16.2.2 getMetric_dd()
6.16.2.3 getMetric_uu()
6.17 FluidVelocityModel Struct Reference
6.17.1 Constructor & Destructor Documentation
6.17.1.1 FluidVelocityModel()
6.17.1.2 ~FluidVelocityModel()
6.17.2 Member Function Documentation
6.17.2.1 GetFourVelocityd()
6.17.2.2 getFullDescriptionStr()
6.17.3 Member Data Documentation
6.17.3.1 m_theMetric
6.18 FourColorScreenDiagnostic Class Reference
6.18.1 Constructor & Destructor Documentation
6.18.1.1 FourColorScreenDiagnostic()
6.18.2 Member Function Documentation
6.18.2.1 FinalDataValDistance()
6.18.2.2 getFinalDataVal()
6.18.2.3 getFullDataStr()
6.18.2.4 getFullDescriptionStr()
6.18.2.5 getNameStr()
6.18.2.6 Reset()
6.18.2.7 UpdateData()
6.18.3 Member Data Documentation
6.18.3.1 m_quadrant
6.19 GeneralCircularRadialFluid Struct Reference
6.19.1 Constructor & Destructor Documentation
6.19.1.1 GeneralCircularRadialFluid()
6.19.2 Member Function Documentation
6.19.2.1 FindISCO()
6.19.2.2 GetChristrRaisedDer()
6.19.2.3 GetCircularVelocityd()
6.19.2.4 GetFourVelocityd()
6.19.2.5 getFullDescriptionStr()
6.19.2.6 GetInsideISCOCircularVelocityd()
6.19.2.7 GetRadialVelocityd()
6.19.3 Member Data Documentation
6.19.3.1 m betaPhi

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. All Terminations	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 lsFinished()	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 ExpInt()	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_InitialSubDividideToFinal	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)7
	6.47.1.1 SquareSubdivisionMeshV2() [1/2])7
	6.47.1.2 SquareSubdivisionMeshV2() [2/2])8
6.47.	2 Member Function Documentation)8
	6.47.2.1 EndCurrentLoop())8
	6.47.2.2 ExpInt())8
	6.47.2.3 GeodesicFinished())8
	6.47.2.4 getCurNrGeodesics())8
	6.47.2.5 GetDown())8
	6.47.2.6 getFullDescriptionStr())9
	6.47.2.7 GetLeft())9
	6.47.2.8 getNewInitConds())9
	6.47.2.9 GetRight())9
	6.47.2.10 GetUp())9
	6.47.2.11 InitializeFirstGrid())9
	6.47.2.12 IsFinished())9
	6.47.2.13 SubdivideAndQueue()	10
	6.47.2.14 UpdateAllWeights()	10
6.47.	3 Member Data Documentation	10
	6.47.3.1 m_ActivePixels	10
	6.47.3.2 m_AllPixels	10
	6.47.3.3 m_CurrentPixelQueue	10
	6.47.3.4 m_CurrentPixelQueueDone	10
	6.47.3.5 m_CurrentPixelUpdating	10
	6.47.3.6 m_InfinitePixels	10
	6.47.3.7 m_InitialPixels	10
	6.47.3.8 m_InitialSubDividideToFinal	10
	6.47.3.9 m_IterationPixels	11
	6.47.3.10 m_MaxPixels	11
	6.47.3.11 m_MaxSubdivide	l 1
	6.47.3.12 m_PixelsIntegrated	11
	6.47.3.13 m_PixelsLeft	l 1
	6.47.3.14 m_RowColumnSize	11
6.48 ST3C	rMetric Class Reference	l 1
6.48.	1 Constructor & Destructor Documentation	13
	6.48.1.1 ST3CrMetric()	13
6.48.	2 Member Function Documentation	13
	6.48.2.1 f_om_phi()	13
	6.48.2.2 f_phi()	13
	6.48.2.3 get_omega()	13
	6.48.2.4 getFullDescriptionStr()	13
	6.48.2.5 getMetric_dd()	13

6.48.2.6 getMetric_uu()	114
6.48.3 Member Data Documentation	
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()	
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	
6.53.1.1 TimeOutTermination()	121
6.53.2 Member Function Documentation	121
6.53.2.1 CheckTermination()	121

6.53.2.2 getFullDescriptionStr()	
6.53.2.3 Reset()	
6.53.3 Member Data Documentation	21
6.53.3.1 m_CurNrSteps	21
6.53.3.2 TermOptions	21
6.54 TimeOutTermOptions Struct Reference	22
6.54.1 Constructor & Destructor Documentation	22
6.54.1.1 TimeOutTermOptions()	22
6.54.2 Member Data Documentation	22
6.54.2.1 MaxSteps	22
6.55 Utilities::Timer Class Reference	23
6.55.1 Member Typedef Documentation	23
6.55.1.1 Clock	23
6.55.1.2 Second	23
6.55.2 Member Function Documentation	23
6.55.2.1 elapsed()	23
6.55.2.2 reset()	23
6.55.3 Member Data Documentation	24
6.55.3.1 m_beg	24
6.56 UpdateFrequency Struct Reference	24
6.56.1 Member Data Documentation	24
6.56.1.1 UpdateFinish	24
6.56.1.2 UpdateNSteps	24
6.56.1.3 UpdateStart	24
6.57 ViewScreen Class Reference	24
6.57.1 Constructor & Destructor Documentation	25
6.57.1.1 ViewScreen() [1/2]	25
6.57.1.2 ViewScreen() [2/2]	25
6.57.2 Member Function Documentation	26
6.57.2.1 ConstructVielbein()	26
6.57.2.2 EndCurrentLoop()	26
6.57.2.3 GeodesicFinished()	26
6.57.2.4 getCurNrGeodesics()	26
6.57.2.5 getFullDescriptionStr()	26
6.57.2.6 lsFinished()	26
6.57.2.7 SetNewInitialConditions()	26
6.57.3 Member Data Documentation	26
6.57.3.1 m_Direction	26
6.57.3.2 m_GeodType	27
6.57.3.3 m_Metric_dd	27
6.57.3.4 m_Pos	27
6.57.3.5 m_rLogScale	27

	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
7	File Documentation	129
	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 LARGECOUNTER_MAX	149
	7.10.1.0 DIVEL MAY	1 10

7.16.2 Typedef Documentation	1	49
7.16.2.1 FourIndex	1	49
7.16.2.2 largecounter	1	49
7.16.2.3 OneIndex	1	50
7.16.2.4 pixelcoord	1	50
7.16.2.5 Point	1	50
7.16.2.6 real	1	50
7.16.2.7 ScreenIndex	1	50
7.16.2.8 ScreenPoint	1	50
7.16.2.9 Singularity	1	50
7.16.2.10 SingularityCoord	1	50
7.16.2.11 ThreeIndex	1	50
7.16.2.12 TwoIndex	1	50
7.16.3 Function Documentation	1	51
7.16.3.1 operator*() [1/2]	1	51
7.16.3.2 operator*() [2/2]	1	51
7.16.3.3 operator+()	1	51
7.16.3.4 operator-()	1	51
7.16.3.5 operator/()	1	51
7.16.3.6 toString() [1/3]	1	51
7.16.3.7 toString() [2/3]	1	52
7.16.3.8 toString() [3/3]	1	52
7.16.4 Variable Documentation	1	52
7.16.4.1 dimension	1	52
7.16.4.2 pi	1	52
7.17 Geometry.h	1	52
7.18 FOORT/src/Header.h File Reference	1	54
7.19 Header.h	1	54
7.20 FOORT/src/InputOutput.cpp File Reference	1	54
7.20.1 Function Documentation	1	55
7.20.1.1 GetLoopMessageFrequency()	1	55
7.20.1.2 ScreenOutput()	1	55
7.20.1.3 SetLoopMessageFrequency()	1	55
7.20.1.4 SetOutputLevel()	1	55
7.21 FOORT/src/InputOutput.h File Reference	1	55
7.21.1 Enumeration Type Documentation	1	56
7.21.1.1 OutputLevel	1	56
7.21.2 Function Documentation	1	56
7.21.2.1 GetLoopMessageFrequency()	1	56
7.21.2.2 ScreenOutput()	1	56
7.21.2.3 SetLoopMessageFrequency()	1	56
7.21.2.4 SetOutputLevel()	1	57

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

	 	 xxi
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
Index		193

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Config	
ConfigReader	13
Integrators	13
tk	1
tk::internal	1
Utilities	1

2 Namespace Index

Hierarchical Index

2.1 Class Hierarchy

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

ConfigReader::ConfigCollection
ConfigReader::ConfigCollection::ConfigSetting
Diagnostic
ClosestRadiusDiagnostic
EquatorialPassesDiagnostic
EquatorialEmissionDiagnostic
FourColorScreenDiagnostic
GeodesicPositionDiagnostic
DiagnosticOptions
ClosestRadiusOptions
EquatorialPassesOptions
EquatorialEmissionOptions
GeodesicPositionOptions
EmissionModel
GLMJohnsonSUEmission
FluidVelocityModel
GeneralCircularRadialFluid
Geodesic
GeodesicOutputHandler
Mesh
InputCertainPixelsMesh
SimpleSquareMesh
SquareSubdivisionMesh
SquareSubdivisionMeshV2
Metric
BosonStarMetric
FlatSpaceMetric
SingularityMetric
ST3CrMetric
SphericalHorizonMetric
JohannsenMetric
KerrMetric
KerrSchildMetric

4 Hierarchical Index

MankoNovikovMetric	79
RasheedLarsenMetric	92
SquareSubdivisionMesh::PixelInfo	. 89
SquareSubdivisionMeshV2::PixelInfo	
std::runtime error	
ConfigReader::ConfigReaderException	27
Source	
NoSource	88
Termination	. 114
BoundarySphereTermination	17
GeneralSingularityTermination	
HorizonTermination	
NaNTermination	
ThetaSingularityTermination	
TimeOutTermination	
TerminationOptions	. 116
BoundarySphereTermOptions	18
GeneralSingularityTermOptions	
HorizonTermOptions	
NaNTermOptions	
ThetaSingularityTermOptions	
TimeOutTermOptions	
Utilities::Timer	. 123
UpdateFrequency	. 124
ViewScreen	. 124

Class Index

3.1 Class List

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

BosonStarMetric
BoundarySphereTermination
BoundarySphereTermOptions
ClosestRadiusDiagnostic
ClosestRadiusOptions
ConfigReader::ConfigCollection
ConfigReader::ConfigReaderException
ConfigReader::ConfigCollection::ConfigSetting
Diagnostic
DiagnosticOptions
EmissionModel
EquatorialEmissionDiagnostic
EquatorialEmissionOptions
EquatorialPassesDiagnostic
EquatorialPassesOptions
FlatSpaceMetric
FluidVelocityModel
FourColorScreenDiagnostic
GeneralCircularRadialFluid
GeneralSingularityTermination
GeneralSingularityTermOptions
Geodesic
GeodesicOutputHandler
GeodesicPositionDiagnostic
GeodesicPositionOptions
GLMJohnsonSUEmission
HorizonTermination
HorizonTermOptions
InputCertainPixelsMesh
JohannsenMetric
KerrMetric
KerrSchildMetric
MankoNovikovMetric
Mesh 8
Metric 8

6 Class Index

NaNTermination	85
NaNTermOptions	87
NoSource	88
SquareSubdivisionMesh::PixelInfo	89
SquareSubdivisionMeshV2::PixelInfo	90
RasheedLarsenMetric	92
SimpleSquareMesh	95
SingularityMetric	97
Source	99
SphericalHorizonMetric	00
SquareSubdivisionMesh	02
SquareSubdivisionMeshV2	06
ST3CrMetric	11
Termination	14
TerminationOptions	16
ThetaSingularityTermination	17
ThetaSingularityTermOptions	19
TimeOutTermination	20
TimeOutTermOptions	22
Utilities::Timer	23
UpdateFrequency	24
ViewScreen	24

File Index

4.1 File List

Here is a list of all files with brief descriptions:

FOORT/src/Config.cpp
FOORT/src/Config.h
FOORT/src/ConfigReader.cpp
FOORT/src/ConfigReader.h
FOORT/src/Diagnostics.cpp
FOORT/src/Diagnostics.h
FOORT/src/DiagnosticsEmission.cpp
FOORT/src/DiagnosticsEmission.h
FOORT/src/Geodesic.cpp
FOORT/src/Geodesic.h
FOORT/src/Geometry.h
FOORT/src/Header.h
FOORT/src/InputOutput.cpp
FOORT/src/InputOutput.h
FOORT/src/Integrators.cpp
FOORT/src/Integrators.h
FOORT/src/Main.cpp
FOORT/src/Mesh.cpp
FOORT/src/Mesh.h
FOORT/src/Metric.cpp
FOORT/src/Metric.h
FOORT/src/Spline.h
FOORT/src/Terminations.cpp
FOORT/src/Terminations.h
FOORT/src/Utilities.cpp
FOORT/src/Utilities.h
FOORT/src/ViewScreen.cpp
FOORT/src/ViewScreen.h

8 File Index

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 the
 Metric)

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()

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

5.1.2.2 GetMesh()

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()

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

5.1.2.4 GetOutputHandler()

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

5.1.2.5 GetSource()

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

5.1.2.6 GetViewScreen()

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

5.1.2.7 InitializeDiagnostics()

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()

5.1.2.9 InitializeTerminations()

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_O_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()

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()

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 Integrator Description

```
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()

5.6.1.2 GetFirstLineInfoString()

5.6.1.3 GetTimeStampString()

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

Other functions in Utilities.

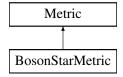
Chapter 6

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 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()

Implements Metric.

6.1.2.3 getMetric_uu()

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]
```

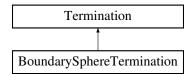
The documentation for this class was generated from the following files:

- 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

 ${\it Boundary Sphere Termination functions}.$

• 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 \sim Termination ()=default

Static Public Attributes

 $\bullet \ \ static \ std:: unique_ptr < BoundarySphereTermOptions > TermOptions \\$

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()

```
\begin{tabular}{ll} {\tt BoundarySphereTermination::} {\tt BoundarySphereTermination (} \\ {\tt Geodesic *const $the Geodesic$)} & [inline] \end{tabular}
```

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]
```

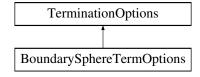
The documentation for this class was generated from the following files:

- 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()

6.3.2 Member Data Documentation

6.3.2.1 rLogScale

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

6.3.2.2 SphereRadius

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

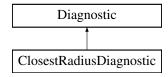
The documentation for this struct was generated from the following file:

· 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()

6.4.2 Member Function Documentation

6.4.2.1 FinalDataValDistance()

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

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

6.4.3.2 m_ClosestRadius

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

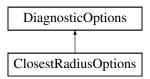
The documentation for this class was generated from the following files:

- 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::st

bool LookupValue (std::string_view SettingName, OutputType &theOutput) const

- $\bullet \ \ \mathsf{template}{<}\mathsf{class} \ \mathsf{OutputType}>$
 - 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()

6.6.2.2 DisplaySetting()

6.6.2.3 DisplayTabs()

6.6.2.4 Exists()

6.6.2.5 GetSettingIndex()

6.6.2.6 IsCollection() [1/2]

6.6.2.7 IsCollection() [2/2]

6.6.2.8 LookupValue() [1/2]

6.6.2.9 LookupValue() [2/2]

6.6.2.10 LookupValueInteger() [1/7]

6.6.2.11 LookupValueInteger() [2/7]

6.6.2.12 LookupValueInteger() [3/7]

6.6.2.13 LookupValueInteger() [4/7]

6.6.2.14 LookupValueInteger() [5/7]

6.6.2.15 LookupValueInteger() [6/7]

6.6.2.16 LookupValueInteger() [7/7]

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]

6.6.2.20 ReadCollection()

6.6.2.21 ReadFile()

6.6.2.22 ReadSettingName()

6.6.2.23 ReadSettingSpecificChar()

6.6.2.24 ReadSettingValue()

6.6.3 Member Data Documentation

6.6.3.1 m Settings

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

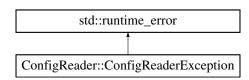
The documentation for this class was generated from the following files:

- 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()

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

6.8.1.2 SettingValue

ConfigSettingValue ConfigReader::ConfigCollection::ConfigSetting::SettingValue

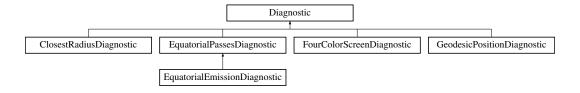
The documentation for this struct was generated from the following file:

• 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]

6.9.1.3 ∼Diagnostic()

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

6.9.2 Member Function Documentation

6.9.2.1 DecideUpdate()

6.9.2.2 FinalDataValDistance()

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]
```

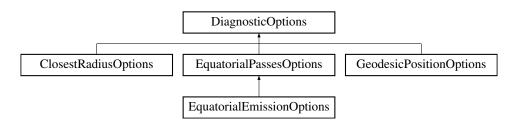
The documentation for this class was generated from the following files:

- 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()

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
```

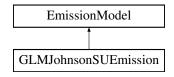
The documentation for this struct was generated from the following file:

• 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
- $\bullet \ \ virtual \ std::string \ \underline{getFullDescriptionStr} \ () \ const$

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()

Implemented in GLMJohnsonSUEmission.

6.11.2.2 getFullDescriptionStr()

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

Emission model functions.

Reimplemented in GLMJohnsonSUEmission.

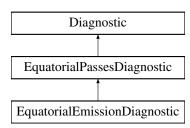
The documentation for this struct was generated from the following files:

- · 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()

6.12.2 Member Function Documentation

6.12.2.1 FinalDataValDistance()

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]
```

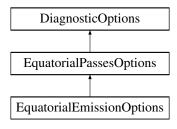
The documentation for this class was generated from the following files:

- 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)

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()

6.13.2 Member Data Documentation

6.13.2.1 EquatPassUpperBound

const int EquatorialEmissionOptions::EquatPassUpperBound

6.13.2.2 GeometricFudgeFactor

 $\verb|const|| \verb|real|| \verb| EquatorialEmissionOptions:: GeometricFudgeFactor||$

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

 $\verb|const| std::unique_ptr<| Fluid Velocity Model| > Equatorial Emission Options:: The Fluid Velocity Model| > The$

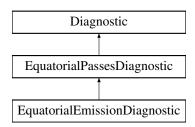
The documentation for this struct was generated from the following file:

• 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()

6.14.2 Member Function Documentation

6.14.2.1 FinalDataValDistance()

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]
```

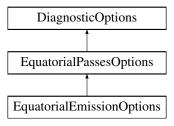
The documentation for this class was generated from the following files:

- FOORT/src/Diagnostics.h
- FOORT/src/Config.cpp
- FOORT/src/Diagnostics.cpp

6.15 Equatorial Passes Options 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()

6.15.2 Member Data Documentation

6.15.2.1 Threshold

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

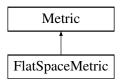
The documentation for this struct was generated from the following file:

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 functions.

6.16.2 Member Function Documentation

6.16.2.1 getFullDescriptionStr()

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

Reimplemented from Metric.

6.16.2.2 getMetric_dd()

Implements Metric.

6.16.2.3 getMetric_uu()

Implements Metric.

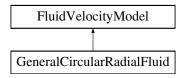
The documentation for this class was generated from the following files:

- 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()

6.17.1.2 ∼FluidVelocityModel()

```
\verb|virtual FluidVelocityModel::\sim FluidVelocityModel () [virtual], [default]|\\
```

6.17.2 Member Function Documentation

6.17.2.1 GetFourVelocityd()

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]
```

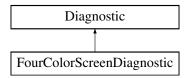
The documentation for this struct was generated from the following files:

- 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()

Implements Diagnostic.

6.18.2.2 getFinalDataVal()

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

6.18.2.3 getFullDataStr()

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

Implements Diagnostic.

