

RAMICESII

Generated by Doxygen 1.9.4



<b>1 Hierarchical Index</b>	<b>1</b>
1.1 Class Hierarchy	1
<b>2 Data Structure Index</b>	<b>3</b>
2.1 Data Structures	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Data Structure Documentation</b>	<b>7</b>
4.1 CatalogueValues Class Reference	7
4.1.1 Constructor & Destructor Documentation	8
4.1.1.1 CatalogueValues()	8
4.1.2 Field Documentation	8
4.1.2.1 AzimuthalResolution	8
4.1.2.2 IsochroneMagnitudeResolution	8
4.1.2.3 IsochroneTimeStep	8
4.1.2.4 RadialResolution	8
4.1.2.5 SampleCount	9
4.1.2.6 SolarRadius	9
4.1.2.7 SynthesisActive	9
4.1.2.8 VerticalHeightPower	9
4.1.2.9 VerticalHeightScaling	9
4.1.2.10 VerticalHeightStart	9
4.2 ElementValues Class Reference	9
4.2.1 Detailed Description	10
4.2.2 Constructor & Destructor Documentation	10
4.2.2.1 ElementValues()	10
4.2.3 Member Function Documentation	10
4.2.3.1 GiveElementsNames()	11
4.2.3.2 Initialise()	11
4.2.4 Field Documentation	11
4.2.4.1 ElementNames	11
4.2.4.2 ProtonCounts	11
4.2.4.3 SolarAbundanceFile	11
4.2.4.4 SolarAbundanceFileDataColumn	12
4.2.4.5 SolarAbundanceFileNameColumn	12
4.2.4.6 SolarAbundances	12
4.3 Galaxy Class Reference	12
4.3.1 Constructor & Destructor Documentation	13
4.3.1.1 Galaxy()	14
4.3.2 Member Function Documentation	14
4.3.2.1 AssignMagnitudes()	14

4.3.2.2 ColdGasMass()	14
4.3.2.3 CompoundScattering()	14
4.3.2.4 ComputeScattering()	14
4.3.2.5 ComputeVisibilityFunction()	14
4.3.2.6 Evolve()	15
4.3.2.7 GasMass()	15
4.3.2.8 GasScaleLength()	15
4.3.2.9 CGMOperations()	15
4.3.2.10 Infall()	15
4.3.2.11 InfallMass()	15
4.3.2.12 InsertInfallingGas()	15
4.3.2.13 LaunchParallelOperation()	16
4.3.2.14 Mass()	16
4.3.2.15 MassHeaders()	16
4.3.2.16 PredictSurfaceDensity()	16
4.3.2.17 RelicMass()	16
4.3.2.18 RingEvolve()	16
4.3.2.19 SaveState()	17
4.3.2.20 SaveState_Enrichment()	17
4.3.2.21 SaveState_Events()	17
4.3.2.22 SaveState_Mass()	17
4.3.2.23 ScatterGas()	17
4.3.2.24 ScatterYields()	17
4.3.2.25 SelectionFunction()	18
4.3.2.26 StarMass()	18
4.3.2.27 StellarSynthesis()	18
4.3.2.28 SynthesiseObservations()	18
4.3.3 Field Documentation	18
4.3.3.1 BrightestStar	18
4.3.3.2 Data	18
4.3.3.3 DimmestStar	19
4.3.3.4 CGM	19
4.3.3.5 Migrator	19
4.3.3.6 ParallelBars	19
4.3.3.7 Param	19
4.3.3.8 RingMasses	19
4.3.3.9 Rings	19
4.3.3.10 SynthesisOutput	19
4.3.3.11 SynthesisProgress	20
4.3.3.12 Threads	20
4.4 GalaxyValues Class Reference	20
4.4.1 Detailed Description	21

4.4.2 Constructor & Destructor Documentation	22
4.4.2.1 GalaxyValues()	22
4.4.3 Member Function Documentation	22
4.4.3.1 Initialise()	22
4.4.4 Field Documentation	22
4.4.4.1 CGM_Mass	22
4.4.4.2 CGMAbsorbing	22
4.4.4.3 InfallMass1	23
4.4.4.4 InfallMass2	23
4.4.4.5 InfallMassMerger	23
4.4.4.6 InfallTime1	23
4.4.4.7 InfallTime2	23
4.4.4.8 InfallTimeMerger	23
4.4.4.9 MaxScaleLength	24
4.4.4.10 MaxSFRFraction	24
4.4.4.11 MergerDelayTime	24
4.4.4.12 MergerTurnOnWidth	24
4.4.4.13 MinScaleLength	24
4.4.4.14 PrimordialHotFraction	24
4.4.4.15 PrimordialMass	25
4.4.4.16 Radius	25
4.4.4.17 Ring0Width	25
4.4.4.18 RingCount	25
4.4.4.19 RingRadius	25
4.4.4.20 RingWidth	25
4.4.4.21 ScaleLengthDelay	26
4.4.4.22 ScaleLengthFinalTime	26
4.4.4.23 ScaleLengthTimeScale	26
4.4.4.24 UsingVariableRingWidth	26
4.5 Gas Class Reference	26
4.5.1 Detailed Description	27
4.5.2 Constructor & Destructor Documentation	27
4.5.2.1 Gas() [1/2]	27
4.5.2.2 Gas() [2/2]	27
4.5.3 Member Function Documentation	27
4.5.3.1 CheckMass()	28
4.5.3.2 Empty()	28
4.5.3.3 Mass() [1/2]	28
4.5.3.4 Mass() [2/2]	28
4.5.3.5 operator[]() [1/2]	28
4.5.3.6 operator[]() [2/2]	29
4.5.3.7 Primordial()	29

4.5.4 Field Documentation	29
4.5.4.1 internal_Mass	29
4.5.4.2 NeedsRecomputing	29
4.5.4.3 Species	29
4.6 GasReservoir Class Reference	30
4.6.1 Detailed Description	31
4.6.2 Constructor & Destructor Documentation	31
4.6.2.1 GasReservoir() [1/2]	31
4.6.2.2 GasReservoir() [2/2]	31
4.6.3 Member Function Documentation	31
4.6.3.1 Absorb() [1/5]	31
4.6.3.2 Absorb() [2/5]	32
4.6.3.3 Absorb() [3/5]	32
4.6.3.4 Absorb() [4/5]	32
4.6.3.5 Absorb() [5/5]	32
4.6.3.6 AbsorbMemory()	32
4.6.3.7 AccretionStream()	33
4.6.3.8 ColdGasMetallicity()	33
4.6.3.9 ColdMass()	33
4.6.3.10 Composition()	33
4.6.3.11 Deplete() [1/2]	33
4.6.3.12 Deplete() [2/2]	34
4.6.3.13 Heat()	34
4.6.3.14 HotMass()	34
4.6.3.15 Mass()	34
4.6.3.16 operator[]() [1/2]	35
4.6.3.17 operator[]() [2/2]	35
4.6.3.18 PassiveCool()	35
4.6.3.19 Primordial()	35
4.6.3.20 TransferAndHeat()	35
4.6.3.21 TransferColdFrom()	36
4.6.3.22 TransferFrom()	36
4.6.3.23 TransferHotFrom()	36
4.6.3.24 Wipe()	36
4.6.4 Field Documentation	36
4.6.4.1 Components	36
4.6.4.2 Param	37
4.7 GasStream Class Reference	37
4.7.1 Detailed Description	38
4.7.2 Constructor & Destructor Documentation	38
4.7.2.1 GasStream() [1/4]	39
4.7.2.2 GasStream() [2/4]	39

4.7.2.3 GasStream() [3/4]	39
4.7.2.4 GasStream() [4/4]	39
4.7.3 Member Function Documentation	40
4.7.3.1 Absorb() [1/3]	40
4.7.3.2 Absorb() [2/3]	40
4.7.3.3 Absorb() [3/3]	40
4.7.3.4 Cold() [1/3]	41
4.7.3.5 Cold() [2/3]	41
4.7.3.6 Cold() [3/3]	41
4.7.3.7 ColdMass() [1/2]	41
4.7.3.8 ColdMass() [2/2]	41
4.7.3.9 ComputeMasses()	42
4.7.3.10 Cool()	42
4.7.3.11 Deplete() [1/2]	42
4.7.3.12 Deplete() [2/2]	42
4.7.3.13 Dirty()	43
4.7.3.14 Heat()	43
4.7.3.15 Hot() [1/3]	43
4.7.3.16 Hot() [2/3]	43
4.7.3.17 Hot() [3/3]	43
4.7.3.18 HotMass() [1/2]	43
4.7.3.19 HotMass() [2/2]	44
4.7.3.20 Mass() [1/2]	44
4.7.3.21 Mass() [2/2]	44
4.7.4 Field Documentation	44
4.7.4.1 internal_Cold	44
4.7.4.2 internal_ColdMass	44
4.7.4.3 internal_Hot	45
4.7.4.4 internal_HotMass	45
4.7.4.5 internal_TotalMass	45
4.7.4.6 NeedsRecomputing	45
4.7.4.7 Source	45
4.8 GlobalParameters Class Reference	45
4.8.1 Detailed Description	46
4.8.2 Constructor & Destructor Documentation	46
4.8.2.1 GlobalParameters()	46
4.8.3 Member Function Documentation	47
4.8.3.1 Initialise()	47
4.8.3.2 SaveInputs()	47
4.8.4 Field Documentation	47
4.8.4.1 Catalogue	47
4.8.4.2 Element	47

4.8.4.3 Galaxy	47
4.8.4.4 Meta	48
4.8.4.5 Migration	48
4.8.4.6 Output	48
4.8.4.7 ParamMembers	48
4.8.4.8 Resources	48
4.8.4.9 Stellar	48
4.8.4.10 Thermal	49
4.8.4.11 Yield	49
4.9 IMF_Functor Class Reference	49
4.9.1 Constructor & Destructor Documentation	49
4.9.1.1 IMF_Functor()	50
4.9.2 Member Function Documentation	50
4.9.2.1 FormationCount()	50
4.9.2.2 IMF()	50
4.9.2.3 MomentCompute()	50
4.9.2.4 Normalise()	50
4.9.2.5 operator()()	50
4.9.2.6 Weighting()	51
4.9.3 Field Documentation	51
4.9.3.1 IMF_MeanMass	51
4.9.3.2 IMF_Normalisation	51
4.9.3.3 IMF_Weighting	51
4.9.3.4 Param	51
4.10 InitialisedData Class Reference	51
4.10.1 Detailed Description	52
4.10.2 Constructor & Destructor Documentation	52
4.10.2.1 InitialisedData()	52
4.10.3 Member Function Documentation	52
4.10.3.1 Log() [1/2]	53
4.10.3.2 Log() [2/2]	53
4.10.3.3 LogFlush()	53
4.10.3.4 NormalDist() [1/2]	53
4.10.3.5 NormalDist() [2/2]	53
4.10.3.6 ProgressBar()	53
4.10.3.7 UniformDist()	54
4.10.3.8 UrgentLog()	54
4.10.4 Field Documentation	54
4.10.4.1 AGBYield	54
4.10.4.2 CCSNYield	54
4.10.4.3 distribution	54
4.10.4.4 ECSNYield	54



4.10.4.5 generator	54
4.10.4.6 IMF	55
4.10.4.7 Isochrones	55
4.10.4.8 NSMYield	55
4.10.4.9 Param	55
4.10.4.10 SLF	55
4.10.4.11 SNIaYield	55
4.11 Integral Struct Reference	55
4.11.1 Field Documentation	56
4.11.1.1 FirstMoment	56
4.11.1.2 ZerothMoment	56
4.12 Interpolator Struct Reference	56
4.12.1 Member Function Documentation	56
4.12.1.1 Interpolate()	57
4.12.2 Field Documentation	57
4.12.2.1 LinearFactor	57
4.12.2.2 LowerID	57
4.12.2.3 UpperID	57
4.13 IsochroneCube Struct Reference	57
4.13.1 Member Function Documentation	58
4.13.1.1 Count()	58
4.13.1.2 Value()	58
4.13.2 Field Documentation	58
4.13.2.1 Data	58
4.13.2.2 Weighting	58
4.14 IsochroneEntry Struct Reference	58
4.14.1 Constructor & Destructor Documentation	59
4.14.1.1 IsochroneEntry()	59
4.14.2 Member Function Documentation	59
4.14.2.1 Countify()	59
4.14.2.2 operator[]() [1/2]	59
4.14.2.3 operator[]() [2/2]	59
4.14.3 Field Documentation	59
4.14.3.1 Properties	59
4.15 IsochroneTracker Class Reference	60
4.15.1 Constructor & Destructor Documentation	60
4.15.1.1 IsochroneTracker()	60
4.15.2 Member Function Documentation	60
4.15.2.1 Construct()	61
4.15.2.2 ExtractSample()	61
4.15.2.3 GetProperties()	61
4.15.2.4 IsoLog()	61

4.15.2.5 NormalSample()	61
4.15.2.6 ParseFile()	61
4.15.2.7 UniformSample()	62
4.15.3 Field Documentation	62
4.15.3.1 CapturedTs	62
4.15.3.2 CapturedZs	62
4.15.3.3 DeltaLogT	62
4.15.3.4 distribution	62
4.15.3.5 generator	62
4.15.3.6 Grid	62
4.15.3.7 isTimeLogUniform	63
4.15.3.8 Param	63
4.15.3.9 UnsortedGrid	63
4.16 IsoMass Class Reference	63
4.16.1 Detailed Description	63
4.16.2 Constructor & Destructor Documentation	64
4.16.2.1 IsoMass() [1/2]	64
4.16.2.2 IsoMass() [2/2]	64
4.16.3 Field Documentation	64
4.16.3.1 BirthIndex	64
4.16.3.2 Count	64
4.16.3.3 DeathIndex	64
4.16.3.4 Isochrone	64
4.16.3.5 MassIndex	65
4.16.3.6 Metallicity	65
4.17 MassReport Struct Reference	65
4.17.1 Field Documentation	65
4.17.1.1 BH	65
4.17.1.2 NS	65
4.17.1.3 Total	66
4.17.1.4 WD	66
4.18 MetaValues Class Reference	66
4.18.1 Detailed Description	67
4.18.2 Constructor & Destructor Documentation	67
4.18.2.1 MetaValues()	67
4.18.3 Member Function Documentation	67
4.18.3.1 Initialise()	67
4.18.4 Field Documentation	67
4.18.4.1 ParallelThreads	67
4.18.4.2 ProgressHashes	68
4.18.4.3 SimulationDuration	68
4.18.4.4 SimulationSteps	68

4.18.4.5 TimeStep	68
4.18.4.6 Verbosity	68
4.19 MigrationMatrix Class Reference	68
4.19.1 Constructor & Destructor Documentation	69
4.19.1.1 MigrationMatrix()	69
4.19.2 Member Function Documentation	69
4.19.2.1 Compound()	69
4.19.2.2 Create()	69
4.19.2.3 DiagonalMultiply()	70
4.19.2.4 Print()	70
4.19.3 Field Documentation	70
4.19.3.1 Grid	70
4.19.3.2 NRings	70
4.19.3.3 Param	70
4.20 MigrationValues Class Reference	70
4.20.1 Detailed Description	71
4.20.2 Constructor & Destructor Documentation	71
4.20.2.1 MigrationValues()	71
4.20.3 Field Documentation	71
4.20.3.1 DispersionOrder	72
4.20.3.2 DispersionTruncation	72
4.20.3.3 InflowActive	72
4.20.3.4 InflowParameterA	72
4.20.3.5 InflowParameterB	72
4.20.3.6 MarkovDispersionStrength	72
4.20.3.7 MaxStealFraction	73
4.21 OutputValues Class Reference	73
4.21.1 Constructor & Destructor Documentation	74
4.21.1.1 OutputValues()	74
4.21.2 Member Function Documentation	74
4.21.2.1 Initialise()	74
4.21.3 Field Documentation	74
4.21.3.1 AbsoluteColdGasFile	74
4.21.3.2 AbsoluteHotGasFile	74
4.21.3.3 ChemicalPrefactor	75
4.21.3.4 ColdGasDataFile	75
4.21.3.5 Config	75
4.21.3.6 EventRateFile	75
4.21.3.7 GalaxyMassFile	75
4.21.3.8 HotGasDataFile	75
4.21.3.9 LogarithmicColdGasFile	76
4.21.3.10 LogarithmicHotGasFile	76

4.21.3.11 Root . . . . .	76
4.21.3.12 StarFile . . . . .	76
4.21.3.13 YieldSubdir . . . . .	76
4.22 ParamList Class Reference . . . . .	77
4.22.1 Detailed Description . . . . .	78
4.22.2 Member Function Documentation . . . . .	78
4.22.2.1 Configure() . . . . .	78
4.22.2.2 Initialise() . . . . .	78
4.22.2.3 StreamContentsTo() . . . . .	78
4.22.3 Field Documentation . . . . .	78
4.22.3.1 argPointers . . . . .	79
4.23 RemnantOutput Struct Reference . . . . .	79
4.23.1 Field Documentation . . . . .	79
4.23.1.1 Mass . . . . .	79
4.23.1.2 Type . . . . .	79
4.24 RemnantPopulation Class Reference . . . . .	79
4.24.1 Constructor & Destructor Documentation . . . . .	80
4.24.1.1 RemnantPopulation() . . . . .	80
4.24.2 Member Function Documentation . . . . .	80
4.24.2.1 Decay() . . . . .	80
4.24.2.2 Feed() [1/2] . . . . .	80
4.24.2.3 Feed() [2/2] . . . . .	81
4.24.2.4 Mass() . . . . .	81
4.24.3 Field Documentation . . . . .	81
4.24.3.1 BlackHoleMass . . . . .	81
4.24.3.2 DormantNSMass . . . . .	81
4.24.3.3 DormantWDMass . . . . .	81
4.24.3.4 LongSNIaBuffer . . . . .	81
4.24.3.5 NSMBuffer . . . . .	81
4.24.3.6 NSMYield . . . . .	82
4.24.3.7 Param . . . . .	82
4.24.3.8 ShortSNIaBuffer . . . . .	82
4.24.3.9 SNIaYield . . . . .	82
4.25 ResourceValues Class Reference . . . . .	82
4.25.1 Constructor & Destructor Documentation . . . . .	83
4.25.1.1 ResourceValues() . . . . .	83
4.25.2 Member Function Documentation . . . . .	83
4.25.2.1 Initialise() . . . . .	83
4.25.3 Field Documentation . . . . .	83
4.25.3.1 IsochroneDirectory . . . . .	83
4.25.3.2 IsochroneRepository . . . . .	84
4.25.3.3 LifeTimeFile . . . . .	84

4.25.3.4 ResourceRoot . . . . .	84
4.25.3.5 WelcomeFile . . . . .	84
4.25.3.6 YieldRoot . . . . .	84
4.26 Ring Class Reference . . . . .	84
4.26.1 Constructor & Destructor Documentation . . . . .	85
4.26.1.1 Ring() . . . . .	85
4.26.2 Member Function Documentation . . . . .	86
4.26.2.1 ComputeSelectionFunction() . . . . .	86
4.26.2.2 Cool() . . . . .	86
4.26.2.3 KillStars() . . . . .	86
4.26.2.4 MakeStars() . . . . .	86
4.26.2.5 Mass() . . . . .	86
4.26.2.6 MetCheck() . . . . .	86
4.26.2.7 SaveChemicalHistory() . . . . .	87
4.26.2.8 SelectionEffect() . . . . .	87
4.26.2.9 Synthesis() . . . . .	87
4.26.2.10 TimeStep() . . . . .	87
4.26.2.11 UpdateMemory() . . . . .	87
4.26.3 Field Documentation . . . . .	87
4.26.3.1 Area . . . . .	88
4.26.3.2 ColdBuffer . . . . .	88
4.26.3.3 Data . . . . .	88
4.26.3.4 Gas . . . . .	88
4.26.3.5 HotBuffer . . . . .	88
4.26.3.6 CGMBuffer . . . . .	88
4.26.3.7 MaxMv . . . . .	88
4.26.3.8 MinMv . . . . .	88
4.26.3.9 Param . . . . .	89
4.26.3.10 Radius . . . . .	89
4.26.3.11 RadiusIndex . . . . .	89
4.26.3.12 SelectionGrid . . . . .	89
4.26.3.13 Stars . . . . .	89
4.26.3.14 Width . . . . .	89
4.27 SimpleYield Class Reference . . . . .	89
4.27.1 Constructor & Destructor Documentation . . . . .	90
4.27.1.1 SimpleYield() . . . . .	90
4.27.2 Member Function Documentation . . . . .	90
4.27.2.1 NSM_Initialise() . . . . .	90
4.27.2.2 operator>() . . . . .	91
4.27.2.3 RemnantInject() . . . . .	91
4.27.2.4 SNIa_Initialise() . . . . .	91
4.27.3 Field Documentation . . . . .	91

4.27.3.1 Grid	91
4.27.3.2 hotInjectionFraction	91
4.27.3.3 Param	91
4.27.3.4 Process	92
4.28 SLF_Functor Class Reference	92
4.28.1 Constructor & Destructor Documentation	92
4.28.1.1 SLF_Functor()	92
4.28.2 Member Function Documentation	92
4.28.2.1 LifeTime()	93
4.28.2.2 operator()()	93
4.28.2.3 PrecomputeGrid()	93
4.28.2.4 PredictLifetime()	93
4.28.2.5 ValueInquiry()	93
4.28.3 Field Documentation	93
4.28.3.1 NotComputed	93
4.28.3.2 Param	94
4.28.3.3 PrecomputedGrid	94
4.29 StarEvents Class Reference	94
4.29.1 Constructor & Destructor Documentation	94
4.29.1.1 StarEvents()	94
4.29.2 Member Function Documentation	95
4.29.2.1 AddHeaders()	95
4.29.2.2 Save()	95
4.29.3 Field Documentation	95
4.29.3.1 AGBDeaths	95
4.29.3.2 CCSN	95
4.29.3.3 ECSN	95
4.29.3.4 Efficiency	95
4.29.3.5 NSM	96
4.29.3.6 NStarsFormed	96
4.29.3.7 SNIa	96
4.29.3.8 StarMassFormed	96
4.30 StarReservoir Class Reference	96
4.30.1 Constructor & Destructor Documentation	97
4.30.1.1 StarReservoir()	97
4.30.2 Member Function Documentation	97
4.30.2.1 AliveMass()	97
4.30.2.2 AssignMagnitudes()	97
4.30.2.3 DeadMass()	98
4.30.2.4 Death()	98
4.30.2.5 Form()	98
4.30.2.6 Observations()	98

4.30.2.7 PrintStatus()	98
4.30.2.8 SaveEventRate()	98
4.30.2.9 SFR_GasLoss()	98
4.30.2.10 YieldsFrom()	99
4.30.3 Field Documentation	99
4.30.3.1 Data	99
4.30.3.2 EventRate	99
4.30.3.3 IMF	99
4.30.3.4 Param	99
4.30.3.5 ParentArea	99
4.30.3.6 ParentRing	99
4.30.3.7 Population	100
4.30.3.8 PopulationIndex	100
4.30.3.9 Remnants	100
4.30.3.10 SLF	100
4.30.3.11 Temp_Mass	100
4.30.3.12 YieldOutput	100
4.31 StellarPopulation Class Reference	100
4.31.1 Constructor & Destructor Documentation	102
4.31.1.1 StellarPopulation()	102
4.31.2 Member Function Documentation	102
4.31.2.1 Active()	102
4.31.2.2 CatalogueEntry()	102
4.31.2.3 CatalogueHeaders()	102
4.31.2.4 Death()	102
4.31.2.5 FormStars()	103
4.31.2.6 FullDeathScan()	103
4.31.2.7 Mass()	103
4.31.2.8 MonotonicDeathScan()	103
4.31.2.9 operator[]() [1/2]	103
4.31.2.10 operator[]() [2/2]	103
4.31.2.11 PrepareIMF()	104
4.31.2.12 RecoverMatter()	104
4.31.2.13 Relic() [1/2]	104
4.31.2.14 Relic() [2/2]	104
4.31.3 Field Documentation	104
4.31.3.1 AGBYield	104
4.31.3.2 Age	104
4.31.3.3 BirthGas	105
4.31.3.4 BirthIndex	105
4.31.3.5 BirthRadius	105
4.31.3.6 CCSNYield	105

4.31.3.7 Data	105
4.31.3.8 DepletionIndex	105
4.31.3.9 Distribution	105
4.31.3.10 ECSNYield	105
4.31.3.11 IMF	106
4.31.3.12 ImmortalStars	106
4.31.3.13 internal_MassCounter	106
4.31.3.14 IsDepleted	106
4.31.3.15 IsLifetimeMonotonic	106
4.31.3.16 Metallicity	106
4.31.3.17 Param	106
4.31.3.18 SLF	106
4.31.3.19 TempGas	107
4.32 StellarValues Class Reference	107
4.32.1 Detailed Description	108
4.32.2 Constructor & Destructor Documentation	108
4.32.2.1 StellarValues()	108
4.32.3 Member Function Documentation	108
4.32.3.1 Initialise()	108
4.32.4 Field Documentation	109
4.32.4.1 EjectionFraction	109
4.32.4.2 FeedbackFactor	109
4.32.4.3 IMF_Slope	109
4.32.4.4 ImmortalMass	109
4.32.4.5 LogZDelta	109
4.32.4.6 LogZGrid	109
4.32.4.7 LogZResolution	110
4.32.4.8 MassDeltas	110
4.32.4.9 MassGrid	110
4.32.4.10 MassResolution	110
4.32.4.11 MaxLogZ	110
4.32.4.12 MaxStellarMass	110
4.32.4.13 MinLogZ	111
4.32.4.14 MinStellarMass	111
4.32.4.15 SchmidtDensityCut	111
4.32.4.16 SchmidtLowPower	111
4.32.4.17 SchmidtMainPower	111
4.32.4.18 SchmidtPrefactor	111
4.33 ThermalValues Class Reference	112
4.33.1 Detailed Description	112
4.33.2 Constructor & Destructor Documentation	112
4.33.2.1 ThermalValues()	113



4.33.3 Field Documentation	113
4.33.3.1 ChimneyFactor	113
4.33.3.2 CoolingPower	113
4.33.3.3 DormantHotFraction	113
4.33.3.4 FeedbackEjectFactor	113
4.33.3.5 GasCoolingTimeScale	113
4.33.3.6 HotInjection_AGB	114
4.33.3.7 HotInjection_CCSN	114
4.33.3.8 HotInjection_NSM	114
4.33.3.9 HotInjection_SNIa	114
4.33.3.10 NumericalResolution	114
4.34 YieldBracket Struct Reference	114
4.34.1 Constructor & Destructor Documentation	115
4.34.1.1 YieldBracket() [1/3]	115
4.34.1.2 YieldBracket() [2/3]	115
4.34.1.3 YieldBracket() [3/3]	115
4.34.2 Member Function Documentation	115
4.34.2.1 Interpolate()	115
4.34.3 Field Documentation	116
4.34.3.1 hasSingle	116
4.34.3.2 isEnclosed	116
4.34.3.3 LowerRidge	116
4.34.3.4 UpperRidge	116
4.35 YieldGrid Class Reference	116
4.35.1 Constructor & Destructor Documentation	117
4.35.1.1 YieldGrid()	117
4.35.2 Member Function Documentation	117
4.35.2.1 AGB_Initialise()	118
4.35.2.2 CCSN_Initialise()	118
4.35.2.3 CreateGrid()	118
4.35.2.4 ECSN_Initialise()	118
4.35.2.5 ElementDestruction()	118
4.35.2.6 ElementProduction()	118
4.35.2.7 GetBracket()	119
4.35.2.8 InitialiseLargeGrid()	119
4.35.2.9 LoadLimongiYields()	119
4.35.2.10 LoadMaederYields()	119
4.35.2.11 LoadMarigoYields()	119
4.35.2.12 LoadOrfeoYields()	119
4.35.2.13 MetallicityInterpolation()	119
4.35.2.14 operator()()	120
4.35.2.15 PurityEnforce()	120

4.35.2.16 SaveGrid()	120
4.35.2.17 StellarInject()	120
4.35.3 Field Documentation	120
4.35.3.1 Grid	120
4.35.3.2 hotInjectionFraction	120
4.35.3.3 MassOffset	121
4.35.3.4 Param	121
4.35.3.5 Process	121
4.35.3.6 RemnantLocation	121
4.35.3.7 RidgeStorage	121
4.35.3.8 SourcePriority	121
4.36 YieldPoint Struct Reference	121
4.36.1 Constructor & Destructor Documentation	122
4.36.1.1 YieldPoint() [1/2]	122
4.36.1.2 YieldPoint() [2/2]	122
4.36.2 Field Documentation	122
4.36.2.1 Mass	122
4.36.2.2 Yield	122
4.37 YieldRidge Class Reference	123
4.37.1 Constructor & Destructor Documentation	123
4.37.1.1 YieldRidge() [1/2]	123
4.37.1.2 YieldRidge() [2/2]	123
4.37.2 Field Documentation	123
4.37.2.1 Points	123
4.37.2.2 Source	124
4.37.2.3 Z	124
4.38 YieldValues Class Reference	124
4.38.1 Constructor & Destructor Documentation	125
4.38.1.1 YieldValues()	125
4.38.2 Member Function Documentation	125
4.38.2.1 Initialise()	125
4.38.3 Field Documentation	125
4.38.3.1 CCSN_MassCut	125
4.38.3.2 CODwarf_MassCut	125
4.38.3.3 Collapse_MassCut	125
4.38.3.4 ECSN_Fraction	126
4.38.3.5 ECSN_MassCut	126
4.38.3.6 MassOverhang	126
4.38.3.7 NSM_ActiveFraction	126
4.38.3.8 NSM_DelayTime	126
4.38.3.9 NSM_Scale	126
4.38.3.10 NSM_TypicalMass	126