6.18.2.4 getFullDescriptionStr()

 $\verb|std::string FourColorScreenDiagnostic::getFullDescriptionStr () const [final], [virtual]|\\$

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]
```

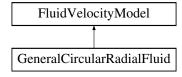
The documentation for this class was generated from the following files:

- 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)
- $\bullet \ \ \mathsf{virtual} \sim \mathsf{FluidVelocityModel} \ (\mathsf{)} \text{=} \mathsf{default}$

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 GetChristrRaisedDer (real r) const

Private Attributes

- const real m_subKeplerParam
- const real m betaR
- const real m_betaPhi
- bool m_ISCOexists {false}
- real m_ISCOr {-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 GetChristrRaisedDer()

6.19.2.3 GetCircularVelocityd()

6.19.2.4 GetFourVelocityd()

Implements FluidVelocityModel.

6.19.2.5 getFullDescriptionStr()

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

FluidVelocityModel functions.

Reimplemented from FluidVelocityModel.

6.19.2.6 GetInsideISCOCircularVelocityd()

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]
```

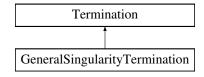
The documentation for this struct was generated from the following files:

- 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()

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()

```
\begin{tabular}{lll} {\bf std::string GeneralSingularityTermination::SingularityToString (} \\ & & int \ singnr) \ const \ \ [private] \end{tabular}
```

6.20.3 Member Data Documentation

6.20.3.1 TermOptions

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

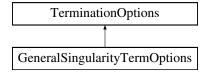
The documentation for this class was generated from the following files:

- 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 \sim 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]

6.22.1.3 Geodesic() [3/3]

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()

```
\verb|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=()

6.22.2.9 Reset()

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 functions.

6.23.2 Member Function Documentation

6.23.2.1 GetFileName()

6.23.2.2 getFullDescriptionStr()

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

6.23.2.3 NewGeodesicOutput()

6.23.2.4 OpenForFirstTime()

6.23.2.5 OutputFinished()

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

6.23.2.6 PrepareForOutput()

6.23.2.7 WriteCachedOutputToFile()

```
\verb"void GeodesicOutputHandler:: \verb"WriteCachedOutputToFile" () \\ \  \  [private]
```

6.23.3 Member Data Documentation

6.23.3.1 m_AllCachedData

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

 $\verb|const std::string GeodesicOutputHandler::m_TimeStamp [private]|\\$

6.23.3.13 m_WriteToConsole

bool GeodesicOutputHandler::m_WriteToConsole {false} [private]

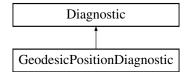
The documentation for this class was generated from the following files:

- FOORT/src/InputOutput.h
- FOORT/src/InputOutput.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()

6.24.2 Member Function Documentation

6.24.2.1 FinalDataValDistance()

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()

```
\verb|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]

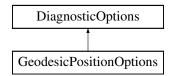
The documentation for this class was generated from the following files:

- 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()

6.25.2 Member Data Documentation

6.25.2.1 OutputNrSteps

```
\verb|const| | \texttt{largecounter}| | GeodesicPositionOptions::OutputNrSteps| |
```

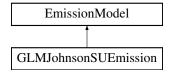
The documentation for this struct was generated from the following file:

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 \sim 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()

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]
```

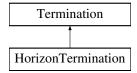
The documentation for this struct was generated from the following files:

- 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 \sim 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()

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]
```

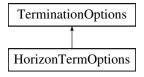
The documentation for this class was generated from the following files:

- 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()

6.28.2 Member Data Documentation

6.28.2.1 AtHorizonEps

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

6.28.2.2 HorizonRadius

```
\verb|const| real | \verb|HorizonTermOptions:: \verb|HorizonRadius||
```

6.28.2.3 rLogScale

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

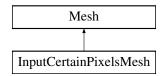
The documentation for this struct was generated from the following file:

• 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]

 $\label{local_continuity} Input Certain Pixels Mesh \ \ () \quad [delete]$

6.29.1.2 InputCertainPixelsMesh() [2/3]

6.29.1.3 InputCertainPixelsMesh() [3/3]

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()

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()

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]
```

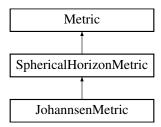
The documentation for this class was generated from the following files:

- 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 JohannsenMetric() [1/2]

```
JohannsenMetric::JohannsenMetric () [delete]
```

6.30.1.2 JohannsenMetric() [2/2]

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

JohannsenMetric functions (implementation by Seppe Staelens)

6.30.2 Member Function Documentation

6.30.2.1 getFullDescriptionStr()

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

Reimplemented from Metric.

6.30.2.2 getMetric_dd()

Implements Metric.

6.30.2.3 getMetric_uu()

Implements Metric.

6.30.3 Member Data Documentation

6.30.3.1 m_alpha13Param

```
const real JohannsenMetric::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]
```

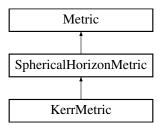
The documentation for this class was generated from the following files:

- 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()

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

Reimplemented from Metric.

6.31.2.2 getMetric_dd()

Implements Metric.

6.31.2.3 getMetric_uu()

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]
```

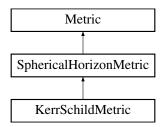
The documentation for this class was generated from the following files:

- 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 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()

Implements Metric.

6.32.2.3 getMetric_uu()

Implements Metric.

6.32.3 Member Data Documentation

6.32.3.1 m_aParam

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

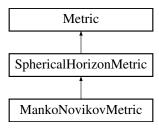
The documentation for this class was generated from the following files:

- 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()

Implements Metric.

6.33.2.3 getMetric_uu()

Implements Metric.

6.34 Mesh Class Reference 81

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]
```

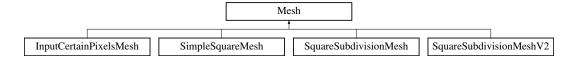
The documentation for this class was generated from the following files:

- 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()

6.34.2 Member Function Documentation

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

6.34.2.1 EndCurrentLoop()

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

Implemented in InputCertainPixelsMesh, SimpleSquareMesh, SquareSubdivisionMesh, and SquareSubdivisionMeshV2.

6.34.2.2 GeodesicFinished()

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.

 $Reimplemented \ in \ Input Certain Pixels Mesh, Simple Square Mesh, Square Subdivision Mesh, and Square Subdivision Mesh V2.$

6.34.2.5 getNewInitConds()

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 (abstract base class) functions.

6.35.2 Member Function Documentation

6.35.2.1 getChristoffel_udd()

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()

6.35.2.4 getMetric_dd()

Implemented in BosonStarMetric, FlatSpaceMetric, JohannsenMetric, KerrMetric, KerrSchildMetric, MankoNovikovMetric, RasheedLarsenMetric, and ST3CrMetric.

6.35.2.5 getMetric_uu()

Implemented in BosonStarMetric, FlatSpaceMetric, JohannsenMetric, KerrMetric, KerrSchildMetric, MankoNovikovMetric, RasheedLarsenMetric, and ST3CrMetric.

6.35.2.6 getRiemann_uddd()

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]
```

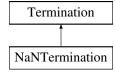
The documentation for this class was generated from the following files:

- 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]
```

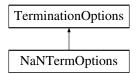
The documentation for this class was generated from the following files:

- 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()

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()

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()

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()

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()

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}
```

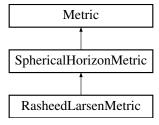
The documentation for this struct was generated from the following file:

• 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]

```
{\tt 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()

Implements Metric.

6.41.2.3 getMetric_uu()

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]
```

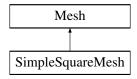
The documentation for this class was generated from the following files:

- 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 \sim 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]

6.42.2 Member Function Documentation

6.42.2.1 EndCurrentLoop()

```
void SimpleSquareMesh::EndCurrentLoop () [final], [virtual]
Implements Mesh.
```

6.42.2.2 GeodesicFinished()

Implements Mesh.

6.42.2.3 getCurNrGeodesics()

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

Implements Mesh.

6.42.2.4 getFullDescriptionStr()

```
\verb|std::string SimpleSquareMesh::getFullDescriptionStr () const [final], [virtual]|\\
```

Mesh (abstract base class) functions.

Reimplemented from Mesh.

6.42.2.5 getNewInitConds()

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]
```

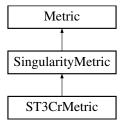
The documentation for this class was generated from the following files:

- 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()

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()

Implemented in NoSource.

6.44.3 Member Data Documentation

6.44.3.1 m_theMetric

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

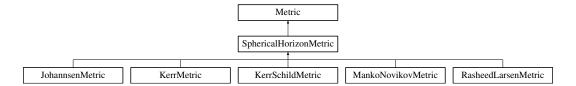
The documentation for this class was generated from the following files:

- 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 functions.

6.45.2 Member Function Documentation

6.45.2.1 getHorizonRadius()

```
{\tt real \ Spherical Horizon Metric::} {\tt get Horizon Radius \ () \ const}
```

6.45.3 Member Data Documentation

6.45.3.1 m_HorizonRadius

```
const real SphericalHorizonMetric::m_HorizonRadius [protected]
```

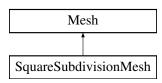
The documentation for this class was generated from the following files:

- 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 \sim Mesh ()=default

Private Member Functions

- void InitializeFirstGrid ()
- void UpdateAllNeighbors ()
- void UpdateAllWeights ()
- · void SubdivideAndQueue (largecounter ind)
- pixelcoord ExpInt (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_InitialSubDividideToFinal
- · 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]

6.46.2 Member Function Documentation

6.46.2.1 EndCurrentLoop()

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

6.46.2.2 ExpInt()

Implements Mesh.

6.46.2.3 GeodesicFinished()

Implements Mesh.

6.46.2.4 getCurNrGeodesics()

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

 ${\bf Square Subdivision Mesh\ functions.}$

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()

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()

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

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

```
\verb|const large counter Square Subdivision Mesh:: \verb|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]

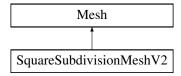
The documentation for this class was generated from the following files:

- 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 ExpInt (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_InitialSubDividideToFinal
- 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]

 ${\tt SquareSubdivisionMeshV2::} {\tt SquareSubdivisionMeshV2} \ \ () \quad [{\tt delete}]$

6.47.1.2 SquareSubdivisionMeshV2() [2/2]

6.47.2 Member Function Documentation

6.47.2.1 EndCurrentLoop()

```
void SquareSubdivisionMeshV2::EndCurrentLoop () [final], [virtual]
Implements Mesh.
```

6.47.2.2 ExpInt()

6.47.2.3 GeodesicFinished()

Implements Mesh.

6.47.2.4 getCurNrGeodesics()

```
largecounter SquareSubdivisionMeshV2::getCurNrGeodesics () const [final], [virtual]
```

SquareSubdivisionMeshV2 functions.

Implements Mesh.

6.47.2.5 GetDown()

6.47.2.6 getFullDescriptionStr()

```
\verb|std::string SquareSubdivisionMeshV2::getFullDescriptionStr () const [final], [virtual]|\\
```

Mesh (abstract base class) functions.

Reimplemented from Mesh.

6.47.2.7 GetLeft()

```
\label{lem:squareSubdivisionMeshV2::GetLeft (pixelInfo * SquareSubdivisionMeshV2::GetLeft (pixelInfo * p, int subdiv) const [private]
```

6.47.2.8 getNewInitConds()

Implements Mesh.

6.47.2.9 GetRight()

6.47.2.10 GetUp()

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()

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

6.47.3.3 m_CurrentPixelQueue

6.47.3.4 m_CurrentPixelQueueDone

6.47.3.5 m_CurrentPixelUpdating

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

 $\verb|const| bool SquareSubdivisionMeshV2::m_InitialSubDividideToFinal [private]| \\$

6.47.3.9 m_IterationPixels

 $\verb|const| | \texttt{largecounter}| | Square Subdivision Mesh V2::m_Iteration Pixels | [private]|$

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]

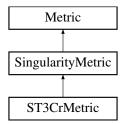
The documentation for this class was generated from the following files:

- 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 I) const
- real f_phi (real phi, real r, real theta, real I, real R) const
- real f_om_phi (real phi, real r, real theta, real I, 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 functions.

6.48.2 Member Function Documentation

6.48.2.1 f_om_phi()

6.48.2.2 f_phi()

6.48.2.3 get_omega()

6.48.2.4 getFullDescriptionStr()

```
std::string ST3CrMetric::getFullDescriptionStr () const [final], [virtual]
```

Reimplemented from Metric.

6.48.2.5 getMetric_dd()

Implements Metric.

6.48.2.6 getMetric_uu()

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()

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]
```

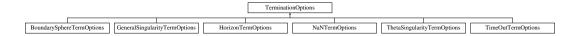
The documentation for this class was generated from the following files:

- 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()

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
```

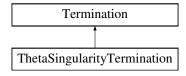
The documentation for this struct was generated from the following file:

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 \sim Termination ()=default

Static Public Attributes

 $\bullet \ \ static \ std::unique_ptr < ThetaSingularityTermOptions > TermOptions \\$

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()

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]
```

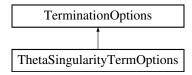
The documentation for this class was generated from the following files:

- 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()

6.52.2 Member Data Documentation

6.52.2.1 ThetaSingEpsilon

```
const real ThetaSingularityTermOptions::ThetaSingEpsilon
```

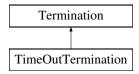
The documentation for this struct was generated from the following file:

• 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()

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:

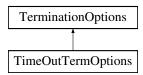
- FOORT/src/Terminations.h
- FOORT/src/Config.cpp
- FOORT/src/Terminations.cpp

122 Class Documentation

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()

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.

124 Class Documentation

6.55.3 Member Data Documentation

6.55.3.1 m_beg

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]

126 Class Documentation

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()

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()

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

128 Class Documentation

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 the
 Metric)

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 131

7.3 Config.h

```
Go to the documentation of this file.
00001 #ifndef _FOORT_CONFIG_H
00002 #define _FOORT_CONFIG_H
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"
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
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 {
             Output level for important missing information that will default
00038
          // (e.g. no metric selected, no diagnostics selected, ...)
constexpr auto Output_Important_Default = OutputLevel::Level_0_WARNING;
00039
00040
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
00054
00055
          // Use configuration to initialize the screen output level
00056
          void InitializeScreenOutput(const ConfigCollection &theCfg);
00057
00058
           // Use configuration to create the correct Metric with specified parameters
00059
          std::unique_ptr<Metric> GetMetric(const ConfigCollection &theCfg);
00060
00061
           // Use configuration to create the correct Source with specified parameters
00062
          std::unique ptr<Source> GetSource(const ConfigCollection &theCfq, const Metric *const theMetric);
00063
00064
           // Use configuration to set the Diagnostics bitflag appropriately;
00065
          // initialize all DiagnosticOptions for all Diagnostics that are turned on;
          // and set bitflag for diagnostic to be used for coarseness evaluating in Mesh void InitializeDiagnostics(const ConfigCollection &theCfg, DiagBitflag &alldiags, DiagBitflag
00066
00067
      &valdiag, const Metric *const theMetric);
00068
00069
           // Use configuration to set the Terminations bitflag appropriately;
00070
           // initialize all TerminationOptions for all Terminations that are turned on;
00071
          void InitializeTerminations(const ConfigCollection &theCfg, TermBitflag &allterms, const Metric
      *const theMetric);
00072
00073
           // Use configuration to create ViewScreen appropriately
          std::unique_ptr<ViewScreen> GetViewScreen(const ConfigCollection &theCfg, DiagBitflag valdiag,
      const Metric *const theMetric);
00075
          // GetViewScreen calls GetMesh to create the correct Mesh
00076
          std::unique_ptr<Mesh> GetMesh(const ConfigCollection &theCfg, DiagBitflag valdiag);
00077
00078
           // Use configuration to return a pointer to the correct integration function to use
          GeodesicIntegratorFunc GetGeodesicIntegrator(const ConfigCollection &theCfg);
00080
00081
          // Use configuration to initialize the output handler
00082
          std::unique_ptr<GeodesicOutputHandler> GetOutputHandler(const ConfigCollection &theCfg,
00083
                                                                      DiagBitflag alldiags, DiagBitflag valdiag,
      std::string FirstLineInfo);
00084
00085 } // end namespace Config
```

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

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
                          // std::string
00005 #include <string>
                              // needed for file and console output
// needed for file input
00006 #include <iostream>
00007 #include <fstream>
                              // needed to create vectors of Settings
00008 #include <vector>
                              // needed for std::variant
00009 #include <variant>
                              // needed for exceptions
00010 #include <stdexcept>
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:
              ConfigReaderException(const std::string &error, std::vector<int> settingtrace = {})
00024
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
          // A collection of configuration settings
00039
00040
          class ConfigCollection
00041
00042
00044
              // Returns true if the collection contains a setting with the given name
00045
              bool Exists(std::string_view SettingName) const;
00046
               \ensuremath{//} Returns total number of settings in this collection
00047
              int NrSettings() const;
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
00060
               // Generic functions that look up the value of a given setting
00061
               // and put it in the output variable
               // Returns true if successful, false if unsuccesful
00062
00063
               // (either due to type mismatch of output and setting value, or if setting doesn't exist)
               // Templated functions implemented below in .h file!
00064
00065
               template <class OutputType>
00066
               bool LookupValue(std::string_view SettingName, OutputType &theOutput) const;
00067
               template <class OutputType>
00068
               bool LookupValue(int SettingIndex, OutputType &theOutput) const;
00069
00070
               // Specific functions for looking up signed integral types
               // 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 // (so e.g. for long long, first long long is tried, then long, then int)
00072
00073
               bool LookupValueInteger(std::string_view SettingName, int &theOutput) const;
00075
               bool LookupValueInteger(std::string_view SettingName, long &theOutput) const;
00076
              bool LookupValueInteger(std::string_view SettingName, long long &theOutput) const;
              // Specific functions for looking up unsigned integral types // These will first test if the setting value is of the signed case, if it is and it is >=0
00077
00078
     then it always fits in
              // the unsigned version; then it will test larger (signed) types and see if the result still
      fits in the unsigned version
              bool LookupValueInteger(std::string_view SettingName, unsigned int &theOutput) const;
00080
00081
              bool LookupValueInteger(std::string_view SettingName, unsigned long &theOutput) const;
```

```
bool LookupValueInteger(std::string_view SettingName, unsigned long long &theOutput) const;
               // Templated functions implemented below in .h file!
00083
00084
               template <class OutputType>
               bool LookupValueInteger(int SettingIndex, OutputType &theOutput) const;
00085
00086
00088
               void DisplayCollection(std::ostream &OutputStream, int Indent = 0) const;
00089
00091
               // this overwrites the current collection with the contents of the config file
00092
               // Will throw ConfigReaderException if parse/syntax error in configuration file
00093
               // Returns true if successful
00094
               bool ReadFile(const std::string &FileName);
00095
00096
          private:
00097
               // A configuration setting value can be an integral type, bool, double, string, or a
      (sub)collection of settings
00098
               // Note: when reading in a configuration file, an integral value will always be stored in the
     smallest possible type!
00099
              using ConfigSettingValue = std::variant<br/>bool, int, long, long long,
00100
00101
                                                          double,
00102
                                                          std::string,
00103
                                                          std::unique_ptr<ConfigCollection»;</pre>
               \ensuremath{//} A setting is a name and setting value
00104
00105
               struct ConfigSetting
00106
               {
00107
                   std::string SettingName;
00108
                   ConfigSettingValue SettingValue;
00109
               }; // The vector of all settings in this collection
00110
               std::vector<ConfigSetting> m_Settings{};
00111
00112
00113
               // Helper function that returns the index of the setting with the given name
00114
               // returns -1 if setting not found
00115
               int GetSettingIndex(std::string_view SettingName) const;
00116
00117
               // Helper functions for DisplayCollection
00118
               // DisplaySetting displays one setting; if it's a subcollection then it calls
               // DisplayCollection on the subcollection
00119
00120
               void DisplaySetting(std::ostream &OutputStream, int SettingIndex, int Indent) const;
00121
               // Output given number of tabs
00122
               void DisplayTabs(std::ostream &OutputStream, int NrTabs) const;
00123
00124
               // Helper functions for ReadFile()
00125
               // These all take an ifstream at a given location and will read in one specific object
00126
00127
               // Read in entire collection (including subcollections)
               // Pre: stream is positioned right after { (or at beginning of file)
// Post: stream is positioned right after } (or at EOF)
00128
00129
               void ReadCollection(std::ifstream &InputFile);
00130
00131
               // Read in a setting name
00132
               // Pre: next non-whitespace character in stream is beginning of name
00133
               // Post: stream has just read in all characters of name (but no more);
00134
               // returns "" if there is no more setting to read in the current collection, in this case
00135
               // '}' has been read in (for subcollections)
               void ReadSettingName(std::ifstream &InputFile, std::string &theName);
// Read in one specific character only (not '/'!)
// Pre: next non-whitespace character in stream is this character
00136
00137
00138
00139
               // Post: stream is just after character
00140
               void ReadSettingSpecificChar(std::ifstream &InputFile, char theChar) const;
00141
               \//\ \mbox{Read} in setting value (could be subcollection)
               // Pre: next non-whitespace character in stream is beginning of value
00142
               // (can be '{' for subcollection, digit for number, '"' for string, or 't'/'f' for boolean)
00143
00144
               // Post: Value has entirely been read in (i.e. next character should be ';')
               void ReadSettingValue(std:ifstream &InputFile, ConfigSettingValue &theValue);
00145
00146
          };
00147
00148 } // end namespace declarations
00149
00152
00153 template <class OutputType>
00154 bool ConfigReader::ConfigCollection::LookupValue(int SettingIndex, OutputType &theOutput) const
00155 {
00156
           // Get the setting value, if it is indeed of this type
          const OutputType *OutputPointer{std::get_if<OutputType>(&m_Settings[SettingIndex].SettingValue)};
00157
00158
          if (OutputPointer)
00159
00160
               // Success, the setting is indeed of this type; set the output accordingly
00161
               theOutput = *OutputPointer;
               return true:
00162
00163
          }
          return false;
00164
00165 }
00166
00167 template <class OutputType>
00168 bool ConfigReader::ConfigCollection::LookupValue(std::string_view SettingName, OutputType &theOutput)
      const
00169 {
```

```
// 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
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()

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 {0b0000'0000'0000'00001}
- 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 {0b0000'0000'00001'0000}

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()

Diagnostic helper function.

7.8.3 Variable Documentation

7.8.3.1 Diag_ClosestRadius

```
DiagBitflag Diag_ClosestRadius {Ob0000'0000'0000'1000} [constexpr]
```

7.8.3.2 Diag_EquatorialEmission

```
DiagBitflag Diag_EquatorialEmission {Ob0000'0000'0001'0000} [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 {Ob0000'0000'0000'0001} [constexpr]
```

7.8.3.6 **Diag_None**

```
DiagBitflag Diag_None {Ob0000'0000'0000'0000} [constexpr]
```

7.9 Diagnostics.h

geodesic.

virtual void UpdateData() = 0;

00085

```
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
00016 #include <cstdint> // 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
           owner Geodesic
00022 // (note "Geodesic.h" is NOT included to avoid header loop, and we do not need Geodesic member
           functions here!)
00023 class Geodesic;
00024
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
{\tt 00032~//~Note:~why~not~std::bitset < size >?~Because~in~this~way~we~can~use~expressions~with~DiagBitflag~in~outled the contraction of the con
          conditional expressions.
00033 // e.g. if ( mydiagbitflag & Diag_FourColorScreen)
00034 using DiagBitflag = std::uint16_t;
00035
00036 // Define a bitflag per existing diagnostic
00037 constexpr DiagBitflag Diag_None(0b0000'0000'0000', 0000' 0000');
00038 constexpr DiagBitflag Diag_GeodesicPosition(0b0000'0000'0000', 0000');
00039 constexpr DiagBitflag Diag_FourColorScreen{0b0000'0000'0000'0010};
00040 constexpr DiagBitflag Diag_EquatorialPasses{0b0000'0000'0000'0100};
00041 constexpr DiagBitflag Diag_ClosestRadius{0b0000'0000'0000'1000};
00042 constexpr DiagBitflag Diag_EquatorialEmission{0b0000'0000'0001'0000};
00043
00045 // Add a DiagBitflag for your new diagnostic. Make sure you use a bitflag that has not been used
          before!
00046 // Sample code:
00047 /*
00048 constexpr DiagBitflag Diag_MyDiag
                                                                                                  { 0b0000'0000'0000'1000 };
00049 */
00051
00052 // This carries the information for a Diagnostic to update itself: if UpdateNsteps > 0, then every so
          many steps.
00053 // If UpdateNSteps == 0, then it only updates at the start and/or finish of integration if the
          appropriate bool is set.
00054 struct UpdateFrequency
00055 {
00056
                   largecounter UpdateNSteps{0}:
00057
                  bool UpdateStart{false};
00058
                  bool UpdateFinish{false};
00059 };
00060
00063
00064 // Abstract base class for all diagnostics
00065 class Diagnostic
00066 {
00067 public:
00068
                  \ensuremath{//} 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
          integrating
00075
                 // a new geodesic.
// The base class implementation only resets m_StepsSinceUpdated
00076
00077
                   // Descendants can override this if they need to reset additional internal variables
00078
00079
00080
                   // virtual destructor to ensure correct destruction of descendants
00081
                  virtual ~Diagnostic() = default;
00082
                   // This is the heart of the Diagnostic. It calls the helper function DecideUpdate(), and if this
00084
                   // returns true, will update its internal status based on the current (new) state of its owner
```

7.9 Diagnostics.h

```
00086
00087
          \ensuremath{//} 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)
          virtual std::string getFullDataStr() const = 0;
00089
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)
          // This should return a number >=0
00096
00097
          virtual real FinalDataValDistance(const std::vector<real> &val1, const std::vector<real> &val2)
      const = 0;
00098
00099
          // Getters for descriptions
          ^{\prime\prime} This returns the name (only) of the Diagnostic, as a string without spaces that will be
00100
     appended
00101
         // to an output file (e.g. prefix_DiagName.ext). Must be implemented!
00102
          virtual std::string getNameStr() const = 0;
          // This returns the full description of the Diagnostic. Default implementation
00103
          // returns GetDiagNameStr()
00104
00105
          virtual std::string getFullDescriptionStr() const;
00106
00107 protected:
          // 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<Diagnosticw;
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
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> &val1, const std::vector<real> &val2) const
     final;
00150
          // Description string getters
std::string getNameStr() const final;
00151
00152
          std::string getFullDescriptionStr() const final;
00153
00154
00155
          \begin{tabular}{ll} // & FourColorScreen & does & not & need & any & (static) & options! \end{tabular}
00156
00157 private:
00158
          // Note initialization to 0; this means the default value returned will be 0
          // (e.g. if Term::BoundarySphere is not reached)
00159
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
          GeodesicPositionDiagnostic(Geodesic *const theGeodesic) : Diagnostic(theGeodesic) {}
00170
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;
          // This returns the final (theta,phi) value of the geodesic
00180
00181
          std::vector<real> getFinalDataVal() const final;
00182
00183
          // This determines the angular distance between two geodesic
          // (based on their final angles on the boundary sphere (theta, phi))
00184
00185
          real FinalDataValDistance(const std::vector<real> &val1, const std::vector<real> &val2) const
     final;
00186
00187
          // Description string getters
          std::string getNameStr() const final;
00188
00189
          std::string getFullDescriptionStr() const final;
00190
00191
          // The options: specifies the update frequency but also how many points we want to output in the
     end
00192
          static std::unique ptr<GeodesicPositionOptions> DiagOptions:
00193
00194 private:
00195
         // Keeps track of the points that are saved
00196
          std::vector<Point> m_AllSavedPoints{};
00197 };
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
          EquatorialPassesDiagnostic(Geodesic *const theGeodesic) : Diagnostic(theGeodesic) {}
00206
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
          real FinalDataValDistance(const std::vector<real> &val1, const std::vector<real> &val2) const
00219
     override;
00220
00221
          // Description string getters
          std::string getNameStr() const override;
00222
00223
         std::string getFullDescriptionStr() const override;
00224
00225
          // Needs the extra option of a threshold
00226
         static std::unique_ptr<EquatorialPassesOptions> DiagOptions;
00227
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
00237 // Forward declaration needed before Diagnostic
00238 struct ClosestRadiusOptions;
00239 // Diagnostic keeps track of the closes radius that the geodesic passes through
00240 // Note: always keeps track of true radius, not log radius
00241 class ClosestRadiusDiagnostic final : public Diagnostic
00242 {
00243 public:
00244
          // Basic constructor only passes on Geodesic pointer to base class constructor
00245
          {\tt ClosestRadiusDiagnostic(Geodesic *const the Geodesic) : Diagnostic(the Geodesic) } \{\}
00246
00247
          // re-initialize closest radius
```

7.9 Diagnostics.h

```
00248
          void Reset() final;
00249
00250
          // Check to see if we have travelled closer
00251
          void UpdateData() final;
00252
00253
          // Return closest radius travelled
          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> &val1, 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
          static std::unique_ptr<ClosestRadiusOptions> DiagOptions;
00265
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 \ // \ {\tt Note that Equatorial Emission Diagnostic inherits from Equatorial Passes Diagnostic,} \\
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:
          // Basic constructor only passes on Geodesic pointer to parent class constructor EquatorialEmissionDiagnostic(Geodesic *const theGeodesic) :
00283
00284
     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
          // using the specified emission model
00290
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> &val1, const std::vector<real> &val2) const
      final;
00299
00300
           // Description string getters
          std::string getNameStr() const final;
00301
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
00313 // Declare your Diagnostic class here, inheriting from Diagnostic.
00314 // Sample code:
00315 /*
00316 // Forward declaration needed before Diagnostic (if using base class DiagnosticOptions, this is
     strictly speaking not necessary)
00317 struct DiagnosticOptions;
00318 // class definition
00319 class MyDiagnostic final : public Diagnostic // good practice to make the class final unless
      descendant classes are possible
00320 (
00321 public:
00322
          // constructor must at least take and pass along the const pointer to the owner Geodesic
00323
          MyDiagnostic(Geodesic* const theGeodesic) : Diagnostic(theGeodesic) {}
00324
00325
          // If you have MyDiagnostic-specific member variables, then they probably need to be reset at the
     beginning
00326
          // of integration for each new geodesic; in that case, you need to override Reset() to do so.
00327
          // Note: make sure to call the base class implementation Diagnostic::Reset()
```

```
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;
          // This should return a vector of real numbers that indicates the final "value" that should be
00338
      associated to
00339
          // \  \, \text{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
00343
         // "distance" of two geodesics (this is used for Mesh refinement)
          real FinalDataValDistance(const std::vector<real>& val1, const std::vector<real>& val2) const
00344
     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
          // where this Diagnostic's output is written. getFullDescriptionStr() is a descriptive string that
00348
     should list
00349
         00350
          std::string getNameStr() const final;
00351
          std::string getFullDescriptionStr() const final;
00352
00353
          // Diagnostic options: basic DiagnosticOptions only contains UpdateFrequency information,
          // if needed, can use a descendant class with more options (see e.g. GeodesicPositionDiagnostic) static std::unique_ptr<DiagnosticOptions> DiagOptions;
00354
00355
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:
          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 struct EmissionModel;
00419 struct FluidVelocityModel;
00420 // EquatorialEmissionDiagnostic needs to keep track of all tuneable parameters of the emission
00421 // Note that this inherits from EquatorialPassesOptions
00422 struct EquatorialEmissionOptions : public EquatorialPassesOptions
00423 {
00424 public:
00425
          EquatorialEmissionOptions (real thefudgefactor, int equatupper,
00426
                                     std::unique_ptr<EmissionModel> theemission,
      std::unique_ptr<FluidVelocityModel> thefluidmodel,
                                     bool rlog, int theredshiftpower, real thethreshold, UpdateFrequency
      thefrequency) : RedShiftPower{theredshiftpower}, RLogScale{rlog},
      GeometricFudgeFactor{thefudgefactor},
00428
      EquatPassUpperBound(equatupper),
00429
      TheEmissionModel{std::move(theemission)}, TheFluidVelocityModel{std::move(thefluidmodel)},
00430
      EquatorialPassesOptions (thethreshold, thefrequency) // constructor for parent class
00431
00432
00433
00434
          // Geometric fudge factor for n>0 passes
00435
          const real GeometricFudgeFactor;
00436
          // Upper bound allowed for contribution to intensity
00437
          const int EquatPassUpperBound;
00438
00439
          // Logarithmic radius scale used or not
00440
          const bool RLogScale;
00441
00442
          // Power of redshift in intensity contribution (should be 3 or 4)
00443
          const int RedShiftPower;
00444
00445
          // The emission model used (see Diagnostics_Emission.h and .cpp for specific models)
00446
          const std::unique ptr<EmissionModel> TheEmissionModel;
          // The fluid velocity model used (see Diagnostics_Emission.h and .cpp for specific models)
00448
          const std::unique_ptr<FluidVelocityModel> TheFluidVelocityModel;
00449 };
00450
00452 // if necessary, define your new DiagnosticOptions class here
00453 // Sample code:
00455 struct MyDiagnosticOptions : public DiagnosticOptions
00456 {
00457 public:
00458
          // Constructor should pass along UpdateFrequency information to base class
          {\tt MyDiagnosticOptions} \ ({\tt UpdateFrequency} \ \ the frequency}) \ \ : \ {\tt DiagnosticOptions} \ ({\tt the frequency}) \ \ //, \ \ (\ldots)
00459
     other initializations
00460
00461
00462
          // other member variables here - make them const!
00463
00464 };
00465 */
00467
00468 #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
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 {\tt GLM}
00029 struct GLMJohnsonSUEmission final : public EmissionModel
00030 {
00031 public:
00032
           // Constructor
           GLMJohnsonSUEmission(real mu, real gamma, real sigma) : m_mu{mu}, m_gamma{gamma}, m_sigma{sigma}
00034
00035
00036
           // Emitted brightness (only depends on radius)
00037
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;
           const real m_sigma;
00047
00048 };
00049
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
          virtual ~FluidVelocityModel() = default;
00060
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
          // Description string getter
00065
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 };
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 {
          // Constructor with three parameters and Metric pointer (which is passed to base class
00077
     constructor)
          GeneralCircularRadialFluid(real subKeplerParam, real betar, real betaphi, const Metric *const
      theMetric) : m_subKeplerParam{fmin(fmax(subKeplerParam, 0.0), 1.0)}, m_betaR{fmin(fmax(betar, 0.0),
00079
     m_betaPhi{fmin(fmax(betaphi, 0.0), 1.0)}, FluidVelocityModel(theMetric)
08000
        {
00081
              // Do some checks on three params, which must lie between 0.0 and 1.0 (note that they are
     adjusted as such in
00082
             // initializer above)
00083
              if (subKeplerParam < 0.0)</pre>
00084
                  ScreenOutput("Sub-Keplerian parameter must be between 0 and 1; adjusting to 0",
     OutputLevel::Level_0_WARNING);
00085
             if (subKeplerParam > 1.0)
00086
                  ScreenOutput("Sub-Keplerian parameter must be between 0 and 1; adjusting to 1",
      OutputLevel::Level_0_WARNING);
00087
00088
              if (betar < 0.0)
                  ScreenOutput("beta_r parameter must be between 0 and 1; adjusting to 0",
00089
     OutputLevel::Level_0_WARNING);
           if (betar > 1.0)
00090
00091
                  ScreenOutput("beta_r parameter must be between 0 and 1; adjusting to 1",
     OutputLevel::Level_0_WARNING);
00092
00093
              if (betaphi < 0.0)
                  ScreenOutput("beta_phi parameter must be between 0 and 1; adjusting to 0",
00094
     OutputLevel::Level_0_WARNING);
00095
             if (betaphi > 1.0)
00096
                  ScreenOutput("beta_phi parameter must be between 0 and 1; adjusting to 1",
     OutputLevel::Level_0_WARNING);
00097
00098
              // Find the (equatorial) ISCO for this Metric
00099
              FindISCO();
00100
          }
00101
00102
          // Get the local four-velocity of the fluid according to this model
00103
          // Note: will always calculate with Point p exactly on the equator theta=pi/2, despite what p[2]
     may be passed
00104
          OneIndex GetFourVelocityd(const Point &p) const final;
00105
00106
          // Description string getter
00107
          std::string getFullDescriptionStr() const final;
00108
00109 private:
00110
          // Three parameters determining the flow
00111
          const real m_subKeplerParam;
00112
          const real m_betaR;
00113
          const real m_betaPhi;
00114
00115
          // Helper function to get circular (sub) Keplerian velocity outside the ISCO
          OneIndex GetCircularVelocityd(const Point &p, bool subKeplerianOn = true) const;

// Helper function to get circular and infalling (sub)Keplerian velocity inside ISCO
00116
00117
          // (according to prescription of Cunningham)
00118
00119
          OneIndex GetInsideISCOCircularVelocityd(const Point &p) const;
00120
00121
          // Helper function to get radial infalling velocity
          OneIndex GetRadialVelocityd(const Point &p) const;
00122
00123
00124
          // Helper function to find the ISCO (called in Constructor)
00125
          void FindISCO();
00126
          \ensuremath{//} ISCO radius and momentum (index down) components
00127
          bool m_ISCOexists{false};
00128
          real m_ISCOr{-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 GetChristrRaisedDer(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
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
00017 #include <string> // for strings
00018 #include <vector> // for std::vector
00019
00022
00023 // Abstract base class
00024 class Source
00025 {
00026 public:
          // Constructor initializes Metric
00027
00028
               Source(const Metric *const theMetric) : m_theMetric{theMetric} {}
00029
00030
               // Virtual destructor to ensure correct descendant destruction
               virtual ~Source() = default;
```

7.15 Geodesic.h

```
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
virtual std::string getFullDescriptionStr() const;
00037
00039 protected:
00040
          \ensuremath{//} 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:
          \ensuremath{//} Simple constructor passes on the Metric pointer to the base constructor
00048
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
          std::string getFullDescriptionStr() const final;
00055
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
          \//\ {\it Default} constructor not allowed
00068
          Geodesic() = delete;
00069
          // Copy constructor or copy assignment not allowed
          Geodesic(const Geodesic &) = delete;
00071
          Geodesic &operator=(const Geodesic &) = delete;
00072
00073
          // Constructor which creates the Geodesic object
          // \ {\tt Takes} \ {\tt the following arguments} \ {\tt which initialize the private member variables} \ {\tt that remain the}
00074
     same over all geodesics
00075
          // - Metric (pointer)
          // - Source (pointer)
00076
          // - Diagnostic bitflag (& value Diagnostic bitflag) (used to create a vector of new instances of
00077
     Diagnostics)
00078
          // - Termination bitflag (used to create a vector of new instances of Terminations)
00079
          ^{\prime\prime} - Geodesic integrator function to use for integrating geodesic equation
          Geodesic(const Metric *const theMetric, const Source *const theSource,
00080
00081
                    DiagBitflag diagbit, DiagBitflag valdiagbit,
                    TermBitflag termbit, GeodesicIntegratorFunc theIntegrator) : m_theMetric{theMetric},
00082
      m_theSource{theSource},
00083
      \verb|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
                                               // Current termination condition (Term::Continue if not done
00098
          Term getTermCondition() const;
      integrating)
00099
          Point getCurrentPos() const;
                                               // Current position
                                              // Current velocity
// Current value of affine parameter
00100
          OneIndex getCurrentVel() const;
00101
          real getCurrentLambda() const;
00102
          ScreenIndex getScreenIndex() const; // screen index
00103
          // Output getters, to be called after the Geodesic terminates
00104
00105
          // This gets the complete output that should be written to the output files;
          // there is one string more than the count of Diagnostics: one string per Diagnostic,
00106
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
```