4.38.3.11 ProcessNames . . . . .	127
4.38.3.12 ProcessTypes . . . . .	127
4.38.3.13 SNIa_ActiveFraction . . . . .	127
4.38.3.14 SNIa_DelayTime . . . . .	127
4.38.3.15 SNIa_LongFraction . . . . .	127
4.38.3.16 SNIa_LongScale . . . . .	127
4.38.3.17 SNIa_ShortScale . . . . .	127
4.38.3.18 SNIa_TypicalMass . . . . .	127
4.38.3.19 TargetNi56Yield . . . . .	127
<b>5 File Documentation</b>	<b>129</b>
5.1 /Users/jf20/Documents/Physics/RAMICES_II/src/Galaxy/Galaxy.h File Reference . . . . .	129
5.1.1 Enumeration Type Documentation . . . . .	129
5.1.1.1 ParallelJob . . . . .	129
5.2 Galaxy.h . . . . .	130
5.3 /Users/jf20/Documents/Physics/RAMICES_II/src/Galaxy/MigrationMatrix.h File Reference . . . . .	131
5.4 MigrationMatrix.h . . . . .	131
5.5 /Users/jf20/Documents/Physics/RAMICES_II/src/Galaxy/Ring.h File Reference . . . . .	132
5.6 Ring.h . . . . .	132
5.7 /Users/jf20/Documents/Physics/RAMICES_II/src/Gas/Gas.h File Reference . . . . .	133
5.8 Gas.h . . . . .	133
5.9 /Users/jf20/Documents/Physics/RAMICES_II/src/Gas/GasReservoir.h File Reference . . . . .	133
5.10 GasReservoir.h . . . . .	134
5.11 /Users/jf20/Documents/Physics/RAMICES_II/src/Gas/GasStream.h File Reference . . . . .	135
5.12 GasStream.h . . . . .	135
5.13 /Users/jf20/Documents/Physics/RAMICES_II/src/Parameters/EnumSets.h File Reference . . . . .	136
5.13.1 Enumeration Type Documentation . . . . .	136
5.13.1.1 ElementID . . . . .	136
5.13.1.2 RemnantType . . . . .	137
5.13.1.3 SourceID . . . . .	137
5.13.1.4 SourceProcess . . . . .	138
5.13.1.5 YieldProcess . . . . .	138
5.14 EnumSets.h . . . . .	138
5.15 /Users/jf20/Documents/Physics/RAMICES_II/src/Parameters/GlobalParameters.h File Reference . . . . .	139
5.15.1 Macro Definition Documentation . . . . .	139
5.15.1.1 PI . . . . .	139
5.16 GlobalParameters.h . . . . .	139
5.17 /Users/jf20/Documents/Physics/RAMICES_II/src/Parameters/InitialisedData.h File Reference . . . . .	140
5.18 InitialisedData.h . . . . .	140
5.19 /Users/jf20/Documents/Physics/RAMICES_II/src/Parameters/List.h File Reference . . . . .	141
5.20 List.h . . . . .	141
5.21 /Users/jf20/Documents/Physics/RAMICES_II/src/Parameters/ParameterLists.h File Reference . . . . .	142

5.22 ParameterLists.h . . . . .	142
5.23 /Users/jf20/Documents/Physics/RAMICES_II/src/Stars/IMF.h File Reference . . . . .	147
5.24 IMF.h . . . . .	147
5.25 /Users/jf20/Documents/Physics/RAMICES_II/src/Stars/IsochroneTracker.h File Reference . . . . .	148
5.25.1 Enumeration Type Documentation . . . . .	148
5.25.1.1 IsochroneProperties . . . . .	148
5.25.2 Variable Documentation . . . . .	149
5.25.2.1 PropertyNames . . . . .	149
5.26 IsochroneTracker.h . . . . .	149
5.27 /Users/jf20/Documents/Physics/RAMICES_II/src/Stars/RemnantPopulation.h File Reference . . . . .	150
5.28 RemnantPopulation.h . . . . .	150
5.29 /Users/jf20/Documents/Physics/RAMICES_II/src/Stars/SLF.h File Reference . . . . .	151
5.30 SLF.h . . . . .	151
5.31 /Users/jf20/Documents/Physics/RAMICES_II/src/Stars/StarEvents.h File Reference . . . . .	151
5.32 StarEvents.h . . . . .	152
5.33 /Users/jf20/Documents/Physics/RAMICES_II/src/Stars/StarReservoir.h File Reference . . . . .	152
5.34 StarReservoir.h . . . . .	152
5.35 /Users/jf20/Documents/Physics/RAMICES_II/src/Stars/StellarPopulation.h File Reference . . . . .	153
5.36 StellarPopulation.h . . . . .	153
5.37 /Users/jf20/Documents/Physics/RAMICES_II/src/Yields/SimpleYield.h File Reference . . . . .	155
5.38 SimpleYield.h . . . . .	155
5.39 /Users/jf20/Documents/Physics/RAMICES_II/src/Yields/YieldGrid.h File Reference . . . . .	155
5.40 YieldGrid.h . . . . .	156
5.41 /Users/jf20/Documents/Physics/RAMICES_II/src/Yields/YieldRidge.h File Reference . . . . .	157
5.42 YieldRidge.h . . . . .	157
<b>Index</b>	<b>159</b>

# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

Galaxy . . . . .	12
Gas . . . . .	26
GasReservoir . . . . .	30
GasStream . . . . .	37
GlobalParameters . . . . .	45
IMF_Functor . . . . .	49
InitialisedData . . . . .	51
Integral . . . . .	55
Interpolator . . . . .	56
IsochroneCube . . . . .	57
IsochroneEntry . . . . .	58
IsochroneTracker . . . . .	60
IsoMass . . . . .	63
MassReport . . . . .	65
MigrationMatrix . . . . .	68
ParamList . . . . .	77
CatalogueValues . . . . .	7
ElementValues . . . . .	9
GalaxyValues . . . . .	20
MetaValues . . . . .	66
MigrationValues . . . . .	70
OutputValues . . . . .	73
ResourceValues . . . . .	82
StellarValues . . . . .	107
ThermalValues . . . . .	112
YieldValues . . . . .	124
RemnantOutput . . . . .	79
RemnantPopulation . . . . .	79
Ring . . . . .	84
SimpleYield . . . . .	89
SLF_Functor . . . . .	92
StarEvents . . . . .	94
StarReservoir . . . . .	96
StellarPopulation . . . . .	100
YieldBracket . . . . .	114
YieldGrid . . . . .	116
YieldPoint . . . . .	121
YieldRidge . . . . .	123



## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">CatalogueValues</a>	7
<a href="#">ElementValues</a>	
The elemental suboptions contains variables and data associated with the solar abundances (and where to locate them), and how to extract and extrapolate the yield data from files	9
<a href="#">Galaxy</a>	12
<a href="#">GalaxyValues</a>	
The galaxy suboptions contains variables associated with the galaxy as a whole, such as the maximum radius, and various mass/infall properties	20
<a href="#">Gas</a>	26
<a href="#">GasReservoir</a>	30
<a href="#">GasStream</a>	37
<a href="#">GlobalParameters</a>	
A package of global parameter objects which can be passed around by reference. Most of the internal values are set as JSL::Argument objects and so can be initialised from the command line / config file	45
<a href="#">IMF_Functor</a>	49
<a href="#">InitialisedData</a>	
These will act like globally-defined functions, but have the scope for modifying themselves as they go along	51
<a href="#">Integral</a>	55
<a href="#">Interpolator</a>	56
<a href="#">IsochroneCube</a>	57
<a href="#">IsochroneEntry</a>	58
<a href="#">IsochroneTracker</a>	60
<a href="#">IsoMass</a>	
A simple struct for tracking the number of stars of a given mass	63
<a href="#">MassReport</a>	65
<a href="#">MetaValues</a>	
The <a href="#">MetaValues</a> contains variables associated with the base-level information about the simulation - the number of cores to access, the timesteps etc	66
<a href="#">MigrationMatrix</a>	68
<a href="#">MigrationValues</a>	
Holds values associated with how matter moves throughout the disc	70
<a href="#">OutputValues</a>	73

[ParamList](#)

A Generic superclass structure so that I can heterogenously loop over the various members of [GlobalParameters](#) without writing it all out arduously. Also provides a consistent interface with the JSL::Argument environment . . . . .

77

[RemnantOutput](#) . . . . . 79

[RemnantPopulation](#) . . . . . 79

[ResourceValues](#) . . . . . 82

[Ring](#) . . . . . 84

[SimpleYield](#) . . . . . 89

[SLF\\_Functor](#) . . . . . 92

[StarEvents](#) . . . . . 94

[StarReservoir](#) . . . . . 96

[StellarPopulation](#) . . . . . 100

[StellarValues](#)

The subset of values associated with stars + their remnants . . . . . 107

[ThermalValues](#)

Thermal suboptions contain variables which deal with the thermal subroutines - cooling timescales injection fractions etc . . . . .

112

[YieldBracket](#) . . . . . 114

[YieldGrid](#) . . . . . 116

[YieldPoint](#) . . . . . 121

[YieldRidge](#) . . . . . 123

[YieldValues](#) . . . . . 124



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

/Users/jf20/Documents/Physics/RAMICES_II/src/Galaxy/ <a href="#">Galaxy.h</a> . . . . .	129
/Users/jf20/Documents/Physics/RAMICES_II/src/Galaxy/ <a href="#">MigrationMatrix.h</a> . . . . .	131
/Users/jf20/Documents/Physics/RAMICES_II/src/Galaxy/ <a href="#">Ring.h</a> . . . . .	132
/Users/jf20/Documents/Physics/RAMICES_II/src/Gas/ <a href="#">Gas.h</a> . . . . .	133
/Users/jf20/Documents/Physics/RAMICES_II/src/Gas/ <a href="#">GasReservoir.h</a> . . . . .	133
/Users/jf20/Documents/Physics/RAMICES_II/src/Gas/ <a href="#">GasStream.h</a> . . . . .	135
/Users/jf20/Documents/Physics/RAMICES_II/src/Parameters/ <a href="#">EnumSets.h</a> . . . . .	136
/Users/jf20/Documents/Physics/RAMICES_II/src/Parameters/ <a href="#">GlobalParameters.h</a> . . . . .	139
/Users/jf20/Documents/Physics/RAMICES_II/src/Parameters/ <a href="#">InitialisedData.h</a> . . . . .	140
/Users/jf20/Documents/Physics/RAMICES_II/src/Parameters/ <a href="#">List.h</a> . . . . .	141
/Users/jf20/Documents/Physics/RAMICES_II/src/Parameters/ <a href="#">ParameterLists.h</a> . . . . .	142
/Users/jf20/Documents/Physics/RAMICES_II/src/Stars/ <a href="#">IMF.h</a> . . . . .	147
/Users/jf20/Documents/Physics/RAMICES_II/src/Stars/ <a href="#">IsochroneTracker.h</a> . . . . .	148
/Users/jf20/Documents/Physics/RAMICES_II/src/Stars/ <a href="#">RemnantPopulation.h</a> . . . . .	150
/Users/jf20/Documents/Physics/RAMICES_II/src/Stars/ <a href="#">SLF.h</a> . . . . .	151
/Users/jf20/Documents/Physics/RAMICES_II/src/Stars/ <a href="#">StarEvents.h</a> . . . . .	151
/Users/jf20/Documents/Physics/RAMICES_II/src/Stars/ <a href="#">StarReservoir.h</a> . . . . .	152
/Users/jf20/Documents/Physics/RAMICES_II/src/Stars/ <a href="#">StellarPopulation.h</a> . . . . .	153
/Users/jf20/Documents/Physics/RAMICES_II/src/Yields/ <a href="#">SimpleYield.h</a> . . . . .	155
/Users/jf20/Documents/Physics/RAMICES_II/src/Yields/ <a href="#">YieldGrid.h</a> . . . . .	155
/Users/jf20/Documents/Physics/RAMICES_II/src/Yields/ <a href="#">YieldRidge.h</a> . . . . .	157



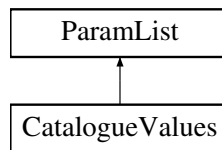
## Chapter 4

# Data Structure Documentation

### 4.1 CatalogueValues Class Reference

```
#include <ParameterLists.h>
```

Inheritance diagram for CatalogueValues:



#### Public Member Functions

- [CatalogueValues](#) ()

*Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.*

#### Data Fields

- Argument< bool > [SynthesisActive](#) = Argument<bool>(true,"stellar-synthesis")
- Argument< double > [IsochroneTimeStep](#) = Argument<double>(0.1,"isochrone-dt")  
*The timesteps used to interpolate isochrones over.*
- Argument< double > [IsochroneMagnitudeResolution](#) = Argument<double>(100,"isochrone-mag-resolution")
- Argument< double > [SolarRadius](#) = Argument<double>(8.2,"solar-radius")
- Argument< int > [RadialResolution](#) = Argument<int>(3,"isochrone-radial-resolution")
- Argument< int > [AzimuthalResolution](#) = Argument<int>(360,"isochrone-radial-resolution")
- Argument< double > [VerticalHeightStart](#) = Argument<double>(0.05,"vertical-height-z0")
- Argument< double > [VerticalHeightScaling](#) = Argument<double>(0.3,"vertical-height-scaling")
- Argument< double > [VerticalHeightPower](#) = Argument<double>(0.66,"vertical-height-power")
- Argument< int > [SampleCount](#) = Argument<int>(10,"catalogue-sample")

## Additional Inherited Members

### 4.1.1 Constructor & Destructor Documentation

#### 4.1.1.1 CatalogueValues()

```
CatalogueValues::CatalogueValues ( ) [inline]
```

Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.

### 4.1.2 Field Documentation

#### 4.1.2.1 AzimuthalResolution

```
Argument<int> CatalogueValues::AzimuthalResolution = Argument<int>(360, "isochrone-radial-resolution")
```

#### 4.1.2.2 IsochroneMagnitudeResolution

```
Argument<double> CatalogueValues::IsochroneMagnitudeResolution = Argument<double>(100, "isochrone-mag-resolut
```

#### 4.1.2.3 IsochroneTimeStep

```
Argument<double> CatalogueValues::IsochroneTimeStep = Argument<double>(0.1, "isochrone-dt")
```

The timesteps used to interpolate isochrones over.

#### 4.1.2.4 RadialResolution

```
Argument<int> CatalogueValues::RadialResolution = Argument<int>(3, "isochrone-radial-resolution")
```

#### 4.1.2.5 SampleCount

```
Argument<int> CatalogueValues::SampleCount = Argument<int>(10, "catalogue-sample")
```

#### 4.1.2.6 SolarRadius

```
Argument<double> CatalogueValues::SolarRadius = Argument<double>(8.2, "solar-radius")
```

#### 4.1.2.7 SynthesisActive

```
Argument<bool> CatalogueValues::SynthesisActive = Argument<bool>(true, "stellar-synthesis")
```

#### 4.1.2.8 VerticalHeightPower

```
Argument<double> CatalogueValues::VerticalHeightPower = Argument<double>(0.66, "vertical-height-power")
```

#### 4.1.2.9 VerticalHeightScaling

```
Argument<double> CatalogueValues::VerticalHeightScaling = Argument<double>(0.3, "vertical-height-scaling")
```

#### 4.1.2.10 VerticalHeightStart

```
Argument<double> CatalogueValues::VerticalHeightStart = Argument<double>(0.05, "vertical-height-z0")
```

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

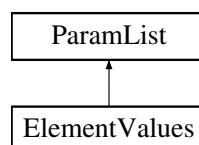
- /Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/[ParameterLists.h](#)

## 4.2 ElementValues Class Reference

The elemental suboptions contains variables and data associated with the solar abundances (and where to locate them), and how to extract and extrapolate the yield data from files.

```
#include <ParameterLists.h>
```

Inheritance diagram for ElementValues:



## Public Member Functions

- [ElementValues](#) ()  
*Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.*
- virtual void [Initialise](#) (std::string resourceRoot)  
*An overload of a normally empty function. Loads in the values fo the solar abundance data file into the SolarAbundances vector.*
- void [GiveElementsNames](#) ()  
*A fairly dumbly-written function which sorts the elemental symbols in ElementNames into the order specified by the global id-enum.*

## Data Fields

- std::vector< std::string > [ElementNames](#)  
*Human readable names for the elements, in the order associated with the ElementIDs. These names are primarily elemental symbols, except Metals, which uses "Z".*
- std::vector< int > [ProtonCounts](#)
- std::vector< double > [SolarAbundances](#)  
*Solar abundances (in mass units) of the elements, in the order associated with the ElementIDs.*
- Argument< std::string > [SolarAbundanceFile](#) = Argument<std::string>("ChemicalData/SolarAbundances←\_Maria.dat", "solar-values-file")  
*The file in which the solar abundances can be found as a csv.*
- Argument< int > [SolarAbundanceFileNameColumn](#) = Argument<int>(0, "solar-values-name-col")  
*The column of the solar abundance files which contains the ElementName for cross matching.*
- Argument< int > [SolarAbundanceFileDataColumn](#) = Argument<int>(3, "solar-values-data-col")  
*The column of the solar abundance file which contains the relevant solar abundance value to be saved to memory.*

## Additional Inherited Members

### 4.2.1 Detailed Description

The elemental suboptions contains variables and data associated with the solar abundances (and where to locate them), and how to extract and extrapolate the yield data from files.

### 4.2.2 Constructor & Destructor Documentation

#### 4.2.2.1 ElementValues()

```
ElementValues::ElementValues ( ) [inline]
```

Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.

### 4.2.3 Member Function Documentation

#### 4.2.3.1 GiveElementsNames()

```
void ElementValues::GiveElementsNames ( )
```

A fairly dumbly-written function which sorts the elemental symbols in ElementNames into the order specified by the global id-enum.

#### 4.2.3.2 Initialise()

```
virtual void ElementValues::Initialise (
    std::string resourceRoot ) [virtual]
```

An overload of a normally empty function. Loads in the values for the solar abundance data file into the Solar↔Abundances vector.

Reimplemented from [ParamList](#).

### 4.2.4 Field Documentation

#### 4.2.4.1 ElementNames

```
std::vector<std::string> ElementValues::ElementNames
```

Human readable names for the elements, in the order associated with the ElementIDs. These names are primarily elemental symbols, except Metals, which uses "Z".

#### 4.2.4.2 ProtonCounts

```
std::vector<int> ElementValues::ProtonCounts
```

#### 4.2.4.3 SolarAbundanceFile

```
Argument<std::string> ElementValues::SolarAbundanceFile = Argument<std::string>("Chemical↔Data/SolarAbundances_Maria.dat","solar-values-file")
```

The file in which the solar abundances can be found as a csv.

#### 4.2.4.4 SolarAbundanceFileDataColumn

```
Argument<int> ElementValues::SolarAbundanceFileDataColumn = Argument<int>(3, "solar-values-data-col")
```

The column of the solar abundance file which contains the relevant solar abundance value to be saved to memory.

#### 4.2.4.5 SolarAbundanceFileNameColumn

```
Argument<int> ElementValues::SolarAbundanceFileNameColumn = Argument<int>(0, "solar-values-name-col")
```

The column of the solar abundance files which contains the ElementName for cross matching.

#### 4.2.4.6 SolarAbundances

```
std::vector<double> ElementValues::SolarAbundances
```

Solar abundances (in mass units) of the elements, in the order associated with the ElementIDs.

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/[ParameterLists.h](#)

## 4.3 Galaxy Class Reference

```
#include <Galaxy.h>
```

### Public Member Functions

- [Galaxy](#) ([InitialisedData](#) &[Data](#))
- void [Evolve](#) ()
- void [SynthesiseObservations](#) ()

### Data Fields

- std::vector< [Ring](#) > [Rings](#)



## Private Member Functions

- void [LaunchParallelOperation](#) (int time, int nOperations, [ParallelJob](#) type)
- double [GasScaleLength](#) (double t)
- double [InfallMass](#) (double t)
- void [InsertInfallingGas](#) (int ring, double amount)
- void [Infall](#) (double t)
- void [RingEvolve](#) (int timestep, int ringStart, int ringEnd)
- void [ScatterYields](#) (int timestep, int ringStart, int ringEnd)
- void [ScatterGas](#) (int timestep)
- void [ComputeScattering](#) (int t)
- void [CompoundScattering](#) (int currentTime, int timeStart, int timeEnd)
- void [AssignMagnitudes](#) (int time, int ringstart, int ringend)
- double [PredictSurfaceDensity](#) (double radius, double width, double totalGasMass, double scalelength)
- double [GasMass](#) ()
- double [ColdGasMass](#) ()
- double [StarMass](#) ()
- void [CGMOperations](#) ()
- double [RelicMass](#) ()
- double [Mass](#) ()
- void [SaveState](#) (double t)
- void [SaveState\\_Mass](#) (double t)
- void [SaveState\\_Enrichment](#) (double t)
- void [SaveState\\_Events](#) (double t)
- void [ComputeVisibilityFunction](#) ()
- void [SelectionFunction](#) (int ringstart, int ringend, int threadID)
- void [StellarSynthesis](#) (int ringstart, int ringend, int threadID)

## Static Private Member Functions

- static std::string [MassHeaders](#) ()

## Private Attributes

- std::vector< std::thread > [Threads](#)
- std::vector< [MigrationMatrix](#) > [Migrator](#)
- [GasReservoir](#) CGM
- const [GlobalParameters](#) & [Param](#)
- [InitialisedData](#) & [Data](#)
- std::vector< double > [RingMasses](#)
- std::vector< std::string > [SynthesisOutput](#)
- std::vector< double > [SynthesisProgress](#)
- double [DimmestStar](#)
- double [BrightestStar](#)
- int [ParallelBars](#) = 0

### 4.3.1 Constructor & Destructor Documentation

#### 4.3.1.1 Galaxy()

```
Galaxy::Galaxy (
    InitialisedData & Data )
```

### 4.3.2 Member Function Documentation

#### 4.3.2.1 AssignMagnitudes()

```
void Galaxy::AssignMagnitudes (
    int time,
    int ringstart,
    int ringend ) [private]
```

#### 4.3.2.2 ColdGasMass()

```
double Galaxy::ColdGasMass ( ) [private]
```

#### 4.3.2.3 CompoundScattering()

```
void Galaxy::CompoundScattering (
    int currentTime,
    int timeStart,
    int timeEnd ) [private]
```

#### 4.3.2.4 ComputeScattering()

```
void Galaxy::ComputeScattering (
    int t ) [private]
```

#### 4.3.2.5 ComputeVisibilityFunction()

```
void Galaxy::ComputeVisibilityFunction ( ) [private]
```

#### 4.3.2.6 Evolve()

```
void Galaxy::Evolve ( )
```

#### 4.3.2.7 GasMass()

```
double Galaxy::GasMass ( ) [private]
```

#### 4.3.2.8 GasScaleLength()

```
double Galaxy::GasScaleLength (
    double t ) [private]
```

#### 4.3.2.9 CGMOperations()

```
void Galaxy::CGMOperations ( ) [private]
```

#### 4.3.2.10 Infall()

```
void Galaxy::Infall (
    double t ) [private]
```

#### 4.3.2.11 InfallMass()

```
double Galaxy::InfallMass (
    double t ) [private]
```

#### 4.3.2.12 InsertInfallingGas()

```
void Galaxy::InsertInfallingGas (
    int ring,
    double amount ) [private]
```

#### 4.3.2.13 LaunchParallelOperation()

```
void Galaxy::LaunchParallelOperation (
    int time,
    int nOperations,
    ParallelJob type ) [private]
```

#### 4.3.2.14 Mass()

```
double Galaxy::Mass ( ) [private]
```

#### 4.3.2.15 MassHeaders()

```
static std::string Galaxy::MassHeaders ( ) [static], [private]
```

#### 4.3.2.16 PredictSurfaceDensity()

```
double Galaxy::PredictSurfaceDensity (
    double radius,
    double width,
    double totalGasMass,
    double scalelength ) [private]
```

#### 4.3.2.17 RelicMass()

```
double Galaxy::RelicMass ( ) [private]
```

#### 4.3.2.18 RingEvolve()

```
void Galaxy::RingEvolve (
    int timestep,
    int ringStart,
    int ringEnd ) [private]
```

#### 4.3.2.19 SaveState()

```
void Galaxy::SaveState (
    double t ) [private]
```

#### 4.3.2.20 SaveState\_Enrichment()

```
void Galaxy::SaveState_Enrichment (
    double t ) [private]
```

#### 4.3.2.21 SaveState\_Events()

```
void Galaxy::SaveState_Events (
    double t ) [private]
```

#### 4.3.2.22 SaveState\_Mass()

```
void Galaxy::SaveState_Mass (
    double t ) [private]
```

#### 4.3.2.23 ScatterGas()

```
void Galaxy::ScatterGas (
    int timestep ) [private]
```

#### 4.3.2.24 ScatterYields()

```
void Galaxy::ScatterYields (
    int timestep,
    int ringStart,
    int ringEnd ) [private]
```

#### 4.3.2.25 SelectionFunction()

```
void Galaxy::SelectionFunction (
    int ringstart,
    int ringend,
    int threadID ) [private]
```

#### 4.3.2.26 StarMass()

```
double Galaxy::StarMass ( ) [private]
```

#### 4.3.2.27 StellarSynthesis()

```
void Galaxy::StellarSynthesis (
    int ringstart,
    int ringend,
    int threadID ) [private]
```

#### 4.3.2.28 SynthesiseObservations()

```
void Galaxy::SynthesiseObservations ( )
```

### 4.3.3 Field Documentation

#### 4.3.3.1 BrightestStar

```
double Galaxy::BrightestStar [private]
```

#### 4.3.3.2 Data

```
InitialisedData& Galaxy::Data [private]
```

#### 4.3.3.3 DimmestStar

```
double Galaxy::DimmestStar [private]
```

#### 4.3.3.4 CGM

```
GasReservoir Galaxy::CGM [private]
```

#### 4.3.3.5 Migrator

```
std::vector<MigrationMatrix> Galaxy::Migrator [private]
```

#### 4.3.3.6 ParallelBars

```
int Galaxy::ParallelBars = 0 [private]
```

#### 4.3.3.7 Param

```
const GlobalParameters& Galaxy::Param [private]
```

#### 4.3.3.8 RingMasses

```
std::vector<double> Galaxy::RingMasses [private]
```

#### 4.3.3.9 Rings

```
std::vector<Ring> Galaxy::Rings
```

#### 4.3.3.10 SynthesisOutput

```
std::vector<std::string> Galaxy::SynthesisOutput [private]
```

#### 4.3.3.11 SynthesisProgress

```
std::vector<double> Galaxy::SynthesisProgress [private]
```

#### 4.3.3.12 Threads

```
std::vector<std::thread> Galaxy::Threads [private]
```

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

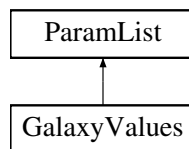
- /Users/jf20/Documents/Physics/RAMICES\_II/src/Galaxy/[Galaxy.h](#)

## 4.4 GalaxyValues Class Reference

The galaxy suboptions contains variables associated with the galaxy as a whole, such as the maximum radius, and various mass/infall properties.

```
#include <ParameterLists.h>
```

Inheritance diagram for GalaxyValues:



### Public Member Functions

- [GalaxyValues](#) ()  
*Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.*
- void [Initialise](#) (std::string resourceRoot)  
*A (hopefully) rarely used function which calls any additional functions which can only be called after the configuration has been run. For most members, this is an empty function.*



## Data Fields

- Argument< int > [RingCount](#) = Argument<int>(100,"rings")  
*The number of annuli into which the galaxy is split.*
- Argument< double > [Radius](#) = Argument<double>(20.0,"radius")  
*The cutoff radius of the galaxy.*
- Argument< bool > [UsingVariableRingWidth](#) = Argument<bool>(false,"variable-ring-width")
- Argument< double > [Ring0Width](#) = Argument<double>(0.05,"inner-ring-width")  
*Width of the innermost ring (kpc)*
- Argument< bool > [CGMAbsorbing](#) = Argument<bool>(true,"cgm-absorb")
- std::vector< double > [RingRadius](#)
- std::vector< double > [RingWidth](#)
- Argument< double > [PrimordialMass](#) = Argument<double>(2,"M0")  
*Initial in-situ mass of the galaxy (assumed to be 100% gas)*
- Argument< double > [PrimordialHotFraction](#) = Argument<double>(0,"primordial-hot")  
*Fraction of primordial gas which is hot.*
- Argument< double > [CGM\\_Mass](#) = Argument<double>(200,"cgm-mass")  
*Initial Mass of the CGM Reservoir.*
- Argument< double > [MinScaleLength](#) = Argument<double>(0.75,"scale-length-min")  
*The initial exponential scale length of the galaxy.*
- Argument< double > [MaxScaleLength](#) = Argument<double>(3.75,"scale-length-max")  
*The exponential scale length that the galaxy achieves at ScaleLengthFinalTime.*
- Argument< double > [ScaleLengthDelay](#) = Argument<double>(1.0,"scale-length-delay")  
*The delay time before the scale length begins to grow.*
- Argument< double > [ScaleLengthTimeScale](#) = Argument<double>(2.0,"scale-length-time")  
*The speed with which the scale length grows.*
- Argument< double > [ScaleLengthFinalTime](#) = Argument<double>(12.0,"scale-length-final")  
*The time at which the scale length stops growing at becomes fixed.*
- Argument< double > [InfallMass1](#) = Argument<double>(50,"M1")  
*The mass of the first (fast) exponential infall.*
- Argument< double > [InfallMass2](#) = Argument<double>(100,"M2")  
*The mass of the second (slow) exponential infall.*
- Argument< double > [InfallTime1](#) = Argument<double>(0.4,"b1")  
*The exponential timescale for the first (fast) exponential infall.*
- Argument< double > [InfallTime2](#) = Argument<double>(6.0,"b2")  
*The exponential timescale for the second (slow) exponential infall.*
- Argument< double > [InfallMassMerger](#) = Argument<double>(0,"merger-mass")
- Argument< double > [InfallTimeMerger](#) = Argument<double>(0.4,"merger-timescale")
- Argument< double > [MergerDelayTime](#) = Argument<double>(8,"merger-delay")
- Argument< double > [MergerTurnOnWidth](#) = Argument<double>(0.3,"merger-width")
- Argument< double > [MaxSFRFraction](#) = Argument<double>(0.95,"max-sfr")  
*maximum fraction which can be removed by SFR + associated feedback*

## Additional Inherited Members

### 4.4.1 Detailed Description

The galaxy suboptions contains variables associated with the galaxy as a whole, such as the maximum radius, and various mass/infall properties.

## 4.4.2 Constructor & Destructor Documentation

### 4.4.2.1 GalaxyValues()

```
GalaxyValues::GalaxyValues ( ) [inline]
```

Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.

## 4.4.3 Member Function Documentation

### 4.4.3.1 Initialise()

```
void GalaxyValues::Initialise (
    std::string resourceRoot ) [virtual]
```

A (hopefully) rarely used function which calls any additional functions which can only be called *after* the configuration has been run. For most members, this is an empty function.

Reimplemented from [ParamList](#).

## 4.4.4 Field Documentation

### 4.4.4.1 CGM\_Mass

```
Argument<double> GalaxyValues::CGM_Mass = Argument<double>(200, "cgm-mass")
```

Initial Mass of the CGM Reservoir.

### 4.4.4.2 CGMAbsorbing

```
Argument<bool> GalaxyValues::CGMAbsorbing = Argument<bool>(true, "cgm-absorb")
```

#### 4.4.4.3 InfallMass1

```
Argument<double> GalaxyValues::InfallMass1 = Argument<double>(50, "M1")
```

The mass of the first (fast) exponential infall.

#### 4.4.4.4 InfallMass2

```
Argument<double> GalaxyValues::InfallMass2 = Argument<double>(100, "M2")
```

The mass of the second (slow) exponential infall.

#### 4.4.4.5 InfallMassMerger

```
Argument<double> GalaxyValues::InfallMassMerger = Argument<double>(0, "merger-mass")
```

#### 4.4.4.6 InfallTime1

```
Argument<double> GalaxyValues::InfallTime1 = Argument<double>(0.4, "b1")
```

The exponential timescale for the first (fast) exponential infall.

#### 4.4.4.7 InfallTime2

```
Argument<double> GalaxyValues::InfallTime2 = Argument<double>(6.0, "b2")
```

The exponential timescale for the second (slow) exponential infall.