```
// to the Geodesic. Will be used to determine "distance" between Geodesics which is used in Mesh
00111
          std::vector<real> getDiagnosticFinalValue() const;
00112
00113 private:
           // These variables define its internal state
00114
00115
          Term m_TermCond{Term::Uninitialized}; // As long as this is Term::Continue, not done integrating
00116
          Point m_CurrentPos();
                                                  // Current position
00117
          OneIndex m_CurrentVel{};
                                                   // Current proper velocity
                                                   // Current value of affine parameter (starts at 0.0)
          real m_curLambda{0.0};
00118
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
          const Source *const m_theSource; // Source for the rhs of the geodesic equation
          // An instance of each Diagnostic and Termination is created for the Geodesic (in its
00127
      constructor);
00128
        \ensuremath{//} so the Geodesic is the owner of these objects.
00129
          const DiagnosticUniqueVector m AllDiagnostics;
          const TerminationUniqueVector m_AllTerminations;
00130
          {\tt const} \ {\tt GeodesicIntegratorFunc} \ {\tt m\_theIntegrator}; \ {\tt //} \ {\tt This} \ {\tt is} \ {\tt the} \ {\tt function} \ {\tt that} \ {\tt will} \ {\tt integrate} \ {\tt the}
00131
      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>
 {\it std::string\ toString\ (const\ std::array} < {\it 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 LARGECOUNTER_MAX

```
#define LARGECOUNTER_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]

7.16.3.2 operator*() [2/2]

7.16.3.3 operator+()

TENSOR ARITHMETIC: addition/subtraction of tensors, scalar multiplication/division.

7.16.3.4 operator-()

7.16.3.5 operator/()

7.16.3.6 toString() [1/3]

PRINTING TENSORS TO STRING.

7.16.3.7 toString() [2/3]

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;
00022 // Note: An unsigned long is guaranteed to be able to hold at least 4 294 967 295 (4.10^{\circ}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 LARGECOUNTER MAX
00033 #define LARGECOUNTER_MAX std::numeric_limits<largecounter>::max()
00034 #endif
00035 #ifndef PIXEL_MAX
00036 #define PIXEL_MAX std::numeric_limits<pixelcoord>::max()
00037 #endif
00038
00043 inline constexpr real pi{3.1415926535};
```

7.17 Geometry.h 153

```
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]);
              theStr += ", ";
00093
00094
00095
          theStr += std::to_string(theTensor[TensorDim - 1]);
          theStr += ")"; // no spaces for the innermost brackets
00096
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
              theStr += std::to_string(theTensor[i]);
00110
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, FourInedex);
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
00125
00126
          for (int i = 0; i < TensorDim - 1; ++i)
00127
00128
              theStr += toString(theTensor[i]);
              theStr += ", ";
00129
00130
00131
          theStr += toString(theTensor[TensorDim - 1]); // the last element doesn't have a comma after it
00132
          theStr += " )"; // All but the innermost brackets have an extra space padding the bracket
00133
00134
```

```
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};
          for (int i = 0; i < TensorDim; ++i)</pre>
00147
              temp[i] = temp[i] + a2[i];
00148
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> &al, const std::array<t, TensorDim>
      &a2)
00156 {
          std::array<t, TensorDim> temp{a1};
for (int i = 0; i < TensorDim; ++i)</pre>
00157
00158
              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 {
          std::array<t, TensorDim> temp{t1};
for (int i = 0; i < TensorDim; ++i)
    temp[i] = static_cast<t>(temp[i] * lambda); // static_cast necessary if t is integral type!
00168
00169
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()

7.20.1.3 SetLoopMessageFrequency()

```
\begin{tabular}{ll} {\tt void SetLoopMessageFrequency (} \\ & {\tt largecounter \ the freq)} \end{tabular}
```

7.20.1.4 SetOutputLevel()

```
void SetOutputLevel (
          OutputLevel theLvl)
```

7.21 FOORT/src/InputOutput.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()

7.21.2.3 SetLoopMessageFrequency()

```
\begin{tabular}{ll} void SetLoopMessageFrequency ( \\ & largecounter \ the freq) \end{tabular}
```

7.22 InputOutput.h 157

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
                              // needed for file ouput
00015 #include <string>
                              // std::string used in various places
                             // needed to create vectors of strings
00016 #include <vector>
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
                                // Coarsest level output; only the major procedures produce output // Subprocedures can also produce output
          Level_1_PROC = 1,
Level_2_SUBPROC = 2,
00025
00026
          Level_3_ALLDETAIL = 3, // Finest level output; all details are shown
00027
          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);
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
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:
          // FilePrefix, TimeStamp, FileExtension, and a vector of strings DiagNames (the names of each of
00056
     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,
largecounter nroutputstocache = LARGECOUNTER_MAX - 1, // note -1,
00060
00061
00062
                                                                                          // since we will
      actually cache one more then this number before outputting everything
00063
                                 largecounter geodperfile = LARGECOUNTER_MAX,
00064
                                 std::string firstlineinfo = "");
00065
00066
          \ensuremath{//} This tells the OutputHandler to prepare for this many geodesic outputs to arrive;
          // the internal state needs to be prepared such that they can come in without providing a data
00067
     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
          // NOTE: this procedure needs to be thread-safe!
          void NewGeodesicOutput(largecounter index, std::vector<std::string> theOutput);
```

```
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
08000
           // Returns full description string of output handler
          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
          // Helper function: return the full file name for the n-th output file // for the Diagnostic diagnr (this is an entry in m_DiagNames) std::string <code>GetFileName(int diagnr, unsigned short filenr) const;</code>
00091
00092
00093
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
           // If this is false, then we are writing to file(s). At any time, if a file I/O error occurs, // the output handler switches to outputting everything to the console
00111
00112
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
           \ensuremath{//} 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
           // The current counter of completely full files (this is kept track of
00122
           // so that it knows what the next file to write output to is)
00123
           // (We had better not have more than 60k files!)
00124
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,
           // this must be written to file(s))
00129
00130
           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
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;
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
00029
           // This is used to avoid dividing by zero
00030
          constexpr real delta_nodiv0 = 1e-20;
00031
          // The amount of any coordinate that we shift to calculate derivatives (using central difference)
inline real Derivative_hval{1e-7};
00032
00033
00034
00035
          // The name of the integrator selected
00036
          inline std::string IntegratorDescription("RK4");
00037
          // Full descriptive string of integrator and all integrator options
00038
00039
          std::string GetFullIntegratorDescription();
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 {\tt GeodesicIntegratorFunc}
00051
          // Using the Runge-Kutta-4 algorithm to integrate the geodesic equation
00052
          void IntegrateGeodesicStep_RK4(Point curpos, OneIndex curvel,
00053
                                           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
          // 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

- 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::PixeIInfo
- class SquareSubdivisionMeshV2
- struct SquareSubdivisionMeshV2::PixeIInfo

7.29 Mesh.h

Go to the documentation of this file.

```
00001 #ifndef _FOORT_MESH_H
00002 #define _FOORT_MESH_H
00003
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
                           // std::unique_ptr
// std::vector
00017 #include <memory>
00018 #include <vector>
00019 #include <array>
                                 // std::array
                                 // for strings
00020 #include <string>
00022 // Abstract Mesh base class
00023 class Mesh
00024 {
00025 public:
         // Basic constructor constructs Diagnostic that is used for determinines "values" and "distances"
00026
      between values
00027
         Mesh(DiagBitflag valdiag)
00028
              // Calling CreateDiagnosticVector in this way will create a vector with exactly one element in
      it,
00029
               // i.e. the Diagnostic we need!
00030
               : m_DistanceDiagnostic{std::move((CreateDiagnosticVector(valdiag, valdiag, nullptr))[0])}
00031
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
```

7.29 Mesh.h 163

```
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"
          // NOTE: despite being a non-const member function, this must be designed to be thread-safe!
virtual void GeodesicFinished(largecounter index, std::vector<real> finalValues) = 0;
00045
00046
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
          // Returns false if the Mesh wants another iteration of pixels to integrate
00051
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
          // distances (using FinalDataValDistance()) between the "values" that are assigned to Geodesics
00059
          const std::unique_ptr<const Diagnostic> m_DistanceDiagnostic;
00060
00061 };
00062
00063 // A simple square mesh that will integrate a square of evenly spaced pixels
00064 class SimpleSquareMesh final : public Mesh
00066 public:
00067
          // Default constructor not possible
00068
          SimpleSquareMesh() = delete;
          // Constructor initializes total number of pixels and passes valdiag to base constructor
00069
00070
          // Note that we static cast the sgrt() to round off the row/column size to an integer number
00071
          SimpleSquareMesh(largecounter totalPixels, DiagBitflag valdiag)
               : m_TotalPixels{static_cast<pixelcoord>(sqrt(totalPixels)) *
00072
      static_cast<pixelcoord>(sqrt(totalPixels))},
00073
                m_RowColumnSize{static_cast<pixelcoord>(sqrt(totalPixels))},
00074
                Mesh (valdiag)
00075
          {
00076
              if constexpr (dimension != 4)
00077
                  ScreenOutput("SimpleSquareMesh only defined in 4D!", OutputLevel::Level_O_WARNING);
00078
00079
00080
          // Declarations of overriding virtual functions
00081
00082
          largecounter getCurNrGeodesics() const final;
00083
00084
          void getNewInitConds(largecounter index, ScreenPoint &newunitpoint, ScreenIndex &newscreenindex)
      const final;
00085
00086
          void GeodesicFinished(largecounter index, std::vector<real> finalValues) final;
00087
00088
          void EndCurrentLoop() final;
00089
00090
          bool IsFinished() const final;
00091
00092
          // Description string getter
00093
          std::string getFullDescriptionStr() const final;
00094
00095 private:
          // Total amount of pixels in grid (is the square of m_RowColumnSize)
00096
00097
          const largecounter m_TotalPixels;
          // Amount of pixels per row or column (square grid)
00098
00099
          const pixelcoord m RowColumnSize;
00100
          // Are we done integrating or not?
          bool m_Finished{false};
00101
00102 };
00103
{\tt 00104} // Mesh which integrates only certain user-inputted pixels
00105 class InputCertainPixelsMesh : public Mesh
00106 {
00107 public:
00108
          // Default constructor not possible
00109
          InputCertainPixelsMesh() = delete;
00110
          // Copy constructor not possible
00111
          InputCertainPixelsMesh(const InputCertainPixelsMesh &) = delete;
          // Constructor given in Mesh.cpp file; constructor asks for input of pixels InputCertainPixelsMesh(largecounter totalPixels, DiagBitflag valdiag);
00112
00113
00114
00115
          // Declarations of overriding virtual functions
00116
00117
          largecounter getCurNrGeodesics() const final:
00118
00119
          void getNewInitConds(largecounter index, ScreenPoint &newunitpoint, ScreenIndex &newscreenindex)
      const final;
00120
00121
          void GeodesicFinished(largecounter index, std::vector<real> finalValues) final;
00122
00123
          void EndCurrentLoop() final;
```

```
00124
          bool IsFinished() const final;
00125
00126
00127
          // Description string getter
          std::string getFullDescriptionStr() const final;
00128
00129
00130 private:
          // Total pixel size of the screen (square grid)
00131
00132
           const pixelcoord m_RowColumnSize;
00133
00134
          // How many pixels have been inputted in total, i.e. need integrating
          largecounter m_TotalPixels{0};
// All pixels' location
00135
00136
          std::vector<ScreenIndex> m_PixelsToIntegrate{};
00137
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 \ensuremath{//} 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:
          // - maxPixels: max. nr of pixels that can be integrated in TOTAL, over all iterations (if 0, then
00153
     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)
          //\ \hbox{- iterationPixels: maximum number of pixels to subdivide in each integration iteration}
00158
          // (max number of pixels that will be integrates is then 5*iterationPixels)
00159
          // - initialSubToFinal: once we decide to subdivide a square, do we automatically keep subdividing
00160
00161
          // until we reach maxSubdivision?
          // - valdiag: the "value" and "distance" Diagnostic to use
00162
          SquareSubdivisionMesh(largecounter maxPixels, largecounter initialPixels, int maxSubdivide,
00163
      largecounter iterationPixels, bool initialSubToFinal,
00164
                                 DiagBitflag valdiag)
              : m_InitialPixels{static_cast<pixelcoord>(sqrt(initialPixels)) *
00165
      static_cast<pixelcoord>(sqrt(initialPixels))},
00166
                m_MaxSubdivide{maxSubdivide},
                m_RowColumnSize{static_cast<pixelcoord>((sqrt(initialPixels) - 1) * ExpInt(2, maxSubdivide -
00167
      1) + 1) \},
00168
                m PixelsLeft{maxPixels}, m MaxPixels{maxPixels}, m InfinitePixels{maxPixels == 0},
      m_IterationPixels{iterationPixels},
00169
                m_InitialSubDividideToFinal{initialSubToFinal}, Mesh(valdiag)
00170
              if constexpr (dimension != 4)
    ScreenOutput("SquareSubdivisionMesh only defined in 4D!", OutputLevel::Level_0_WARNING);
00171
00172
00173
00174
              // DEBUG message for constructor (can delete)
              ScreenOutput("SquareSubdivisionMesh constructed: maxPixels: " + (m_InfinitePixels ? "infinite"
00175
      : 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
          std::string getFullDescriptionStr() const final;
00194
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)
```

7.29 Mesh.h 165

```
const pixelcoord m_RowColumnSize;
           // How many pixels per iteration can we subdivide?
00203
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?
          const bool m_InitialSubDividideToFinal;
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
               \ensuremath{//} The level at which the pixel has been subdivided
               // Note: initial grid pixels are at 1; pixels at 0 are pixels that cannot be subdivided
00225
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)
               // between its values and those of its right, lower, and right-lower neighbors.
00231
00232
               real Weight {-1};
00233
00234
               \ensuremath{//} The values associated to this pixel (as calculated by the value Diagnostic)
00235
               std::vector<real> DiagValue{};
00236
               // Where its lower and right neighbors are located in m AllPixels
00237
               // Note: the pixel with index 0 is (0,0) and can never be the lower or right neighbor of any
00238
     other pixel!
00239
               largecounter LowerNbrIndex{0};
00240
              largecounter RightNbrIndex{0};
00241
           // The current queue of pixels to be integrated
00242
          std::vector<PixelInfo> m_CurrentPixelQueue{};
// A bool for every pixel in the current queue: gets set to true when the pixel is done
00243
00244
      integrating and gets its values returned
00245
          std::vector<bool> m_CurrentPixelQueueDone{};
00246
          // All pixels that have been integrated already (so does not include the pixels in the current
      queue)
00247
          std::vector<PixelInfo> m AllPixels{};
00248
00249
           // Initializes the first nxn screen in m_CurrentPixelQueue
00250
          void InitializeFirstGrid();
00251
00252
           // Updates the neighbors of all pixels (that should have neighbors and don't yet) in m_AllPixels
00253
          void UpdateAllNeighbors();
00254
           // Updates all weights of the pixels in m_AllPixels with weight < 0 and subdiv > 0 and subdiv <
      m MaxSubdivide
00256
          // Assumes all squares have neighbors assigned correctly
00257
          void UpdateAllWeights();
00258
          // This will take the pixel \texttt{m\_AllPixels[ind]} and subdivide it, // adding up to <=5 pixels to the <code>CurrentPixelQueue</code>
00259
00260
           void SubdivideAndQueue(largecounter ind);
00261
00262
00263
           // Helper function to exponentiate an int to an int
00264
          // Note: the result can be larger than fits in an int, but the base is always 2 and the exp is
      always
00265
          // a number <=m_MaxSubdivide (which is an int)</pre>
00266
          pixelcoord ExpInt(int base, int exp);
00267 };
00268
00269 // Adaptive subdivision Mesh: starts with evenly spaced, square Mesh,
00270 // then decides to subdivide certain squares of pixels into smaller squares, 00271 // based on which pixels have a bigger "weight", which is defined as the maximum
00272 // "distance" (using the Diagnostic value distance) between the upper-left
00273 // vertex of the square with the other three vertices of the square.
00274 // V2: new way of dealing with neighbors and looping over pixels
00275 \ \text{class SquareSubdivisionMeshV2} : public Mesh
00276 {
00277 public:
00278
           // default constructor not possible
           SquareSubdivisionMeshV2() = delete;
00279
00280
          // Constructor must be called with arguments:
          // - maxPixels: max. nr of pixels that can be integrated in TOTAL, over all iterations (if 0, then
00281
      this is infinite,
00282
          // i.e. we keep integrating until all squares are maximally subdivided or have weight 0)
```

```
// - initialPixels: initial number of pixels to integrate (spaced equally over the screen)
               // - maxSubdivide: maximum number of times that we can subdivide squares (note: 1 denotes the
00284
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)
               // - initialSubToFinal: once we decide to subdivide a square, do we automatically keep subdividing
00288
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,
                                                 DiagBitflag valdiag)
00292
00293
                     : m_InitialPixels{static_cast<pixelcoord>(sqrt(initialPixels)) *
         static_cast<pixelcoord>(sqrt(initialPixels))},
00294
                       m_MaxSubdivide(maxSubdivide),
00295
                        \label{eq:m_RowColumnSize} $$ m_RowColumnSize{static_cast<pixelcoord>((sqrt(initialPixels) - 1) * ExpInt(2, maxSubdivide - 1
         1) + 1) \},
00296
                       m_PixelsLeft{maxPixels}, m_MaxPixels{maxPixels}, m_InfinitePixels{maxPixels == 0},
        m_IterationPixels{iterationPixels},
00297
                       m_InitialSubDividideToFinal{initialSubToFinal}, Mesh(valdiag)
00298
00299
                     if constexpr (dimension != 4)
                           ScreenOutput("SquareSubdivisionMeshV2 only defined in 4D!", OutputLevel::Level_0_WARNING);
00300
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)
               const pixelcoord m_RowColumnSize;
00327
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?
               const largecounter m_MaxPixels;
00331
00332
               // If we decide to subdivide a square, do we automatically subdivide it further to the max level?
               const bool m_InitialSubDividideToFinal;
00333
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!
                     // The weight is determined as the max of the distance (as calculated by the value Diagnostic)
00358
00359
                     // between its values and those of its right, lower, and right-lower neighbors.
00360
                     real Weight{-1};
00361
                     \ensuremath{//} The values associated to this pixel (as calculated by the value Diagnostic)
00362
00363
                     std::vector<real> DiagValue{};
```

```
00364
00365
               // Pointers to its neighbors
00366
               PixelInfo *LeftNbr{nullptr};
               PixelInfo *RightNbr{nullptr};
00367
               PixelInfo *UpNbr{nullptr};
PixelInfo *DownNbr{nullptr};
00368
00369
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
          // This list is only used to store the owner pointers (and thus the objects) of all pixels. // The other pixel vectors are used to iterate through (and need random access)
00375
00376
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{};
           // 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
           // Updates all weights of the pixels in m_CurrentPixelUpdating; // these should have subdiv > 0 and subdiv < m_MaxSubdivide, and all their neigbors assigned
00391
00392
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;
          PixelInfo *GetLeft(PixelInfo *p, int subdiv) const;
00401
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
           // Note: the result can be larger than fits in an int, but the base is always 2 and the exp is
00408
      always
00409
          // a number <=m_MaxSubdivide (which is an int)</pre>
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.
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
00015
00016 // The abstract base class for all Metrics.
00017 class Metric
00018 {
00019 public:
00020
          // Virtual destructor to ensure correct destruction of descendants
00021
          virtual ~Metric() = default;
00022
00023
          Metric(bool rlogscale = false);
00024
00025
           \ensuremath{//} Basic functions that return the metric with indices down or up:
00026
           \ensuremath{//} pure virtual as they must be defined in the descendant classes.
00027
00028
           // Get the metric at Point p, indices down
00029
           virtual TwoIndex getMetric_dd(const Point &p) const = 0;
00030
           // Get the metric at Point p, indices up
00031
          virtual TwoIndex getMetric_uu(const Point &p) const = 0;
00032
00033
           // The following functions return the Christoffel and other derivative quantities of the metric.
00034
           // They are implemented for this base class, BUT are left as virtual functions to allow for
00035
           // other metrics to implement their own (more efficient)
00036
           // way of calculating them, if so desired.
00037
          // Get the Christoffel symbol, indices up-down-down virtual ThreeIndex getChristoffel_udd(const Point &p) const;
00038
00039
00040
           // Get the Riemann tensor, indices up-down-down-down
00041
          virtual FourIndex getRiemann_uddd(const Point &p) const;
00042
           // Get the Kretschmann scalar
00043
           virtual real getKretschmann(const Point &p) const;
00044
00045
           // Function to get the description of the metric
          // (used for outputting to the screen while running and possibly to the output files)
// There is a base class implementation of this function returning an undescriptive string
00046
00047
00048
           virtual std::string getFullDescriptionStr() const;
00049
00050
          bool getrLogScale() const;
00051
00052 protected:
          // The symmetries (coordinate Killing vectors) of the metric. Should be set by descendant
00053
      constructor.
```

7.32 Metric.h 169

```
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:
        // 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
          // Constructor setting parameter a
00092
         KerrMetric(real aParam, bool rLogScale = false, real mParam = 1.);
00093
00094
          \ensuremath{//} 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
          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:
         // Rasheed-Larsen is specified by four parameters
00121
00122
         const real m aParam:
00123
         const real m_mParam;
00124
          const real m_pParam;
00125
         const real m_qParam;
00126
00127 public:
          // No default constructor allowed, must specify parameters
00128
          RasheedLarsenMetric() = delete;
00129
00130
          // Constructor setting parameter a
00131
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 };
```

```
00142 // Johanssen black hole metric (implementation by Seppe Staelens)
00143 class JohannsenMetric final : public SphericalHorizonMetric
00144 {
00145 private:
          // Johannsen up to first order in deviation function is specified by five parameters (if M=1)
00146
          const real m_aParam;
00148
          const real m_alpha13Param;
00149
          const real m_alpha22Param;
00150
          const real m_alpha52Param;
          const real m_eps3Param;
00151
00152
00153 public:
00154
          // No default constructor allowed, must specify parameters
00155
          JohannsenMetric() = delete;
00156
          // Constructor setting parameter a
JohannsenMetric(real aParam, real alpha13Param, real alpha22Param, real alpha52Param, real
00157
00158
     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
          std::string getFullDescriptionStr() const final;
00165
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:
          // No default constructor allowed, must specify parameters
00181
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 // Note that this should be between -1 and 1 since M=1
00200
00201
          const real m_aParam;
00202
00203 public:
00204
          \ensuremath{//} 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
          \begin{tabular}{ll} // & Constructor & that initializes & singularities \\ \end{tabular}
00223
          SingularityMetric(std::vector<Singularity> thesings, bool rLogScale);
00224
```

7.32 Metric.h 171

```
// Getter functions for the two member variables
           std::vector<Singularity> getSingularities() const;
00226
00227
00228 protected:
          // All singularities of the metric
00229
          const std::vector<Singularity> m_AllSingularities;
00230
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
           TwoIndex getMetric_dd(const Point &p) const final;
00241
00242
           TwoIndex getMetric_uu(const Point &p) const final;
00243
00244
           // The description string getter
          std::string getFullDescriptionStr() const final;
00245
00246
00247 private:
00248
          const real m_P;
00249
          const real m_q0;
00250
          const real m_lambda;
00251
          real get_omega(real r, real theta, real 1) const;
real f_phi(real phi, real r, real theta, real 1, real R) const;
real f_om_phi(real phi, real r, real theta, real 1, real R) const;
00252
00253
00254
00255 };
00256
00257 // Boson star with solitonic potential (sigma = 0.06, phi_c = 0.044) (implementation by Seppe
00258 class BosonStarMetric final : public Metric
00259 {
00260 public:
           // 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;
           TwoIndex getMetric_uu(const Point &p) const final;
00265
           \ensuremath{//} The override of the description string getter
00266
          std::string getFullDescriptionStr() const final;
00267
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
           // These MUST be implemented
00291
00292
           TwoIndex getMetric_dd(const Point& p) const final;
00293
           TwoIndex getMetric_uu(const Point& p) const final;
00294
          // The description string getter // This is optional (but recommended) to implement; if not implemented, \,
00295
00296
00297
           // the base class Metric::getFullDescriptionStr() will be called instead
00298
           std::string getFullDescriptionStr() const final;
00299
00300 private:
          // good practice to have all const params (initialized in the constructor) // since the metric cannot change after initialization // const params...;
00301
00302
00303
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
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
           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 <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.
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
00051
           namespace tk
00052
00053
00054
               // spline interpolation
00055
                class spline
```

7.34 Spline.h 173

```
00056
00057
                public:
00058
                    // spline types
00059
                    enum spline_type
00060
00061
                                                // linear interpolation
                         linear = 10.
                         cspline = 30,
                                                // cubic splines (classical C^2)
00062
00063
                         cspline_hermite = 31 // cubic hermite splines (local, only C^1)
00064
00065
                    \ensuremath{//} boundary condition type for the spline end-points
00066
00067
                    enum bd_type
00068
                    {
00069
                         first_deriv = 1,
00070
                         second_deriv = 2
00071
                    };
00072
00073
                protected:
                   std::vector<double> m_x, m_y; // x,y coordinates of points
00075
                    // interpolation parameters
                    // f(x) = a_i + b_i * (x-x_i) + c_i * (x-x_i)^2 + d_i * (x-x_i)^3
00076
                    // where a_i = y_i, or else it won't go through grid points
std::vector<double> m_b, m_c, m_d; // spline coefficients
double m_c0; // for left extrapolation
00077
00078
00079
08000
                    spline_type m_type;
                    bd_type m_left, m_right;
00081
00082
                    double m_left_value, m_right_value;
00083
                    bool m_made_monotonic;
                                                              // calculate c_i, d_i from b_i
00084
                    void set_coeffs_from_b();
                    size_t find_closest(double x) const; // closest idx so that m_x[idx]<=x</pre>
00085
00086
00087
                public:
                   ^{\prime\prime} // default constructor: set boundary condition to be zero curvature ^{\prime\prime} // at both ends, i.e. natural splines
00088
00089
00090
                    spline() : m_type(cspline),
00091
                                m_left(second_deriv), m_right(second_deriv),
00092
                                \label{eq:m_left_value} \\ \texttt{m\_left\_value(0.0), m\_right\_value(0.0), m\_made\_monotonic(false)} \\
                    {
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,
                            bd_type right = second_deriv, double right_value = 0.0) : m_type(type),
00100
00101
                                                                                               m_left(left),
      m_right(right),
00102
                                                                                               m left value(left value),
      m_right_value(right_value),
00103
                                                                                               m made monotonic(false)
      // false correct here: make_monotonic() sets it
00104
00105
                         this->set_points(X, Y, m_type);
00106
                         if (make_monotonic)
00107
00108
                             this->make monotonic();
00110
                    }
00111
00112
                    // modify boundary conditions: if called it must be before set_points()
                    void set_boundary(bd_type left, double left_value,
00113
00114
                                        bd_type right, double right_value);
00115
00116
                    // set all data points (cubic_spline=false means linear interpolation)
00117
                    void set_points(const std::vector<double> &x,
00118
                                      const std::vector<double> &y,
00119
                                       spline_type type = cspline);
00120
00121
                    // adjust coefficients so that the spline becomes piecewise monotonic
00122
                    // where possible
                        this is done by adjusting slopes at grid points by a non-negative
00123
00124
                          factor and this will break C^2
00125
                         this can also break boundary conditions if adjustments need to
                    // this can also break boundary conditions if adjustments need
// be made at the boundary points
// returns false if no adjustments have been made, true otherwise
00126
00127
00128
                    bool make_monotonic();
00129
00130
                    // evaluates the spline at point x
00131
                    double operator()(double x) const;
00132
                    double deriv(int order, double x) const;
00133
00134
                    // returns the input data points
00135
                    std::vector<double> get_x() const { return m_x; }
00136
                    std::vector<double> get_y() const { return m_y; }
00137
                    double get_x_min() const
00138
00139
                         assert(!m x.emptv());
```

```
return m_x.front();
00142
                   double get_x_max() const
00143
00144
                       assert(!m_x.empty());
00145
                       return m_x.back();
00146
00147
00148 #ifdef HAVE_SSTREAM
00149
                  // spline info string, i.e. spline type, boundary conditions etc.
                  std::string info() const;
00150
00151 #endif // HAVE_SSTREAM
00152
             };
00153
00154
              namespace internal
00155
00156
                   // band matrix solver
00157
00158
                  class band_matrix
00159
00160
                  private:
00161
                       std::vector<std::vector<double> m_upper; // upper band
                       std::vector<std::vector<double> m_lower; // lower band
00162
00163
                   public:
00164
                       band_matrix() {};
                                                                  // constructor
                       band_matrix(int dim, int n_u, int n_l); // constructor
00165
00166
                       ~band_matrix() {};
00167
                       void resize(int dim, int n_u, int n_l); // init with dim,n_u,n_l
                                                                  // matrix dimension
00168
                       int dim() const;
00169
                       int num_upper() const
00170
00171
                           return (int)m_upper.size() - 1;
00172
00173
                       int num_lower() const
00174
                           return (int)m_lower.size() - 1;
00175
00176
                       // access operator
00178
                       double &operator()(int i, int j);
00179
                       double operator()(int i, int j) const; // read
00180
                       // we can store an additional diagonal (in m_lower)
                       double &saved_diag(int i);
00181
00182
                       double saved diag(int i) const;
00183
                       void lu_decompose();
                       std::vector<double> r_solve(const std::vector<double> &b) const;
00184
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;
                  m_left_value = left_value;
m_right_value = right_value;
00205
00206
00207
              }
00208
              void spline::set_coeffs_from_b()
00210
00211
                   assert(m_x.size() == m_y.size());
                  assert(m_x.size() == m_b.size());
assert(m_x.size() > 2);
00212
00213
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
                   for (size_t i = 0; i < n - 1; i++)</pre>
00220
00221
00222
                       const double h = m_x[i + 1] - m_x[i];
00223
                       // from continuity and differentiability condition
00224
                        \label{eq:mcii} $$ m_c[i] = (3.0 * (m_y[i + 1] - m_y[i]) / h - (2.0 * m_b[i] + m_b[i + 1])) / h; $$
                       // from differentiability condition  m\_d[i] = ((m\_b[i+1] - m\_b[i]) \ / \ (3.0 \ * \ h) \ - \ 2.0 \ / \ 3.0 \ * \ m\_c[i]) \ / \ h; 
00225
00226
```

7.34 Spline.h 175

```
00227
                   }
00228
                   // for left extrapolation coefficients
00229
00230
                   m_c0 = (m_left == first_deriv) ? 0.0 : m_c[0];
00231
00232
               void spline::set_points(const std::vector<double> &x,
00234
                                         const std::vector<double> &y,
                                         spline_type type)
00235
00236
               {
00237
                   assert(x.size() == y.size());
                   assert(x.size() > 2);
00238
00239
                   m type = type;
00240
                   m_made_monotonic = false;
00241
                   m_x = x;
                   m_y = y;
int n = (int)x.size();
00242
00243
                   // check strict monotonicity of input vector \mathbf{x} for (int i=0; i < n-1; i++)
00244
00245
00246
                        assert(m_x[i] < m_x[i + 1]);
00247
00248
                   }
00249
                   if (type == linear)
00250
00251
00252
                        // linear interpolation
                        m_d.resize(n);
00253
00254
                        m_c.resize(n);
00255
                        m_b.resize(n);
                        for (int i = 0; i < n - 1; i++)</pre>
00256
00257
00258
                            m_d[i] = 0.0;
00259
                            m_c[i] = 0.0;
00260
                            00261
                        ^{\prime} // ignore boundary conditions, set slope equal to the last segment
00262
                       m_b[n-1] = m_b[n-2];

m_c[n-1] = 0.0;
00263
00264
00265
                        m_d[n - 1] = 0.0;
00266
00267
                   else if (type == cspline)
00268
                        // classical cubic splines which are C^2 (twice cont differentiable)
00269
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);
for (int i = 1; i < n - 1; i++)</pre>
00276
00277
                            00278
00279
                            A(i, i + 1) = 1.0 / 3.0 * (x[i + 1] - x[i]);

rhs[i] = (y[i + 1] - y[i]) / (x[i + 1] - x[i]) - (y[i] - y[i - 1]) / (x[i] - x[i - 1])
00280
00281
      11);
00282
00283
                        // boundary conditions
00284
                        if (m_left == spline::second_deriv)
00285
                            // 2*c[0] = f''
00286
                            A(0, 0) = 2.0;

A(0, 1) = 0.0;
00287
00288
00289
                            rhs[0] = m_left_value;
00290
00291
                        else if (m_left == spline::first_deriv)
00292
                            // b[0] = f', needs to be re-expressed in terms of c:
00293
                            // (2c[0]+c[1])(x[1]-x[0]) = 3((y[1]-y[0])/(x[1]-x[0]) - f')
00294
                            A(0, 0) = 2.0 * (x[1] - x[0]);

A(0, 1) = 1.0 * (x[1] - x[0]);
00295
00296
                            rhs[0] = 3.0 * ((y[1] - y[0]) / (x[1] - x[0]) - m_left_value);
00297
00298
00299
                        else
00300
                        {
00301
                            assert(false);
00302
00303
                        if (m_right == spline::second_deriv)
00304
00305
                            // 2*c[n-1] = f''
                            A(n-1, n-1) = 2.0;

A(n-1, n-2) = 0.0;
00306
00307
00308
                            rhs[n - 1] = m_right_value;
00309
00310
                        else if (m_right == spline::first_deriv)
00311
00312
                            // b[n-1] = f', needs to be re-expressed in terms of c:
```

```
// (c[n-2]+2c[n-1])(x[n-1]-x[n-2])
                                                                    // = 3 (f' - (y[n-1]-y[n-2])/(x[n-1]-x[n-2]))

A(n-1, n-1) = 2.0 * (x[n-1] - x[n-2]);

A(n-1, n-2) = 1.0 * (x[n-1] - x[n-2]);
00314
00315
00316
                                                                    00317
               21));
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
00333
               + 1]) * (x[i + 1] - x[i]);
00334
00335
                                                          // for the right extrapolation coefficients (zero cubic term)
                                                          // f_{n-1}(x) = y_{n-1} + b*(x-x_{n-1}) + c*(x-x_{n-1})^2 double h = x[n - 1] - x[n - 2];
00337
00338
                                                          // m_c[n-1] is determined by the boundary condition
00339
                                                          m_d[n - 1] = 0.0;
                                                         m_b[n-1] = 3.0 * m_d[n-2] * h * h + 2.0 * m_c[n-2] * h + m_b[n-2]; // = 0.00 * m_c[n-2] * h + m_b[n-2]; // = 0.00 * m_c[n-2] * h + m_b[n-2]; // = 0.00 * m_c[n-2] * h + m_b[n-2]; // = 0.00 * m_c[n-2] * h + m_b[n-2]; // = 0.00 * m_c[n-2] * h + m_b[n-2]; // = 0.00 * m_c[n-2] * h + m_b[n-2]; // = 0.00 * m_c[n-2] * h + m_b[n-2] * h + m_b
00340
               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
                                                          // hermite cubic splines which are C^1 (cont. differentiable)
00346
00347
                                                          // and derivatives are specified on each grid point
00348
                                                           // (here we use 3-point finite differences)
00349
                                                          m_b.resize(n);
                                                          m_c.resize(n);
00350
00351
                                                          m_d.resize(n);
00352
                                                          // set b to match 1st order derivative finite difference
00353
                                                          for (int i = 1; i < n - 1; i++)</pre>
00354
                                                                     const double h = m_x[i + 1] - m_x[i];
00355
00356
                                                                    const double hl = m_x[i] - m_x[i - 1];
                                                                    m_b[i] = -h / (hl * (hl + h)) * m_y[i - 1] + (h - hl) / (hl * h) * m_y[i] + hl /
00357
               (h * (hl + h)) * m_y[i + 1];
00358
                                                          // boundary conditions determine b[0] and b[n-1]
00359
00360
                                                          if (m_left == first_deriv)
00361
00362
                                                                    m_b[0] = m_left_value;
00363
00364
                                                          else if (m left == second deriv)
00365
                                                                    const double h = m_x[1] - m_x[0];

m_b[0] = 0.5 * (-m_b[1] - 0.5 * m_left_value * h + 3.0 * (m_y[1] - m_y[0]) / h);
00366
00367
00368
00369
                                                          else
00370
00371
                                                                    assert(false);
00372
00373
                                                           if (m_right == first_deriv)
00374
00375
                                                                    m_b[n - 1] = m_{right\_value};
                                                                    m_c[n-1] = 0.0;
00376
00377
00378
                                                           else if (m_right == second_deriv)
00379
00380
                                                                    const double h = m_x[n - 1] - m_x[n - 2];
                                                                    m_b[n-1] = 0.5 * (-m_b[n-2] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] - m_b[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * (m_y[n-1] + 0.5 * m_right_value * h + 3.0 * m_right_value * h
00381
               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
```

7.34 Spline.h 177

```
assert(false);
00396
00397
                   \label{eq:coefficients} \ensuremath{//} \ensuremath{\text{for left extrapolation coefficients}}
00398
00399
                   m_c0 = (m_left == first_deriv) ? 0.0 : m_c[0];
00400
               }
00402
               bool spline::make_monotonic()
00403
00404
                   assert(m_x.size() == m_y.size());
                   assert(m_x.size() == m_b.size());
00405
                   assert(m_x.size() > 2);
00406
00407
                   bool modified = false;
                   const int n = (int)m_x.size();
00408
00409
                   // make sure: input data monotonic increasing --> b_i>=0
00410
                                  input data monotonic decreasing --> b_i<=0
                   for (int i = 0; i < n; i++)
00411
00412
                        int im1 = std::max(i - 1, 0);
                       00414
00415
00416
                            ((m_y[im1] >= m_y[i]) && (m_y[i] >= m_y[ip1]) && 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)
                   // ensure a sufficient criteria for monotonicity is satisfied:
// sqrt(b[i]^2+b[i+1]^2) <= 3 |avg|, with avg=(y[i+1]-y[i])/h,</pre>
00423
00424
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;
                       if (avg == 0.0 && (m_b[i] != 0.0 || m_b[i + 1] != 0.0))
00429
00430
                            modified = true;
m_b[i] = 0.0;
00431
00432
00433
                            m_b[i + 1] = 0.0;
00434
                       else if ((m_b[i] >= 0.0 && m_b[i + 1] >= 0.0 && avg > 0.0) || (m_b[i] <= 0.0 && m_b[i + 1] <= 0.0 && avg < 0.0))
00435
00436
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]
                                modified = true;

m_b[i] *= (3.0 / r);

m_b[i + 1] *= (3.0 / r);
00444
00446
00447
00448
                       }
                   }
00449
00450
                   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] \le x (return 0 if x \le x[0])
00461
               \verb|size_t spline::find_closest(|double x)| const|\\
00462
00463
                   std::vector<double>::const iterator it;
                   00464
00465
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.:
                       - Clenshaw
00473
00474
                        - Even-Odd method by A.C.R. Newbery
00475
                       - Compensated Horner Scheme
                   size_t n = m_x.size();
00477
                   size_t idx = find_closest(x);
00478
00479
                   double h = x - m_x[idx];
                   double interpol;
if (x < m_x[0])</pre>
00480
00481
```