#### 4.4.4.8 InfallTimeMerger

```
Argument<double> GalaxyValues::InfallTimeMerger = Argument<double>(0.4, "merger-timescale")
```

#### 4.4.4.9 MaxScaleLength

```
Argument<double> GalaxyValues::MaxScaleLength = Argument<double>(3.75, "scale-length-max")
```

The exponential scale length that the galaxy achieves at ScaleLengthFinalTime.

#### 4.4.4.10 MaxSFRFraction

```
Argument<double> GalaxyValues::MaxSFRFraction = Argument<double>(0.95, "max-sfr")
```

maximum fraction which can be removed by SFR + associated feedback

#### 4.4.4.11 MergerDelayTime

```
Argument<double> GalaxyValues::MergerDelayTime = Argument<double>(8, "merger-delay")
```

#### 4.4.4.12 MergerTurnOnWidth

```
Argument<double> GalaxyValues::MergerTurnOnWidth = Argument<double>(0.3, "merger-width")
```

#### 4.4.4.13 MinScaleLength

```
Argument<double> GalaxyValues::MinScaleLength = Argument<double>(0.75, "scale-length-min")
```

The initial exponential scale length of the galaxy.

#### 4.4.4.14 PrimordialHotFraction

```
Argument<double> GalaxyValues::PrimordialHotFraction = Argument<double>(0, "primordial-hot")
```

Fraction of primordial gas which is hot.

#### 4.4.4.15 PrimordialMass

```
Argument<double> GalaxyValues::PrimordialMass = Argument<double>(2, "M0")
```

Initial in-situ mass of the galaxy (assumed to be 100% gas)

#### 4.4.4.16 Radius

```
Argument<double> GalaxyValues::Radius = Argument<double>(20.0, "radius")
```

The cutoff radius of the galaxy.

#### 4.4.4.17 Ring0Width

```
Argument<double> GalaxyValues::Ring0Width = Argument<double>(0.05, "inner-ring-width")
```

Width of the innermost ring (kpc)

#### 4.4.4.18 RingCount

```
Argument<int> GalaxyValues::RingCount = Argument<int>(100, "rings")
```

The number of annuli into which the galaxy is split.

#### 4.4.4.19 RingRadius

```
std::vector<double> GalaxyValues::RingRadius
```

#### 4.4.4.20 RingWidth

```
std::vector<double> GalaxyValues::RingWidth
```

#### 4.4.4.21 ScaleLengthDelay

```
Argument<double> GalaxyValues::ScaleLengthDelay = Argument<double>(1.0, "scale-length-delay")
```

The delay time before the scale length begins to grow.

#### 4.4.4.22 ScaleLengthFinalTime

```
Argument<double> GalaxyValues::ScaleLengthFinalTime = Argument<double>(12.0, "scale-length-final")
```

The time at which the scale length stops growing at becomes fixed.

#### 4.4.4.23 ScaleLengthTimeScale

```
Argument<double> GalaxyValues::ScaleLengthTimeScale = Argument<double>(2.0, "scale-length-time")
```

The speed with which the scale length grows.

#### 4.4.4.24 UsingVariableRingWidth

```
Argument<bool> GalaxyValues::UsingVariableRingWidth = Argument<bool>(false, "variable-ring-width")
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/[ParameterLists.h](#)

## 4.5 Gas Class Reference

```
#include <Gas.h>
```

### Public Member Functions

- [Gas](#) ()  
*Default Constructor initialises the chunk of gas to have zero mass.*
- [Gas](#) (const std::vector< double > &elements)  
*Turns the elements array into the [Species](#) entity, otherwise does nothing interesting.*
- double [Mass](#) ()
- double [Mass](#) () const
- double & [operator\[\]](#) ([ElementID](#) id)
- const double & [operator\[\]](#) ([ElementID](#) id) const  
*An annoyingly necessary redeclaration for when the object is const and normal references don't behave nicely.*

## Static Public Member Functions

- static [Gas Primordial](#) (double mass)
- static [Gas Empty](#) ()

## Private Member Functions

- void [CheckMass](#) ()

## Private Attributes

- `std::vector< double >` [Species](#)  
*The central mass array. Has [ElementCount](#) elements, indexed by *ElementID*.*
- bool [NeedsRecomputing](#)
- double [internal\\_Mass](#)

### 4.5.1 Detailed Description

The most basic gas object there is – an array of element-masses and some rules for interacting with it.

### 4.5.2 Constructor & Destructor Documentation

#### 4.5.2.1 [Gas\(\)](#) [1/2]

```
Gas::Gas ( )
```

Default Constructor initialises the chunk of gas to have zero mass.

#### 4.5.2.2 [Gas\(\)](#) [2/2]

```
Gas::Gas (
    const std::vector< double > & elements )
```

Turns the elements array into the [Species](#) entity, otherwise does nothing interesting.

### 4.5.3 Member Function Documentation

#### 4.5.3.1 CheckMass()

```
void Gas::CheckMass ( ) [private]
```

#### 4.5.3.2 Empty()

```
static Gas Gas::Empty ( ) [static]
```

##### Returns

A default-constructed object, but name is clear that the object is empty

#### 4.5.3.3 Mass() [1/2]

```
double Gas::Mass ( )
```

##### Returns

The current total mass within the [Species](#) array

#### 4.5.3.4 Mass() [2/2]

```
double Gas::Mass ( ) const
```

#### 4.5.3.5 operator[]() [1/2]

```
double & Gas::operator[] (
    ElementID id )
```

##### Returns

A reference to the indexed member of [Species](#), allowing for vector like access



#### 4.5.3.6 operator[]() [2/2]

```
const double & Gas::operator[] (
    ElementID id ) const
```

An annoyingly necessary redeclaration for when the object is const and normal references don't behave nicely.

#### 4.5.3.7 Primordial()

```
static Gas Gas::Primordial (
    double mass ) [static]
```

##### Returns

A gas object of the specified mass but with a primordial elemental abundance distribution ( $X = 0.75$ ,  $Y = 0.25$  etc)

### 4.5.4 Field Documentation

#### 4.5.4.1 internal\_Mass

```
double Gas::internal_Mass [private]
```

#### 4.5.4.2 NeedsRecomputing

```
bool Gas::NeedsRecomputing [private]
```

#### 4.5.4.3 Species

```
std::vector<double> Gas::Species [private]
```

The central mass array. Has [ElementCount](#) elements, indexed by ElementID.

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Gas/[Gas.h](#)

## 4.6 GasReservoir Class Reference

```
#include <GasReservoir.h>
```

### Public Member Functions

- [GasReservoir](#) ()  
*Default constructor. Initialises the [Components](#) and gives them their [SourceProcess](#) ID.*
- [GasReservoir](#) (const [GlobalParameters](#) &param)  
*useful constructor*
- [GasStream](#) & [operator\[\]](#) ([SourceProcess](#) source)  
*A vector-like access overload, allowing indexing into the [Components](#) vector using appropriate #SourceProcesses.*
- const [GasStream](#) & [operator\[\]](#) ([SourceProcess](#) source) const  
*An annoyingly necessary redeclaration for when constant references don't want to play ball.*
- double [Mass](#) ()
- double [ColdMass](#) ()
- double [HotMass](#) ()
- void [Absorb](#) (const [GasReservoir](#) &givingGas)  
*Transfer the contents of the input reservoir and sum them into the reservoir.*
- void [Absorb](#) (const [GasStream](#) &givingGas)  
*Transfer the contents of the input stream into the element of [Components](#) indicated by the input's [GasStream::Source](#) flag.*
- void [Absorb](#) (const [GasStream](#) &givingGas, double fraction)
- void [Absorb](#) (const std::vector< [GasStream](#) > &givingGas)
- void [Absorb](#) (const std::vector< [GasStream](#) > &givingGas, double fraction)
- void [AbsorbMemory](#) (int t, const [GasStream](#) &input)
- void [Deplete](#) (double amountToLose)  
*Calls [GasStream::Deplete\(double\)](#) on each element of [Components](#), keeping the relative mass contribution of each component equal.*
- void [Wipe](#) ()  
*Wipes all mass from the reservoir.*
- void [Deplete](#) (double amountToLose\_Cold, double amountToLose\_Hot)  
*Calls [GasStream::Deplete\(double, double\)](#) on each element of [Components](#), keeping the relative hot mass and cold mass contribution of each component equal.*
- void [Heat](#) (double amountToHeat)  
*Heats up the specified amount of gas into the hot reservoir, keeping the elemental abundances of the cold gas reservoir constant.*
- void [PassiveCool](#) (double dt, bool isCGM)  
*Executes the usual cooling mechanism.*
- void [TransferFrom](#) ([GasReservoir](#) &givingGas, double massToMove)  
*Transfers the specified amount of mass across from the target, removing the mass from the target and adding it to the current object. Maintains the thermal, source and elemental ratios of the source object.*
- void [TransferColdFrom](#) ([GasReservoir](#) &givingGas, double massToMove)  
*Transfers the specified amount of mass across from the target, removing the mass from the target and adding it to the current object. Maintains the source and elemental ratios of the source object.*
- void [TransferHotFrom](#) ([GasReservoir](#) &givingGas, double massToMove)
- void [TransferAndHeat](#) ([GasReservoir](#) &givingGas, double massToMove)
- [GasStream](#) [AccretionStream](#) (double amountToLose)  
*Extracts the chosen amount of cold gas from the reservoir, and puts it into an accretion stream.*
- double [ColdGasMetallicity](#) () const
- const std::vector< [GasStream](#) > & [Composition](#) () const

## Static Public Member Functions

- static [GasReservoir Primordial](#) (double mass, const [GlobalParameters](#) &param)  
*Generates a primordial gas reservoir of the specified mass – only the ::Primordial component is populated, with the nature of that component determined by several key parameters in [GlobalParameters](#).*

## Private Attributes

- std::vector< [GasStream](#) > [Components](#)  
*A representation of the total amount of gas within the reservoir, separated by the origin of the gas.*
- const [GlobalParameters](#) & [Param](#)

### 4.6.1 Detailed Description

A [GasReservoir](#) is a heterogenously sourced pool of gas, such as those found within each ring, or representing the CGM. In practicality, they are a container for a vector of [GasStream](#) objects + assorted ways for these objects to interact with one another

### 4.6.2 Constructor & Destructor Documentation

#### 4.6.2.1 GasReservoir() [1/2]

```
GasReservoir::GasReservoir ( )
```

Default constructor. Initialises the [Components](#) and gives them their [SourceProcess](#) ID.

#### 4.6.2.2 GasReservoir() [2/2]

```
GasReservoir::GasReservoir (
    const GlobalParameters & param )
```

useful constructor

### 4.6.3 Member Function Documentation

#### 4.6.3.1 Absorb() [1/5]

```
void GasReservoir::Absorb (
    const GasReservoir & givingGas )
```

Transfer the contents of the input reservoir and sum them into the reservoir.

## Parameters

<i>givingGas</i>	the reservoir which will be summed into the current object (unaltered)
------------------	--

**4.6.3.2 Absorb()** [2/5]

```
void GasReservoir::Absorb (
    const GasStream & givingGas )
```

Transfer the contents of the input stream into the element of [Components](#) indicated by the input's [GasStream::Source](#) flag.

## Parameters

<i>givingGas</i>	the stream which is absorbed into the reservoir (unaltered)
------------------	---

**4.6.3.3 Absorb()** [3/5]

```
void GasReservoir::Absorb (
    const GasStream & givingGas,
    double fraction )
```

**4.6.3.4 Absorb()** [4/5]

```
void GasReservoir::Absorb (
    const std::vector< GasStream > & givingGas )
```

**4.6.3.5 Absorb()** [5/5]

```
void GasReservoir::Absorb (
    const std::vector< GasStream > & givingGas,
    double fraction )
```

**4.6.3.6 AbsorbMemory()**

```
void GasReservoir::AbsorbMemory (
    int t,
    const GasStream & input )
```

#### 4.6.3.7 AccretionStream()

```
GasStream GasReservoir::AccretionStream (
    double amountToLose )
```

Extracts the chosen amount of cold gas from the reservoir, and puts it into an accretion stream.

#### 4.6.3.8 ColdGasMetallicity()

```
double GasReservoir::ColdGasMetallicity ( ) const
```

#### 4.6.3.9 ColdMass()

```
double GasReservoir::ColdMass ( )
```

##### Returns

The current total cold-gas mass of the reservoir, the sum of [GasStream::ColdMass\(\)](#) calls over the [Components](#) vector.

#### 4.6.3.10 Composition()

```
const std::vector< GasStream > & GasReservoir::Composition ( ) const
```

#### 4.6.3.11 Deplete() [1/2]

```
void GasReservoir::Deplete (
    double amountToLose )
```

Calls [GasStream::Deplete\(double\)](#) on each element of [Components](#), keeping the relative mass contribution of each component equal.

##### Parameters

<i>amountToLose</i>	The total amount of mass to be lost from the reservoir (shared amongst components)
---------------------	--

#### 4.6.3.12 Deplete() [2/2]

```
void GasReservoir::Deplete (
    double amountToLose_Cold,
    double amountToLose_Hot )
```

Calls [GasStream::Deplete\(double, double\)](#) on each element of [Components](#), keeping the relative hot mass and cold mass contribution of each component equal.

##### Parameters

<i>amountToLose_Cold</i>	The total amount of cold gas mass to be lost from the reservoir (shared amongst components)
<i>amountToLose_Hot</i>	The total amount of hot gas mass to be lost from the reservoir (shared amongst components)

#### 4.6.3.13 Heat()

```
void GasReservoir::Heat (
    double amoutToHeat )
```

Heats up the specified amount of gas into the hot reservoir, keeping the elemental abundances of the cold gas reservoir constant.

#### 4.6.3.14 HotMass()

```
double GasReservoir::HotMass ( )
```

##### Returns

The current total hot-gas mass of the reservoir, the sum of [GasStream::HotMass\(\)](#) calls over the [Components](#) vector.

#### 4.6.3.15 Mass()

```
double GasReservoir::Mass ( )
```

##### Returns

The current total mass of the reservoir, the sum of [GasStream::Mass\(\)](#) calls over the [Components](#) vector.

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

```
GasStream & GasReservoir::operator[] (
    SourceProcess source )
```

A vector-like access overload, allowing indexing into the [Components](#) vector using appropriate #SourceProcesses.

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

```
const GasStream & GasReservoir::operator[] (
    SourceProcess source ) const
```

An annoyingly necessary redeclaration for when constant references don't want to play ball.

**4.6.3.18 PassiveCool()**

```
void GasReservoir::PassiveCool (
    double dt,
    bool isCGM )
```

Executes the usual cooling mechanism.

**4.6.3.19 Primordial()**

```
static GasReservoir GasReservoir::Primordial (
    double mass,
    const GlobalParameters & param ) [static]
```

Generates a primordial gas reservoir of the specified mass – only the ::Primordial component is populated, with the nature of that component determined by several key parameters in [GlobalParameters](#).

**Parameters**

<i>mass</i>	The total mass of the new reservoir
<i>param</i>	A reference to the global parameter set - required for primordial abundances and hot-gas fractions

**4.6.3.20 TransferAndHeat()**

```
void GasReservoir::TransferAndHeat (
    GasReservoir & givingGas,
    double massToMove )
```

#### 4.6.3.21 TransferColdFrom()

```
void GasReservoir::TransferColdFrom (
    GasReservoir & givingGas,
    double massToMove )
```

Transfers the specified amount of mass across from the target, removing the mass from the target and adding it to the current object. Maintains the source and elemental ratios of the source object.

#### 4.6.3.22 TransferFrom()

```
void GasReservoir::TransferFrom (
    GasReservoir & givingGas,
    double massToMove )
```

Transfers the specified amount of mass across from the target, removing the mass from the target and adding it to the current object. Maintains the thermal, source and elemental ratios of the source object.

#### 4.6.3.23 TransferHotFrom()

```
void GasReservoir::TransferHotFrom (
    GasReservoir & givingGas,
    double massToMove )
```

#### 4.6.3.24 Wipe()

```
void GasReservoir::Wipe ( )
```

Wipes all mass from the reservoir.

### 4.6.4 Field Documentation

#### 4.6.4.1 Components

```
std::vector<GasStream> GasReservoir::Components [private]
```

A representation of the total amount of gas within the reservoir, separated by the origin of the gas.



## 4.6.4.2 Param

```
const GlobalParameters& GasReservoir::Param [private]
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Gas/GasReservoir.h

## 4.7 GasStream Class Reference

```
#include <GasStream.h>
```

## Public Member Functions

- const [Gas](#) & [Hot](#) () const
- const [Gas](#) & [Cold](#) () const
- double & [Hot](#) ([ElementID](#) el)
- double & [Cold](#) ([ElementID](#) el)
- const double & [Hot](#) ([ElementID](#) el) const
- const double & [Cold](#) ([ElementID](#) el) const
- [GasStream](#) ()  
*Default constructor – assigns itself as [Unknown](#), with zero mass.*
- [GasStream](#) ([SourceProcess](#) source)  
*Gives itself zero mass in both components.*
- [GasStream](#) ([SourceProcess](#) source, const [Gas](#) &hot, const [Gas](#) &cold)  
*Initialises itself to the correct hot/cold components.*
- [GasStream](#) ([SourceProcess](#) source, const [Gas](#) &gas, double hotFraction)  
*Splits a single [Gas](#) object up into fractions determined by the hot fraction.*
- double [Mass](#) ()  
*This function attempts to be clever. Checks the [#NedsRecomputing](#) flag, and if necessary, calls [ComputeMasses\(\)](#), then sets the flag accordingly. Should reduce the number of loops needed.*
- double [Mass](#) () const
- double [HotMass](#) ()  
*See [Mass\(\)](#) for details.*
- double [ColdMass](#) ()  
*See [Mass\(\)](#) for details.*
- double [HotMass](#) () const  
*As with [HotMass\(\)](#), but returns only the last computed value...assumes proper normalisation before casting to const!*
- double [ColdMass](#) () const  
*As with [ColdMass\(\)](#), but returns only the last computed value...assumes proper normalisation before casting to const!*
- void [Deplete](#) (double amountToRemove)  
*Removes (i.e. throws away) the chosen amount of mass, keeping the hot/cold ratio and the elemental abundances the same.*
- void [Heat](#) (double amountToHeat)  
*Moves the specified mass from the cold reservoir into the hot reservoir, keeping the cold elemental abundances constant.*
- void [Cool](#) (double amountToCool)  
*Moves the specified mass from the hot reservoir to the cold reservoir, keeping the hot elemental abundances constant.*
- void [Deplete](#) (double amountToRemove\_Cold, double amountToRemove\_Hot)

*Removes (i.e. throws away) the mass from the hot and cold components, keeping the individual elemental abundances the same.*

- void [Absorb](#) (const [GasStream](#) &input)  
*Adds the gas contained within the input to the current stream.*
- void [Absorb](#) (const [GasStream](#) &input, double fraction)  
*Adds the specified fraction of gas contained within the input to the current stream.*
- void [Absorb](#) (const [Gas](#) &input, double hotFraction)  
*Adds the gas to the current stream, splitting it according to the hotFraction.*
- void [Dirty](#) ()  
*Sets the [NeedsRecomputing](#) flag to true. Used to inform the Stream that someone (most probably a call through [GasReservoir::operator\[\]](#)) has gone over its head and touched its internal workings.*

## Data Fields

- [SourceProcess Source](#)  
*A label identifying where and how this gas stream was created.*

## Private Member Functions

- void [ComputeMasses](#) ()  
*If [NeedsRecomputing](#) is true, this recalculates [internal\\_HotMass](#) and [internal\\_ColdMass](#).*

## Private Attributes

- bool [NeedsRecomputing](#)  
*A flag used to prevent excessive recomputing of the internal masses. By design, if [Absorb\(\)](#), [Deplete\(\)](#) etc. calls are avoided, the components of [Hot](#) and [Cold](#) are invariant, so the values of [Mass\(\)](#) are constant. Therefore if this flag is false, simply returns the last computed value of the mass value.*
- double [internal\\_HotMass](#)  
*The last computed value of [Hot.Gas::Mass\(\)](#)*
- double [internal\\_ColdMass](#)  
*The last computed value of [Cold.Gas::Mass\(\)](#)*
- double [internal\\_TotalMass](#)  
*The sum of [internal\\_HotMass](#) and [internal\\_ColdMass](#).*
- [Gas internal\\_Hot](#)  
*The container for the hot component.*
- [Gas internal\\_Cold](#)  
*The container for the cold component.*

### 4.7.1 Detailed Description

A gas stream is how a homogeneously-sourced set of gas gets moved around . They are created through [Source](#)↔ Events - such as CCSN or accretion events.

### 4.7.2 Constructor & Destructor Documentation

#### 4.7.2.1 GasStream() [1/4]

```
GasStream::GasStream ( )
```

Default constructor – assigns itself as [Unknown](#), with zero mass.

#### 4.7.2.2 GasStream() [2/4]

```
GasStream::GasStream (
    SourceProcess source )
```

Gives itself zero mass in both components.

##### Parameters

<i>source</i>	The value of <a href="#">Source</a> which this object inherits
---------------	--

#### 4.7.2.3 GasStream() [3/4]

```
GasStream::GasStream (
    SourceProcess source,
    const Gas & hot,
    const Gas & cold )
```

Initialises itself to the correct hot/cold components.

##### Parameters

<i>source</i>	The value of <a href="#">Source</a> which this object inherits
<i>hot</i>	The gas component which is copied into <a href="#">Hot</a>
<i>cold</i>	The gas component which is copied into <a href="#">Cold</a>

#### 4.7.2.4 GasStream() [4/4]

```
GasStream::GasStream (
    SourceProcess source,
    const Gas & gas,
    double hotFraction )
```

Splits a single [Gas](#) object up into fractions determined by the hot fraction.

## Parameters

<i>source</i>	The value of <a href="#">Source</a> which this object inherits
<i>gas</i>	The total gas mass of the new stream
<i>hotFraction</i>	the fraction of the gas which is put into <a href="#">Hot</a> , keeping the elemental abundances the same

### 4.7.3 Member Function Documentation

#### 4.7.3.1 Absorb() [1/3]

```
void GasStream::Absorb (
    const Gas & input,
    double hotFraction )
```

Adds the gas to the current stream, splitting it according to the hotFraction.

## Parameters

<i>input</i>	The object which donates the gas (unaltered)
<i>hotFraction</i>	The amount of the input which goes into the <a href="#">Hot</a> stream

#### 4.7.3.2 Absorb() [2/3]

```
void GasStream::Absorb (
    const GasStream & input )
```

Adds the gas contained within the input to the current stream.

## Parameters

<i>input</i>	The object which donates the gas (unaltered)
--------------	--

#### 4.7.3.3 Absorb() [3/3]

```
void GasStream::Absorb (
    const GasStream & input,
    double fraction )
```

Adds the specified fractiongas contained within the input to the current stream.

## Parameters

<i>input</i>	The object which donates the gas (unaltered)
<i>fraction</i>	the fraction of the input object which is absorbed

**4.7.3.4 Cold()** [1/3]

```
const Gas & GasStream::Cold ( ) const
```

**4.7.3.5 Cold()** [2/3]

```
double & GasStream::Cold (
    ElementID el )
```

**4.7.3.6 Cold()** [3/3]

```
const double & GasStream::Cold (
    ElementID el ) const
```

**4.7.3.7 ColdMass()** [1/2]

```
double GasStream::ColdMass ( )
```

See [Mass\(\)](#) for details.

## Returns

The current mass of the [Cold](#) component

**4.7.3.8 ColdMass()** [2/2]

```
double GasStream::ColdMass ( ) const
```

As with [ColdMass\(\)](#), but returns only the last computed value...assumes proper normalisation before casting to const!

#### 4.7.3.9 ComputeMasses()

```
void GasStream::ComputeMasses ( ) [private]
```

If [NeedsRecomputing](#) is true, this recalculates [internal\\_HotMass](#) and [internal\\_ColdMass](#).

#### 4.7.3.10 Cool()

```
void GasStream::Cool (
    double amountToCool )
```

Moves the specified mass from the hot reservoir to the cold reservoir, keeping the hot elemental abundances constant.

#### 4.7.3.11 Deplete() [1/2]

```
void GasStream::Deplete (
    double amountToRemove )
```

Removes (i.e. throws away) the chosen amount of mass, keeping the hot/cold ratio and the elemental abundances the same.

##### Parameters

<i>amountToRemove</i>	The amount of mass to lose from the stream
-----------------------	--

#### 4.7.3.12 Deplete() [2/2]

```
void GasStream::Deplete (
    double amountToRemove_Cold,
    double amountToRemove_Hot )
```

Removes (i.e. throws away) the mass from the hot and cold components, keeping the individual elemental abundances the same.

##### Parameters

<i>amountToRemove_Cold,the</i>	amount of cold gas to lose
<i>amountToRemove_Hot,the</i>	amount of hot gas to lose

#### 4.7.3.13 Dirty()

```
void GasStream::Dirty ( )
```

Sets the [NeedsRecomputing](#) flag to true. Used to inform the Stream that someone (most probably a call through [GasReservoir::operator\[\]](#)) has gone over its head and touched its internal workings.

#### 4.7.3.14 Heat()

```
void GasStream::Heat (
    double amountToHeat )
```

Moves the specified mass from the cold reservoir into the hot reservoir, keeping the cold elemental abundances constant.

#### 4.7.3.15 Hot() [1/3]

```
const Gas & GasStream::Hot ( ) const
```

#### 4.7.3.16 Hot() [2/3]

```
double & GasStream::Hot (
    ElementID el )
```

#### 4.7.3.17 Hot() [3/3]

```
const double & GasStream::Hot (
    ElementID el ) const
```

#### 4.7.3.18 HotMass() [1/2]

```
double GasStream::HotMass ( )
```

See [Mass\(\)](#) for details.

##### Returns

The current mass of the [Hot](#) component

#### 4.7.3.19 HotMass() [2/2]

```
double GasStream::HotMass ( ) const
```

As with [HotMass\(\)](#), but returns only the last computed value...assumes proper normalisation before casting to const!

#### 4.7.3.20 Mass() [1/2]

```
double GasStream::Mass ( )
```

This function attempts to be clever. Checks the #NedsRecomputing flag, and if necessary, calls [ComputeMasses\(\)](#), then sets the flag accordingly. Should reduce the number of loops needed.

##### Returns

The current total mass of the stream

#### 4.7.3.21 Mass() [2/2]

```
double GasStream::Mass ( ) const
```

### 4.7.4 Field Documentation

#### 4.7.4.1 internal\_Cold

```
Gas GasStream::internal_Cold [private]
```

The container for the cold component.

#### 4.7.4.2 internal\_ColdMass

```
double GasStream::internal_ColdMass [private]
```

The last computed value of [Cold.Gas::Mass\(\)](#)



#### 4.7.4.3 internal\_Hot

```
Gas GasStream::internal_Hot [private]
```

The container for the hot component.

#### 4.7.4.4 internal\_HotMass

```
double GasStream::internal_HotMass [private]
```

The last computed value of [Hot.Gas::Mass\(\)](#)

#### 4.7.4.5 internal\_TotalMass

```
double GasStream::internal_TotalMass [private]
```

The sum of [internal\\_HotMass](#) and [internal\\_ColdMass](#).

#### 4.7.4.6 NeedsRecomputing

```
bool GasStream::NeedsRecomputing [private]
```

A flag used to prevent excessive recomputing of the internal masses. By design, if [Absorb\(\)](#), [Deplete\(\)](#) etc. calls are avoided, the components of [Hot](#) and [Cold](#) are invariant, so the values of [Mass\(\)](#) are constant. Therefore if this flag is false, simply returns the last computed value of the mass value.

#### 4.7.4.7 Source

```
SourceProcess GasStream::Source
```

A label identifying where and how this gas stream was created.

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

- [/Users/jf20/Documents/Physics/RAMICES\\_II/src/Gas/GasStream.h](#)

## 4.8 GlobalParameters Class Reference

A package of global parameter objects which can be passed around by reference. Most of the internal values are set as `JSL::Argument` objects and so can be initialised from the command line / config file.

```
#include <GlobalParameters.h>
```

## Public Member Functions

- [GlobalParameters](#) ()  
*Does absolutely nothing!*
- void [Initialise](#) (int argc, char \*argv[])  
*Loops over the ParamMembers and initialises their values according to the [ParamList](#) object.*
- void [SaveInputs](#) ()  
*Writes the inputs to file as a mock-config file so that this exact simulation can be rerun.*

## Data Fields

- [MetaValues](#) [Meta](#)  
*Simulation values - timescales, number of threads etc.*
- [OutputValues](#) [Output](#)  
*Output directory values – directory information etc.*
- [ResourceValues](#) [Resources](#)  
*Resource directory values.*
- [ElementValues](#) [Element](#)  
*Abundance data + resource location data.*
- [StellarValues](#) [Stellar](#)  
*Stellar limits (mass, metallicity)*
- [YieldValues](#) [Yield](#)  
*Yield stuff.*
- [ThermalValues](#) [Thermal](#)  
*Hot gas cooling/injection parameters.*
- [MigrationValues](#) [Migration](#)  
*Migration stuff.*
- [CatalogueValues](#) [Catalogue](#)  
*Catalogue Synthesis stuff.*
- [GalaxyValues](#) [Galaxy](#)  
*Galactic size/ evolution parameters.*
- std::vector< [ParamList](#) \* > [ParamMembers](#) = {&[Meta](#),&[Output](#),&[Resources](#),&[Element](#),&[Stellar](#),&[Thermal](#),&[Galaxy](#),&[Yield](#),&[Migration](#)}  
*A heterogeneous pointer array, which allows for a nice loop over the members. Any new parameter pack needs to be inserted here so that the member values can be initialised.*

### 4.8.1 Detailed Description

A package of global parameter objects which can be passed around by reference. Most of the internal values are set as JSL::Argument objects and so can be initialised from the command line / config file.

### 4.8.2 Constructor & Destructor Documentation

#### 4.8.2.1 GlobalParameters()

```
GlobalParameters::GlobalParameters ( )
```

Does absolutely nothing!

### 4.8.3 Member Function Documentation

#### 4.8.3.1 Initialise()

```
void GlobalParameters::Initialise (
    int argc,
    char * argv[] )
```

Loops over the ParamMembers and initialises their values according to the [ParamList](#) object.

#### 4.8.3.2 SaveInputs()

```
void GlobalParameters::SaveInputs ( )
```

Writes the inputs to file as a mock-config file so that this exact simulation can be rerun.

### 4.8.4 Field Documentation

#### 4.8.4.1 Catalogue

[CatalogueValues](#) GlobalParameters::Catalogue

Catalogue Synthesis stuff.

#### 4.8.4.2 Element

[ElementValues](#) GlobalParameters::Element

Abundance data + resource location data.

#### 4.8.4.3 Galaxy

[GalaxyValues](#) GlobalParameters::Galaxy

Galactic size/ evolution parameters.

#### 4.8.4.4 Meta

`MetaValues` `GlobalParameters::Meta`

Simulation values - timescales, number of threads etc.

#### 4.8.4.5 Migration

`MigrationValues` `GlobalParameters::Migration`

Migration stuff.