```
{
00483
                         // extrapolation to the left
                        interpol = (m_c0 * h + m_b[0]) * h + m_y[0];
00484
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
                    {
                         // interpolation
00493
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);
                    size_t n = m_x.size();
size_t idx = find_closest(x);
00502
00503
00504
                    double h = x - m_x[idx];
00505
00506
                    double interpol;
00507
                    if (x < m_x[0])</pre>
00508
00509
                        \ensuremath{//} 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;
default:
00518
                            interpol = 0.0;
00520
                             break;
00521
00522
                    else if (x > m_x[n - 1])
00523
00524
00525
                        // extrapolation to the right
00526
                        switch (order)
00527
00528
                        case 1:
                            interpol = 2.0 * m_c[n - 1] * h + m_b[n - 1];
00529
00530
                            break;
                        case 2:
00531
                           interpol = 2.0 * m_c[n - 1];
break;
00532
00533
00534
                        default:
                           interpol = 0.0;
00535
00536
                            break;
00537
                        }
00539
00540
                        // interpolation
00541
00542
                        switch (order)
00543
00544
                        case 1:
                          interpol = (3.0 * m_d[idx] * h + 2.0 * m_c[idx]) * h + m_b[idx];
break;
00545
00546
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 #ifdef HAVE SSTREAM
               std::string spline::info() const
00562
                    std::stringstream ss;
00564
                   ss « "type " « m_type « ", left boundary deriv " « m_left « " = "; ss « m_left_value « ", right boundary deriv " « m_right « " = "; ss « m_right_value « std::endl; if (m_made_monotonic)
00565
00566
00567
00568
```

7.34 Spline.h 179

```
{
00570
                         ss « "(spline has been adjusted for piece-wise monotonicity)";
00571
                     }
                    return ss.str();
00572
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
                    void band_matrix::resize(int dim, int n_u, int n_l)
00586
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);
for (size_t i = 0; i < m_upper.size(); i++)</pre>
00593
00594
00595
                             m_upper[i].resize(dim);
00596
00597
                         for (size_t i = 0; i < m_lower.size(); i++)</pre>
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
                    {
                         int k = j - i; // what band is the entry assert((i >= 0) && (i < dim()) && (j >= 0) && (j < dim())); assert((-num_lower() <= k) && (k <= num_upper()));
00618
00619
00620
00621
                         // k=0 -> diagonal, k<0 lower left part, k>0 upper right part
                         if (k >= 0)
00622
00623
                             return m_upper[k][i];
                         else
00624
00625
                             return m_lower[-k][i];
00626
00627
                    double band_matrix::operator()(int i, int j) const
00628
                         int k = j - i; // what band is the entry
00629
                         assert((i >= 0) && (i < dim()) && (j >= 0) && (j < dim()));
assert((-num_lower() <= k) && (k <= num_upper()));
00630
00631
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
                    ^{\prime} // second diag (used in LU decomposition), saved in m_lower
00638
00639
                    double band_matrix::saved_diag(int i) const
00640
                         assert((i >= 0) && (i < dim()));
return m_lower[0][i];</pre>
00641
00642
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
                         int i_max, j_max;
00653
00654
                         int j_min;
00655
                         double x;
```

```
00657
                        // preconditioning
                        // normalize column i so that a_ii=1
00658
                        for (int i = 0; i < this->dim(); i++)
00659
00660
00661
                            assert(this->operator()(i, i) != 0.0);
                            this->saved_diag(i) = 1.0 / this->operator()(i, i);
00662
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++)</pre>
00666
00667
                                 this->operator()(i, j) *= this->saved_diag(i);
00668
00669
                            this->operator()(i, i) = 1.0; // prevents rounding errors
00670
                        }
00671
                        // Gauss\ LR-Decomposition
00672
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++)</pre>
00677
                            {
00678
                                 assert(this->operator()(k, k) != 0.0);
                                 asset(tilis=>operator()(i, k) / this=>operator()(k, k);
this=>operator()(i, k) = -x; // assembly part of L
j_max = std::min(this=>dim() - 1, k + this=>num_upper());
00679
00681
00682
                                 for (int j = k + 1; j \le j_{max}; j++)
00683
                                     // assembly part of R
00684
                                     this->operator()(i, j) = this->operator()(i, j) + x * this->operator()(k,
00685
      j);
00686
00687
00688
                       }
00689
                    // solves Ly=b
00690
00691
                   std::vector<double> band_matrix::l_solve(const std::vector<double> &b) const
00692
00693
                        assert(this->dim() == (int)b.size());
00694
                        std::vector<double> x(this->dim());
00695
                        int j_start;
00696
                        double sum:
00697
                        for (int i = 0; i < this->dim(); i++)
00698
00699
                            sum = 0;
00700
                            j_start = std::max(0, i - this->num_lower());
                            for (int j = j_start; j < i; j++)
    sum += this->operator()(i, j) * x[j];
00701
00702
00703
                            x[i] = (b[i] * this -> saved diag(i)) - sum;
00704
                        }
00705
00706
                    // solves Rx=y
00707
00708
                   std::vector<double> band_matrix::r_solve(const std::vector<double> &b) const
00709
00710
                        assert(this->dim() == (int)b.size());
00711
                        std::vector<double> x(this->dim());
00712
                        int j_stop;
                        double sum;
00713
                        for (int i = this - > dim() - 1; i >= 0; i--)
00714
00715
00716
                            sum = 0;
00717
                            j_stop = std::min(this->dim() - 1, i + this->num_upper());
00718
                              or (int j = i + 1; j <= j_stop; j++)
                                sum += this->operator()(i, j) * x[j];
00719
00720
                            x[i] = (b[i] - sum) / this->operator()(i, i);
00721
                        }
00722
                        return x:
00723
                    }
00724
00725
                    std::vector<double> band_matrix::lu_solve(const std::vector<double> &b,
00726
                                                                  bool is_lu_decomposed)
00727
00728
                        assert(this->dim() == (int)b.size());
00729
                        std::vector<double> x, y;
00730
                        if (is_lu_decomposed == false)
00731
00732
                            this->lu_decompose();
00733
00734
                        v = this -> 1 solve(b);
00735
                        x = this -> r_solve(y);
00736
                        return x;
00737
                    }
00738
00739
               } // namespace internal
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()

Termination helper function.

7.36 FOORT/src/Terminations.h File Reference

```
#include "Geometry.h"
#include <cstdint>
#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 {0b0000'0000'0000'00001}
- 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

7.36.3 Function Documentation

7.36.3.1 CreateTerminationVector()

Termination helper function.

7.36.4 Variable Documentation

7.36.4.1 Term_BoundarySphere

```
TermBitflag Term_BoundarySphere {Ob0000'0000'0000'0001} [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 {Ob0000'0000'0000'0010} [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
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;
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{0b0000'0000'1000};
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 {
           Uninitialized = -1, // Geodesic has not been properly initialized yet with initial
00051
position/velocity
00052 Continu
                                   \ensuremath{//} All is right, continue integrating geodesic
                               // STOP, encountered horizon (set by HorizonTermination)
   // STOP, encountered boundary sphere (set by BoundarySphereTermination)
// STOP, taken too many steps (set by TimeOutTermination)
00053
           Horizon,
00054
           BoundarySphere,
00055
           TimeOut,
           ThetaSingularity, // STOP, too close to polar coordinate Singularity, ...
NaN. // STOP, NaN encountered in geodesic position or velocity

its apparentered (of any codimension)
00056
                                   // STOP, too close to polar coordinate singularity (theta = 0 or theta = pi/2)
00057
           NaN.
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
           // a new geodesic.
// The base class implementation only resets m_StepsSinceUpdated
00085
00086
            // Descendants can override this if they need to reset additional internal variables
00087
00088
           virtual void Reset();
00089
00090
            // virtual destructor to ensure correct destruction of descendants
00091
           virtual ~Termination() = default;
00092
```

7.37 Terminations.h 185

```
// Function that is called to determine whether Termination wants to
          // terminate the Geodesic. Returns Term::Continue if no termination wanted,
00094
00095
          // otherwise it returns the appropriate Term condition
00096
          virtual Term CheckTermination() = 0;
00097
00098
          // This returns the full description of the Termination
          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
          // UpdateNSteps (which is set to 0 if we always update)
00106
00107
          bool DecideUpdate(largecounter UpdateNSteps);
00108
         // The termination is itself in charge of keeping track of how many steps it has been since it has
00109
     been updated
00110
         // The Termination's TerminationOptions struct tells it how many steps it needs to wait between
     updates
00111
         largecounter m StepsSinceUpdated{};
00112 };
00113
00114 // The owner vector of derived Termination classes
00115 using TerminationUniqueVector = std::vector<std::unique_ptr<Termination»;
00116
00118 TerminationUniqueVector CreateTerminationVector(TermBitflag termflags, Geodesic *const theGeodesic);
00119
00122
00123 // Forward declaration needed before Termination
00124 struct HorizonTermOptions;
00125 // Horizon termination: terminate geodesics if they get too close to the horizon (returns
      Term::Horizon)
00126 class HorizonTermination final : public Termination
00127 {
00128 public:
          // Basic constructor only passes on Geodesic pointer to base class constructor
00130
          HorizonTermination(Geodesic *const theGeodesic) : Termination(theGeodesic) {}
00131
00132
          \ensuremath{//} 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
     allowed from the horizon)
00139
         static std::unique_ptr<HorizonTermOptions> TermOptions;
00140 };
00141
00142 // Forward declaration needed before Termination
00143 struct BoundarySphereTermOptions;
00144 // The Boundary Sphere: this terminates the geodesic (and returns Term::BoundarySphere) if 00145 // the geodesic reaches outside of the boundary sphere 00146 class BoundarySphereTermination final : public Termination
00147 {
00148 public:
00149
          // Basic constructor only passes on Geodesic pointer to base class constructor
00150
          BoundarySphereTermination(Geodesic *const theGeodesic) : Termination(theGeodesic) {}
00151
00152
          // Check if we have passed the boundary sphere
00153
          Term CheckTermination() final;
00154
00155
          // Description string
00156
          std::string getFullDescriptionStr() const final;
00157
00158
         // The options that the BoundarySphereTermination keeps (contains the radius of the boundary
     sphere)
00159
         static std::unique_ptr<BoundarySphereTermOptions> TermOptions;
00160 };
00161
00162 // Forward declaration needed before Termination
00163 struct TimeOutTermOptions:
00164 // The Time Out: this terminates the geodesic if too many steps have been
00165 // taken in its integration (and returns Term::TimeOut)
00166 class TimeOutTermination final : public Termination
00167 {
00168 public:
00169
          // Basic constructor only passes on Geodesic pointer to base class constructor
00170
          TimeOutTermination(Geodesic *const theGeodesic) : Termination(theGeodesic) {}
00171
00172
          // This descendant needs to override Reset in order to also reset m_CurNrSteps
00173
          void Reset() final;
00174
          // Check if we have already taken too many steps
00175
00176
          Term CheckTermination() final:
```

```
00178
          // Description string
00179
          std::string getFullDescriptionStr() const final;
00180
00181
          // The options that the TimeOutTermination keeps (contains max number of steps allowed)
          static std::unique_ptr<TimeOutTermOptions> TermOptions;
00182
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
          \ensuremath{//} 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
     specifies
00205
         // any additional options the Termination needs)
          static std::unique_ptr<ThetaSingularityTermOptions> TermOptions;
00206
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
allowed from the horizon)
00225 static atd.
          // Options (contains horizon radius, if we are using logarithmic r coordinate, and distance
         static std::unique_ptr<NaNTermOptions> TermOptions;
00226 };
00227
00228 // Forward declaration needed before Termination
00229 struct GeneralSingularityTermOptions;
00230 // General singularity: terminate geodesic if it comes too close to one of a given number of
      (arbitrary codimension) singularities
00231 class GeneralSingularityTermination final : public Termination
00232 {
00233 public:
00234
          \ensuremath{//} Basic constructor only passes on Geodesic pointer to base class constructor
00235
          GeneralSingularityTermination(Geodesic *const theGeodesic) : Termination(theGeodesic) {}
00236
00237
          // Check if we are too close to the horizon
00238
          Term CheckTermination() final;
00239
00240
          // Description string
00241
          std::string getFullDescriptionStr() const final;
00242
00243
          // Options (contains horizon radius, if we are using logarithmic r coordinate, and distance
     allowed from the horizon)
00244
          static std::unique_ptr<GeneralSingularityTermOptions> TermOptions;
00245
00246 private:
00247
         std::string SingularityToString(int singnr) const;
00248 };
00249
00251 // Declare your Termination class here, inheriting from Termination.
00252 // Sample code:
00253 /*
00254 // Forward declaration needed before Termination
00255 struct TerminationOptions; // possibly will instead need to declare descendant options struct
00256 class MyTermination final : public Termination // good practice to make the class final unless
      descendant classes are possible
00257 (
00258 public:
00259
         // Constructor must at least pass on Geodesic pointer to base class constructor
```

7.37 Terminations.h 187

```
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)
               // Note: make sure to call the base class implementation Termination::Reset()
00263
                 // from within your implementation of MyTermination::Reset(), so that the base class internal
00264
         variable is also reset!
00265
                void Reset() final;
00266
00267
                 // Check the specific termination condition
00268
                Term CheckTermination() final;
00269
00270
                // Description string
                std::string getFullDescriptionStr() const final;
00271
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;
00277 private:
00278
                // any private member variables that are needed to keep track of things
00279 1:
00280 */
00282
00286 // Base class for TerminationOptions. Other Terminations can inherit from here if they require more
         options.
00287 struct TerminationOptions
00288 {
00289 public:
00290
                 // Basic constructor only sets the number of steps between updates
00291
                 TerminationOptions(largecounter Nsteps) : UpdateEveryNSteps{Nsteps}
00292
00293
00294
00295
                 // virtual destructor to ensure correct destruction of descendants
                virtual ~TerminationOptions() = default;
00296
00297
00298
                const largecounter UpdateEveryNSteps;
00299 };
00300
00301 // Options class for HorizonTermination; keeps track of location of horizon radius and the epsilon to
          terminate away from the horizon
00302 struct HorizonTermOptions : public TerminationOptions
00303 {
00304 public:
              HorizonTermOptions (real theHorizonRadius, bool therLogScale, real theAtHorizonEps, largecounter
00305
         Nsteps) : Horizon Radius \{the Horizon Radius\}, \ At Horizon Eps \{the At Horizon Eps\}, \ r Log Scale \{ther Log Scale\}, \ r Log Scale \{ther Log Scale \{ther Log Scale}, \ r Log Scale}, \ r Log Scale \{ther Log Scale}, \ r Lo
          TerminationOptions(Nsteps)
00306
                 {
00307
00308
00309
                const real HorizonRadius;
                const real AtHorizonEps;
00310
00311
                const bool rLogScale;
00312 };
00313
00314 // Options class for BoundarySphere; has to keep track of the BoundarySphere's radius
{\tt 00315~struct~BoundarySphereTermOptions:public~TerminationOptions}
00316 (
00317 public:
00318
                BoundarySphereTermOptions(real theRadius, bool therLogScale, largecounter Nsteps) :
         SphereRadius{theRadius}, rLogScale{therLogScale},
00319
         TerminationOptions(Nsteps)
00320
00321
00322
00323
               const real SphereRadius;
00324
                const bool rLogScale;
00325 };
00326
00327 // Options class for TimeOut; has to keep track of the max. number of integration steps allowed
00328 struct TimeOutTermOptions : public TerminationOptions
00329 {
00330 public:
00331
               TimeOutTermOptions(largecounter MaxStepsAllowed, largecounter Nsteps) : MaxSteps{MaxStepsAllowed},
         TerminationOptions(Nsteps)
00332
                {
00333
                 }
00334
00335
                 const largecounter MaxSteps;
00336 };
00337
00338 // Options class for ThetaSingularityTermination
00339 struct ThetaSingularityTermOptions : public TerminationOptions
```

```
00340 {
          ThetaSingularityTermOptions(real epsilon, largecounter Nsteps) : ThetaSingEpsilon{epsilon},
     TerminationOptions(Nsteps)
00343
00344
00346
          const real ThetaSingEpsilon;
00347 };
00348
00349 // Options class for TimeOut; has to keep track of the max. number of integration steps allowed
{\tt 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
{\tt 00360 \ struct \ GeneralSingularityTermOptions: public \ TerminationOptions}
00361 {
00362 public:
         GeneralSingularityTermOptions(std::vector<Singularity> sings,
                                         real eps, bool consoleoutputon, bool therlogscale, largecounter
00364
00365
              : Singularities{std::move(sings)}, Epsilon{eps}, OutputToConsole{consoleoutputon},
00366
                rLogScale{therlogscale},
00367
                TerminationOptions(Nsteps) {}
00368
00369
         const std::vector<Singularity> Singularities;
00370
          const real Epsilon;
00371
          const bool OutputToConsole;
00372
          const bool rLogScale;
00373 };
00374
00376 // Add your new TerminationOptions struct here, inheriting from TerminationOptions (if needed)
00377 // Sample code:
00378 /*
{\tt 00379 \ struct \ MyTermOptions \ : \ public \ TerminationOptions}
00380 {
00381 public:
         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>
```

7.40 Utilities.h

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 all-diags, 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
          private:
00028
00029
              // Type aliases to make accessing nested type easier
using Clock = std::chrono::steady_clock;
00030
00031
               using Second = std::chrono::duration<double, std::ratio<1»;
00032
00033
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

Null	
Timelike	
Spacelike	

7.43 ViewScreen.h

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 { Null = 0,00024 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 $\ensuremath{//}$ - physical position and looking direction; 00039 // - screen dimensions (in dimensions of length) 00040 // - the Mesh used (ViewScreen must become a owner of this object!) // - the Metric used (ViewScreen is NOT the owner of the Metric) 00041 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 // At the moment, we don't even use the direction; we are always pointed towards the origin if $(m_Direction != Point\{0, -1, 0, 0\})$ 00050 00051 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 // At the moment, we are only integrating null geodesics 00056 if (m_GeodType != GeodesicType::Null) 00057 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 // Heart of the ViewScreen: here, the ViewScreen is asked to provide initial conditions // for the geodesic nr index of the current iteration; based on the screen index 00067 00068 // that the Mesh gives, it sets up these physical initial conditions. 00069 void SetNewInitialConditions(largecounter index, Point &pos, OneIndex &vel, ScreenIndex &scrIndex) 00070 00071 00072 // These member functions essentially pass on information to/from the Mesh $\,$ bool IsFinished() const; // Does the ViewScreen (i.e. the Mesh) want to integrate 00073 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! void GeodesicFinished(largecounter index, std::vector<real> finalValues); // This geodesic has been integrated, returning its final "values" 00077 00078 00079 // Description string getter (spaces allowed), also will contain information about the Mesh 08000 std::string getFullDescriptionStr() const;

```
00081
00082 private:
00083
           \ensuremath{//} The metric at the position of the viewscreen (we only need indices down)
00084
           TwoIndex m_Metric_dd{};
          // The vielbein used to transform from the curved spacetime at the viewscreen to a locally flat
00085
      frame
00086
          TwoIndex m_Vielbein{};
00087
00088
           \ensuremath{//} 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)
           const ScreenPoint m_ScreenSize;
00095
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
           \ensuremath{//} 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
          // The const pointer to the Mesh we are using to determine pixels to be integrated const std::unique_ptr<Mesh> m_theMesh;
00108
00109
00110 };
00111
00112 #endif
```

Index

\sim Diagnostic	Utilities::Timer, 123
Diagnostic, 30	ClosestRadiusDiagnostic, 20
\sim DiagnosticOptions	ClosestRadiusDiagnostic, 21
DiagnosticOptions, 32	DiagOptions, 22
\sim EmissionModel	FinalDataValDistance, 21
EmissionModel, 33	getFinalDataVal, 21
\sim FluidVelocityModel	getFullDataStr, 21
FluidVelocityModel, 44	getFullDescriptionStr, 21
\sim Mesh	getNameStr, 21
Mesh, 82	m ClosestRadius, 22
\sim Metric	Reset, 21
Metric, 84	UpdateData, 21
~Source	ClosestRadiusOptions, 22
Source, 99	ClosestRadiusOptions, 23
\sim Termination	RLogScale, 23
Termination, 115	Config, 9
\sim TerminationOptions	ConfigCollection, 10
TerminationOptions, 116	GetGeodesicIntegrator, 10
,	GetMesh, 10
AtHorizonEps	GetMetric, 10
HorizonTermOptions, 68	GetOutputHandler, 10
	GetSource, 10
BosonStarMetric, 15	GetViewScreen, 11
BosonStarMetric, 16	InitializeDiagnostics, 11
getFullDescriptionStr, 16	InitializeScreenOutput, 11
getMetric_dd, 16	InitializeTerminations, 11
getMetric_uu, 16	Output_Important_Default, 11
m_mSpline, 16	Output_Other_Default, 11
m_PhiSpline, 16	SettingError, 10
BoundarySphere	Config.h
Terminations.h, 182	CONFIGURATION_MODE, 130
BoundarySphereTermination, 17	ConfigCollection
BoundarySphereTermination, 18	Config, 10
CheckTermination, 18	ConfigReader, 12
getFullDescriptionStr, 18	MaxStreamSize, 12
TermOptions, 18	ConfigReader::ConfigCollection, 23
BoundarySphereTermOptions, 18	ConfigSettingValue, 24
BoundarySphereTermOptions, 19	DisplayCollection, 24
rLogScale, 19	DisplaySetting, 24
SphereRadius, 19	DisplayTabs, 24
	Exists, 24
CheckTermination	GetSettingIndex, 25
BoundarySphereTermination, 18	IsCollection, 25
GeneralSingularityTermination, 51	LookupValue, 25
HorizonTermination, 67	LookupValueInteger, 25, 26
NaNTermination, 86	m Settings, 27
Termination, 115	NrSettings, 26
ThetaSingularityTermination, 118	operator[], 26
TimeOutTermination, 121	ReadCollection, 26
Clock	neaucollection, 20

ReadFile, 27	m_StepsSinceUpdated, 31
ReadSettingName, 27	Reset, 31
ReadSettingSpecificChar, 27	UpdateData, 31
ReadSettingValue, 27	DiagnosticOptions, 31
ConfigReader::ConfigCollection::ConfigSetting, 28	\sim DiagnosticOptions, 32
SettingName, 28	DiagnosticOptions, 32
SettingValue, 28	theUpdateFrequency, 32
ConfigReader::ConfigReaderException, 27	Diagnostics.cpp
ConfigReaderException, 28	CreateDiagnosticVector, 135
m_settingtrace, 28	Diagnostics.h
trace, 28	CreateDiagnosticVector, 137
ConfigReaderException	Diag_ClosestRadius, 137
ConfigReader::ConfigReaderException, 28	Diag_EquatorialEmission, 137
ConfigSettingValue	Diag_EquatorialPasses, 137
ConfigReader::ConfigCollection, 24	Diag_FourColorScreen, 137
CONFIGURATION_MODE	Diag_GeodesicPosition, 137
Config.h, 130	Diag_None, 137
ConstructVielbein	DiagBitflag, 136
ViewScreen, 126	DiagnosticUniqueVector, 136
Continue	DiagnosticUniqueVector
Terminations.h, 182	Diagnostics.h, 136
CreateDiagnosticVector	DiagOptions
Diagnostics.cpp, 135	ClosestRadiusDiagnostic, 22
Diagnostics.h, 137	EquatorialEmissionDiagnostic, 36
CreateTerminationVector	EquatorialPassesDiagnostic, 41
Terminations.cpp, 181	GeodesicPositionDiagnostic, 63
Terminations.h, 183	DiagValue
	SquareSubdivisionMesh::PixeIInfo, 90
DecideUpdate	SquareSubdivisionMeshV2::PixelInfo, 91
Diagnostic, 30	dimension
Termination, 115	Geometry.h, 152
delta_nodiv0	DisplayCollection
Integrators, 13	ConfigReader::ConfigCollection, 24
Derivative_hval	DisplaySetting
Integrators, 13	ConfigReader::ConfigCollection, 24
Diag_ClosestRadius	DisplayTabs
Diagnostics.h, 137	ConfigReader::ConfigCollection, 24
Diag_EquatorialEmission	DownNbr
Diagnostics.h, 137	SquareSubdivisionMeshV2::PixelInfo, 91
Diag_EquatorialPasses	
Diagnostics.h, 137	elapsed
Diag_FourColorScreen	Utilities::Timer, 123
Diagnostics.h, 137	EmissionModel, 32
Diag_GeodesicPosition	\sim EmissionModel, 33
Diagnostics.h, 137	GetEmission, 33
Diag_None	getFullDescriptionStr, 33
Diagnostics.h, 137	EndCurrentLoop
DiagBitflag	InputCertainPixelsMesh, 70
Diagnostics.h, 136	Mesh, 82
Diagnostic, 29	SimpleSquareMesh, 96
\sim Diagnostic, 30	SquareSubdivisionMesh, 103
DecideUpdate, 30	SquareSubdivisionMeshV2, 108
Diagnostic, 29	ViewScreen, 126
FinalDataValDistance, 30	Epsilon
getFinalDataVal, 30	GeneralSingularityTermOptions, 53
getFullDataStr, 30	epsilon
getFullDescriptionStr, 30	Integrators, 13
getNameStr, 30	EquatorialEmissionDiagnostic, 33
m_OwnerGeodesic, 31	DiagOptions, 36

EquatorialEmissionDiagnostic, 35	\sim FluidVelocityModel, 44
FinalDataValDistance, 35	FluidVelocityModel, 44
getFinalDataVal, 35	GetFourVelocityd, 44
getFullDataStr, 35	getFullDescriptionStr, 44
getFullDescriptionStr, 35	m_theMetric, 45
getNameStr, 35	FOORT/src/Config.cpp, 129
m_Intensity, 36	FOORT/src/Config.h, 129, 131
Reset, 36	FOORT/src/ConfigReader.cpp, 132
UpdateData, 36	FOORT/src/ConfigReader.h, 132, 133
EquatorialEmissionOptions, 36	FOORT/src/Diagnostics.cpp, 135
EquatorialEmissionOptions, 37	FOORT/src/Diagnostics.h, 135, 138
EquatPassUpperBound, 38	FOORT/src/DiagnosticsEmission.cpp, 143
GeometricFudgeFactor, 38	FOORT/src/DiagnosticsEmission.h, 144
RedShiftPower, 38	FOORT/src/Geodesic.cpp, 146
RLogScale, 38	FOORT/src/Geodesic.h, 146
TheEmissionModel, 38	FOORT/src/Geometry.h, 148, 152
TheFluidVelocityModel, 38	FOORT/src/Header.h, 154
•	FOORT/src/InputOutput.cpp, 154
EquatorialPassesDiagnostic, 38	
DiagOptions, 41	FOORT/src/InputOutput.h, 155, 157
EquatorialPassesDiagnostic, 39	FOORT/src/Integrators.cpp, 158
FinalDataValDistance, 40	FOORT/src/Integrators.h, 158, 160
getFinalDataVal, 40	FOORT/src/Main.cpp, 160
getFullDataStr, 40	FOORT/src/Mesh.cpp, 161
getFullDescriptionStr, 40	FOORT/src/Mesh.h, 162
getNameStr, 40	FOORT/src/Metric.cpp, 167
m_EquatPasses, 41	FOORT/src/Metric.h, 167, 168
m_PrevTheta, 41	FOORT/src/Spline.h, 172
Reset, 40	FOORT/src/Terminations.cpp, 181
UpdateData, 40	FOORT/src/Terminations.h, 181, 184
EquatorialPassesOptions, 41	FOORT/src/Utilities.cpp, 188
EquatorialPassesOptions, 42	FOORT/src/Utilities.h, 188, 189
Threshold, 42	FOORT/src/ViewScreen.cpp, 190
EquatPassUpperBound	FOORT/src/ViewScreen.h, 190, 191
EquatorialEmissionOptions, 38	FourColorScreenDiagnostic, 45
Exists	FinalDataValDistance, 46
ConfigReader::ConfigCollection, 24	FourColorScreenDiagnostic, 46
ExpInt	getFinalDataVal, 46
SquareSubdivisionMesh, 103	getFullDataStr, 46
SquareSubdivisionMeshV2, 108	getFullDescriptionStr, 46
f are unlei	getNameStr, 47
f_om_phi	m_quadrant, 47
ST3CrMetric, 113	Reset, 47
f_phi	UpdateData, 47
ST3CrMetric, 113	FourIndex
FinalDataValDistance	Geometry.h, 149
ClosestRadiusDiagnostic, 21	One and Observation Product Florid 47
Diagnostic, 30	GeneralCircularRadialFluid, 47
EquatorialEmissionDiagnostic, 35	FindISCO, 49
EquatorialPassesDiagnostic, 40	GeneralCircularRadialFluid, 48
FourColorScreenDiagnostic, 46	GetChristrRaisedDer, 49
GeodesicPositionDiagnostic, 62	GetCircularVelocityd, 49
FindISCO	GetFourVelocityd, 49
GeneralCircularRadialFluid, 49	getFullDescriptionStr, 49
FlatSpaceMetric, 42	GetInsideISCOCircularVelocityd, 49
FlatSpaceMetric, 43	GetRadialVelocityd, 49
getFullDescriptionStr, 43	m_betaPhi, 50
getMetric_dd, 43	m_betaR, 50
getMetric_uu, 43	m_ISCOexists, 50
FluidVelocityModel, 44	m_ISCOpphi, 50

m_ISCOpt, 50	m_FirstLineInfoString, 60
m_ISCOr, 50	m_nrGeodesicsPerFile, 60
m_subKeplerParam, 50	m_nrOutputsToCache, 60
GeneralSingularity	m_PrevCached, 60
Terminations.h, 182	m_PrintFirstLineInfo, 60
GeneralSingularityTermination, 50	m_TimeStamp, 60
CheckTermination, 51	m_WriteToConsole, 60
GeneralSingularityTermination, 51	NewGeodesicOutput, 58
getFullDescriptionStr, 51	OpenForFirstTime, 59
SingularityToString, 52	OutputFinished, 59
TermOptions, 52	PrepareForOutput, 59
GeneralSingularityTermOptions, 52	WriteCachedOutputToFile, 59
Epsilon, 53	GeodesicPositionDiagnostic, 61
GeneralSingularityTermOptions, 53	DiagOptions, 63
OutputToConsole, 53	FinalDataValDistance, 62
rLogScale, 53	GeodesicPositionDiagnostic, 62
Singularities, 53	getFinalDataVal, 62
Geodesic, 54	getFullDataStr, 62
Geodesic, 54	getFullDescriptionStr, 62
getAllOutputStr, 55	getNameStr, 62
getCurrentLambda, 55	m_AllSavedPoints, 63
getCurrentPos, 55	Reset, 62
getCurrentVel, 55	UpdateData, 62
getDiagnosticFinalValue, 55	GeodesicPositionOptions, 63
getScreenIndex, 55	GeodesicPositionOptions, 64
getTermCondition, 55	OutputNrSteps, 64
m_AllDiagnostics, 56	GeodesicType
m_AllTerminations, 56	ViewScreen.h, 190
m_curLambda, 56	GeometricFudgeFactor
m_CurrentPos, 56	EquatorialEmissionOptions, 38
m_CurrentVel, 56	Geometry.h
m_ScreenIndex, 56	dimension, 152
m_TermCond, 56	FourIndex, 149
m_theIntegrator, 56	largecounter, 149
m_theMetric, 57	LARGECOUNTER_MAX, 149
m_theSource, 57	OneIndex, 149
operator=, 55	operator+, 151
Reset, 55	operator-, 151
Update, 56	operator/, 151
GeodesicFinished	operator*, 151
InputCertainPixelsMesh, 70	pi, 152
Mesh, 82	PIXEL_MAX, 149
SimpleSquareMesh, 96	pixelcoord, 150
SquareSubdivisionMesh, 103	Point, 150
SquareSubdivisionMeshV2, 108	real, 150
ViewScreen, 126	ScreenIndex, 150
GeodesicIntegratorFunc	ScreenPoint, 150
Integrators.h, 159	Singularity, 150
GeodesicOutputHandler, 57	SingularityCoord, 150
GeodesicOutputHandler, 58	ThreeIndex, 150
GetFileName, 58	toString, 151, 152
getFullDescriptionStr, 58	TwoIndex, 150
m_AllCachedData, 59	get_omega
m_CurrentFullFiles, 59	ST3CrMetric, 113
m_CurrentGeodesicsInFile, 59	GetAdaptiveStep
m_DiagNames, 59	Integrators, 12
m_FileExtension, 59	getAllOutputStr
m_FilePrefix, 60	Geodesic, 55

getChristoffel_udd	FluidVelocityModel, 44
Metric, 84	FourColorScreenDiagnostic, 46
GetChristrRaisedDer	GeneralCircularRadialFluid, 49
GeneralCircularRadialFluid, 49	GeneralSingularityTermination, 51
GetCircularVelocityd	GeodesicOutputHandler, 58
GeneralCircularRadialFluid, 49	GeodesicPositionDiagnostic, 62
getCurNrGeodesics	GLMJohnsonSUEmission, 65
InputCertainPixelsMesh, 70	HorizonTermination, 67
Mesh, 82	InputCertainPixelsMesh, 70
SimpleSquareMesh, 96	JohannsenMetric, 73
SquareSubdivisionMesh, 104	KerrMetric, 76
SquareSubdivisionMeshV2, 108	KerrSchildMetric, 78
ViewScreen, 126	MankoNovikovMetric, 80
getCurrentLambda	Mesh, 82
Geodesic, 55	Metric, 84
getCurrentPos	NaNTermination, 86
Geodesic, 55	NoSource, 89
getCurrentVel	RasheedLarsenMetric, 94
Geodesic, 55	SimpleSquareMesh, 96
GetDiagNameStrings	Source, 99
Utilities, 14	SquareSubdivisionMesh, 104
getDiagnosticFinalValue	SquareSubdivisionMeshV2, 108
Geodesic, 55	ST3CrMetric, 113
GetDown	Termination, 115
SquareSubdivisionMeshV2, 108	ThetaSingularityTermination, 118
GetEmission	TimeOutTermination, 121
EmissionModel, 33	ViewScreen, 126
GLMJohnsonSUEmission, 65	GetFullIntegratorDescription
GetFileName	Integrators, 12
GeodesicOutputHandler, 58	GetGeodesicIntegrator
getFinalDataVal	Config, 10
ClosestRadiusDiagnostic, 21	getHorizonRadius
Diagnostic, 30	SphericalHorizonMetric, 101
EquatorialEmissionDiagnostic, 35	GetInsideISCOCircularVelocityd
EquatorialPassesDiagnostic, 40	GeneralCircularRadialFluid, 49
FourColorScreenDiagnostic, 46	getKretschmann
GeodesicPositionDiagnostic, 62	Metric, 84
GetFirstLineInfoString	GetLeft
Utilities, 14	SquareSubdivisionMeshV2, 109
GetFourVelocityd	GetLoopMessageFrequency
FluidVelocityModel, 44	InputOutput.cpp, 155
GeneralCircularRadialFluid, 49	InputOutput.h, 156
getFullDataStr	GetMesh
ClosestRadiusDiagnostic, 21	Config, 10
Diagnostic, 30	GetMetric
EquatorialEmissionDiagnostic, 35	Config, 10
EquatorialPassesDiagnostic, 40	getMetric_dd
FourColorScreenDiagnostic, 46	BosonStarMetric, 16
GeodesicPositionDiagnostic, 62	FlatSpaceMetric, 43
getFullDescriptionStr	JohannsenMetric, 73
BosonStarMetric, 16	KerrMetric, 76
BoundarySphereTermination, 18	KerrSchildMetric, 78
ClosestRadiusDiagnostic, 21	MankoNovikovMetric, 80
Diagnostic, 30	Metric, 84
EmissionModel, 33	RasheedLarsenMetric, 94
EquatorialEmissionDiagnostic, 35	ST3CrMetric, 113
EquatorialPassesDiagnostic, 40	getMetric_uu
FlatSpaceMetric, 43	BosonStarMetric, 16

FlatSpaceMetric, 43	Terminations.h, 182
JohannsenMetric, 73	HorizonRadius
KerrMetric, 76	HorizonTermOptions, 68
KerrSchildMetric, 78	HorizonTermination, 66
MankoNovikovMetric, 80	CheckTermination, 67
Metric, 84	getFullDescriptionStr, 67
RasheedLarsenMetric, 94	HorizonTermination, 67
ST3CrMetric, 113	TermOptions, 67
getNameStr	HorizonTermOptions, 67
ClosestRadiusDiagnostic, 21	AtHorizonEps, 68
Diagnostic, 30	HorizonRadius, 68
EquatorialEmissionDiagnostic, 35	HorizonTermOptions, 68
EquatorialPassesDiagnostic, 40	rLogScale, 68
FourColorScreenDiagnostic, 47	landa
GeodesicPositionDiagnostic, 62	Index
getNewInitConds	SquareSubdivisionMesh::PixelInfo, 90
InputCertainPixelsMesh, 70	SquareSubdivisionMeshV2::PixelInfo, 91
Mesh, 82	InitializeDiagnostics
SimpleSquareMesh, 96	Config, 11
SquareSubdivisionMesh, 104	InitializeFirstGrid
SquareSubdivisionMeshV2, 109	SquareSubdivisionMesh, 104
GetOutputHandler	SquareSubdivisionMeshV2, 109
Config, 10	InitializeScreenOutput
GetRadialVelocityd	Config, 11
GeneralCircularRadialFluid, 49	InitializeTerminations
getRiemann_uddd	Config, 11
Metric, 85	InputCertainPixelsMesh, 69
GetRight	EndCurrentLoop, 70
SquareSubdivisionMeshV2, 109	GeodesicFinished, 70
getrLogScale	getCurNrGeodesics, 70
Metric, 85	getFullDescriptionStr, 70
getScreenIndex	getNewInitConds, 70
Geodesic, 55	InputCertainPixelsMesh, 69, 70
GetSettingIndex	IsFinished, 71
ConfigReader::ConfigCollection, 25	m_Finished, 71
getSingularities	m_PixelsToIntegrate, 71
SingularityMetric, 98	m_RowColumnSize, 71
GetSource	m_TotalPixels, 71
Config, 10	InputOutput.cpp
getSource	GetLoopMessageFrequency, 155
NoSource, 89	ScreenOutput, 155
	SetLoopMessageFrequency, 155
Source, 99	SetOutputLevel, 155
getTermCondition	InputOutput.h
Geodesic, 55	GetLoopMessageFrequency, 156
GetTimeStampString	Level 0 WARNING, 156
Utilities, 14	Level_1_PROC, 156
GetUp	Level_2_SUBPROC, 156
SquareSubdivisionMeshV2, 109	Level_3_ALLDETAIL, 156
GetViewScreen	Level_4_DEBUG, 156
Config, 11	MaxLevel, 156
GLMJohnsonSUEmission, 64	OutputLevel, 156