#### 4.8.4.6 Output

`OutputValues` `GlobalParameters::Output`

Output directory values – directory information etc.

#### 4.8.4.7 ParamMembers

```
std::vector<ParamList *> GlobalParameters::ParamMembers = {&Meta, &Output, &Resources, &Element, &Stellar, &Therma
```

A heterogeneous pointer array, which allows for a nice loop over the members. Any new parameter pack needs to be inserted here so that the member values can be initialised.

#### 4.8.4.8 Resources

`ResourceValues` `GlobalParameters::Resources`

Resource directory values.

#### 4.8.4.9 Stellar

`StellarValues` `GlobalParameters::Stellar`

Stellar limits (mass, metallicity)

#### 4.8.4.10 Thermal

`ThermalValues` `GlobalParameters::Thermal`

Hot gas cooling/injection parameters.

#### 4.8.4.11 Yield

`YieldValues` `GlobalParameters::Yield`

Yield stuff.

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

- `/Users/jf20/Documents/Physics/RAMICES_II/src/Parameters/GlobalParameters.h`

## 4.9 IMF\_Functor Class Reference

```
#include <IMF.h>
```

### Public Member Functions

- `IMF_Functor` (const `GlobalParameters` &param)
- double `operator()` (double mass)
- double `FormationCount` (double formationMass) const
- double `Weighting` (int i) const

### Private Member Functions

- `IntegralMomentCompute` (double start, double stop, int resolution)
- void `Normalise` ()
- double `IMF` (double mass)

### Private Attributes

- const `GlobalParameters` & `Param`
- double `IMF_Normalisation`
- double `IMF_MeanMass`
- `std::vector< double >` `IMF_Weighting`

### 4.9.1 Constructor & Destructor Documentation

#### 4.9.1.1 IMF\_Functor()

```
IMF_Functor::IMF_Functor (
    const GlobalParameters & param )
```

### 4.9.2 Member Function Documentation

#### 4.9.2.1 FormationCount()

```
double IMF_Functor::FormationCount (
    double formationMass ) const
```

#### 4.9.2.2 IMF()

```
double IMF_Functor::IMF (
    double mass ) [private]
```

#### 4.9.2.3 MomentCompute()

```
Integral IMF_Functor::MomentCompute (
    double start,
    double stop,
    int resolution ) [private]
```

#### 4.9.2.4 Normalise()

```
void IMF_Functor::Normalise ( ) [private]
```

#### 4.9.2.5 operator()()

```
double IMF_Functor::operator() (
    double mass )
```

#### 4.9.2.6 Weighting()

```
double IMF_Functor::Weighting (
    int i ) const
```

### 4.9.3 Field Documentation

#### 4.9.3.1 IMF\_MeanMass

```
double IMF_Functor::IMF_MeanMass [private]
```

#### 4.9.3.2 IMF\_Normalisation

```
double IMF_Functor::IMF_Normalisation [private]
```

#### 4.9.3.3 IMF\_Weighting

```
std::vector<double> IMF_Functor::IMF_Weighting [private]
```

#### 4.9.3.4 Param

```
const GlobalParameters& IMF_Functor::Param [private]
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/[IMF.h](#)

## 4.10 InitialisedData Class Reference

These will act like globally-defined functions, but have the scope for modifying themselves as they go along.

```
#include <InitialisedData.h>
```

## Public Member Functions

- [InitialisedData](#) (const [GlobalParameters](#) &param)
- void [Log](#) (const std::string &input) const
- void [Log](#) (const std::string &input, int importance) const
- void [LogFlush](#) () const
- void [UrgentLog](#) (const std::string &input) const
- void [ProgressBar](#) (int &currentBars, int currentStep, int totalSteps)
- double [NormalDist](#) ()
- double [NormalDist](#) (double mu, double sigma)
- double [UniformDist](#) (double lowerBound, double upperBound)

## Data Fields

- const [IMF\\_Functor](#) IMF
- [SLF\\_Functor](#) SLF
- const [GlobalParameters](#) & Param
- const [YieldGrid](#) CCSNYield
- const [YieldGrid](#) AGBYield
- const [YieldGrid](#) ECSNYield
- const [SimpleYield](#) SNIaYield
- const [SimpleYield](#) NSMYield
- [IsochroneTracker](#) Isochrones

## Private Attributes

- std::default\_random\_engine [generator](#)
- std::normal\_distribution< double > [distribution](#)

### 4.10.1 Detailed Description

These will act like globally-defined functions, but have the scope for modifying themselves as they go along.

### 4.10.2 Constructor & Destructor Documentation

#### 4.10.2.1 InitialisedData()

```
InitialisedData::InitialisedData (
    const GlobalParameters & param )
```

### 4.10.3 Member Function Documentation



#### 4.10.3.1 Log() [1/2]

```
void InitialisedData::Log (
    const std::string & input ) const
```

#### 4.10.3.2 Log() [2/2]

```
void InitialisedData::Log (
    const std::string & input,
    int importance ) const
```

#### 4.10.3.3 LogFlush()

```
void InitialisedData::LogFlush ( ) const
```

#### 4.10.3.4 NormalDist() [1/2]

```
double InitialisedData::NormalDist ( )
```

#### 4.10.3.5 NormalDist() [2/2]

```
double InitialisedData::NormalDist (
    double mu,
    double sigma )
```

#### 4.10.3.6 ProgressBar()

```
void InitialisedData::ProgressBar (
    int & currentBars,
    int currentStep,
    int totalSteps )
```

#### 4.10.3.7 UniformDist()

```
double InitialisedData::UniformDist (
    double lowerBound,
    double upperBound )
```

#### 4.10.3.8 UrgentLog()

```
void InitialisedData::UrgentLog (
    const std::string & input ) const
```

### 4.10.4 Field Documentation

#### 4.10.4.1 AGBYield

```
const YieldGrid InitialisedData::AGBYield
```

#### 4.10.4.2 CCSNYield

```
const YieldGrid InitialisedData::CCSNYield
```

#### 4.10.4.3 distribution

```
std::normal_distribution<double> InitialisedData::distribution [private]
```

#### 4.10.4.4 ECSNYield

```
const YieldGrid InitialisedData::ECSNYield
```

#### 4.10.4.5 generator

```
std::default_random_engine InitialisedData::generator [private]
```

#### 4.10.4.6 IMF

```
const IMF_Functor InitialisedData::IMF
```

#### 4.10.4.7 Isochrones

```
IsochroneTracker InitialisedData::Isochrones
```

#### 4.10.4.8 NSMYield

```
const SimpleYield InitialisedData::NSMYield
```

#### 4.10.4.9 Param

```
const GlobalParameters& InitialisedData::Param
```

#### 4.10.4.10 SLF

```
SLF_Functor InitialisedData::SLF
```

#### 4.10.4.11 SNIaYield

```
const SimpleYield InitialisedData::SNIaYield
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/[InitialisedData.h](#)

## 4.11 Integral Struct Reference

```
#include <IMF.h>
```

## Data Fields

- double [ZerothMoment](#)
- double [FirstMoment](#)

### 4.11.1 Field Documentation

#### 4.11.1.1 FirstMoment

```
double Integral::FirstMoment
```

#### 4.11.1.2 ZerothMoment

```
double Integral::ZerothMoment
```

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

- [/Users/jf20/Documents/Physics/RAMICES\\_II/src/Stars/IMF.h](#)

## 4.12 Interpolator Struct Reference

```
#include <YieldGrid.h>
```

### Public Member Functions

- double [Interpolate](#) (double lower, double upper)

### Data Fields

- int [UpperID](#)
- int [LowerID](#)
- double [LinearFactor](#)

### 4.12.1 Member Function Documentation

#### 4.12.1.1 Interpolate()

```
double Interpolator::Interpolate (  
    double lower,  
    double upper ) [inline]
```

### 4.12.2 Field Documentation

#### 4.12.2.1 LinearFactor

```
double Interpolator::LinearFactor
```

#### 4.12.2.2 LowerID

```
int Interpolator::LowerID
```

#### 4.12.2.3 UpperID

```
int Interpolator::UpperID
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Yields/[YieldGrid.h](#)

## 4.13 IsochroneCube Struct Reference

```
#include <IsochroneTracker.h>
```

### Public Member Functions

- int [Count](#) () const
- double [Value](#) (int entry, [IsochroneProperties](#) p) const

### Data Fields

- std::vector< double > [Weighting](#)
- std::vector< [IsochroneEntry](#) \* > [Data](#)

### 4.13.1 Member Function Documentation

#### 4.13.1.1 Count()

```
int IsochroneCube::Count ( ) const [inline]
```

#### 4.13.1.2 Value()

```
double IsochroneCube::Value (
    int entry,
    IsochroneProperties p ) const [inline]
```

### 4.13.2 Field Documentation

#### 4.13.2.1 Data

```
std::vector<IsochroneEntry *> IsochroneCube::Data
```

#### 4.13.2.2 Weighting

```
std::vector<double> IsochroneCube::Weighting
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/[IsochroneTracker.h](#)

## 4.14 IsochroneEntry Struct Reference

```
#include <IsochroneTracker.h>
```

### Public Member Functions

- [IsochroneEntry](#) ()
- double & [operator\[\]](#) ([IsochroneProperties](#) p)
- const double & [operator\[\]](#) ([IsochroneProperties](#) p) const
- int [Countify](#) ()

## Data Fields

- `std::vector< double >` [Properties](#)

### 4.14.1 Constructor & Destructor Documentation

#### 4.14.1.1 IsochroneEntry()

```
IsochroneEntry::IsochroneEntry ( ) [inline]
```

### 4.14.2 Member Function Documentation

#### 4.14.2.1 Countify()

```
int IsochroneEntry::Countify ( ) [inline]
```

#### 4.14.2.2 operator[]() [1/2]

```
double & IsochroneEntry::operator[] (   
    IsochroneProperties p ) [inline]
```

#### 4.14.2.3 operator[]() [2/2]

```
const double & IsochroneEntry::operator[] (   
    IsochroneProperties p ) const [inline]
```

### 4.14.3 Field Documentation

#### 4.14.3.1 Properties

```
std::vector<double> IsochroneEntry::Properties
```

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

- `/Users/jf20/Documents/Physics/RAMICES_II/src/Stars/IsochroneTracker.h`

## 4.15 IsochroneTracker Class Reference

```
#include <IsochroneTracker.h>
```

### Public Member Functions

- [IsochroneTracker](#) (const [GlobalParameters](#) &param)
- void [Construct](#) ()
- [IsochroneCube](#) [GetProperties](#) (int mass, double z, double age)

### Private Member Functions

- void [IsoLog](#) (std::string val)
- void [ParseFile](#) (std::string file)
- double [NormalSample](#) (double mu, double sigma)
- double [UniformSample](#) (double lowerBound, double upperBound)
- void [ExtractSample](#) ([IsochroneCube](#) &output, int sampleMass, double sampleZ, double sampleAge)

### Private Attributes

- const [GlobalParameters](#) [Param](#)
- std::vector< double > [CapturedZs](#)
- std::vector< double > [CapturedTs](#)
- std::vector< std::vector< std::vector< [IsochroneEntry](#) > > > [Grid](#)
- std::vector< std::vector< std::vector< [IsochroneEntry](#) > > > [UnsortedGrid](#)
- bool [isTimeLogUniform](#)
- double [DeltaLogT](#)
- std::default\_random\_engine [generator](#)
- std::normal\_distribution< double > [distribution](#)

### 4.15.1 Constructor & Destructor Documentation

#### 4.15.1.1 IsochroneTracker()

```
IsochroneTracker::IsochroneTracker (
    const GlobalParameters & param )
```

### 4.15.2 Member Function Documentation



#### 4.15.2.1 Construct()

```
void IsochroneTracker::Construct ( )
```

#### 4.15.2.2 ExtractSample()

```
void IsochroneTracker::ExtractSample (
    IsochroneCube & output,
    int sampleMass,
    double sampleZ,
    double sampleAge ) [private]
```

#### 4.15.2.3 GetProperties()

```
IsochroneCube IsochroneTracker::GetProperties (
    int mass,
    double z,
    double age )
```

#### 4.15.2.4 IsoLog()

```
void IsochroneTracker::IsoLog (
    std::string val ) [private]
```

#### 4.15.2.5 NormalSample()

```
double IsochroneTracker::NormalSample (
    double mu,
    double sigma ) [private]
```

#### 4.15.2.6 ParseFile()

```
void IsochroneTracker::ParseFile (
    std::string file ) [private]
```

#### 4.15.2.7 UniformSample()

```
double IsochroneTracker::UniformSample (
    double lowerBound,
    double upperBound ) [private]
```

### 4.15.3 Field Documentation

#### 4.15.3.1 CapturedTs

```
std::vector<double> IsochroneTracker::CapturedTs [private]
```

#### 4.15.3.2 CapturedZs

```
std::vector<double> IsochroneTracker::CapturedZs [private]
```

#### 4.15.3.3 DeltaLogT

```
double IsochroneTracker::DeltaLogT [private]
```

#### 4.15.3.4 distribution

```
std::normal_distribution<double> IsochroneTracker::distribution [private]
```

#### 4.15.3.5 generator

```
std::default_random_engine IsochroneTracker::generator [private]
```

#### 4.15.3.6 Grid

```
std::vector<std::vector<std::vector<IsochroneEntry> > > IsochroneTracker::Grid [private]
```

#### 4.15.3.7 isTimeLogUniform

```
bool IsochroneTracker::isTimeLogUniform [private]
```

#### 4.15.3.8 Param

```
const GlobalParameters IsochroneTracker::Param [private]
```

#### 4.15.3.9 UnsortedGrid

```
std::vector<std::vector<std::vector<IsochroneEntry> > > IsochroneTracker::UnsortedGrid [private]
```

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

- [/Users/jf20/Documents/Physics/RAMICES\\_II/src/Stars/IsochroneTracker.h](#)

## 4.16 IsoMass Class Reference

A simple struct for tracking the number of stars of a given mass.

```
#include <StellarPopulation.h>
```

### Public Member Functions

- [IsoMass](#) ()
- [IsoMass](#) (double n, int m, double z, int birth, int death)

### Data Fields

- int [MassIndex](#)
- double [Count](#)
- double [Metallicity](#)
- int [BirthIndex](#)
- int [DeathIndex](#)
- [IsochroneCube](#) [Isochrone](#)

#### 4.16.1 Detailed Description

A simple struct for tracking the number of stars of a given mass.

## 4.16.2 Constructor & Destructor Documentation

### 4.16.2.1 IsoMass() [1/2]

```
IsoMass::IsoMass ( )
```

### 4.16.2.2 IsoMass() [2/2]

```
IsoMass::IsoMass (
    double n,
    int m,
    double z,
    int birth,
    int death )
```

## 4.16.3 Field Documentation

### 4.16.3.1 BirthIndex

```
int IsoMass::BirthIndex
```

### 4.16.3.2 Count

```
double IsoMass::Count
```

### 4.16.3.3 DeathIndex

```
int IsoMass::DeathIndex
```

### 4.16.3.4 Isochrone

```
IsochroneCube IsoMass::Isochrone
```

#### 4.16.3.5 MassIndex

```
int IsoMass::MassIndex
```

#### 4.16.3.6 Metallicity

```
double IsoMass::Metallicity
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/[StellarPopulation.h](#)

## 4.17 MassReport Struct Reference

```
#include <RemnantPopulation.h>
```

### Data Fields

- double [Total](#)
- double [WD](#)
- double [NS](#)
- double [BH](#)

### 4.17.1 Field Documentation

#### 4.17.1.1 BH

```
double MassReport::BH
```

#### 4.17.1.2 NS

```
double MassReport::NS
```

#### 4.17.1.3 Total

```
double MassReport::Total
```

#### 4.17.1.4 WD

```
double MassReport::WD
```

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

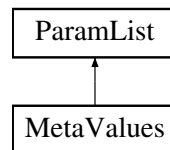
- /Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/[RemnantPopulation.h](#)

## 4.18 MetaValues Class Reference

The [MetaValues](#) contains variables associated with the base-level information about the simulation - the number of cores to access, the timesteps etc.

```
#include <ParameterLists.h>
```

Inheritance diagram for MetaValues:



### Public Member Functions

- [MetaValues](#) ()  
*Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.*
- virtual void [Initialise](#) (std::string resourceRoot)  
*An overload of a normally empty function. Computes the value of [SimulationSteps](#).*

### Data Fields

- Argument< int > [Verbosity](#) = Argument<int>(1, "verbose")  
*Controls whether the funky ASCII welcome message is played at the beginning of the code.*
- Argument< int > [ParallelThreads](#) = Argument<int>(3, "thread")  
*The maximum number of parallel threads which can be active at any given time.*
- Argument< double > [TimeStep](#) = Argument<double>(0.01, "timestep")  
*The top level timestep used in the main chemical loop.*
- Argument< double > [SimulationDuration](#) = Argument<double>(10.0, "duration")
- Argument< int > [ProgressHashes](#) = Argument<int>(32, "progress-hashes")  
*The number of hashes used to display progress bars.*
- int [SimulationSteps](#)  
*The number of timesteps in the simulation, computed from [SimulationDuration](#) and [TimeStep](#).*

## Additional Inherited Members

### 4.18.1 Detailed Description

The [MetaValues](#) contains variables associated with the base-level information about the simulation - the number of cores to access, the timesteps etc.

### 4.18.2 Constructor & Destructor Documentation

#### 4.18.2.1 MetaValues()

```
MetaValues::MetaValues ( ) [inline]
```

Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.

### 4.18.3 Member Function Documentation

#### 4.18.3.1 Initialise()

```
virtual void MetaValues::Initialise (
    std::string resourceRoot ) [virtual]
```

An overload of a normally empty function. Computes the value of [SimulationSteps](#).

Reimplemented from [ParamList](#).

### 4.18.4 Field Documentation

#### 4.18.4.1 ParallelThreads

```
Argument<int> MetaValues::ParallelThreads = Argument<int>(3, "thread")
```

The maximum number of parallel threads which can be active at any given time.

#### 4.18.4.2 ProgressHashes

```
Argument<int> MetaValues::ProgressHashes = Argument<int>(32, "progress-hashes")
```

The number of hashes used to display progress bars.

#### 4.18.4.3 SimulationDuration

```
Argument<double> MetaValues::SimulationDuration = Argument<double>(10.0, "duration")
```

#### 4.18.4.4 SimulationSteps

```
int MetaValues::SimulationSteps
```

The number of timesteps in the simulation, computed from [SimulationDuration](#) and [TimeStep](#).

#### 4.18.4.5 TimeStep

```
Argument<double> MetaValues::TimeStep = Argument<double>(0.01, "timestep")
```

The top level timestep used in the main chemical loop.

#### 4.18.4.6 Verbosity

```
Argument<int> MetaValues::Verbosity = Argument<int>(1, "verbose")
```

Controls whether the funky ASCII welcome message is played at the beginning of the code.

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

- [/Users/jf20/Documents/Physics/RAMICES\\_II/src/Parameters/ParameterLists.h](#)

## 4.19 MigrationMatrix Class Reference

```
#include <MigrationMatrix.h>
```



## Public Member Functions

- [MigrationMatrix](#) ([InitialisedData](#) &Data)
- void [Create](#) (const std::vector< double > &masses)
- void [Compound](#) (const [MigrationMatrix](#) &newTime)
- void [Print](#) ()

## Data Fields

- std::vector< std::vector< double > > [Grid](#)

## Private Member Functions

- std::vector< std::vector< double > > [DiagonalMultiply](#) (const std::vector< std::vector< double > > &a, const std::vector< std::vector< double > > &b, int diagonalDistance)

## Private Attributes

- int [NRings](#)
- const [GlobalParameters](#) & [Param](#)

### 4.19.1 Constructor & Destructor Documentation

#### 4.19.1.1 MigrationMatrix()

```
MigrationMatrix::MigrationMatrix (  
    InitialisedData & Data )
```

### 4.19.2 Member Function Documentation

#### 4.19.2.1 Compound()

```
void MigrationMatrix::Compound (  
    const MigrationMatrix & newTime )
```

#### 4.19.2.2 Create()

```
void MigrationMatrix::Create (  
    const std::vector< double > & masses )
```

#### 4.19.2.3 DiagonalMultiply()

```
std::vector< std::vector< double > > MigrationMatrix::DiagonalMultiply (
    const std::vector< std::vector< double > > & a,
    const std::vector< std::vector< double > > & b,
    int diagonalDistance ) [private]
```

#### 4.19.2.4 Print()

```
void MigrationMatrix::Print ( )
```

### 4.19.3 Field Documentation

#### 4.19.3.1 Grid

```
std::vector<std::vector<double> > MigrationMatrix::Grid
```

#### 4.19.3.2 NRings

```
int MigrationMatrix::NRings [private]
```

#### 4.19.3.3 Param

```
const GlobalParameters& MigrationMatrix::Param [private]
```

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

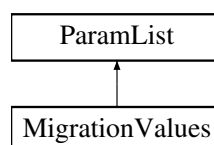
- [/Users/jf20/Documents/Physics/RAMICES\\_II/src/Galaxy/MigrationMatrix.h](#)

## 4.20 MigrationValues Class Reference

Holds values associated with how matter moves throughout the disc.

```
#include <ParameterLists.h>
```

Inheritance diagram for MigrationValues:



## Public Member Functions

- [MigrationValues](#) ()

Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.

## Data Fields

- Argument< bool > [InflowActive](#) = Argument<bool>(true,"inflow-on")  
*Turns on or off the gas inflow in the disc.*
- Argument< double > [InflowParameterA](#) = Argument<double>(0.33,"inflow-a")  
*A parameter to do with the inflow weighting scheme (icky)*
- Argument< double > [InflowParameterB](#) = Argument<double>(0.53,"inflow-b")  
*A parameter to do with the inflow weighting scheme (icky)*
- Argument< double > [MaxStealFraction](#) = Argument<double>(0.95,"max-steal")  
*The maximum amount of gas which can be moved moved between rings during the inflow portion.*
- Argument< double > [MarkovDispersionStrength](#) = Argument<double>(0.2,"mixing-strength")  
*The strength of the random-walk mixing process, in units of  $1e-3 \text{ kpc}^2/\text{Gyr}$ .*
- Argument< int > [DispersionOrder](#) = Argument<int>(3,"mixing-order")  
*The order to which the mixing matrix is computed - note that higher values allow instantaneous dispersion to higher radii.*
- Argument< double > [DispersionTruncation](#) = Argument<double>(1e-10,"mixing-truncation")

## Additional Inherited Members

### 4.20.1 Detailed Description

Holds values associated with how matter moves throughout the disc.

### 4.20.2 Constructor & Destructor Documentation

#### 4.20.2.1 MigrationValues()

```
MigrationValues::MigrationValues ( ) [inline]
```

Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.

### 4.20.3 Field Documentation

#### 4.20.3.1 DispersionOrder

```
Argument<int> MigrationValues::DispersionOrder = Argument<int>(3, "mixing-order")
```

The order to which the mixing matrix is computed - note that higher values allow instantaneous dispersion to higher radii.

#### 4.20.3.2 DispersionTruncation

```
Argument<double> MigrationValues::DispersionTruncation = Argument<double>(1e-10, "mixing-truncation")
```

#### 4.20.3.3 InflowActive

```
Argument<bool> MigrationValues::InflowActive = Argument<bool>(true, "inflow-on")
```

Turns on or off the gas inflow in the disc.

#### 4.20.3.4 InflowParameterA

```
Argument<double> MigrationValues::InflowParameterA = Argument<double>(0.33, "inflow-a")
```

A parameter to do with the inflow weighting scheme (icky)

#### 4.20.3.5 InflowParameterB

```
Argument<double> MigrationValues::InflowParameterB = Argument<double>(0.53, "inflow-b")
```

A parameter to do with the inflow weighting scheme (icky)

#### 4.20.3.6 MarkovDispersionStrength

```
Argument<double> MigrationValues::MarkovDispersionStrength = Argument<double>(0.2, "mixing-strength")
```

The strength of the random-walk mixing process, in units of  $1e-3 \text{ kpc}^2/\text{Gyr}$ .

### 4.20.3.7 MaxStealFraction

```
Argument<double> MigrationValues::MaxStealFraction = Argument<double>(0.95, "max-steal")
```

The maximum amount of gas which can be moved moved between rings during the inflow portion.

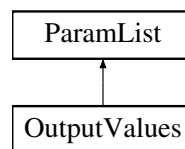
The documentation for this class was generated from the following file:

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/[ParameterLists.h](#)

## 4.21 OutputValues Class Reference

```
#include <ParameterLists.h>
```

Inheritance diagram for OutputValues:



### Public Member Functions

- [OutputValues](#) ()  
*Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.*
- virtual void [Initialise](#) (std::string resourceRoot)  
*An overload of a normally empty function. Goes through and creates the necessary directory structure.*

### Data Fields

- Argument< std::string > [Root](#) = Argument<std::string>("Output/", "output")  
*The name of the output directory into which the output will be saved.*
- Argument< std::string > [Config](#) = Argument<std::string>("rerun.config", "config-out")  
*The name for the output config file which would replicate this simulation.*
- Argument< std::string > [YieldSubdir](#) = Argument<std::string>("Yields/", "yield-dir")
- Argument< std::string > [GalaxyMassFile](#) = Argument<std::string>("Mass.dat", "galaxy-mass-file")  
*The name of the file containing galactic-scale mass information.*
- Argument< std::string > [EventRateFile](#) = Argument<std::string>("Events.dat", "event-rate-file")
- Argument< std::string > [StarFile](#) = Argument<std::string>("StellarCatalogue.dat", "ring-data-stars")  
*The ring-star data identifier.*
- Argument< std::string > [ChemicalPrefactor](#) = Argument<std::string>("Enrichment\_", "enrichment-base")  
*The enrichment file identifier.*
- Argument< std::string > [ColdGasDataFile](#) = Argument<std::string>("ColdGas.dat", "enrichment-cold")  
*The cold gas filename.*
- Argument< std::string > [HotGasDataFile](#) = Argument<std::string>("HotGas.dat", "enrichment-hot")  
*The hot gas filename.*
- std::string [LogarithmicColdGasFile](#)
- std::string [AbsoluteColdGasFile](#)
- std::string [LogarithmicHotGasFile](#)
- std::string [AbsoluteHotGasFile](#)

## Additional Inherited Members

### 4.21.1 Constructor & Destructor Documentation

#### 4.21.1.1 OutputValues()

```
OutputValues::OutputValues ( ) [inline]
```

Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.

### 4.21.2 Member Function Documentation

#### 4.21.2.1 Initialise()

```
virtual void OutputValues::Initialise (
    std::string resourceRoot ) [virtual]
```

An overload of a normally empty function. Goes through and creates the necessary directory structure.

Reimplemented from [ParamList](#).

### 4.21.3 Field Documentation

#### 4.21.3.1 AbsoluteColdGasFile

```
std::string OutputValues::AbsoluteColdGasFile
```

#### 4.21.3.2 AbsoluteHotGasFile

```
std::string OutputValues::AbsoluteHotGasFile
```

### 4.21.3.3 ChemicalPrefactor

```
Argument<std::string> OutputValues::ChemicalPrefactor = Argument<std::string>("Enrichment_↵", "enrichment-base")
```

The enrichment file identifier.

### 4.21.3.4 ColdGasDataFile

```
Argument<std::string> OutputValues::ColdGasDataFile = Argument<std::string>("ColdGas.dat", "enrichment-cold")
```

The cold gas filename.

### 4.21.3.5 Config

```
Argument<std::string> OutputValues::Config = Argument<std::string>("rerun.config", "config-out")
```

The name for the output config file which would replicate this simulation.

### 4.21.3.6 EventRateFile

```
Argument<std::string> OutputValues::EventRateFile = Argument<std::string>("Events.dat", "event-rate-file")
```

### 4.21.3.7 GalaxyMassFile

```
Argument<std::string> OutputValues::GalaxyMassFile = Argument<std::string>("Mass.dat", "galaxy-mass-file")
```

The name of the file containing galactic-scale mass information.

### 4.21.3.8 HotGasDataFile

```
Argument<std::string> OutputValues::HotGasDataFile = Argument<std::string>("HotGas.dat", "enrichment-hot")
```

The hot gas filename.

#### 4.21.3.9 LogarithmicColdGasFile

```
std::string OutputValues::LogarithmicColdGasFile
```

#### 4.21.3.10 LogarithmicHotGasFile

```
std::string OutputValues::LogarithmicHotGasFile
```

#### 4.21.3.11 Root

```
Argument<std::string> OutputValues::Root = Argument<std::string>("Output/", "output")
```

The name of the output directory into which the output will be saved.

#### 4.21.3.12 StarFile

```
Argument<std::string> OutputValues::StarFile = Argument<std::string>("StellarCatalogue.↵  
dat", "ring-data-stars")
```

The ring-star data identifier.

#### 4.21.3.13 YieldSubdir

```
Argument<std::string> OutputValues::YieldSubdir = Argument<std::string>("Yields/", "yield-dir")
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/[ParameterLists.h](#)

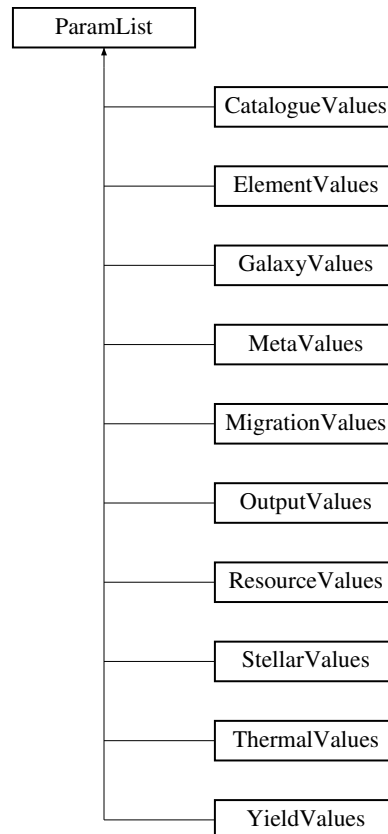


## 4.22 ParamList Class Reference

A Generic superclass structure so that I can heterogenously loop over the various members of [GlobalParameters](#) without writing it all out arduously. Also provides a consistent interface with the `JSL::Argument` environment.

```
#include <List.h>
```

Inheritance diagram for ParamList:



### Public Member Functions

- void [Configure](#) (int argc, char \*argv[])  
*Loops over all members of the `argPointers` array and calls the configuration/command line API on them to initialise the members of the child classes.*
- virtual void [Initialise](#) (std::string resourceRoot)  
*A (hopefully) rarely used function which calls any additional functions which can only be called after the configuration has been run. For most members, this is an empty function.*
- void [StreamContentsTo](#) (std::stringstream &stream)

### Protected Attributes

- std::vector< JSL::ArgumentInterface \* > [argPointers](#)  
*A list of pointers to member variables of the child classes which want to be initialised against command line / config-file values. Any Argument objects not added to this array will not be initialised! This array should be allocated during the individual subclass constructors.*

### 4.22.1 Detailed Description

A Generic superclass structure so that I can heterogenously loop over the various members of [GlobalParameters](#) without writing it all out arduously. Also provides a consistent interface with the JSL::Argument environment.