GetEmission, 65	ScreenOutput, 156
getFullDescriptionStr, 65	
GLMJohnsonSUEmission, 65	SetLoopMessageFrequency, 156
m_gamma, 65	SetOutputLevel, 156
m_mu, 65	IntegrateGeodesicStep_RK4
m_sigma, 65	Integrators, 13
	IntegrateGeodesicStep_Verlet
Horizon	Integrators, 13

IntegratorDescription	InputOutput.h, 156
Integrators, 13	Level 2 SUBPROC
Integrators, 12	InputOutput.h, 156
delta_nodiv0, 13	Level_3_ALLDETAIL
Derivative_hval, 13	InputOutput.h, 156
epsilon, 13	Level_4_DEBUG
•	
GetAdaptiveStep, 12	InputOutput.h, 156
GetFullIntegratorDescription, 12	LoadPrecompiledOptions
IntegrateGeodesicStep_RK4, 13	Main.cpp, 161
IntegrateGeodesicStep_Verlet, 13	LookupValue
IntegratorDescription, 13	ConfigReader::ConfigCollection, 25
SmallestPossibleStepsize, 13	LookupValueInteger
VerletVelocityTolerance, 14	ConfigReader::ConfigCollection, 25, 26
Integrators.h	LowerNbrIndex
GeodesicIntegratorFunc, 159	SquareSubdivisionMesh::PixeIInfo, 90
IsCollection	
ConfigReader::ConfigCollection, 25	m_ActivePixels
IsFinished	SquareSubdivisionMeshV2, 110
InputCertainPixelsMesh, 71	m_AllCachedData
Mesh, 83	GeodesicOutputHandler, 59
SimpleSquareMesh, 96	m_AllDiagnostics
SquareSubdivisionMesh, 104	Geodesic, 56
SquareSubdivisionMeshV2, 109	m_AllPixels
ViewScreen, 126	SquareSubdivisionMesh, 105
Viewocieen, 120	SquareSubdivisionMeshV2, 110
JohannsenMetric, 71	m AllSavedPoints
getFullDescriptionStr, 73	GeodesicPositionDiagnostic, 63
getMetric_dd, 73	m_AllSingularities
getMetric_uu, 73	SingularityMetric, 98
-	m AllTerminations
JohannsenMetric, 73	Geodesic, 56
m_alpha13Param, 73	
m_alpha22Param, 73	m_alpha13Param
m_alpha52Param, 74	JohannsenMetric, 73
m_aParam, 74	m_alpha22Param
m_eps3Param, 74	JohannsenMetric, 73
	m_alpha3Param
KerrMetric, 74	MankoNovikovMetric, 81
getFullDescriptionStr, 76	m_alpha52Param
getMetric_dd, 76	JohannsenMetric, 74
getMetric_uu, 76	m_alphaParam
KerrMetric, 75	MankoNovikovMetric, 81
m_aParam, 76	m_aParam
m_mParam, 76	JohannsenMetric, 74
KerrSchildMetric, 76	KerrMetric, 76
getFullDescriptionStr, 78	KerrSchildMetric, 78
getMetric dd, 78	MankoNovikovMetric, 81
getMetric uu, 78	RasheedLarsenMetric, 94
KerrSchildMetric, 77	m_beg
m aParam, 78	Utilities::Timer, 124
m_ar aram, / o	m_betaPhi
largecounter	GeneralCircularRadialFluid, 50
Geometry.h, 149	
LARGECOUNTER MAX	m_betaR
Geometry.h, 149	GeneralCircularRadialFluid, 50
LeftNbr	m_ClosestRadius
	ClosestRadiusDiagnostic, 22
SquareSubdivisionMeshV2::PixelInfo, 91	m_curLambda
Level_0_WARNING	Geodesic, 56
InputOutput.h, 156	m_CurNrSteps
Level_1_PROC	TimeOutTermination, 121

m_CurrentFullFiles	m_ISCOr
GeodesicOutputHandler, 59	GeneralCircularRadialFluid, 50
m CurrentGeodesicsInFile	m IterationPixels
GeodesicOutputHandler, 59	SquareSubdivisionMesh, 105
m CurrentPixelQueue	SquareSubdivisionMeshV2, 110
SquareSubdivisionMesh, 105	m kParam
SquareSubdivisionMeshV2, 110	MankoNovikovMetric, 81
m CurrentPixelQueueDone	m lambda
SquareSubdivisionMesh, 105	ST3CrMetric, 114
SquareSubdivisionMeshV2, 110	m MaxPixels
m_CurrentPixelUpdating	SquareSubdivisionMesh, 105
SquareSubdivisionMeshV2, 110	SquareSubdivisionMeshV2, 111
m_CurrentPos	m MaxSubdivide
Geodesic, 56	SquareSubdivisionMesh, 105
m CurrentVel	SquareSubdivisionMeshV2, 111
Geodesic, 56	m Metric dd
m_DiagNames	ViewScreen, 127
GeodesicOutputHandler, 59	m mParam
m Direction	KerrMetric, 76
ViewScreen, 126	RasheedLarsenMetric, 94
m DistanceDiagnostic	m_mSpline
Mesh, 83	BosonStarMetric, 16
m_eps3Param	m mu
JohannsenMetric, 74	GLMJohnsonSUEmission, 65
m_EquatPasses	m nrGeodesicsPerFile
EquatorialPassesDiagnostic, 41	GeodesicOutputHandler, 60
m FileExtension	m_nrOutputsToCache
GeodesicOutputHandler, 59	GeodesicOutputHandler, 60
m FilePrefix	m_OwnerGeodesic
GeodesicOutputHandler, 60	Diagnostic, 31
m Finished	Termination, 116
InputCertainPixelsMesh, 71	m_P
SimpleSquareMesh, 97	ST3CrMetric, 114
m_FirstLineInfoString	m_PhiSpline
GeodesicOutputHandler, 60	BosonStarMetric, 16
m gamma	m_PixelsIntegrated
GLMJohnsonSUEmission, 65	SquareSubdivisionMeshV2, 111
m_GeodType	m_PixelsLeft
ViewScreen, 126	SquareSubdivisionMesh, 106
m HorizonRadius	SquareSubdivisionMeshV2, 111
SphericalHorizonMetric, 101	m_PixelsToIntegrate
m InfinitePixels	InputCertainPixelsMesh, 71
SquareSubdivisionMesh, 105	m_Pos
SquareSubdivisionMeshV2, 110	ViewScreen, 127
m_InitialPixels	m_pParam
SquareSubdivisionMesh, 105	RasheedLarsenMetric, 94
SquareSubdivisionMeshV2, 110	m PrevCached
m InitialSubDividideToFinal	GeodesicOutputHandler, 60
SquareSubdivisionMesh, 105	m PrevTheta
SquareSubdivisionMeshV2, 110	EquatorialPassesDiagnostic, 41
m_Intensity	m PrintFirstLineInfo
EquatorialEmissionDiagnostic, 36	GeodesicOutputHandler, 60
m ISCOexists	m_q0
GeneralCircularRadialFluid, 50	ST3CrMetric, 114
m_ISCOpphi	m_qParam
GeneralCircularRadialFluid, 50	RasheedLarsenMetric, 94
m_ISCOpt	m_quadrant
GeneralCircularRadialFluid. 50	FourColorScreenDiagnostic. 47

m_rLogScale	m_alpha3Param, 81
Metric, 85	m_alphaParam, 81
ViewScreen, 127	m aParam, 81
m RowColumnSize	m_kParam, 81
InputCertainPixelsMesh, 71	MankoNovikovMetric, 80
SimpleSquareMesh, 97	MaxLevel
SquareSubdivisionMesh, 106	InputOutput.h, 156
SquareSubdivisionMeshV2, 111	MaxSteps
	•
m_ScreenCenter	TimeOutTermOptions, 122
ViewScreen, 127	MaxStreamSize
m_ScreenIndex	ConfigReader, 12
Geodesic, 56	Maxterms
m_ScreenSize	Terminations.h, 182
ViewScreen, 127	Mesh, 81
m_Settings	\sim Mesh, 82
ConfigReader::ConfigCollection, 27	EndCurrentLoop, 82
m_settingtrace	GeodesicFinished, 82
ConfigReader::ConfigReaderException, 28	getCurNrGeodesics, 82
m_sigma	getFullDescriptionStr, 82
GLMJohnsonSUEmission, 65	getNewInitConds, 82
m_StepsSinceUpdated	IsFinished, 83
Diagnostic, 31	m_DistanceDiagnostic, 83
Termination, 116	Mesh, 82
m_subKeplerParam	Metric, 83
GeneralCircularRadialFluid, 50	∼Metric, 84
	getChristoffel_udd, 84
m_Symmetries	_
Metric, 85	getFullDescriptionStr, 84
m_TermCond	getKretschmann, 84
Geodesic, 56	getMetric_dd, 84
m_theIntegrator	getMetric_uu, 84
Geodesic, 56	getRiemann_uddd, 85
m_theMesh	getrLogScale, 85
ViewScreen, 127	m_rLogScale, 85
m_theMetric	m_Symmetries, 85
FluidVelocityModel, 45	Metric, 84
Geodesic, 57	
Source, 100	NaN
ViewScreen, 127	Terminations.h, 182
m theSource	NaNTermination, 85
Geodesic, 57	CheckTermination, 86
m_TimeStamp	getFullDescriptionStr, 86
GeodesicOutputHandler, 60	NaNTermination, 86
m TotalPixels	TermOptions, 87
InputCertainPixelsMesh, 71	NaNTermOptions, 87
SimpleSquareMesh, 97	NaNTermOptions, 88
• • •	OutputToConsole, 88
m_Vielbein	NewGeodesicOutput
ViewScreen, 127	GeodesicOutputHandler, 58
m_WriteToConsole	•
GeodesicOutputHandler, 60	NoSource, 88
main	getFullDescriptionStr, 89
Main.cpp, 161	getSource, 89
Main.cpp	NoSource, 89
LoadPrecompiledOptions, 161	NrSettings
main, 161	ConfigReader::ConfigCollection, 26
MankoNovikovMetric, 79	Null
getFullDescriptionStr, 80	ViewScreen.h, 190
getMetric_dd, 80	
getMetric_uu, 80	OneIndex
J	Geometry.h, 149

OpenForFirstTime	ReadSettingValue
GeodesicOutputHandler, 59	ConfigReader::ConfigCollection, 27
operator+	real
Geometry.h, 151	Geometry.h, 150
operator-	RedShiftPower
Geometry.h, 151	EquatorialEmissionOptions, 38
operator/	Reset
Geometry.h, 151	ClosestRadiusDiagnostic, 21
operator=	Diagnostic, 31
Geodesic, 55	EquatorialEmissionDiagnostic, 36
operator[] ConfigReader::ConfigCollection, 26	EquatorialPassesDiagnostic, 40 FourColorScreenDiagnostic, 47
operator*	Geodesic, 55
Geometry.h, 151	GeodesicPositionDiagnostic, 62
Output_Important_Default	Termination, 115
Config, 11	TimeOutTermination, 121
Output_Other_Default	reset
Config, 11	Utilities::Timer, 123
OutputFinished	RightNbr
GeodesicOutputHandler, 59	SquareSubdivisionMeshV2::PixelInfo, 91
OutputLevel	RightNbrIndex
InputOutput.h, 156	SquareSubdivisionMesh::PixelInfo, 90
OutputNrSteps	RLogScale
GeodesicPositionOptions, 64	ClosestRadiusOptions, 23
OutputToConsole	EquatorialEmissionOptions, 38
GeneralSingularityTermOptions, 53	rLogScale
NaNTermOptions, 88	BoundarySphereTermOptions, 19
,	GeneralSingularityTermOptions, 53
pi	HorizonTermOptions, 68
Geometry.h, 152	
PIXEL_MAX	ScreenIndex
Geometry.h, 149	Geometry.h, 150
pixelcoord	ScreenOutput
Geometry.h, 150	InputOutput.cpp, 155
PixelInfo	InputOutput.h, 156
SquareSubdivisionMesh::PixelInfo, 90	ScreenPoint 450
SquareSubdivisionMeshV2::PixeIInfo, 91	Geometry.h, 150
Point	Second
Geometry.h, 150	Utilities::Timer, 123
PrepareForOutput	SEdiagNbr
GeodesicOutputHandler, 59	SquareSubdivisionMeshV2::PixelInfo, 91
RasheedLarsenMetric, 92	SetLoopMessageFrequency InputOutput.cpp, 155
getFullDescriptionStr, 94	InputOutput.h, 156
getMetric_dd, 94	SetNewInitialConditions
getMetric_uu, 94	ViewScreen, 126
m_aParam, 94	SetOutputLevel
m_mParam, 94	InputOutput.cpp, 155
m_pParam, 94	InputOutput.h, 156
m_qParam, 94	SettingError
RasheedLarsenMetric, 93	Config, 10
ReadCollection	SettingName
ConfigReader::ConfigCollection, 26	ConfigReader::ConfigCollection::ConfigSetting, 28
ReadFile	Setting Value
ConfigReader::ConfigCollection, 27	ConfigReader::ConfigCollection::ConfigSetting, 28
ReadSettingName	SimpleSquareMesh, 95
ConfigReader::ConfigCollection, 27	EndCurrentLoop, 96
ReadSettingSpecificChar	GeodesicFinished, 96
ConfigReader::ConfigCollection, 27	getCurNrGeodesics, 96
J	3

getFullDescriptionStr, 96	UpdateAllWeights, 105
getNewInitConds, 96	SquareSubdivisionMesh::PixelInfo, 89
IsFinished, 96	DiagValue, 90
m_Finished, 97	Index, 90
m_RowColumnSize, 97	LowerNbrIndex, 90
m_TotalPixels, 97	PixelInfo, 90
SimpleSquareMesh, 95	RightNbrIndex, 90
Singularities	SubdivideLevel, 90
GeneralSingularityTermOptions, 53	Weight, 90
Singularity	SquareSubdivisionMeshV2, 106
Geometry.h, 150	EndCurrentLoop, 108
SingularityCoord	Explnt, 108
Geometry.h, 150	GeodesicFinished, 108
SingularityMetric, 97	getCurNrGeodesics, 108
getSingularities, 98	GetDown, 108
m_AllSingularities, 98	getFullDescriptionStr, 108
SingularityMetric, 98	GetLeft, 109
SingularityToString	getNewInitConds, 109
GeneralSingularityTermination, 52	GetRight, 109
SmallestPossibleStepsize	GetUp, 109
Integrators, 13	InitializeFirstGrid, 109
Source, 99	IsFinished, 109
~Source, 99	m_ActivePixels, 110
getFullDescriptionStr, 99	m_AllPixels, 110
getSource, 99	m_CurrentPixelQueue, 110
m_theMetric, 100	m_CurrentPixelQueueDone, 110
Source, 99	m_CurrentPixelUpdating, 110
Spacelike	
•	m_InfinitePixels, 110
ViewScreen.h, 190	m_InitialPixels, 110
SphereRadius Reundam SphereTermOntions 10	m_InitialSubDividideToFinal, 110
BoundarySphereTermOptions, 19	m_IterationPixels, 110
SphericalHorizonMetric, 100	m_MaxPixels, 111
getHorizonRadius, 101	m_MaxSubdivide, 111
m_HorizonRadius, 101	m_PixelsIntegrated, 111
SphericalHorizonMetric, 101	m_PixelsLeft, 111
SquareSubdivisionMesh, 102	m_RowColumnSize, 111
EndCurrentLoop, 103	SquareSubdivisionMeshV2, 107
Explnt, 103	SubdivideAndQueue, 109
GeodesicFinished, 103	UpdateAllWeights, 110
getCurNrGeodesics, 104	SquareSubdivisionMeshV2::PixelInfo, 90
getFullDescriptionStr, 104	DiagValue, 91
getNewInitConds, 104	DownNbr, 91
InitializeFirstGrid, 104	Index, 91
IsFinished, 104	LeftNbr, 91
m_AllPixels, 105	PixelInfo, 91
m_CurrentPixelQueue, 105	RightNbr, 91
m_CurrentPixelQueueDone, 105	SEdiagNbr, 91
m_InfinitePixels, 105	SubdivideLevel, 91
m_InitialPixels, 105	UpNbr, 92
m_InitialSubDividideToFinal, 105	Weight, 92
m_IterationPixels, 105	ST3CrMetric, 111
m_MaxPixels, 105	f_om_phi, 113
m_MaxSubdivide, 105	f_phi, 113
m_PixelsLeft, 106	get_omega, 113
m_RowColumnSize, 106	getFullDescriptionStr, 113
SquareSubdivisionMesh, 103	getMetric_dd, 113
SubdivideAndQueue, 104	getMetric_uu, 113
UpdateAllNeighbors, 104	m_lambda, 114
1 2 == =/ =	_ = ===================================

m_P, 114	TermBitflag, 182
m_q0, 114	TerminationUniqueVector, 182
ST3CrMetric, 113	ThetaSingularity, 182
SubdivideAndQueue	TimeOut, 182
SquareSubdivisionMesh, 104	Uninitialized, 182
SquareSubdivisionMeshV2, 109	TerminationUniqueVector
SubdivideLevel	Terminations.h, 182
SquareSubdivisionMesh::PixeIInfo, 90	TermOptions
SquareSubdivisionMeshV2::PixelInfo, 91	BoundarySphereTermination, 18
·	GeneralSingularityTermination, 52
Term	HorizonTermination, 67
Terminations.h, 182	NaNTermination, 87
Term_BoundarySphere	ThetaSingularityTermination, 118
Terminations.h, 183	TimeOutTermination, 121
Term_GeneralSingularity	TheEmissionModel
Terminations.h, 183	EquatorialEmissionOptions, 38
Term_Horizon	TheFluidVelocityModel
Terminations.h, 183	EquatorialEmissionOptions, 38
Term_NaN	ThetaSingEpsilon
Terminations.h, 183	ThetaSingularityTermOptions, 119
Term_None	ThetaSingularity ThetaSingularity
Terminations.h, 183	Terminations.h, 182
Term_ThetaSingularity	ThetaSingularityTermination, 117
Terminations.h, 183	CheckTermination, 118
Term TimeOut	
Terminations.h, 183	getFullDescriptionStr, 118
TermBitflag	TermOptions, 118
Terminations.h, 182	ThetaSingularityTermination, 118
Termination, 114	ThetaSingularityTermOptions, 119
~Termination, 115	ThetaSingEpsilon, 119
CheckTermination, 115	ThetaSingularityTermOptions, 119
DecideUpdate, 115	theUpdateFrequency
getFullDescriptionStr, 115	DiagnosticOptions, 32
m_OwnerGeodesic, 116	ThreeIndex
m_StepsSinceUpdated, 116	Geometry.h, 150
Reset, 115	Threshold
Termination, 115	EquatorialPassesOptions, 42
TerminationOptions, 116	Timelike
~TerminationOptions, 116	ViewScreen.h, 190
TerminationOptions, 116	TimeOut
·	Terminations.h, 182
UpdateEveryNSteps, 117	TimeOutTermination, 120
Terminations.cpp	CheckTermination, 121
CreateTerminationVector, 181	getFullDescriptionStr, 121
Terminations.h	m_CurNrSteps, 121
BoundarySphere, 182	Reset, 121
Continue, 182	TermOptions, 121
CreateTerminationVector, 183	TimeOutTermination, 121
GeneralSingularity, 182	TimeOutTermOptions, 122
Horizon, 182	MaxSteps, 122
Maxterms, 182	TimeOutTermOptions, 122
NaN, 182	tk, 14
Term, 182	tk::internal, 14
Term_BoundarySphere, 183	toString
Term_GeneralSingularity, 183	Geometry.h, 151, 152
Term_Horizon, 183	trace
Term_NaN, 183	ConfigReader::ConfigReaderException, 28
Term_None, 183	TwoIndex
Term_ThetaSingularity, 183	Geometry.h, 150
Term_TimeOut, 183	, , - -

Uninitialized	m_Vielbein, 127
Terminations.h, 182	SetNewInitialConditions, 126
Update	ViewScreen, 125
Geodesic, 56	ViewScreen.h
UpdateAllNeighbors	GeodesicType, 190
SquareSubdivisionMesh, 104	Null, 190
UpdateAllWeights	Spacelike, 190
SquareSubdivisionMesh, 105	Timelike, 190
SquareSubdivisionMeshV2, 110	
UpdateData	Weight
ClosestRadiusDiagnostic, 21	SquareSubdivisionMesh::PixelInfo, 90
Diagnostic, 31	SquareSubdivisionMeshV2::PixeIInfo, 92
EquatorialEmissionDiagnostic, 36	WriteCachedOutputToFile
EquatorialPassesDiagnostic, 40	GeodesicOutputHandler, 59
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	
Geooria, 125	
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	