### 4.22.2 Member Function Documentation

#### 4.22.2.1 Configure()

```
void ParamList::Configure (
    int argc,
    char * argv[] )
```

Loops over all members of the argPointers array and calls the configuration/command line API on them to initialise the members of the child classes.

#### 4.22.2.2 Initialise()

```
virtual void ParamList::Initialise (
    std::string resourceRoot ) [inline], [virtual]
```

A (hopefully) rarely used function which calls any additional functions which can only be called *after* the configuration has been run. For most members, this is an empty function.

Reimplemented in [MetaValues](#), [OutputValues](#), [ResourceValues](#), [ElementValues](#), [StellarValues](#), [YieldValues](#), and [GalaxyValues](#).

#### 4.22.2.3 StreamContentsTo()

```
void ParamList::StreamContentsTo (
    std::stringstream & stream )
```

### 4.22.3 Field Documentation

### 4.22.3.1 argPointers

```
std::vector<JSL::ArgumentInterface *> ParamList::argPointers [protected]
```

A list of pointers to member variables of the child classes which want to be initialised against command line / config-file values. Any Argument objects not added to this array will not be initialised! This array should be allocated during the individual subclass constructors.

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

- [/Users/jf20/Documents/Physics/RAMICES\\_II/src/Parameters/List.h](#)

## 4.23 RemnantOutput Struct Reference

```
#include <YieldGrid.h>
```

### Data Fields

- [RemnantType Type](#)
- double [Mass](#)

### 4.23.1 Field Documentation

#### 4.23.1.1 Mass

```
double RemnantOutput::Mass
```

#### 4.23.1.2 Type

```
RemnantType RemnantOutput::Type
```

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

- [/Users/jf20/Documents/Physics/RAMICES\\_II/src/Yields/YieldGrid.h](#)

## 4.24 RemnantPopulation Class Reference

```
#include <RemnantPopulation.h>
```

## Public Member Functions

- [RemnantPopulation](#) ([InitialisedData](#) &data)
- void [Feed](#) (int *timeIndex*, double *bhMass*, double *wdMass*, double *nsMass*)
- void [Feed](#) (int *timeIndex*, [RemnantOutput](#) rem)
- void [Decay](#) (int *currentTime*, std::vector< [GasReservoir](#) > &*scatteringReservoir*, [StarEvents](#) &*EventRate*)
- [MassReport](#) *Mass* ()

## Private Attributes

- std::vector< double > [ShortSNIaBuffer](#)
- std::vector< double > [LongSNIaBuffer](#)
- std::vector< double > [NSMBuffer](#)
- const [SimpleYield](#) & [SNIaYield](#)
- const [SimpleYield](#) & [NSMYield](#)
- double [BlackHoleMass](#)
- double [DormantWDMass](#)
- double [DormantNSMass](#)
- const [GlobalParameters](#) & [Param](#)

## 4.24.1 Constructor & Destructor Documentation

### 4.24.1.1 RemnantPopulation()

```
RemnantPopulation::RemnantPopulation (
    InitialisedData & data )
```

## 4.24.2 Member Function Documentation

### 4.24.2.1 Decay()

```
void RemnantPopulation::Decay (
    int currentTime,
    std::vector< GasReservoir > & scatteringReservoir,
    StarEvents & EventRate )
```

### 4.24.2.2 Feed() [1/2]

```
void RemnantPopulation::Feed (
    int timeIndex,
    double bhMass,
    double wdMass,
    double nsMass )
```

#### 4.24.2.3 Feed() [2/2]

```
void RemnantPopulation::Feed (
    int timeIndex,
    RemnantOutput rem )
```

#### 4.24.2.4 Mass()

```
MassReport RemnantPopulation::Mass ( )
```

### 4.24.3 Field Documentation

#### 4.24.3.1 BlackHoleMass

```
double RemnantPopulation::BlackHoleMass [private]
```

#### 4.24.3.2 DormantNSMass

```
double RemnantPopulation::DormantNSMass [private]
```

#### 4.24.3.3 DormantWDMass

```
double RemnantPopulation::DormantWDMass [private]
```

#### 4.24.3.4 LongSNIaBuffer

```
std::vector<double> RemnantPopulation::LongSNIaBuffer [private]
```

#### 4.24.3.5 NSMBuffer

```
std::vector<double> RemnantPopulation::NSMBuffer [private]
```

#### 4.24.3.6 NSMYield

```
const SimpleYield& RemnantPopulation::NSMYield [private]
```

#### 4.24.3.7 Param

```
const GlobalParameters& RemnantPopulation::Param [private]
```

#### 4.24.3.8 ShortSNIaBuffer

```
std::vector<double> RemnantPopulation::ShortSNIaBuffer [private]
```

#### 4.24.3.9 SNIaYield

```
const SimpleYield& RemnantPopulation::SNIaYield [private]
```

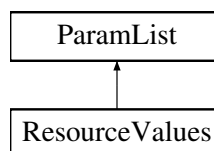
The documentation for this class was generated from the following file:

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/[RemnantPopulation.h](#)

## 4.25 ResourceValues Class Reference

```
#include <ParameterLists.h>
```

Inheritance diagram for ResourceValues:



### Public Member Functions

- [ResourceValues](#) ()  
*Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.*
- virtual void [Initialise](#) (std::string resourceRoot)  
*An overload of a normally empty function. Goes through and creates the necessary directory structure.*

## Data Fields

- `Argument< std::string > WelcomeFile = Argument<std::string>("welcome.dat","welcome-file")`  
*The location of the funky ASCII welcome message.*
- `Argument< std::string > ResourceRoot = Argument<std::string>("Resources/","resource")`  
*The location of the directory which the code looks for its expected resource file structure.*
- `Argument< std::string > YieldRoot = Argument<std::string>("ChemicalData/","yield-root")`  
*The location of the directory within ResourceRoot which houses the stellar yield data.*
- `Argument< std::string > IsochroneDirectory = Argument<std::string>("Isochrones/","iso-dir")`
- `Argument< std::string > LifeTimeFile = Argument<std::string>("LifetimeGrid.dat","lifetime-file")`
- `Argument< std::string > IsochroneRepository = Argument<std::string>("NewPadova/","iso-repo")`

## Additional Inherited Members

### 4.25.1 Constructor & Destructor Documentation

#### 4.25.1.1 ResourceValues()

```
ResourceValues::ResourceValues ( ) [inline]
```

Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.

### 4.25.2 Member Function Documentation

#### 4.25.2.1 Initialise()

```
virtual void ResourceValues::Initialise (
    std::string resourceRoot ) [virtual]
```

An overload of a normally empty function. Goes through and creates the necessary directory structure.

Reimplemented from [ParamList](#).

### 4.25.3 Field Documentation

#### 4.25.3.1 IsochroneDirectory

```
Argument<std::string> ResourceValues::IsochroneDirectory = Argument<std::string>("Isochrones/","iso-dir")
```

#### 4.25.3.2 IsochroneRepository

```
Argument<std::string> ResourceValues::IsochroneRepository = Argument<std::string>("New↵  
Padova/", "iso-repo")
```

#### 4.25.3.3 LifeTimeFile

```
Argument<std::string> ResourceValues::LifeTimeFile = Argument<std::string>("LifetimeGrid.↵  
dat", "lifetime-file")
```

#### 4.25.3.4 ResourceRoot

```
Argument<std::string> ResourceValues::ResourceRoot = Argument<std::string>("Resources/", "resource")
```

The location of the directory which the code looks for its expected resource file structure.

#### 4.25.3.5 WelcomeFile

```
Argument<std::string> ResourceValues::WelcomeFile = Argument<std::string>("welcome.dat", "welcome-file")
```

The location of the funky ASCII welcome message.

#### 4.25.3.6 YieldRoot

```
Argument<std::string> ResourceValues::YieldRoot = Argument<std::string>("ChemicalData/", "yield-root")
```

The location of the directory within ResourceRoot which houses the stellar yield data.

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/[ParameterLists.h](#)

## 4.26 Ring Class Reference

```
#include <Ring.h>
```



## Public Member Functions

- [Ring](#) (int radiusIndex, double mass, [InitialisedData](#) &data)  
*Initialises itself into a primordial state.*
- double [Mass](#) ()
- void [MakeStars](#) ()
- void [KillStars](#) (int time)
- void [Cool](#) ()
- void [TimeStep](#) (int t)
- void [UpdateMemory](#) (int t)
- void [SaveChemicalHistory](#) (int t, std::stringstream &absoluteStreamCold, std::stringstream &logarithmicStreamCold, std::stringstream &absoluteStreamHot, std::stringstream &logarithmicStreamHot)
- double [SelectionEffect](#) (double Mv, double age)
- void [ComputeSelectionFunction](#) (const double brightLimit, const double dimLimit)
- void [MetCheck](#) (const std::string &location)
- std::string [Synthesis](#) (const [StellarPopulation](#) &targetPopulation, double migrationFraction, double originRadius, double &totalSynthesised)

## Data Fields

- const double [Radius](#)
- const double [Width](#)
- double [Area](#)
- [StarReservoir](#) Stars
- [GasReservoir](#) Gas
- [GasReservoir](#) CGMBuffer

## Private Attributes

- int [RadiusIndex](#)
- [InitialisedData](#) & [Data](#)
- const [GlobalParameters](#) & [Param](#)
- std::vector< std::vector< double > > [ColdBuffer](#)
- std::vector< std::vector< double > > [HotBuffer](#)
- std::vector< std::vector< double > > [SelectionGrid](#)
- double [MinMv](#)
- double [MaxMv](#)

### 4.26.1 Constructor & Destructor Documentation

#### 4.26.1.1 Ring()

```
Ring::Ring (
    int radiusIndex,
    double mass,
    InitialisedData & data )
```

Initialises itself into a primordial state.

## 4.26.2 Member Function Documentation

### 4.26.2.1 ComputeSelectionFunction()

```
void Ring::ComputeSelectionFunction (
    const double brightLimit,
    const double dimLimit )
```

### 4.26.2.2 Cool()

```
void Ring::Cool ( )
```

### 4.26.2.3 KillStars()

```
void Ring::KillStars (
    int time )
```

### 4.26.2.4 MakeStars()

```
void Ring::MakeStars ( )
```

### 4.26.2.5 Mass()

```
double Ring::Mass ( )
```

### 4.26.2.6 MetCheck()

```
void Ring::MetCheck (
    const std::string & location )
```

#### 4.26.2.7 SaveChemicalHistory()

```
void Ring::SaveChemicalHistory (
    int t,
    std::stringstream & absoluteStreamCold,
    std::stringstream & logarithmicStreamCold,
    std::stringstream & absoluteStreamHot,
    std::stringstream & logarithmicStreamHot )
```

#### 4.26.2.8 SelectionEffect()

```
double Ring::SelectionEffect (
    double Mv,
    double age )
```

#### 4.26.2.9 Synthesis()

```
std::string Ring::Synthesis (
    const StellarPopulation & targetPopulation,
    double migrationFraction,
    double originRadius,
    double & totalSynthesised )
```

#### 4.26.2.10 TimeStep()

```
void Ring::TimeStep (
    int t )
```

#### 4.26.2.11 UpdateMemory()

```
void Ring::UpdateMemory (
    int t )
```

### 4.26.3 Field Documentation

#### 4.26.3.1 Area

```
double Ring::Area
```

#### 4.26.3.2 ColdBuffer

```
std::vector<std::vector<double> > Ring::ColdBuffer [private]
```

#### 4.26.3.3 Data

```
InitialisedData& Ring::Data [private]
```

#### 4.26.3.4 Gas

```
GasReservoir Ring::Gas
```

#### 4.26.3.5 HotBuffer

```
std::vector<std::vector<double> > Ring::HotBuffer [private]
```

#### 4.26.3.6 CGMBuffer

```
GasReservoir Ring::CGMBuffer
```

#### 4.26.3.7 MaxMv

```
double Ring::MaxMv [private]
```

#### 4.26.3.8 MinMv

```
double Ring::MinMv [private]
```

#### 4.26.3.9 Param

```
const GlobalParameters& Ring::Param [private]
```

#### 4.26.3.10 Radius

```
const double Ring::Radius
```

#### 4.26.3.11 RadiusIndex

```
int Ring::RadiusIndex [private]
```

#### 4.26.3.12 SelectionGrid

```
std::vector<std::vector<double> > Ring::SelectionGrid [private]
```

#### 4.26.3.13 Stars

```
StarReservoir Ring::Stars
```

#### 4.26.3.14 Width

```
const double Ring::Width
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Galaxy/[Ring.h](#)

## 4.27 SimpleYield Class Reference

```
#include <SimpleYield.h>
```

## Public Member Functions

- `SimpleYield` (const `GlobalParameters` &param, `YieldProcess` Process)
- void `operator()` (`GasReservoir` &scatteringReservoir, double nObjects) const

## Data Fields

- const `SourceProcess` Process

## Private Member Functions

- void `NSM_Initialise` ()
- void `SNla_Initialise` ()
- void `RemnantInject` (`GasReservoir` &scatteringReservoir, double Nstars) const

## Private Attributes

- double `hotInjectionFraction`
- `std::vector< double >` `Grid`
- const `GlobalParameters` & `Param`

## 4.27.1 Constructor & Destructor Documentation

### 4.27.1.1 SimpleYield()

```
SimpleYield::SimpleYield (
    const GlobalParameters & param,
    YieldProcess Process )
```

## 4.27.2 Member Function Documentation

### 4.27.2.1 NSM\_Initialise()

```
void SimpleYield::NSM_Initialise ( ) [private]
```

#### 4.27.2.2 operator()

```
void SimpleYield::operator() (
    GasReservoir & scatteringReservoir,
    double nObjects ) const
```

#### 4.27.2.3 RemnantInject()

```
void SimpleYield::RemnantInject (
    GasReservoir & scatteringReservoir,
    double Nstars ) const [private]
```

#### 4.27.2.4 SNIa\_Initialise()

```
void SimpleYield::SNIa_Initialise ( ) [private]
```

### 4.27.3 Field Documentation

#### 4.27.3.1 Grid

```
std::vector<double> SimpleYield::Grid [private]
```

#### 4.27.3.2 hotInjectionFraction

```
double SimpleYield::hotInjectionFraction [private]
```

#### 4.27.3.3 Param

```
const GlobalParameters& SimpleYield::Param [private]
```

#### 4.27.3.4 Process

```
const SourceProcess SimpleYield::Process
```

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

- [/Users/jf20/Documents/Physics/RAMICES\\_II/src/Yields/SimpleYield.h](#)

## 4.28 SLF\_Functor Class Reference

```
#include <SLF.h>
```

### Public Member Functions

- [SLF\\_Functor](#) (const [GlobalParameters](#) &[Param](#))
- int [operator\(\)](#) (int mass, double metallicity)
- double [PredictLifetime](#) (double mass, double logmetallicity)

### Private Member Functions

- double [ValueInquiry](#) (int m, int z)
- double [LifeTime](#) (int mass, int metallicity)
- void [PrecomputeGrid](#) ()

### Private Attributes

- const [GlobalParameters](#) & [Param](#)
- std::vector< std::vector< double > > [PrecomputedGrid](#)
- const double [NotComputed](#) = -1

## 4.28.1 Constructor & Destructor Documentation

### 4.28.1.1 SLF\_Functor()

```
SLF_Functor::SLF_Functor (
    const GlobalParameters & Param )
```

## 4.28.2 Member Function Documentation



#### 4.28.2.1 LifeTime()

```
double SLF_Functor::LifeTime (
    int mass,
    int metallicity ) [private]
```

#### 4.28.2.2 operator()

```
int SLF_Functor::operator() (
    int mass,
    double metallicity )
```

#### 4.28.2.3 PrecomputeGrid()

```
void SLF_Functor::PrecomputeGrid ( ) [private]
```

#### 4.28.2.4 PredictLifetime()

```
double SLF_Functor::PredictLifetime (
    double mass,
    double logmetallicity )
```

#### 4.28.2.5 ValueInquiry()

```
double SLF_Functor::ValueInquiry (
    int m,
    int z ) [private]
```

### 4.28.3 Field Documentation

#### 4.28.3.1 NotComputed

```
const double SLF_Functor::NotComputed = -1 [private]
```

#### 4.28.3.2 Param

```
const GlobalParameters& SLF_Functor::Param [private]
```

#### 4.28.3.3 PrecomputedGrid

```
std::vector<std::vector<double> > SLF_Functor::PrecomputedGrid [private]
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/[SLF.h](#)

## 4.29 StarEvents Class Reference

```
#include <StarEvents.h>
```

### Public Member Functions

- [StarEvents](#) ()
- void [AddHeaders](#) (std::stringstream &output)
- void [Save](#) (std::stringstream &output, double timestep)

### Data Fields

- double [StarMassFormed](#)
- double [NStarsFormed](#)
- double [CCSN](#)
- double [AGBDeaths](#)
- double [NSM](#)
- double [SNIa](#)
- double [ECSN](#)
- double [Efficiency](#)

### 4.29.1 Constructor & Destructor Documentation

#### 4.29.1.1 StarEvents()

```
StarEvents::StarEvents ( )
```

## 4.29.2 Member Function Documentation

### 4.29.2.1 AddHeaders()

```
void StarEvents::AddHeaders (
    std::stringstream & output )
```

### 4.29.2.2 Save()

```
void StarEvents::Save (
    std::stringstream & output,
    double timestep )
```

## 4.29.3 Field Documentation

### 4.29.3.1 AGBDeaths

```
double StarEvents::AGBDeaths
```

### 4.29.3.2 CCSN

```
double StarEvents::CCSN
```

### 4.29.3.3 ECSN

```
double StarEvents::ECSN
```

### 4.29.3.4 Efficiency

```
double StarEvents::Efficiency
```

#### 4.29.3.5 NSM

```
double StarEvents::NSM
```

#### 4.29.3.6 NStarsFormed

```
double StarEvents::NStarsFormed
```

#### 4.29.3.7 SNIa

```
double StarEvents::SNIa
```

#### 4.29.3.8 StarMassFormed

```
double StarEvents::StarMassFormed
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/[StarEvents.h](#)

## 4.30 StarReservoir Class Reference

```
#include <StarReservoir.h>
```

### Public Member Functions

- [StarReservoir](#) (int parentRing, [InitialisedData](#) &data)
- double [AliveMass](#) ()
- [MassReport](#) [DeadMass](#) ()
- void [Observations](#) ()
- void [Form](#) ([GasReservoir](#) &gas, [GasReservoir](#) &cgm)
- void [Death](#) (int currentTime)
- void [PrintStatus](#) (int t)
- const std::vector< [GasStream](#) > & [YieldsFrom](#) (int t)
- void [SaveEventRate](#) (int t, std::stringstream &output)
- void [AssignMagnitudes](#) ()

### Data Fields

- std::vector< [StellarPopulation](#) > [Population](#)
- std::vector< [GasReservoir](#) > [YieldOutput](#)

## Private Member Functions

- double [SFR\\_GasLoss](#) (double coldMass, double hotMass, double ejectFactor)

## Private Attributes

- [RemnantPopulation](#) [Remnants](#)
- const int [ParentRing](#)
- double [ParentArea](#)
- double [Temp\\_Mass](#)
- const [IMF\\_Functor](#) & [IMF](#)
- [SLF\\_Functor](#) & [SLF](#)
- int [PopulationIndex](#)
- std::vector< [StarEvents](#) > [EventRate](#)
- [InitialisedData](#) & [Data](#)
- const [GlobalParameters](#) & [Param](#)

## 4.30.1 Constructor & Destructor Documentation

### 4.30.1.1 StarReservoir()

```
StarReservoir::StarReservoir (
    int parentRing,
    InitialisedData & data )
```

## 4.30.2 Member Function Documentation

### 4.30.2.1 AliveMass()

```
double StarReservoir::AliveMass ( )
```

### 4.30.2.2 AssignMagnitudes()

```
void StarReservoir::AssignMagnitudes ( )
```

#### 4.30.2.3 DeadMass()

```
MassReport StarReservoir::DeadMass ( )
```

#### 4.30.2.4 Death()

```
void StarReservoir::Death (
    int currentTime )
```

#### 4.30.2.5 Form()

```
void StarReservoir::Form (
    GasReservoir & gas,
    GasReservoir & cgm )
```

#### 4.30.2.6 Observations()

```
void StarReservoir::Observations ( )
```

#### 4.30.2.7 PrintStatus()

```
void StarReservoir::PrintStatus (
    int t )
```

#### 4.30.2.8 SaveEventRate()

```
void StarReservoir::SaveEventRate (
    int t,
    std::stringstream & output )
```

#### 4.30.2.9 SFR\_GasLoss()

```
double StarReservoir::SFR_GasLoss (
    double coldMass,
    double hotMass,
    double ejectFactor ) [private]
```

#### 4.30.2.10 YieldsFrom()

```
const std::vector< GasStream > & StarReservoir::YieldsFrom (
    int t )
```

### 4.30.3 Field Documentation

#### 4.30.3.1 Data

```
InitialisedData & StarReservoir::Data [private]
```

#### 4.30.3.2 EventRate

```
std::vector<StarEvents> StarReservoir::EventRate [private]
```

#### 4.30.3.3 IMF

```
const IMF_Functor & StarReservoir::IMF [private]
```

#### 4.30.3.4 Param

```
const GlobalParameters & StarReservoir::Param [private]
```

#### 4.30.3.5 ParentArea

```
double StarReservoir::ParentArea [private]
```

#### 4.30.3.6 ParentRing

```
const int StarReservoir::ParentRing [private]
```

#### 4.30.3.7 Population

```
std::vector<StellarPopulation> StarReservoir::Population
```

#### 4.30.3.8 PopulationIndex

```
int StarReservoir::PopulationIndex [private]
```

#### 4.30.3.9 Remnants

```
RemnantPopulation StarReservoir::Remnants [private]
```

#### 4.30.3.10 SLF

```
SLF_Functor& StarReservoir::SLF [private]
```

#### 4.30.3.11 Temp\_Mass

```
double StarReservoir::Temp_Mass [private]
```

#### 4.30.3.12 YieldOutput

```
std::vector<GasReservoir> StarReservoir::YieldOutput
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/[StarReservoir.h](#)

## 4.31 StellarPopulation Class Reference

```
#include <StellarPopulation.h>
```



## Public Member Functions

- [StellarPopulation](#) ([InitialisedData](#) &data, int parentRing)
- void [PrepareIMF](#) ()
- int [FormStars](#) (double formingMass, int timeIndex, [GasReservoir](#) &formingGas)  
*Returns the number of stars formed (spread across all mass grids)*
- double [Mass](#) ()
- [IsoMass](#) & [Relic](#) ()
- const [IsoMass](#) & [Relic](#) () const
- [IsoMass](#) & [operator\[\]](#) (int i)
- const [IsoMass](#) & [operator\[\]](#) (int i) const
- bool [Active](#) ()
- void [Death](#) (int time, std::vector< [GasReservoir](#) > &TemporalYieldGrid, [RemnantPopulation](#) &remnants, [StarEvents](#) &EventRate)
- std::string [CatalogueHeaders](#) ()
- std::string [CatalogueEntry](#) (std::vector< int > popEntry, int m, double currentRadius, double birthRadius) const

## Data Fields

- int [BirthRadius](#)
- double [Metallicity](#)
- int [BirthIndex](#)
- std::vector< [IsoMass](#) > [Distribution](#)
- [IsoMass](#) [ImmortalStars](#)
- std::vector< [GasStream](#) > [BirthGas](#)
- double [Age](#)

## Private Member Functions

- void [MonotonicDeathScan](#) (int time, std::vector< [GasReservoir](#) > &[YieldGrid](#), [RemnantPopulation](#) &remnants, [StarEvents](#) &eventRate)
- void [FullDeathScan](#) (int time)
- void [RecoverMatter](#) (int time, int nstars, int mass, [GasReservoir](#) &temporalYieldGrid, [RemnantPopulation](#) &remnants)

## Private Attributes

- const [GlobalParameters](#) & [Param](#)
- const [IMF\\_Functor](#) & [IMF](#)
- [SLF\\_Functor](#) & [SLF](#)
- const [YieldGrid](#) & [CCSNYield](#)
- const [YieldGrid](#) & [ECSNYield](#)
- const [YieldGrid](#) & [AGBYield](#)
- [InitialisedData](#) & [Data](#)
- bool [IsLifetimeMonotonic](#)
- bool [IsDepleted](#)
- int [DepletionIndex](#)
- double [internal\\_MassCounter](#)
- [Gas](#) [TempGas](#)

## 4.31.1 Constructor & Destructor Documentation

### 4.31.1.1 StellarPopulation()

```
StellarPopulation::StellarPopulation (
    InitialisedData & data,
    int parentRing )
```

## 4.31.2 Member Function Documentation

### 4.31.2.1 Active()

```
bool StellarPopulation::Active ( )
```

### 4.31.2.2 CatalogueEntry()

```
std::string StellarPopulation::CatalogueEntry (
    std::vector< int > popEntry,
    int m,
    double currentRadius,
    double birthRadius ) const
```

### 4.31.2.3 CatalogueHeaders()

```
std::string StellarPopulation::CatalogueHeaders ( )
```

### 4.31.2.4 Death()

```
void StellarPopulation::Death (
    int time,
    std::vector< GasReservoir > & TemporalYieldGrid,
    RemnantPopulation & remnants,
    StarEvents & EventRate )
```

#### 4.31.2.5 FormStars()

```
int StellarPopulation::FormStars (
    double formingMass,
    int timeIndex,
    GasReservoir & formingGas )
```

Returns the number of stars formed (spread across all mass grids)

#### 4.31.2.6 FullDeathScan()

```
void StellarPopulation::FullDeathScan (
    int time ) [private]
```

#### 4.31.2.7 Mass()

```
double StellarPopulation::Mass ( )
```

#### 4.31.2.8 MonotonicDeathScan()

```
void StellarPopulation::MonotonicDeathScan (
    int time,
    std::vector< GasReservoir > & YieldGrid,
    RemnantPopulation & remnants,
    StarEvents & eventRate ) [private]
```

#### 4.31.2.9 operator[]() [1/2]

```
IsoMass & StellarPopulation::operator[] (
    int i )
```

#### 4.31.2.10 operator[]() [2/2]

```
const IsoMass & StellarPopulation::operator[] (
    int i ) const
```

#### 4.31.2.11 PrepareIMF()

```
void StellarPopulation::PrepareIMF ( )
```

#### 4.31.2.12 RecoverMatter()

```
void StellarPopulation::RecoverMatter (
    int time,
    int nstars,
    int mass,
    GasReservoir & temporalYieldGrid,
    RemnantPopulation & remnants ) [private]
```

#### 4.31.2.13 Relic() [1/2]

```
IsoMass & StellarPopulation::Relic ( )
```

#### 4.31.2.14 Relic() [2/2]

```
const IsoMass & StellarPopulation::Relic ( ) const
```

### 4.31.3 Field Documentation

#### 4.31.3.1 AGBYield

```
const YieldGrid& StellarPopulation::AGBYield [private]
```

#### 4.31.3.2 Age

```
double StellarPopulation::Age
```

#### 4.31.3.3 BirthGas

```
std::vector<GasStream> StellarPopulation::BirthGas
```

#### 4.31.3.4 BirthIndex

```
int StellarPopulation::BirthIndex
```

#### 4.31.3.5 BirthRadius

```
int StellarPopulation::BirthRadius
```

#### 4.31.3.6 CCSNYield

```
const YieldGrid& StellarPopulation::CCSNYield [private]
```

#### 4.31.3.7 Data

```
InitialisedData& StellarPopulation::Data [private]
```

#### 4.31.3.8 DepletionIndex

```
int StellarPopulation::DepletionIndex [private]
```

#### 4.31.3.9 Distribution

```
std::vector<IsoMass> StellarPopulation::Distribution
```

#### 4.31.3.10 ECSNYield

```
const YieldGrid& StellarPopulation::ECSNYield [private]
```

#### 4.31.3.11 IMF

```
const IMF_Functor& StellarPopulation::IMF [private]
```

#### 4.31.3.12 ImmortalStars

```
IsoMass StellarPopulation::ImmortalStars
```

#### 4.31.3.13 internal\_MassCounter

```
double StellarPopulation::internal_MassCounter [private]
```

#### 4.31.3.14 IsDepleted

```
bool StellarPopulation::IsDepleted [private]
```

#### 4.31.3.15 IsLifetimeMonotonic

```
bool StellarPopulation::IsLifetimeMonotonic [private]
```

#### 4.31.3.16 Metallicity

```
double StellarPopulation::Metallicity
```

#### 4.31.3.17 Param

```
const GlobalParameters& StellarPopulation::Param [private]
```

#### 4.31.3.18 SLF

```
SLF_Functor& StellarPopulation::SLF [private]
```

### 4.31.3.19 TempGas

`Gas StellarPopulation::TempGas [private]`

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

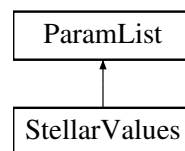
- `/Users/jf20/Documents/Physics/RAMICES_II/src/Stars/StellarPopulation.h`

## 4.32 StellarValues Class Reference

The subset of values associated with stars + their remnants.

```
#include <ParameterLists.h>
```

Inheritance diagram for StellarValues:



### Public Member Functions

- `StellarValues ()`  
*Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.*
- `void Initialise (std::string resourceRoot)`  
*Initialises the mass grid etc.*

### Data Fields

- `Argument< double > MaxStellarMass = Argument<double>(100,"mass-max")`  
*Minimum stellar mass that IMF can generate.*
- `Argument< double > MinStellarMass = Argument<double>(0.1,"mass-min")`  
*Maximum stellar mass that IMF can generate.*
- `Argument< double > ImmortalMass = Argument<double>(0.3,"mass-immortal")`  
*Mass of stars which we consider immortal without checking their isochrones.*
- `Argument< int > MassResolution = Argument<int>(199,"mass-resolution")`  
*Number of points along the stellar mass grid.*
- `std::vector< double > MassGrid`  
*A grid which holds the masses onto which all interpolation will take place. Allows for the possibility of non-uniform steps. The values are the centre of each mass divide.*
- `std::vector< double > MassDeltas`  
*The corresponding widths of each interval on the mass line.*
- `Argument< double > MinLogZ = Argument<double>(-6,"logz-min")`  
*Minimum Z that the ILM(??) can consider.*
- `Argument< double > MaxLogZ = Argument<double>(-0.1,"logz-max")`  
*Maximum Z that the ILM(??) can consider.*

- `Argument< int > LogZResolution = Argument<int>(100,"logz-resolution")`  
*Z Resolution.*
- `std::vector< double > LogZGrid`  
*As with MassGrid, but for metallicity (assumed to be always uniform in log-space)*
- `double LogZDelta`
- `Argument< double > EjectionFraction = Argument<double>(0.45,"eject")`  
*The fraction of supernovae ejecta which is thrown into the CGM.*
- `Argument< double > FeedbackFactor = Argument<double>(0.5,"mass-load")`  
*For every 1 solar mass of stars which form, this fraction of gas is heated into the hot phase.*
- `Argument< double > SchmidtMainPower = Argument<double>(1.4,"schmidt-main")`  
*The normal Kennicutt-Schmidt power law index.*
- `Argument< double > SchmidtLowPower = Argument<double>(4.0,"schmidt-low")`  
*The low-density Kennicutt-Schmidt power law index.*
- `Argument< double > SchmidtDensityCut = Argument<double>(0,"schmidt-cut")`  
*The density cut for the low/high density switchover in Schmidt power law.*
- `Argument< double > SchmidtPrefactor = Argument<double>(2,"schmidt-factor")`  
*The Schmidt prefactor.*
- `Argument< double > IMF_Slope = Argument<double>(2.3,"imf-slope")`  
*The slope of the high-mass tail fo the IMF.*

## Additional Inherited Members

### 4.32.1 Detailed Description

The subset of values associated with stars + their remnants.

### 4.32.2 Constructor & Destructor Documentation

#### 4.32.2.1 StellarValues()

```
StellarValues::StellarValues ( ) [inline]
```

Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.

### 4.32.3 Member Function Documentation

#### 4.32.3.1 Initialise()

```
void StellarValues::Initialise (
    std::string resourceRoot ) [virtual]
```

Initialises the mass grid etc.

Reimplemented from [ParamList](#).



## 4.32.4 Field Documentation

### 4.32.4.1 EjectionFraction

```
Argument<double> StellarValues::EjectionFraction = Argument<double>(0.45, "eject")
```

The fraction of supernovae ejecta which is thrown into the CGM.

### 4.32.4.2 FeedbackFactor

```
Argument<double> StellarValues::FeedbackFactor = Argument<double>(0.5, "mass-load")
```

For every 1 solar mass of stars which form, this fraction of gas is heated into the hot phase.

### 4.32.4.3 IMF\_Slope

```
Argument<double> StellarValues::IMF_Slope = Argument<double>(2.3, "imf-slope")
```

The slope of the high-mass tail to the IMF.

### 4.32.4.4 ImmortalMass

```
Argument<double> StellarValues::ImmortalMass = Argument<double>(0.3, "mass-immortal")
```

Mass of stars which we consider immortal without checking their isochrones.

### 4.32.4.5 LogZDelta

```
double StellarValues::LogZDelta
```

### 4.32.4.6 LogZGrid

```
std::vector<double> StellarValues::LogZGrid
```

As with MassGrid, but for metallicity (assumed to be always uniform in log-space)

#### 4.32.4.7 LogZResolution

```
Argument<int> StellarValues::LogZResolution = Argument<int>(100, "logz-resolution")
```

Z Resolution.

#### 4.32.4.8 MassDeltas

```
std::vector<double> StellarValues::MassDeltas
```

The corresponding widths of each interval on the mass line.

#### 4.32.4.9 MassGrid

```
std::vector<double> StellarValues::MassGrid
```

A grid which holds the masses onto which all interpolation will take place. Allows for the possibility of non-uniform steps. The values are the centre of each mass divide.

#### 4.32.4.10 MassResolution

```
Argument<int> StellarValues::MassResolution = Argument<int>(199, "mass-resolution")
```

Number of points along the stellar mass grid.

#### 4.32.4.11 MaxLogZ

```
Argument<double> StellarValues::MaxLogZ = Argument<double>(-0.1, "logz-max")
```

Maximum Z that the ILM(??) can consider.

#### 4.32.4.12 MaxStellarMass

```
Argument<double> StellarValues::MaxStellarMass = Argument<double>(100, "mass-max")
```

Minimum stellar mass that IMF can generate.

#### 4.32.4.13 MinLogZ

```
Argument<double> StellarValues::MinLogZ = Argument<double>(-6, "logz-min")
```

Minimum Z that the ILM(??) can consider.

#### 4.32.4.14 MinStellarMass

```
Argument<double> StellarValues::MinStellarMass = Argument<double>(0.1, "mass-min")
```

Maximum stellar mass that IMF can generate.

#### 4.32.4.15 SchmidtDensityCut

```
Argument<double> StellarValues::SchmidtDensityCut = Argument<double>(0, "schmidt-cut")
```

The density cut for the low/high density switchover in Schmidt power law.

#### 4.32.4.16 SchmidtLowPower

```
Argument<double> StellarValues::SchmidtLowPower = Argument<double>(4.0, "schmidt-low")
```

The low-density Kennicutt-Schmidt power law index.

#### 4.32.4.17 SchmidtMainPower

```
Argument<double> StellarValues::SchmidtMainPower = Argument<double>(1.4, "schmidt-main")
```

The normal Kennicutt-Schmidt power law index.

#### 4.32.4.18 SchmidtPrefactor

```
Argument<double> StellarValues::SchmidtPrefactor = Argument<double>(2, "schmidt-factor")
```

The Schmidt prefactor.

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

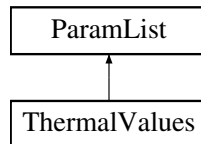
- /Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/[ParameterLists.h](#)

## 4.33 ThermalValues Class Reference

Thermal suboptions contain variables which deal with the thermal subroutines - cooling timescales injection fractions etc.

```
#include <ParameterLists.h>
```

Inheritance diagram for ThermalValues:



### Public Member Functions

- [ThermalValues](#) ()

*Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.*

### Data Fields

- Argument< double > [HotInjection\\_CCSN](#) = Argument<double>(0.7,"fh-ccsn")  
*Fraction of CCSN ejecta which is put into the hot phase.*
- Argument< double > [HotInjection\\_NSM](#) = Argument<double>(0.4,"fh-nsm")  
*Fraction of NSM ejecta which is put into the hot phase.*
- Argument< double > [HotInjection\\_AGB](#) = Argument<double>(0.7,"fh-agb")  
*Fraction of AGB ejecta which is put into the hot phase.*
- Argument< double > [FeedbackEjectFactor](#) = Argument<double>(0,"feedback-eject")
- Argument< double > [ChimneyFactor](#) = Argument<double>(0,"chimney")
- Argument< double > [HotInjection\\_SNIa](#) = Argument<double>(0.99,"fh-sn1a")  
*Fraction of SNIa ejecta which is put into the hot phase.*
- Argument< double > [GasCoolingTimeScale](#) = Argument<double>(1,"cool")  
*The exponential timescale over which the hot gas cools into the cold gas.*
- Argument< int > [NumericalResolution](#) = Argument<int>(30,"cool-resolution")
- Argument< double > [DormantHotFraction](#) = Argument<double>(1e-20,"dormant-hot-frac")
- Argument< double > [CoolingPower](#) = Argument<double>(1,"cooling-index")

### Additional Inherited Members

#### 4.33.1 Detailed Description

Thermal suboptions contain variables which deal with the thermal subroutines - cooling timescales injection fractions etc.

#### 4.33.2 Constructor & Destructor Documentation

#### 4.33.2.1 ThermalValues()

```
ThermalValues::ThermalValues ( ) [inline]
```

Boring constructor – slots in the relevant arguments into the ParamList::argPointer array.

### 4.33.3 Field Documentation

#### 4.33.3.1 ChimneyFactor

```
Argument<double> ThermalValues::ChimneyFactor = Argument<double>(0,"chimney")
```

#### 4.33.3.2 CoolingPower

```
Argument<double> ThermalValues::CoolingPower = Argument<double>(1,"cooling-index")
```

#### 4.33.3.3 DormantHotFraction

```
Argument<double> ThermalValues::DormantHotFraction = Argument<double>(1e-20,"dormant-hot-frac")
```

#### 4.33.3.4 FeedbackEjectFactor

```
Argument<double> ThermalValues::FeedbackEjectFactor = Argument<double>(0,"feedback-eject")
```

#### 4.33.3.5 GasCoolingTimeScale

```
Argument<double> ThermalValues::GasCoolingTimeScale = Argument<double>(1,"cool")
```

The exponential timescale over which the hot gas cools into the cold gas.

#### 4.33.3.6 HotInjection\_AGB

```
Argument<double> ThermalValues::HotInjection_AGB = Argument<double>(0.7, "fh-agb")
```

Fraction of AGB ejecta which is put into the hot phase.

#### 4.33.3.7 HotInjection\_CCSN

```
Argument<double> ThermalValues::HotInjection_CCSN = Argument<double>(0.7, "fh-ccsn")
```

Fraction of CCSN ejecta which is put into the hot phase.

#### 4.33.3.8 HotInjection\_NSM

```
Argument<double> ThermalValues::HotInjection_NSM = Argument<double>(0.4, "fh-nsm")
```

Fraction of NSM ejecta which is put into the hot phase.

#### 4.33.3.9 HotInjection\_SNIa

```
Argument<double> ThermalValues::HotInjection_SNIa = Argument<double>(0.99, "fh-sni1a")
```

Fraction of SNIa ejecta which is put into the hot phase.

#### 4.33.3.10 NumericalResolution

```
Argument<int> ThermalValues::NumericalResolution = Argument<int>(30, "cool-resolution")
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/[ParameterLists.h](#)

## 4.34 YieldBracket Struct Reference

```
#include <YieldRidge.h>
```

## Public Member Functions

- [YieldBracket](#) ()
- [YieldBracket](#) ([YieldRidge](#) r1, [YieldRidge](#) r2)
- [YieldBracket](#) ([YieldRidge](#) r1)
- double [Interpolate](#) (double mass, double z)

## Data Fields

- bool [isEnclosed](#)
- bool [hasSingle](#)
- [YieldRidge](#) [UpperRidge](#)
- [YieldRidge](#) [LowerRidge](#)

## 4.34.1 Constructor & Destructor Documentation

### 4.34.1.1 YieldBracket() [1/3]

```
YieldBracket::YieldBracket ( ) [inline]
```

### 4.34.1.2 YieldBracket() [2/3]

```
YieldBracket::YieldBracket (
    YieldRidge r1,
    YieldRidge r2 ) [inline]
```

### 4.34.1.3 YieldBracket() [3/3]

```
YieldBracket::YieldBracket (
    YieldRidge r1 ) [inline]
```

## 4.34.2 Member Function Documentation

### 4.34.2.1 Interpolate()

```
double YieldBracket::Interpolate (
    double mass,
    double z )
```

### 4.34.3 Field Documentation

#### 4.34.3.1 hasSingle

```
bool YieldBracket::hasSingle
```

#### 4.34.3.2 isEnclosed

```
bool YieldBracket::isEnclosed
```

#### 4.34.3.3 LowerRidge

```
YieldRidge YieldBracket::LowerRidge
```

#### 4.34.3.4 UpperRidge

```
YieldRidge YieldBracket::UpperRidge
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Yields/[YieldRidge.h](#)

## 4.35 YieldGrid Class Reference

```
#include <YieldGrid.h>
```

### Public Member Functions

- [YieldGrid](#) (const [GlobalParameters](#) &param, [YieldProcess](#) Process)
- [RemnantOutput operator\(\)](#) ([GasReservoir](#) &scatteringReservoir, double Nstars, int mass, double z, const std::vector< [GasStream](#) > &birthGas) const

### Data Fields

- const [SourceProcess](#) Process



## Private Member Functions

- void [CCSN\\_Initialise](#) ()
- void [AGB\\_Initialise](#) ()
- void [ECSN\\_Initialise](#) ()
- void [InitialiseLargeGrid](#) (int mSize, int zSize)
- [RemnantOutput StellarInject](#) ([GasReservoir](#) &scatteringReservoir, double Nstars, int mass, double z, const std::vector< [GasStream](#) > &birthGas) const
- void [LoadOrfeoYields](#) ()
- void [LoadMarigoYields](#) ()
- void [LoadLimongiYields](#) ()
- void [LoadMaederYields](#) ()
- [Interpolator MetallicityInterpolation](#) (double z) const
- double [ElementProduction](#) ([ElementID](#) element, double synthesisFraction, double ejectaMass, std::vector< [GasStream](#) > &output, const std::vector< [GasStream](#) > &birthStreams) const
- void [ElementDestruction](#) ([ElementID](#) element, double synthesisFraction, double ejectaMass, std::vector< [GasStream](#) > &output, const std::vector< [GasStream](#) > &birthStreams) const
- void [CreateGrid](#) ()
- [YieldBracket GetBracket](#) (int id, double mass, double z, bool overhang)
- void [SaveGrid](#) (std::string name)
- void [PurityEnforce](#) ()

## Private Attributes

- const [GlobalParameters](#) & [Param](#)
- std::vector< std::vector< std::vector< double > > > [Grid](#)
- double [hotInjectionFraction](#)
- int [MassOffset](#)
- std::vector< std::vector< [YieldRidge](#) > > [RidgeStorage](#)
- int [RemnantLocation](#)
- std::vector< std::vector< int > > [SourcePriority](#)

## 4.35.1 Constructor & Destructor Documentation

### 4.35.1.1 YieldGrid()

```
YieldGrid::YieldGrid (
    const GlobalParameters & param,
    YieldProcess Process )
```

## 4.35.2 Member Function Documentation

#### 4.35.2.1 AGB\_Initialise()

```
void YieldGrid::AGB_Initialise ( ) [private]
```

#### 4.35.2.2 CCSN\_Initialise()

```
void YieldGrid::CCSN_Initialise ( ) [private]
```

#### 4.35.2.3 CreateGrid()

```
void YieldGrid::CreateGrid ( ) [private]
```

#### 4.35.2.4 ECSN\_Initialise()

```
void YieldGrid::ECSN_Initialise ( ) [private]
```

#### 4.35.2.5 ElementDestruction()

```
void YieldGrid::ElementDestruction (
    ElementID element,
    double synthesisFraction,
    double ejectaMass,
    std::vector< GasStream > & output,
    const std::vector< GasStream > & birthStreams ) const [private]
```

#### 4.35.2.6 ElementProduction()

```
double YieldGrid::ElementProduction (
    ElementID element,
    double synthesisFraction,
    double ejectaMass,
    std::vector< GasStream > & output,
    const std::vector< GasStream > & birthStreams ) const [private]
```

#### 4.35.2.7 GetBracket()

```
YieldBracket YieldGrid::GetBracket (
    int id,
    double mass,
    double z,
    bool overhang ) [private]
```

#### 4.35.2.8 InitialiseLargeGrid()

```
void YieldGrid::InitialiseLargeGrid (
    int mSize,
    int zSize ) [private]
```

#### 4.35.2.9 LoadLimongiYields()

```
void YieldGrid::LoadLimongiYields ( ) [private]
```

#### 4.35.2.10 LoadMaederYields()

```
void YieldGrid::LoadMaederYields ( ) [private]
```

#### 4.35.2.11 LoadMarigoYields()

```
void YieldGrid::LoadMarigoYields ( ) [private]
```

#### 4.35.2.12 LoadOrfeoYields()

```
void YieldGrid::LoadOrfeoYields ( ) [private]
```

#### 4.35.2.13 MetallicityInterpolation()

```
Interpolator YieldGrid::MetallicityInterpolation (
    double z ) const [private]
```

#### 4.35.2.14 operator()()

```
RemnantOutput YieldGrid::operator() (
    GasReservoir & scatteringReservoir,
    double Nstars,
    int mass,
    double z,
    const std::vector< GasStream > & birthGas ) const
```

#### 4.35.2.15 PurityEnforce()

```
void YieldGrid::PurityEnforce ( ) [private]
```

#### 4.35.2.16 SaveGrid()

```
void YieldGrid::SaveGrid (
    std::string name ) [private]
```

#### 4.35.2.17 StellarInject()

```
RemnantOutput YieldGrid::StellarInject (
    GasReservoir & scatteringReservoir,
    double Nstars,
    int mass,
    double z,
    const std::vector< GasStream > & birthGas ) const [private]
```

### 4.35.3 Field Documentation

#### 4.35.3.1 Grid

```
std::vector<std::vector<std::vector<double> > > YieldGrid::Grid [private]
```

#### 4.35.3.2 hotInjectionFraction

```
double YieldGrid::hotInjectionFraction [private]
```

#### 4.35.3.3 MassOffset

```
int YieldGrid::MassOffset [private]
```

#### 4.35.3.4 Param

```
const GlobalParameters& YieldGrid::Param [private]
```

#### 4.35.3.5 Process

```
const SourceProcess YieldGrid::Process
```

#### 4.35.3.6 RemnantLocation

```
int YieldGrid::RemnantLocation [private]
```

#### 4.35.3.7 RidgeStorage

```
std::vector<std::vector<YieldRidge> > YieldGrid::RidgeStorage [private]
```

#### 4.35.3.8 SourcePriority

```
std::vector<std::vector<int> > YieldGrid::SourcePriority [private]
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Yields/[YieldGrid.h](#)

## 4.36 YieldPoint Struct Reference

```
#include <YieldRidge.h>
```

## Public Member Functions

- [YieldPoint](#) (double m, double y)
- [YieldPoint](#) ()

## Data Fields

- double [Mass](#)
- double [Yield](#)

## 4.36.1 Constructor & Destructor Documentation

### 4.36.1.1 [YieldPoint\(\)](#) [1/2]

```
YieldPoint::YieldPoint (  
    double m,  
    double y ) [inline]
```

### 4.36.1.2 [YieldPoint\(\)](#) [2/2]

```
YieldPoint::YieldPoint ( ) [inline]
```

## 4.36.2 Field Documentation

### 4.36.2.1 [Mass](#)

```
double YieldPoint::Mass
```

### 4.36.2.2 [Yield](#)

```
double YieldPoint::Yield
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Yields/[YieldRidge.h](#)

## 4.37 YieldRidge Class Reference

```
#include <YieldRidge.h>
```

### Public Member Functions

- [YieldRidge](#) ()
- [YieldRidge](#) ([SourceID](#) source, double z, int nPoints)

### Data Fields

- [SourceID](#) [Source](#)
- double [Z](#)
- std::vector< [YieldPoint](#) > [Points](#)

### 4.37.1 Constructor & Destructor Documentation

#### 4.37.1.1 YieldRidge() [1/2]

```
YieldRidge::YieldRidge ( )
```

#### 4.37.1.2 YieldRidge() [2/2]

```
YieldRidge::YieldRidge (
    SourceID source,
    double z,
    int nPoints )
```

### 4.37.2 Field Documentation

#### 4.37.2.1 Points

```
std::vector<YieldPoint> YieldRidge::Points
```

### 4.37.2.2 Source

[SourceID](#) `YieldRidge::Source`

### 4.37.2.3 Z

`double YieldRidge::Z`

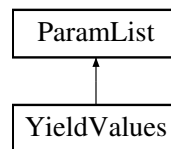
The documentation for this class was generated from the following file:

- `/Users/jf20/Documents/Physics/RAMICES_II/src/Yields/YieldRidge.h`

## 4.38 YieldValues Class Reference

```
#include <ParameterLists.h>
```

Inheritance diagram for YieldValues:



### Public Member Functions

- [YieldValues](#) ()
- void [Initialise](#) (std::string resourceRoot)  
*Initialises the mass grid etc.*

### Data Fields

- Argument< double > [TargetNi56Yield](#) = Argument<double>(0.1,"ideal-ni56")
- Argument< double > [MassOverhang](#) = Argument<double>(5,"yield-mass-overhang")
- std::vector< std::string > [ProcessNames](#)
- std::vector< [SourceProcess](#) > [ProcessTypes](#)
- Argument< double > [SNIa\\_DelayTime](#) = Argument<double>(0.2,"sn1a-delay")  
*Time before SNIa can turn on, in Gyr.*
- Argument< double > [SNIa\\_ActiveFraction](#) = Argument<double>(0.1,"sn1a-fraction")
- Argument< double > [SNIa\\_LongFraction](#) = Argument<double>(0.99,"sn1a-fraction-long")
- Argument< double > [SNIa\\_ShortScale](#) = Argument<double>(0.1,"sn1a-short-decay")
- Argument< double > [SNIa\\_TypicalMass](#) = Argument<double>(1.37,"sn1a-progenitor-mass")
- Argument< double > [NSM\\_TypicalMass](#) = Argument<double>(1.4,"nsm-progenitor-mass")
- Argument< double > [SNIa\\_LongScale](#) = Argument<double>(1.5,"sn1a-long-decay")
- Argument< double > [CCSN\\_MassCut](#) = Argument<double>(10,"ccsn-mass")
- Argument< double > [ECSN\\_MassCut](#) = Argument<double>(8.5,"ecsn-mass")
- Argument< double > [CODwarf\\_MassCut](#) = Argument<double>(3.2,"co-mass")
- Argument< double > [Collapse\\_MassCut](#) = Argument<double>(40,"bh-mass")
- Argument< double > [ECSN\\_Fraction](#) = Argument<double>(0,"ecsn-fraction")  
*Fraction of stars in ECSN mass range which go ECSN vs CCSN.*
- Argument< double > [NSM\\_DelayTime](#) = Argument<double>(0.02,"nsm-delay")
- Argument< double > [NSM\\_ActiveFraction](#) = Argument<double>(0.001,"nsm-fraction")
- Argument< double > [NSM\\_Scale](#) = Argument<double>(10,"nsm-decay")



## Additional Inherited Members

### 4.38.1 Constructor & Destructor Documentation

#### 4.38.1.1 YieldValues()

```
YieldValues::YieldValues ( ) [inline]
```

### 4.38.2 Member Function Documentation

#### 4.38.2.1 Initialise()

```
void YieldValues::Initialise (
    std::string resourceRoot ) [virtual]
```

Initialises the mass grid etc.

Reimplemented from [ParamList](#).

### 4.38.3 Field Documentation

#### 4.38.3.1 CCSN\_MassCut

```
Argument<double> YieldValues::CCSN_MassCut = Argument<double>(10, "ccsn-mass")
```

#### 4.38.3.2 CODwarf\_MassCut

```
Argument<double> YieldValues::CODwarf_MassCut = Argument<double>(3.2, "co-mass")
```

#### 4.38.3.3 Collapse\_MassCut

```
Argument<double> YieldValues::Collapse_MassCut = Argument<double>(40, "bh-mass")
```

#### 4.38.3.4 ECSN\_Fraction

```
Argument<double> YieldValues::ECSN_Fraction = Argument<double>(0, "ecsn-fraction")
```

Fraction of stars in ECSN mass range which go ECSN vs CCSN.

#### 4.38.3.5 ECSN\_MassCut

```
Argument<double> YieldValues::ECSN_MassCut = Argument<double>(8.5, "ecsn-mass")
```

#### 4.38.3.6 MassOverhang

```
Argument<double> YieldValues::MassOverhang = Argument<double>(5, "yield-mass-overhang")
```

#### 4.38.3.7 NSM\_ActiveFraction

```
Argument<double> YieldValues::NSM_ActiveFraction = Argument<double>(0.001, "nsm-fraction")
```

#### 4.38.3.8 NSM\_DelayTime

```
Argument<double> YieldValues::NSM_DelayTime = Argument<double>(0.02, "nsm-delay")
```

#### 4.38.3.9 NSM\_Scale

```
Argument<double> YieldValues::NSM_Scale = Argument<double>(10, "nsm-decay")
```

#### 4.38.3.10 NSM\_TypicalMass

```
Argument<double> YieldValues::NSM_TypicalMass = Argument<double>(1.4, "nsm-progenitor-mass")
```

#### 4.38.3.11 ProcessNames

```
std::vector<std::string> YieldValues::ProcessNames
```

#### 4.38.3.12 ProcessTypes

```
std::vector<SourceProcess> YieldValues::ProcessTypes
```

#### 4.38.3.13 SNla\_ActiveFraction

```
Argument<double> YieldValues::SNla_ActiveFraction = Argument<double>(0.1, "snla-fraction")
```

#### 4.38.3.14 SNla\_DelayTime

```
Argument<double> YieldValues::SNla_DelayTime = Argument<double>(0.2, "snla-delay")
```

Time before SNla can turn on, in Gyr.

#### 4.38.3.15 SNla\_LongFraction

```
Argument<double> YieldValues::SNla_LongFraction = Argument<double>(0.99, "snla-fraction-long")
```

#### 4.38.3.16 SNla\_LongScale

```
Argument<double> YieldValues::SNla_LongScale = Argument<double>(1.5, "snla-long-decay")
```

#### 4.38.3.17 SNla\_ShortScale

```
Argument<double> YieldValues::SNla_ShortScale = Argument<double>(0.1, "snla-short-decay")
```

#### 4.38.3.18 SNla\_TypicalMass

```
Argument<double> YieldValues::SNla_TypicalMass = Argument<double>(1.37, "snla-progenitor-mass")
```

#### 4.38.3.19 TargetNi56Yield

```
Argument<double> YieldValues::TargetNi56Yield = Argument<double>(0.1, "ideal-ni56")
```

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

- /Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/[ParameterLists.h](#)



## Chapter 5

# File Documentation

### 5.1 /Users/jf20/Documents/Physics/RAMICES\_II/src/Galaxy/Galaxy.h File Reference

```
#include "../Parameters/InitialisedData.h"
#include "Ring.h"
#include "../Gas/GasReservoir.h"
#include "../Stars/IMF.h"
#include "../Stars/SLF.h"
#include "MigrationMatrix.h"
#include <sstream>
#include <iomanip>
#include <thread>
#include <future>
```

#### Data Structures

- class [Galaxy](#)

#### Enumerations

- enum [ParallelJob](#) {  
    [RingStep](#) , [Compounding](#) , [Scattering](#) , [AssignIsochrones](#) ,  
    [Synthesis](#) , [Selection](#) }

#### 5.1.1 Enumeration Type Documentation

##### 5.1.1.1 ParallelJob

```
enum ParallelJob
```

## Enumerator

RingStep	
Compounding	
Scattering	
AssignIsochrones	
Synthesis	
Selection	

## 5.2 Galaxy.h

[Go to the documentation of this file.](#)

```

1 #pragma once
2 #include "../Parameters/InitialisedData.h"
3 #include "Ring.h"
4 #include "../Gas/GasReservoir.h"
5 #include "../Stars/IMF.h"
6 #include "../Stars/SLF.h"
7 #include "MigrationMatrix.h"
8 #include <sstream>
9 #include <iomanip>
10 #include <thread>
11 #include <future>
12
13 enum ParallelJob {RingStep, Compounding, Scattering, AssignIsochrones, Synthesis, Selection};
14
15 class Galaxy
16 {
17     public:
18         Galaxy(InitialisedData & Data);
19         void Evolve();
20         void SynthesiseObservations();
21         std::vector<Ring> Rings;
22     private:
23
24         std::vector<std::thread> Threads;
25         std::vector<MigrationMatrix> Migrator;
26         GasReservoir CGM;
27         const GlobalParameters & Param;
28
29         void LaunchParallelOperation(int time, int nOperations, ParallelJob type);
30
31         //Infall Stuff
32         double GasScaleLength(double t);
33         double InfallMass(double t);
34         void InsertInfallingGas(int ring, double amount);
35         void Infall(double t);
36
37         //Star Formation
38
39         void RingEvolve(int timestep, int ringStart, int ringEnd);
40         void ScatterYields(int timestep, int ringStart, int ringEnd);
41         void ScatterGas(int timestep);
42
43
44         void ComputeScattering(int t);
45         void CompoundScattering(int currentTime, int timeStart, int timeEnd);
46
47         void AssignMagnitudes(int time, int ringstart, int ringend);
48
49         double PredictSurfaceDensity(double radius, double width, double totalGasMass, double
scalelength);
50         double GasMass();
51         double ColdGasMass();
52         double StarMass();
53         void CGMOperations();
54
55         double RelicMass();
56         double Mass();
57         void SaveState(double t);
58         void SaveState_Mass(double t);
59         void SaveState_Enrichment(double t);
60         void SaveState_Events(double t);
61

```

```

62     static std::string MassHeaders();
63
64     InitialisedData & Data;
65
66     std::vector<double> RingMasses;
67
68     void ComputeVisibilityFunction();
69     void SelectionFunction(int ringstart, int ringend, int threadID);
70     void StellarSynthesis(int ringstart, int ringend, int threadID);
71     std::vector<std::string> SynthesisOutput;
72     std::vector<double> SynthesisProgress;
73
74     double DimmestStar;
75     double BrightestStar;
76     int ParallelBars = 0;
77
78 };

```

## 5.3 /Users/jf20/Documents/Physics/RAMICES\_II/src/Galaxy/MigrationMatrix.h File Reference

```

#include <vector>
#include "../Parameters/InitialisedData.h"
#include <iomanip>

```

### Data Structures

- class [MigrationMatrix](#)

## 5.4 MigrationMatrix.h

[Go to the documentation of this file.](#)

```

1 #pragma once
2 #include <vector>
3 #include "../Parameters/InitialisedData.h"
4 #include <iomanip>
5 class MigrationMatrix
6 {
7     public:
8         MigrationMatrix(InitialisedData & Data);
9
10        std::vector<std::vector<double>> Grid;
11
12        void Create(const std::vector<double> & masses);
13
14        void Compound(const MigrationMatrix & newTime);
15        void Print();
16    private:
17        int NRings;
18        const GlobalParameters & Param;
19        std::vector<std::vector<double>> DiagonalMultiply(const std::vector<std::vector<double>> & a, const
std::vector<std::vector<double>> & b, int diagonalDistance);
20
21 };

```

## 5.5 /Users/jf20/Documents/Physics/RAMICES\_II/src/Galaxy/Ring.h File Reference

```
#include <vector>
#include <iomanip>
#include "../Parameters/InitialisedData.h"
#include "../Gas/GasReservoir.h"
#include "../Stars/StarReservoir.h"
#include "../Stars/IMF.h"
#include "../Stars/SLF.h"
```

### Data Structures

- class [Ring](#)

## 5.6 Ring.h

[Go to the documentation of this file.](#)

```
1 #pragma once
2 #include <vector>
3 #include <iomanip>
4 #include "../Parameters/InitialisedData.h"
5 #include "../Gas/GasReservoir.h"
6 #include "../Stars/StarReservoir.h"
7 #include "../Stars/IMF.h"
8 #include "../Stars/SLF.h"
9 class Ring
10 {
11     public:
12         Ring(int radiusIndex, double mass, InitialisedData & data);
13
14         double Mass();
15         const double Radius;
16         const double Width;
17         double Area;
18         //Relic Reservoir
19         StarReservoir Stars;
20         GasReservoir Gas;
21         GasReservoir CGMBuffer;
22         void MakeStars();
23         void KillStars(int time);
24         void Cool();
25         void TimeStep(int t);
26         void UpdateMemory(int t);
27
28         void SaveChemicalHistory(int t, std::stringstream & absoluteStreamCold, std::stringstream &
29         logarithmicStreamCold, std::stringstream & absoluteStreamHot, std::stringstream &
30         logarithmicStreamHot);
31
32         double SelectionEffect(double Mv, double age);
33
34         void ComputeSelectionFunction(const double brightLimit, const double dimLimit);
35         void MetCheck(const std::string & location);
36         std::string Synthesis(const StellarPopulation & targetPopulation, double migrationFraction,
37         double originRadius, double & totalSynthesised);
38     private:
39         //~ std::vector<GasReservoir> PreviousEnrichment;
40         int RadiusIndex;
41
42
43         InitialisedData & Data;
44         const GlobalParameters & Param;
45
46         std::vector<std::vector<double>> ColdBuffer;
47         std::vector<std::vector<double>> HotBuffer;
48
49         std::vector<std::vector<double>> SelectionGrid;
50         double MinMv;
51         double MaxMv;
52 };
53 ;;
```



## 5.7 /Users/jf20/Documents/Physics/RAMICES\_II/src/Gas/Gas.h File Reference

```
#include "../Parameters/GlobalParameters.h"
```

### Data Structures

- class [Gas](#)

## 5.8 Gas.h

[Go to the documentation of this file.](#)

```
1 #pragma once
2 #include "../Parameters/GlobalParameters.h"
3
4
5
6
7 class Gas
8 {
9     public:
10
11
12     Gas();
13
14     Gas(const std::vector<double> & elements);
15
16     double Mass();
17
18     double Mass() const;
19
20     double & operator[] (ElementID id);
21
22     const double & operator[] (ElementID id) const;
23
24     static Gas Primordial(double mass);
25
26     static Gas Empty();
27
28     private:
29     std::vector<double> Species;
30
31     void CheckMass();
32
33     bool NeedsRecomputing;
34     double internal_Mass;
35 };
36
37
```

## 5.9 /Users/jf20/Documents/Physics/RAMICES\_II/src/Gas/GasReservoir.h File Reference

```
#include <vector>
#include "../Parameters/GlobalParameters.h"
#include "GasStream.h"
#include <sstream>
```

### Data Structures

- class [GasReservoir](#)

## 5.10 GasReservoir.h

[Go to the documentation of this file.](#)

```

1  #pragma once
2  #include <vector>
3  #include "../Parameters/GlobalParameters.h"
4  #include "GasStream.h"
5  #include <sstream>
12 class GasReservoir
13 {
14     public:
15
16         GasReservoir();
17
18         GasReservoir(const GlobalParameters & param);
19
20         GasStream & operator[] (SourceProcess source);
21
22         const GasStream & operator[] (SourceProcess source) const;
23
24
25         double Mass();
26
27         double ColdMass();
28
29         double HotMass();
30
31         void Absorb(const GasReservoir & givingGas);
32
33
34         void Absorb(const GasStream & givingGas);
35
36         void Absorb(const GasStream & givingGas, double fraction);
37
38         void Absorb(const std::vector<GasStream> & givingGas);
39
40         void Absorb(const std::vector<GasStream> & givingGas, double fraction);
41
42         void AbsorbMemory(int t, const GasStream & input);
43
44         void Deplete(double amountToLose);
45
46
47         void Wipe();
48
49         void Deplete(double amountToLose_Cold, double amountToLose_Hot);
50
51         void Heat(double amoutToHeat);
52
53         void PassiveCool(double dt, bool isCGM);
54
55         void TransferFrom(GasReservoir & givingGas, double massToMove);
56
57         void TransferColdFrom(GasReservoir & givingGas, double massToMove);
58
59         void TransferHotFrom(GasReservoir & givingGas, double massToMove);
60
61         void TransferAndHeat(GasReservoir & givingGas, double massToMove);
62         GasStream AccretionStream(double amountToLose);
63
64         static GasReservoir Primordial(double mass, const GlobalParameters & param);
65
66         double ColdGasMetallicity() const;
67
68         const std::vector<GasStream> & Composition() const;
69     private:
70
71         std::vector<GasStream> Components;
72
73
74         const GlobalParameters & Param;
75
76 };
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

## 5.11 /Users/jf20/Documents/Physics/RAMICES\_II/src/Gas/GasStream.h File Reference

```
#include "../Parameters/GlobalParameters.h"
#include "Gas.h"
```

### Data Structures

- class [GasStream](#)

## 5.12 GasStream.h

[Go to the documentation of this file.](#)

```
1 #pragma once
2 #include "../Parameters/GlobalParameters.h"
3 #include "Gas.h"
4
5 class GasStream
6 {
7     public:
8         SourceProcess Source;
9
10         const Gas & Hot() const;
11         const Gas & Cold() const;
12
13         double & Hot(ElementID el);
14         double & Cold(ElementID el);
15
16         const double & Hot(ElementID el) const;
17         const double & Cold(ElementID el) const;
18
19         GasStream();
20         GasStream(SourceProcess source);
21         GasStream(SourceProcess source, const Gas & hot, const Gas & cold);
22         GasStream(SourceProcess source, const Gas & gas, double hotFraction);
23
24         double Mass();
25
26         double Mass() const;
27         double HotMass();
28
29         double ColdMass();
30
31         double HotMass() const;
32         double ColdMass() const;
33
34         void Deplete(double amountToRemove);
35
36         void Heat(double amountToHeat);
37
38         void Cool(double amountToCool);
39         //~ void DepleteFraction(double fraction)
40         //~ {
41             //~ Deplete(fraction* Mass());
42         //~ }
43
44         void Deplete(double amountToRemove_Cold, double amountToRemove_Hot);
45
46         void Absorb(const GasStream & input);
47         void Absorb(const GasStream & input, double fraction);
48         void Absorb(const Gas & input, double hotFraction);
49         void Dirty();
50
51     private:
```

```

82     bool NeedsRecomputing;
83
84     double internal_HotMass;
85
86     double internal_ColdMass;
87
88     double internal_TotalMass;
89
90     void ComputeMasses();
91
92     Gas internal_Hot;
93
94     Gas internal_Cold;
95 };
96
97
98
99
100
101
102

```

## 5.13 /Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/EnumSets.h File Reference

### Enumerations

- enum [ElementID](#) {  
[Hydrogen](#) , [Helium](#) , [Metals](#) , [Iron](#) ,  
[Oxygen](#) , [Magnesium](#) , [Carbon](#) , [Silicon](#) ,  
[Calcium](#) , [Manganese](#) , [Chromium](#) , [Cobalt](#) ,  
[Europium](#) , [ElementCount](#) }
- enum [SourceProcess](#) { [Accreted](#) , [Stellar](#) , [Remnant](#) , [ProcessCount](#) }  
*Defines a globally recognised set of aliases for sources of gas, used to label [GasStream](#) objects.*
- enum [YieldProcess](#) {  
[CCSN](#) , [ECSN](#) , [SNIa](#) , [NSM](#) ,  
[AGB](#) , [YieldCount](#) }
- enum [RemnantType](#) {  
[DormantDwarf](#) , [CODwarf](#) , [NeutronStar](#) , [DormantNS](#) ,  
[MergerNS](#) , [BlackHole](#) }
- enum [SourceID](#) {  
[Orfeo](#) , [Marigo](#) , [Limongi](#) , [Maeder](#) ,  
[Mixed](#) , [Unknown](#) , [SourceCount](#) }  
*Enums to identify the theoretical basis for different yield tables.*

### 5.13.1 Enumeration Type Documentation

#### 5.13.1.1 ElementID

```
enum ElementID
```

Defines a globally recognised ordering of the elements + provides them with a nice readable name.

#### Enumerator

Hydrogen	
Helium	
Metals	

## Enumerator

Iron	
Oxygen	
Magnesium	
Carbon	
Silicon	
Calcium	
Manganese	
Chromium	
Cobalt	
Europium	
ElementCount	The final entry should always be ElementCount, as the numbering system inherent in enums makes this true only if it is the last entry!

## 5.13.1.2 RemnantType

```
enum RemnantType
```

## Enumerator

DormantDwarf	
CODwarf	
NeutronStar	
DormantNS	
MergerNS	
BlackHole	

## 5.13.1.3 SourceID

```
enum SourceID
```

Enums to identify the theoretical basis for different yield tables.

## Enumerator

Orfeo	
Marigo	
Limongi	
Maeder	
Mixed	
Unknown	
SourceCount	

### 5.13.1.4 SourceProcess

enum [SourceProcess](#)

Defines a globally recognised set of aliases for sources of gas, used to label [GasStream](#) objects.

#### Enumerator

Accreted	
Stellar	
Remnant	
ProcessCount	As with ElementID, final entry is used to count the number of elements. This must always be the final entry.

### 5.13.1.5 YieldProcess

enum [YieldProcess](#)

#### Enumerator

CCSN	
ECSN	
SN Ia	
NSM	
AGB	
YieldCount	

## 5.14 EnumSets.h

[Go to the documentation of this file.](#)

```

1 #pragma once
2
3
4
5
6
7 enum ElementID
8     {Hydrogen, Helium, Metals, Iron, Oxygen, Magnesium, Carbon, Silicon, Calcium, Manganese, Chromium, Cobalt, Europium,
9
10     ElementCount
11     };
12
13
14 enum SourceProcess {Accreted, Stellar, Remnant,
15
16     ProcessCount
17     };
18
19
20 enum YieldProcess {CCSN, ECSN, SN Ia, NSM, AGB, YieldCount};
21
22 enum RemnantType {DormantDwarf, CODwarf, NeutronStar, DormantNS, MergerNS, BlackHole};
23
24
25 enum SourceID {Orfeo, Marigo, Limongi, Maeder, Mixed, Unknown, SourceCount};

```

## 5.15 /Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/GlobalParameters.h File Reference

```
#include <string>
#include <vector>
#include "JSL.h"
#include <sstream>
#include "ParameterLists.h"
#include "List.h"
#include "EnumSets.h"
```

### Data Structures

- class [GlobalParameters](#)

*A package of global parameter objects which can be passed around by reference. Most of the internal values are set as JSL::Argument objects and so can be initialised from the command line / config file.*

### Macros

- #define [PI](#) 3.14159265358979323846

#### 5.15.1 Macro Definition Documentation

##### 5.15.1.1 PI

```
#define PI 3.14159265358979323846
```

## 5.16 GlobalParameters.h

[Go to the documentation of this file.](#)

```
1 #pragma once
2 #include <string>
3 #include <vector>
4 #include "JSL.h"
5 #include <sstream>
6 #include "ParameterLists.h"
7 #include "List.h"
8 #include "EnumSets.h"
9
10 #define PI 3.14159265358979323846
11
12 //Options classes are sets of initialisations of Argument objects, such that they can be separated &
    categorised
13
14
15
17 class GlobalParameters
18 {
19
20     public:
```

```

22     MetaValues Meta;
23
24     OutputValues Output;
25
26     ResourceValues Resources;
27
28     ElementValues Element;
29
30     StellarValues Stellar;
31
32     YieldValues Yield;
33
34     ThermalValues Thermal;
35
36     MigrationValues Migration;
37
38     CatalogueValues Catalogue;
39
40     GalaxyValues Galaxy;
41
42     std::vector<ParamList *> ParamMembers =
43     {&Meta,&Output,&Resources,&Element,&Stellar,&Thermal,&Galaxy,&Yield,&Migration,&Catalogue};
44
45     GlobalParameters();
46
47     void Initialise(int argc, char* argv[]);
48
49     void SaveInputs();
50
51 };

```

## 5.17 /Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/InitialisedData.h File Reference

```

#include <random>
#include "../Stars/IMF.h"
#include "../Stars/SLF.h"
#include "../Stars/IsochroneTracker.h"
#include "../Yields/YieldGrid.h"
#include "../Yields/SimpleYield.h"
#include "GlobalParameters.h"

```

### Data Structures

- class [InitialisedData](#)

*These will act like globally-defined functions, but have the scope for modifying themselves as they go along.*

## 5.18 InitialisedData.h

[Go to the documentation of this file.](#)

```

1 #pragma once
2 class YieldGrid;
3 class SimpleYield;
4 #include <random>
5 #include "../Stars/IMF.h"
6 #include "../Stars/SLF.h"
7 #include "../Stars/IsochroneTracker.h"
8 #include "../Yields/YieldGrid.h"
9 #include "../Yields/SimpleYield.h"
10 #include "GlobalParameters.h"
11
12 class InitialisedData

```



```

14 {
15     public:
16         const IMF_Functor IMF;
17         SLF_Functor SLF;
18         const GlobalParameters & Param;
19         const YieldGrid CCSNYield;
20         const YieldGrid AGBYield;
21         const YieldGrid ECSNYield;
22         const SimpleYield SNiAYield;
23         const SimpleYield NSMYield;
24         IsochroneTracker Isochrones;
25         InitialisedData(const GlobalParameters & param);
26
27         void Log(const std::string & input) const;
28         void Log(const std::string & input, int importance) const;
29         void LogFlush() const;
30         void UrgentLog(const std::string & input) const;
31
32         void ProgressBar(int & currentBars, int currentStep, int totalSteps);
33         double NormalDist();
34         double NormalDist(double mu, double sigma);
35         double UniformDist(double lowerBound, double upperBound);
36     private:
37         std::default_random_engine generator;
38         std::normal_distribution<double> distribution;
39
40
41 };

```

## 5.19 /Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/List.h File Reference

```

#include "JSL.h"
#include <vector>
#include <string>
#include <sstream>

```

### Data Structures

- class [ParamList](#)

*A Generic superclass structure so that I can heterogenously loop over the various members of [GlobalParameters](#) without writing it all out arduously. Also provides a consistent interface with the [JSL::Argument](#) environment.*

## 5.20 List.h

[Go to the documentation of this file.](#)

```

1 #pragma once
2 #include "JSL.h"
3 #include <vector>
4 #include <string>
5 #include <sstream>
6 using JSL::Argument;
7
9 class ParamList
10 {
11     public:
12         void Configure(int argc, char * argv[]);
13
14         void virtual Initialise(std::string resourceRoot){};
15
16         void StreamContentsTo(std::stringstream & stream);
17     protected:
18
19         std::vector<JSL::ArgumentInterface *> argPointers;
20
21
22
23
24 };

```

## 5.21 /Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/↵ ParameterLists.h File Reference

```
#include "JSL.h"
#include "List.h"
#include "EnumSets.h"
```

### Data Structures

- class [MetaValues](#)  
*The [MetaValues](#) contains variables associated with the base-level information about the simulation - the number of cores to access, the timesteps etc.*
- class [OutputValues](#)
- class [ResourceValues](#)
- class [ElementValues](#)  
*The elemental suboptions contains variables and data associated with the solar abundances (and where to locate them), and how to extract and extrapolate the yield data from files.*
- class [StellarValues](#)  
*The subset of values associated with stars + their remnants.*
- class [YieldValues](#)
- class [ThermalValues](#)  
*Thermal suboptions contain variables which deal with the thermal subroutines - cooling timescales injection fractions etc.*
- class [MigrationValues](#)  
*Holds values associated with how matter moves throughout the disc.*
- class [CatalogueValues](#)
- class [GalaxyValues](#)  
*The galaxy suboptions contains variables associated with the galaxy as a whole, such as the maximum radius, and various mass/infall properties.*

## 5.22 ParameterLists.h

[Go to the documentation of this file.](#)

```
1 #pragma once
2 #include "JSL.h"
3 #include "List.h"
4 #include "EnumSets.h"
5 using JSL::Argument;
6
7
9 class MetaValues : public ParamList
10 {
11
12     public:
14         Argument<int> Verbosity = Argument<int>(1, "verbose");
15
17         Argument<int> ParallelThreads = Argument<int>(3, "thread");
18
20         Argument<double> TimeStep = Argument<double>(0.01, "timestep");
21
22         //~ //!The total duration of the chemical simulation
23         Argument<double> SimulationDuration = Argument<double>(10.0, "duration");
24
26         Argument<int> ProgressHashes = Argument<int>(32, "progress-hashes");
27
28
30         int SimulationSteps;
```

```

31
32
33     MetaValues()
34     {
35         argPointers = {&Verbosity,&ParallelThreads,&TimeStep,&SimulationDuration,&ProgressHashes};
36     };
37
38     virtual void Initialise(std::string resourceRoot);
39 };
40
41
42
43 class OutputValues : public ParamList
44 {
45     public:
46         Argument<std::string> Root = Argument<std::string>("Output/", "output");
47
48         Argument<std::string> Config = Argument<std::string>("rerun.config", "config-out");
49
50         Argument<std::string> YieldSubdir = Argument<std::string>("Yields/", "yield-dir");
51
52         Argument<std::string> GalaxyMassFile = Argument<std::string>("Mass.dat", "galaxy-mass-file");
53
54         Argument<std::string> EventRateFile = Argument<std::string>("Events.dat", "event-rate-file");
55
56         Argument<std::string> StarFile = Argument<std::string>("StellarCatalogue.dat", "ring-data-stars");
57
58         Argument<std::string> ChemicalPrefactor =
59         Argument<std::string>("Enrichment_", "enrichment-base");
60
61         Argument<std::string> ColdGasDataFile= Argument<std::string>("ColdGas.dat", "enrichment-cold");
62
63         Argument<std::string> HotGasDataFile= Argument<std::string>("HotGas.dat", "enrichment-hot");
64
65         std::string LogarithmicColdGasFile;
66         std::string AbsoluteColdGasFile;
67         std::string LogarithmicHotGasFile;
68         std::string AbsoluteHotGasFile;
69
70         OutputValues()
71         {
72             argPointers = {&Root, &GalaxyMassFile,&StarFile,
73             &ChemicalPrefactor,&ColdGasDataFile,&HotGasDataFile,&YieldSubdir};
74         };
75
76         virtual void Initialise(std::string resourceRoot);
77 };
78
79 class ResourceValues : public ParamList
80 {
81     public:
82         Argument<std::string> WelcomeFile = Argument<std::string>("welcome.dat", "welcome-file");
83
84         Argument<std::string> ResourceRoot = Argument<std::string>("Resources/", "resource");
85
86         Argument<std::string> YieldRoot = Argument<std::string>("ChemicalData/", "yield-root");
87
88         Argument<std::string> IsochroneDirectory = Argument<std::string>("Isochrones/", "iso-dir");
89
90         Argument<std::string> LifeTimeFile = Argument<std::string>("LifetimeGrid.dat", "lifetime-file");
91
92         Argument<std::string> IsochroneRepository = Argument<std::string>("NewPadova/", "iso-repo");
93
94         ResourceValues()
95         {
96             argPointers = {&WelcomeFile,
97             &ResourceRoot,&YieldRoot,&IsochroneDirectory,&LifeTimeFile,&IsochroneRepository};
98         };
99
100         virtual void Initialise(std::string resourceRoot);
101 };
102
103 class ElementValues : public ParamList
104 {
105     public:
106         std::vector<std::string> ElementNames;
107
108         // The values of Z associated with each element
109         std::vector<int> ProtonCounts;
110
111         std::vector<double> SolarAbundances;
112
113         Argument<std::string> SolarAbundanceFile =
114         Argument<std::string>("ChemicalData/SolarAbundances_Maria.dat", "solar-values-file");

```

```

134
136     Argument<int> SolarAbundanceFileNameColumn = Argument<int>(0, "solar-values-name-col");
137
139     Argument<int> SolarAbundanceFileDataColumn = Argument<int>(3, "solar-values-data-col");
140
141
142
144     ElementValues()
145     {
146         argPointers = {&SolarAbundanceFile, &SolarAbundanceFileDataColumn,
&SolarAbundanceFileNameColumn};
147     };
148
150     virtual void Initialise(std::string resourceRoot);
151
153     void GiveElementsNames();
154
155 };
156
158 class StellarValues : public ParamList
159 {
160
161     public:
163     Argument<double> MaxStellarMass = Argument<double>(100, "mass-max");
164
166     Argument<double> MinStellarMass = Argument<double>(0.1, "mass-min");
167
169     Argument<double> ImmortalMass = Argument<double>(0.3, "mass-immortal");
170
172     Argument<int> MassResolution = Argument<int>(199, "mass-resolution");
173
175     std::vector<double> MassGrid;
176
178     std::vector<double> MassDeltas;
179
180
181
183     Argument<double> MinLogZ = Argument<double>(-6, "logz-min");
184
186     Argument<double> MaxLogZ = Argument<double>(-0.1, "logz-max");
187
189     Argument<int> LogZResolution = Argument<int>(100, "logz-resolution");
190
192     std::vector<double> LogZGrid;
193     double LogZDelta;
194
196     Argument<double> EjectionFraction = Argument<double>(0.45, "eject");
197
199     Argument<double> FeedbackFactor = Argument<double>(0.5, "mass-load");
200
201
202
204     Argument<double> SchmidtMainPower = Argument<double>(1.4, "schmidt-main");
205
207     Argument<double> SchmidtLowPower = Argument<double>(4.0, "schmidt-low");
208
210     Argument<double> SchmidtDensityCut = Argument<double>(0, "schmidt-cut");
211
213     Argument<double> SchmidtPrefactor = Argument<double>(2, "schmidt-factor");
214
216     Argument<double> IMF_Slope = Argument<double>(2.3, "imf-slope");
217
218
219
221     StellarValues()
222     {
223         argPointers = {&MaxStellarMass, &MinStellarMass, &ImmortalMass, &MassResolution, &MinLogZ,
&MaxLogZ, &LogZResolution, &EjectionFraction, &SchmidtMainPower, &SchmidtLowPower, &SchmidtDensityCut,
&SchmidtPrefactor, &FeedbackFactor};
224     }
225
227     void Initialise(std::string resourceRoot);
228 };
229
230
231 class YieldValues : public ParamList
232 {
233     public:
234     Argument<double> TargetNi56Yield = Argument<double>(0.1, "ideal-ni56");
235     Argument<double> MassOverhang = Argument<double>(5, "yield-mass-overhang");
236     std::vector<std::string> ProcessNames;
237     std::vector<SourceProcess> ProcessTypes;
238
240     Argument<double> SNiA_DelayTime = Argument<double>(0.2, "snia-delay");
241
242     Argument<double> SNiA_ActiveFraction = Argument<double>(0.1, "snia-fraction");
243

```

```

244     Argument<double> SNiA_LongFraction = Argument<double>(0.99,"snla-fraction-long");
245
246     Argument<double> SNiA_ShortScale = Argument<double>(0.1,"snla-short-decay");
247
248     Argument<double> SNiA_TypicalMass = Argument<double>(1.37,"snla-progenitor-mass");
249
250     Argument<double> NSM_TypicalMass = Argument<double>(1.4,"nsm-progenitor-mass");
251
252     Argument<double> SNiA_LongScale = Argument<double>(1.5,"snla-long-decay");
253
254     Argument<double> CCSN_MassCut = Argument<double>(10,"ccsn-mass");
255
256     Argument<double> ECSN_MassCut = Argument<double>(8.5,"ecsn-mass");
257     Argument<double> CODwarf_MassCut = Argument<double>(3.2,"co-mass");
258     Argument<double> Collapse_MassCut = Argument<double>(40,"bh-mass");
259
260     Argument<double> ECSN_Fraction = Argument<double>(0,"ecsn-fraction");
261
262     Argument<double> NSM_DelayTime = Argument<double>(0.02,"nsm-delay");
263     Argument<double> NSM_ActiveFraction = Argument<double>(0.001,"nsm-fraction");
264     Argument<double> NSM_Scale = Argument<double>(10,"nsm-decay");
265
266     YieldValues()
267     {
268     }
269     {
270         argPointers = {&SNiA_DelayTime, &SNiA_ShortScale, &SNiA_LongScale, &NSM_DelayTime, &
271             SNiA_ActiveFraction, &SNiA_LongFraction, &CCSN_MassCut, &NSM_ActiveFraction,
272             &NSM_Scale, &SNiA_TypicalMass, &NSM_TypicalMass, &TargetNi56Yield, &ECSN_Fraction};
273     }
274     void Initialise(std::string resourceRoot);
275 };
276
277 class ThermalValues : public ParamList
278 {
279
280
281     public:
282     Argument<double> HotInjection_CCSN = Argument<double>(0.7,"fh-ccsn");
283
284     Argument<double> HotInjection_NSM = Argument<double>(0.4,"fh-nsm");
285
286     Argument<double> HotInjection_AGB = Argument<double>(0.7,"fh-agb");
287
288     Argument<double> FeedbackEjectFactor = Argument<double>(0,"feedback-eject");
289
290     Argument<double> ChimneyFactor = Argument<double>(0,"chimney");
291
292     Argument<double> HotInjection_SNiA = Argument<double>(0.99,"fh-snla");
293
294     Argument<double> GasCoolingTimeScale = Argument<double>(1,"cool");
295
296     Argument<int> NumericalResolution = Argument<int>(30,"cool-resolution");
297
298     Argument<double> DormantHotFraction = Argument<double>(1e-20,"dormant-hot-frac");
299
300     Argument<double> CoolingPower = Argument<double>(1,"cooling-index");
301     ThermalValues()
302     {
303         argPointers = {&HotInjection_CCSN, &HotInjection_NSM, &HotInjection_SNiA,
304             &GasCoolingTimeScale, &HotInjection_AGB, &NumericalResolution, &DormantHotFraction, &CoolingPower,
305             &FeedbackEjectFactor, &ChimneyFactor};
306     }
307 };
308
309 class MigrationValues: public ParamList
310 {
311
312
313     public:
314     Argument<bool> InflowActive = Argument<bool>(true,"inflow-on");
315     Argument<double> InflowParameterA = Argument<double>(0.33,"inflow-a");
316
317     Argument<double> InflowParameterB = Argument<double>(0.53,"inflow-b");
318
319     Argument<double> MaxStealFraction = Argument<double>(0.95,"max-steal");
320
321     Argument<double> MarkovDispersionStrength = Argument<double>(0.2,"mixing-strength");
322
323     Argument<int> DispersionOrder = Argument<int>(3,"mixing-order");
324
325     Argument<double> DispersionTruncation = Argument<double>(1e-10,"mixing-truncation");
326
327     MigrationValues()
328     {
329         argPointers =
330         {&InflowParameterA, &InflowParameterB, &MaxStealFraction, &MarkovDispersionStrength, &DispersionOrder,
331             &DispersionTruncation, &InflowActive};
332     }
333 };

```

```

342     }
343
344
345 };
346
347 class CatalogueValues: public ParamList
348 {
349     public:
350
351         Argument<bool> SynthesisActive = Argument<bool>(true, "stellar-synthesis");
352
353         Argument<double> IsochroneTimeStep = Argument<double>(0.1, "isochrone-dt");
354
355         Argument<double> IsochroneMagnitudeResolution =
356             Argument<double>(100, "isochrone-mag-resolution");
357
358         Argument<double> SolarRadius = Argument<double>(8.2, "solar-radius");
359
360         Argument<int> RadialResolution = Argument<int>(3, "isochrone-radial-resolution");
361         Argument<int> AzimuthalResolution = Argument<int>(360, "isochrone-radial-resolution");
362
363
364
365         Argument<double> VerticalHeightStart = Argument<double>(0.05, "vertical-height-z0");
366         Argument<double> VerticalHeightScaling = Argument<double>(0.3, "vertical-height-scaling");
367         Argument<double> VerticalHeightPower = Argument<double>(0.66, "vertical-height-power");
368
369
370         Argument<int> SampleCount = Argument<int>(10, "catalogue-sample");
371
372         CatalogueValues()
373         {
374             argPointers =
375                 {&SynthesisActive, &IsochroneTimeStep, &IsochroneMagnitudeResolution, &SolarRadius, &RadialResolution, &AzimuthalResolution,
376                 };
377
378 };
379
380 class GalaxyValues : public ParamList
381 {
382     public:
383
384         Argument<int> RingCount = Argument<int>(100, "rings");
385
386
387         Argument<double> Radius = Argument<double>(20.0, "radius");
388
389         Argument<bool> UsingVariableRingWidth = Argument<bool>(false, "variable-ring-width");
390
391         Argument<double> Ring0Width = Argument<double>(0.05, "inner-ring-width");
392
393         Argument<bool> CGMAbsorbing = Argument<bool>(true, "cgm-absorb");
394
395         std::vector<double> RingRadius;
396         std::vector<double> RingWidth;
397
398         Argument<double> PrimordialMass = Argument<double>(2, "M0");
399
400         Argument<double> PrimordialHotFraction = Argument<double>(0, "primordial-hot");
401
402         Argument<double> CGM_Mass = Argument<double>(200, "cgm-mass");
403
404         Argument<double> MinScaleLength = Argument<double>(0.75, "scale-length-min");
405         Argument<double> MaxScaleLength = Argument<double>(3.75, "scale-length-max");
406
407         Argument<double> ScaleLengthDelay = Argument<double>(1.0, "scale-length-delay");
408         Argument<double> ScaleLengthTimeScale = Argument<double>(2.0, "scale-length-time");
409         Argument<double> ScaleLengthFinalTime = Argument<double>(12.0, "scale-length-final");
410
411         Argument<double> InfallMass1 = Argument<double>(50, "M1");
412         Argument<double> InfallMass2 = Argument<double>(100, "M2");
413         Argument<double> InfallTime1 = Argument<double>(0.4, "b1");
414         Argument<double> InfallTime2 = Argument<double>(6.0, "b2");
415         Argument<double> InfallMassMerger = Argument<double>(0, "merger-mass");
416         Argument<double> InfallTimeMerger = Argument<double>(0.4, "merger-timescale");
417         Argument<double> MergerDelayTime = Argument<double>(8, "merger-delay");

```

```

445
446     Argument<double> MergerTurnOnWidth = Argument<double>(0.3, "merger-width");
447
448     Argument<double> MaxSFRFraction = Argument<double>(0.95, "max-sfr");
449
450     GalaxyValues()
451     {
452         argPointers = {&RingCount, &PrimordialMass, &PrimordialHotFraction, &CGM\_Mass, &Radius,
453             &MinScaleLength, &MaxScaleLength, &ScaleLengthDelay, &ScaleLengthTimeScale, &ScaleLengthFinalTime,
454             &InfallMass1, &InfallMass2, &InfallTime1, &InfallTime2, &MaxSFRFraction, &Ring0Width,
455             &UsingVariableRingWidth, &CGMAbsorbing, &InfallMassMerger, &InfallTimeMerger, &
456             &MergerDelayTime, &MergerTurnOnWidth};
457     }
458     void Initialise(std::string resourceRoot);
459 };

```

## 5.23 /Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/IMF.h File Reference

```
#include "../Parameters/GlobalParameters.h"
```

### Data Structures

- struct [Integral](#)
- class [IMF\\_Functor](#)

## 5.24 IMF.h

[Go to the documentation of this file.](#)

```

1 #pragma once
2 #include "../Parameters/GlobalParameters.h"
3
4
5
6 struct Integral
7 {
8     double ZerothMoment;
9     double FirstMoment;
10 };
11 class IMF\_Functor
12 {
13     public:
14         IMF\_Functor(const GlobalParameters & param);
15
16         double operator() (double mass);
17
18         double FormationCount(double formationMass) const;
19         double Weighting(int i) const;
20
21     private:
22         const GlobalParameters & Param;
23
24         double IMF\_Normalisation;
25         double IMF\_MeanMass;
26         Integral MomentCompute(double start, double stop, int resolution);
27         void Normalise();
28         double IMF(double mass);
29         std::vector<double> IMF\_Weighting;
30 };
31

```

## 5.25 /Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/IsochroneTracker.h File Reference

```
#include "../Parameters/GlobalParameters.h"
#include <filesystem>
#include <random>
```

### Data Structures

- struct [IsochroneEntry](#)
- struct [IsochroneCube](#)
- class [IsochroneTracker](#)

### Enumerations

- enum [IsochroneProperties](#) {  
[logL](#) , [BolometricMag](#) , [UMag](#) , [BMag](#) ,  
[VMag](#) , [RMag](#) , [IMag](#) , [JMag](#) ,  
[HMag](#) , [KMag](#) , [TEff](#) , [Logg](#) ,  
[PropertyCount](#) }

### Variables

- const std::vector< std::string > [PropertyNames](#) = {"logL", "[BolometricMag](#)", "[UMag](#)", "[BMag](#)", "[VMag](#)", "[RMag](#)", "[IMag](#)", "[JMag](#)", "[HMag](#)", "[KMag](#)", "[TEff](#)", "[Logg](#)"}

#### 5.25.1 Enumeration Type Documentation

##### 5.25.1.1 IsochroneProperties

```
enum IsochroneProperties
```

##### Enumerator

logL	
BolometricMag	
UMag	
BMag	
VMag	
RMag	
IMag	
JMag	
HMag	
KMag	
TEff	
Logg	
PropertyCount	



## 5.25.2 Variable Documentation

### 5.25.2.1 PropertyNames

```
const std::vector<std::string> PropertyNames = {"logL", "BolometricMag", "UMag", "BMag", "VMag",
"RMag", "IMag", "JMag", "HMag", "KMag", "TEff", "Logg"}
```

## 5.26 IsochroneTracker.h

[Go to the documentation of this file.](#)

```
1 #pragma once
2 #include "../Parameters/GlobalParameters.h"
3 #include <filesystem>
4 #include <random>
5
6
7 enum IsochroneProperties {logL, BolometricMag, UMag, BMag, VMag, RMag, IMag, JMag, HMag, KMag, TEff, Logg,
    PropertyCount};
8 const std::vector<std::string> PropertyNames = {"logL", "BolometricMag", "UMag", "BMag", "VMag",
    "RMag", "IMag", "JMag", "HMag", "KMag", "TEff", "Logg"};
9 struct IsochroneEntry
10 {
11     std::vector<double> Properties;
12     IsochroneEntry()
13     {
14         Properties = std::vector<double>(PropertyCount, 999.0);
15     }
16     double & operator[] (IsochroneProperties p)
17     {
18         return Properties[p];
19     }
20     const double & operator[] (IsochroneProperties p) const
21     {
22         return Properties[p];
23     }
24     int Countify()
25     {
26         return Properties.size();
27     }
28 };
29 struct IsochroneCube
30 {
31     std::vector<double> Weighting;
32     std::vector<IsochroneEntry *> Data;
33     int Count() const
34     {
35         return Data.size();
36     }
37     double Value(int entry, IsochroneProperties p) const
38     {
39         return Data[entry]->Properties[p];
40     }
41 };
42
43 class IsochroneTracker
44 {
45
46
47 public:
48     IsochroneTracker(const GlobalParameters & param);
49     void Construct();
50
51     IsochroneCube GetProperties(int mass, double z, double age);
52
53 private:
54     const GlobalParameters Param;
55     void IsoLog(std::string val);
56     void ParseFile(std::string file);
57     std::vector<double> CapturedZs;
58     std::vector<double> CapturedTs;
59     std::vector<std::vector<std::vector<IsochroneEntry>>> Grid;
60     std::vector<std::vector<std::vector<IsochroneEntry>>> UnsortedGrid;
```

```

61     bool isTimeLogUniform;
62     double DeltaLogT;
63
64     std::default_random_engine generator;
65     std::normal_distribution<double> distribution;
66
67     double NormalSample(double mu, double sigma);
68     double UniformSample(double lowerBound, double upperBound);
69
70     void ExtractSample(IsochroneCube & output, int sampleMass, double sampleZ, double sampleAge);
71 };

```

## 5.27 /Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/RemnantPopulation.h File Reference

```

#include "../Yields/YieldGrid.h"
#include "../Yields/SimpleYield.h"
#include "../Parameters/InitialisedData.h"
#include "StarEvents.h"

```

### Data Structures

- struct [MassReport](#)
- class [RemnantPopulation](#)

## 5.28 RemnantPopulation.h

[Go to the documentation of this file.](#)

```

1 #pragma once
2 #include "../Yields/YieldGrid.h"
3 #include "../Yields/SimpleYield.h"
4 #include "../Parameters/InitialisedData.h"
5 #include "StarEvents.h"
6 struct MassReport
7 {
8     double Total;
9     double WD;
10    double NS;
11    double BH;
12 };
13
14 class RemnantPopulation
15 {
16 public:
17     RemnantPopulation(InitialisedData & data);
18
19
20     void Feed(int timeIndex, double bhMass, double wdMass, double nsMass);
21     void Feed(int timeIndex, RemnantOutput rem);
22     void Decay(int currentTime, std::vector<GasReservoir> & scatteringReservoir, StarEvents &
EventRate);
23     MassReport Mass();
24 private:
25     //the MASS of remnants created at each time
26     std::vector<double> ShortSNiABuffer;
27     std::vector<double> LongSNiABuffer;
28     std::vector<double> NSMBuffer;
29
30     const SimpleYield & SNIaYield;
31     const SimpleYield & NSMYield;
32     double BlackHoleMass;
33     double DormantWDMass;
34     double DormantNSMass;
35     const GlobalParameters & Param;
36 };

```

## 5.29 /Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/SLF.h File Reference

```
#include "../Parameters/GlobalParameters.h"
#include <algorithm>
```

### Data Structures

- class [SLF\\_Functor](#)

## 5.30 SLF.h

[Go to the documentation of this file.](#)

```
1 #pragma once
2 #include "../Parameters/GlobalParameters.h"
3 #include <algorithm>
4 class SLF_Functor
5 {
6     public:
7         SLF_Functor(const GlobalParameters & Param);
8
9         int operator() (int mass, double metallicity);
10        double PredictLifetime(double mass, double logmetallicity);
11    private:
12
13        double ValueInquiry(int m, int z);
14        const GlobalParameters & Param;
15
16        double LifeTime(int mass, int metallicity);
17        void PrecomputeGrid();
18        std::vector<std::vector<double>> PrecomputedGrid;
19        const double NotComputed = -1;
20
21 };
```

## 5.31 /Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/StarEvents.h File Reference

```
#include <sstream>
```

### Data Structures

- class [StarEvents](#)

## 5.32 StarEvents.h

[Go to the documentation of this file.](#)

```
1 #pragma once
2 #include <sstream>
3 class StarEvents
4 {
5     public:
6         double StarMassFormed;
7         double NStarsFormed;
8         double CCSN;
9         double AGBDeaths;
10        double NSM;
11        double SNIa;
12        double ECSN;
13        double Efficiency;
14        StarEvents();
15
16        void AddHeaders(std::stringstream & output);
17        void Save(std::stringstream & output, double timestep);
18 };
```

## 5.33 /Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/StarReservoir.h File Reference

```
#include <vector>
#include "../Parameters/InitialisedData.h"
#include "../Gas/GasReservoir.h"
#include "RemnantPopulation.h"
#include "IsochroneTracker.h"
#include "StellarPopulation.h"
#include <sstream>
#include "IMF.h"
#include "SLF.h"
#include "StarEvents.h"
```

### Data Structures

- class [StarReservoir](#)

## 5.34 StarReservoir.h

[Go to the documentation of this file.](#)

```
1 #pragma once
2
3 #include <vector>
4 #include "../Parameters/InitialisedData.h"
5 #include "../Gas/GasReservoir.h"
6 #include "RemnantPopulation.h"
7 #include "IsochroneTracker.h"
8 #include "StellarPopulation.h"
9 #include "IsochroneTracker.h"
10 #include <sstream>
11 #include "IMF.h"
12 #include "SLF.h"
13 #include "StarEvents.h"
14 class StarReservoir
15 {
16     public:
```

```

17     StarReservoir(int parentRing, InitialisedData & data);
18     std::vector<StellarPopulation> Population;
19     double AliveMass();
20     MassReport DeadMass();
21
22
23     void Observations();
24     void Form(GasReservoir & gas, GasReservoir & cgm);
25     void Death(int currentTime);
26     void PrintStatus(int t);
27     const std::vector<GasStream> & YieldsFrom(int t);
28     void SaveEventRate(int t, std::stringstream & output);
29     void AssignMagnitudes();
30     std::vector<GasReservoir> YieldOutput;
31 private:
32
33     RemnantPopulation Remnants;
34
35     double SFR_GasLoss(double coldMass, double hotMass, double ejectFactor);
36     const int ParentRing;
37     double ParentArea;
38     double Temp_Mass;
39     const IMF_Functor & IMF;
40     SLF_Functor & SLF;
41     int PopulationIndex;
42
43     std::vector<StarEvents> EventRate;
44
45
46
47     InitialisedData & Data;
48     const GlobalParameters & Param;
49 };

```

## 5.35 /Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/StellarPopulation.h File Reference

```

#include <vector>
#include "../Parameters/InitialisedData.h"
#include "IMF.h"
#include "../Gas/GasReservoir.h"
#include "RemnantPopulation.h"
#include "SLF.h"
#include "StarEvents.h"
#include "IsochroneTracker.h"

```

### Data Structures

- class [IsoMass](#)  
A simple struct for tracking the number of stars of a given mass.
- class [StellarPopulation](#)

## 5.36 StellarPopulation.h

[Go to the documentation of this file.](#)

```

1 #pragma once
2 #include <vector>
3
4 #include "../Parameters/InitialisedData.h"
5 #include "IMF.h"
6 #include "../Gas/GasReservoir.h"

```

```

7 #include "RemnantPopulation.h"
8 #include "SLF.h"
9 #include "StarEvents.h"
10 #include "IsochroneTracker.h"
11
12
13
14
15 class IsoMass
16 {
17     public:
18         int MassIndex;
19         double Count;
20         double Metallicity;
21         int BirthIndex;
22         int DeathIndex;
23         IsochroneCube Isochrone;
24         IsoMass();
25         IsoMass(double n, int m, double z, int birth, int death);
26 };
27
28
29 class StellarPopulation
30 {
31     public:
32         StellarPopulation(InitialisedData & data, int parentRing);
33
34         void PrepareIMF();
35         int BirthRadius;
36         double Metallicity;
37         int BirthIndex;
38
39         int FormStars(double formingMass, int timeIndex, GasReservoir & formingGas);
40         double Mass();
41         IsoMass & Relic();
42         const IsoMass & Relic() const;
43         IsoMass & operator[](int i);
44         const IsoMass & operator[](int i) const;
45         bool Active();
46
47         void Death(int time, std::vector<GasReservoir> & TemporalYieldGrid, RemnantPopulation & remnants,
48             StarEvents & EventRate);
49         std::vector<IsoMass> Distribution;
50         IsoMass ImmortalStars;
51
52         std::vector<GasStream> BirthGas;
53
54         std::string CatalogueHeaders();
55         std::string CatalogueEntry(std::vector<int> popEntry, int m, double currentRadius, double
56             birthRadius) const;
57         double Age;
58     private:
59         const GlobalParameters & Param;
60
61         const IMF_Functor & IMF;
62         SLF_Functor & SLF;
63         const YieldGrid & CCSNYield;
64         const YieldGrid & ECSNYield;
65         const YieldGrid & AGBYield;
66         InitialisedData & Data;
67         bool IsLifetimeMonotonic;
68         bool IsDepleted;
69         int DepletionIndex;
70
71         double internal_MassCounter;
72
73         void MonotonicDeathScan(int time, std::vector<GasReservoir> & YieldGrid, RemnantPopulation &
74             remnants, StarEvents & eventRate);
75         void FullDeathScan(int time);
76
77         void RecoverMatter(int time, int nstars, int mass, GasReservoir & temporalYieldGrid,
78             RemnantPopulation & remnants);
79         Gas TempGas;
80 };

```

## 5.37 /Users/jf20/Documents/Physics/RAMICES\_II/src/Yields/SimpleYield.h File Reference

```
#include "../Parameters/GlobalParameters.h"
#include "../Gas/GasReservoir.h"
```

### Data Structures

- class [SimpleYield](#)

## 5.38 SimpleYield.h

[Go to the documentation of this file.](#)

```
1 #pragma once
2 #include "../Parameters/GlobalParameters.h"
3 #include "../Gas/GasReservoir.h"
4
5 class SimpleYield
6 {
7     public:
8         const SourceProcess Process;
9         SimpleYield(const GlobalParameters & param, YieldProcess Process);
10
11         void operator()(GasReservoir & scatteringReservoir, double nObjects) const;
12
13     private:
14
15         double hotInjectionFraction;
16         void NSM_Initialise();
17         void SNIa_Initialise();
18
19         void RemnantInject( GasReservoir & scatteringReservoir, double Nstars) const;
20         std::vector<double> Grid;
21         const GlobalParameters & Param;
22 };
```

## 5.39 /Users/jf20/Documents/Physics/RAMICES\_II/src/Yields/YieldGrid.h File Reference

```
#include "../Parameters/GlobalParameters.h"
#include "../Gas/GasReservoir.h"
#include "YieldRidge.h"
```

### Data Structures

- struct [RemnantOutput](#)
- struct [Interpolator](#)
- class [YieldGrid](#)

## 5.40 YieldGrid.h

[Go to the documentation of this file.](#)

```

1 #pragma once
2
3 //~ #include "../Stars/RemnantPopulation.h"
4 #include "../Parameters/GlobalParameters.h"
5 #include "../Gas/GasReservoir.h"
6 #include "YieldRidge.h"
7 //~ #include <ostream>
8
9 struct RemnantOutput
10 {
11     RemnantType Type;
12     double Mass;
13 };
14 struct Interpolator
15 {
16     int UpperID;
17     int LowerID;
18     double LinearFactor;
19     double Interpolate(double lower, double upper)
20     {
21         double val = lower + LinearFactor*(upper - lower);
22         if (val < 0 && (lower > 0 || upper > 0))
23         {
24             val = 0;
25         }
26         return val;
27     }
28 };
29
30 class YieldGrid
31 {
32 public:
33     const SourceProcess Process;
34     YieldGrid(const GlobalParameters & param, YieldProcess Process);
35
36     RemnantOutput operator()(GasReservoir & scatteringReservoir, double Nstars, int mass, double z,
37     const std::vector<GasStream> & birthGas) const;
38 private:
39     const GlobalParameters & Param;
40     std::vector<std::vector<std::vector<double>>> Grid;
41
42     double hotInjectionFraction;
43
44     void CCSN_Initialise();
45
46     void AGB_Initialise();
47     void ECSN_Initialise();
48     void InitialiseLargeGrid(int mSize, int zSize);
49     //~ GasStream TempStream;
50     //~allows the grid size to be truncated for CCSN etc.
51     int MassOffset;
52     RemnantOutput StellarInject( GasReservoir & scatteringReservoir, double Nstars, int mass, double
53     z, const std::vector<GasStream> & birthGas) const;
54
55     void LoadOrfeoYields();
56     void LoadMarigoYields();
57     void LoadLimongiYields();
58     void LoadMaederYields();
59     std::vector<std::vector<YieldRidge> RidgeStorage;
60     int RemnantLocation;
61
62     Interpolator MetallicityInterpolation(double z) const;
63
64     double ElementProduction(ElementID element, double synthesisFraction, double
65     ejectaMass, std::vector<GasStream> & output, const std::vector<GasStream> & birthStreams) const;
66     void ElementDestruction(ElementID element, double synthesisFraction, double ejectaMass,
67     std::vector<GasStream> & output, const std::vector<GasStream> & birthStreams) const;
68
69     // Creation properties
70
71     void CreateGrid();
72     YieldBracket GetBracket(int id, double mass, double z, bool overhang);
73     //~ std::vector<int> SourcePriority;
74     std::vector<std::vector<int> SourcePriority;
75     void SaveGrid(std::string name);
76     void PurityEnforce();
77 };

```



## 5.41 /Users/jf20/Documents/Physics/RAMICES\_II/src/Yields/YieldRidge.h File Reference

```
#include "../Parameters/GlobalParameters.h"
```

### Data Structures

- struct [YieldPoint](#)
- class [YieldRidge](#)
- struct [YieldBracket](#)

## 5.42 YieldRidge.h

[Go to the documentation of this file.](#)

```
1 #pragma once
2
3 #include "../Parameters/GlobalParameters.h"
4 struct YieldPoint
5 {
6     double Mass;
7     double Yield;
8     YieldPoint(double m, double y): Mass(m), Yield(y)
9     {
10
11     }
12     YieldPoint()
13     {
14         YieldPoint(0,0);
15     }
16 };
17
18
19
20 class YieldRidge
21 {
22     public:
23         SourceID Source;
24         double Z;
25         std::vector<YieldPoint> Points;
26         YieldRidge();
27         YieldRidge(SourceID source, double z, int nPoints);
28
29 };
30
31 struct YieldBracket
32 {
33     bool isEnclosed;
34     bool hasSingle;
35     YieldRidge UpperRidge;
36     YieldRidge LowerRidge;
37
38     YieldBracket()
39     {
40         isEnclosed = false;
41         hasSingle = false;
42     }
43     YieldBracket(YieldRidge r1, YieldRidge r2): UpperRidge(r1), LowerRidge(r2)
44     {
45         isEnclosed = true;
46         hasSingle = false;
47     }
48     YieldBracket(YieldRidge r1): UpperRidge(r1), LowerRidge(r1)
49     {
50         isEnclosed = true;
51         hasSingle = true;
52     }
53
54     double Interpolate(double mass, double z);
55 };
```



# Index

/Users/jf20/Documents/Physics/RAMICES\_II/src/Galaxy/Galaxy.h, Memory  
129, 130  
GasReservoir, 32  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Galaxy/MacroMatrix.h,  
131  
EnumSets.h, 138  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Galaxy/RingAccretionStream  
132  
GasReservoir, 32  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Gas/Gas.Active  
133  
StellarPopulation, 102  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Gas/Gas.Active.h,  
133, 134  
StarEvents, 95  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Gas/Gas.Stream.h,  
135  
EnumSets.h, 138  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/ACB\_Units.h,  
136, 138  
YieldGrid, 117  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/ACB\_DealParameters.h,  
139  
StarEvents, 95  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/ACB\_InitialisedData.h,  
140  
InitialisedData, 54  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/ListStellarPopulation, 104  
141  
Age  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Parameters/ParameterList, 104  
142  
StellarPopulation  
AliveMass  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/IMF.h, StarReservoir, 97  
147  
Area  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/IsochroneTracker.h,  
148, 149  
argPointers  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/RemnantPopulation, 78  
150  
AssignIsochrones  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/SLF.h, Galaxy.h, 130  
151  
AssignMagnitudes  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/StarEvents, 14  
151, 152  
StarReservoir, 97  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/StarReservoir.h,  
152  
Resolution  
CatalogueValues, 8  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Stars/StellarPopulation.h,  
153  
BH  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Yields/SimpleYield.h, MassReport, 65  
155  
BirthGas  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Yields/YieldGrid.h, StellarPopulation, 104  
155, 156  
BirthIndex  
/Users/jf20/Documents/Physics/RAMICES\_II/src/Yields/YieldRidge.h, genMass, 64  
157  
StellarPopulation, 105  
BirthRadius  
StellarPopulation, 105  
BlackHole  
EnumSets.h, 137  
BlackHoleMass  
RemnantPopulation, 81  
BMag  
IsochroneTracker.h, 148

AbsoluteColdGasFile  
OutputValues, 74  
AbsoluteHotGasFile  
OutputValues, 74  
Absorb  
GasReservoir, 31, 32  
GasStream, 40

- BolometricMag
  - IsochroneTracker.h, [148](#)
- BrightestStar
  - Galaxy, [18](#)
- Calcium
  - EnumSets.h, [137](#)
- CapturedTs
  - IsochroneTracker, [62](#)
- CapturedZs
  - IsochroneTracker, [62](#)
- Carbon
  - EnumSets.h, [137](#)
- Catalogue
  - GlobalParameters, [47](#)
- CatalogueEntry
  - StellarPopulation, [102](#)
- CatalogueHeaders
  - StellarPopulation, [102](#)
- CatalogueValues, [7](#)
  - AzimuthalResolution, [8](#)
  - CatalogueValues, [8](#)
  - IsochroneMagnitudeResolution, [8](#)
  - IsochroneTimeStep, [8](#)
  - RadialResolution, [8](#)
  - SampleCount, [8](#)
  - SolarRadius, [9](#)
  - SynthesisActive, [9](#)
  - VerticalHeightPower, [9](#)
  - VerticalHeightScaling, [9](#)
  - VerticalHeightStart, [9](#)
- CCSN
  - EnumSets.h, [138](#)
  - StarEvents, [95](#)
- CCSN\_Initialise
  - YieldGrid, [118](#)
- CCSN\_MassCut
  - YieldValues, [125](#)
- CCSNYield
  - InitialisedData, [54](#)
  - StellarPopulation, [105](#)
- CGM
  - Galaxy, [19](#)
- CGM\_Mass
  - GalaxyValues, [22](#)
- CGMAbsorbing
  - GalaxyValues, [22](#)
- CGMBuffer
  - Ring, [88](#)
- CGMOperations
  - Galaxy, [15](#)
- CheckMass
  - Gas, [27](#)
- ChemicalPrefactor
  - OutputValues, [74](#)
- ChimneyFactor
  - ThermalValues, [113](#)
- Chromium
  - EnumSets.h, [137](#)
- Cobalt
  - EnumSets.h, [137](#)
- CODwarf
  - EnumSets.h, [137](#)
- CODwarf\_MassCut
  - YieldValues, [125](#)
- Cold
  - GasStream, [41](#)
- ColdBuffer
  - Ring, [88](#)
- ColdGasDataFile
  - OutputValues, [75](#)
- ColdGasMass
  - Galaxy, [14](#)
- ColdGasMetallicity
  - GasReservoir, [33](#)
- ColdMass
  - GasReservoir, [33](#)
  - GasStream, [41](#)
- Collapse\_MassCut
  - YieldValues, [125](#)
- Components
  - GasReservoir, [36](#)
- Composition
  - GasReservoir, [33](#)
- Compound
  - MigrationMatrix, [69](#)
- Compounding
  - Galaxy.h, [130](#)
- CompoundScattering
  - Galaxy, [14](#)
- ComputeMasses
  - GasStream, [41](#)
- ComputeScattering
  - Galaxy, [14](#)
- ComputeSelectionFunction
  - Ring, [86](#)
- ComputeVisibilityFunction
  - Galaxy, [14](#)
- Config
  - OutputValues, [75](#)
- Configure
  - ParamList, [78](#)
- Construct
  - IsochroneTracker, [60](#)
- Cool
  - GasStream, [42](#)
  - Ring, [86](#)
- CoolingPower
  - ThermalValues, [113](#)
- Count
  - IsochroneCube, [58](#)
  - IsoMass, [64](#)
- Countify
  - IsochroneEntry, [59](#)
- Create
  - MigrationMatrix, [69](#)
- CreateGrid

- YieldGrid, 118
- Data
  - Galaxy, 18
  - IsochroneCube, 58
  - Ring, 88
  - StarReservoir, 99
  - StellarPopulation, 105
- DeadMass
  - StarReservoir, 97
- Death
  - StarReservoir, 98
  - StellarPopulation, 102
- DeathIndex
  - IsoMass, 64
- Decay
  - RemnantPopulation, 80
- DeltaLogT
  - IsochroneTracker, 62
- Deplete
  - GasReservoir, 33
  - GasStream, 42
- DepletionIndex
  - StellarPopulation, 105
- DiagonalMultiply
  - MigrationMatrix, 69
- DimmestStar
  - Galaxy, 18
- Dirty
  - GasStream, 42
- DispersionOrder
  - MigrationValues, 71
- DispersionTruncation
  - MigrationValues, 72
- Distribution
  - StellarPopulation, 105
- distribution
  - InitialisedData, 54
  - IsochroneTracker, 62
- DormantDwarf
  - EnumSets.h, 137
- DormantHotFraction
  - ThermalValues, 113
- DormantNS
  - EnumSets.h, 137
- DormantNSMass
  - RemnantPopulation, 81
- DormantWDMass
  - RemnantPopulation, 81
- ECSN
  - EnumSets.h, 138
  - StarEvents, 95
- ECSN\_Fraction
  - YieldValues, 125
- ECSN\_Initialise
  - YieldGrid, 118
- ECSN\_MassCut
  - YieldValues, 126
- ECSNYield
  - InitialisedData, 54
  - StellarPopulation, 105
- Efficiency
  - StarEvents, 95
- EjectionFraction
  - StellarValues, 109
- Element
  - GlobalParameters, 47
- ElementCount
  - EnumSets.h, 137
- ElementDestruction
  - YieldGrid, 118
- ElementID
  - EnumSets.h, 136
- ElementNames
  - ElementValues, 11
- ElementProduction
  - YieldGrid, 118
- ElementValues, 9
  - ElementNames, 11
  - ElementValues, 10
  - GiveElementsNames, 10
  - Initialise, 11
  - ProtonCounts, 11
  - SolarAbundanceFile, 11
  - SolarAbundanceFileDataColumn, 11
  - SolarAbundanceFileNameColumn, 12
  - SolarAbundances, 12
- Empty
  - Gas, 28
- EnumSets.h
  - Accreted, 138
  - AGB, 138
  - BlackHole, 137
  - Calcium, 137
  - Carbon, 137
  - CCSN, 138
  - Chromium, 137
  - Cobalt, 137
  - CODwarf, 137
  - DormantDwarf, 137
  - DormantNS, 137
  - ECSN, 138
  - ElementCount, 137
  - ElementID, 136
  - Europium, 137
  - Helium, 136
  - Hydrogen, 136
  - Iron, 137
  - Limongi, 137
  - Maeder, 137
  - Magnesium, 137
  - Manganese, 137
  - Marigo, 137
  - MergerNS, 137
  - Metals, 136
  - Mixed, 137

- NeutronStar, [137](#)
- NSM, [138](#)
- Orfeo, [137](#)
- Oxygen, [137](#)
- ProcessCount, [138](#)
- Remnant, [138](#)
- RemnantType, [137](#)
- Silicon, [137](#)
- SNla, [138](#)
- SourceCount, [137](#)
- SourceID, [137](#)
- SourceProcess, [138](#)
- Stellar, [138](#)
- Unknown, [137](#)
- YieldCount, [138](#)
- YieldProcess, [138](#)
- Europium
  - EnumSets.h, [137](#)
- EventRate
  - StarReservoir, [99](#)
- EventRateFile
  - OutputValues, [75](#)
- Evolve
  - Galaxy, [14](#)
- ExtractSample
  - IsochroneTracker, [61](#)
- Feed
  - RemnantPopulation, [80](#)
- FeedbackEjectFactor
  - ThermalValues, [113](#)
- FeedbackFactor
  - StellarValues, [109](#)
- FirstMoment
  - Integral, [56](#)
- Form
  - StarReservoir, [98](#)
- FormationCount
  - IMF\_Functor, [50](#)
- FormStars
  - StellarPopulation, [102](#)
- FullDeathScan
  - StellarPopulation, [103](#)
- Galaxy, [12](#)
  - AssignMagnitudes, [14](#)
  - BrightestStar, [18](#)
  - CGM, [19](#)
  - CGMOperations, [15](#)
  - ColdGasMass, [14](#)
  - CompoundScattering, [14](#)
  - ComputeScattering, [14](#)
  - ComputeVisibilityFunction, [14](#)
  - Data, [18](#)
  - DimmestStar, [18](#)
  - Evolve, [14](#)
  - Galaxy, [13](#)
  - GasMass, [15](#)
  - GasScaleLength, [15](#)
  - GlobalParameters, [47](#)
  - Infall, [15](#)
  - InfallMass, [15](#)
  - InsertInfallingGas, [15](#)
  - LaunchParallelOperation, [15](#)
  - Mass, [16](#)
  - MassHeaders, [16](#)
  - Migrator, [19](#)
  - ParallelBars, [19](#)
  - Param, [19](#)
  - PredictSurfaceDensity, [16](#)
  - RelicMass, [16](#)
  - RingEvolve, [16](#)
  - RingMasses, [19](#)
  - Rings, [19](#)
  - SaveState, [16](#)
  - SaveState\_Enrichment, [17](#)
  - SaveState\_Events, [17](#)
  - SaveState\_Mass, [17](#)
  - ScatterGas, [17](#)
  - ScatterYields, [17](#)
  - SelectionFunction, [17](#)
  - StarMass, [18](#)
  - StellarSynthesis, [18](#)
  - SynthesiseObservations, [18](#)
  - SynthesisOutput, [19](#)
  - SynthesisProgress, [19](#)
  - Threads, [20](#)
- Galaxy.h
  - AssignIsochrones, [130](#)
  - Compounding, [130](#)
  - ParallelJob, [129](#)
  - RingStep, [130](#)
  - Scattering, [130](#)
  - Selection, [130](#)
  - Synthesis, [130](#)
- GalaxyMassFile
  - OutputValues, [75](#)
- GalaxyValues, [20](#)
  - CGM\_Mass, [22](#)
  - CGMAbsorbing, [22](#)
  - GalaxyValues, [22](#)
  - InfallMass1, [22](#)
  - InfallMass2, [23](#)
  - InfallMassMerger, [23](#)
  - InfallTime1, [23](#)
  - InfallTime2, [23](#)
  - InfallTimeMerger, [23](#)
  - Initialise, [22](#)
  - MaxScaleLength, [23](#)
  - MaxSFRFraction, [24](#)
  - MergerDelayTime, [24](#)
  - MergerTurnOnWidth, [24](#)
  - MinScaleLength, [24](#)
  - PrimordialHotFraction, [24](#)
  - PrimordialMass, [24](#)
  - Radius, [25](#)
  - Ring0Width, [25](#)

- RingCount, 25
- RingRadius, 25
- RingWidth, 25
- ScaleLengthDelay, 25
- ScaleLengthFinalTime, 26
- ScaleLengthTimeScale, 26
- UsingVariableRingWidth, 26
- Gas, 26
  - CheckMass, 27
  - Empty, 28
  - Gas, 27
  - internal\_Mass, 29
  - Mass, 28
  - NeedsRecomputing, 29
  - operator[], 28
  - Primordial, 29
  - Ring, 88
  - Species, 29
- GasCoolingTimeScale
  - ThermalValues, 113
- GasMass
  - Galaxy, 15
- GasReservoir, 30
  - Absorb, 31, 32
  - AbsorbMemory, 32
  - AccretionStream, 32
  - ColdGasMetallicity, 33
  - ColdMass, 33
  - Components, 36
  - Composition, 33
  - Deplete, 33
  - GasReservoir, 31
  - Heat, 34
  - HotMass, 34
  - Mass, 34
  - operator[], 34, 35
  - Param, 36
  - PassiveCool, 35
  - Primordial, 35
  - TransferAndHeat, 35
  - TransferColdFrom, 35
  - TransferFrom, 36
  - TransferHotFrom, 36
  - Wipe, 36
- GasScaleLength
  - Galaxy, 15
- GasStream, 37
  - Absorb, 40
  - Cold, 41
  - ColdMass, 41
  - ComputeMasses, 41
  - Cool, 42
  - Deplete, 42
  - Dirty, 42
  - GasStream, 38, 39
  - Heat, 43
  - Hot, 43
  - HotMass, 43
  - internal\_Cold, 44
  - internal\_ColdMass, 44
  - internal\_Hot, 44
  - internal\_HotMass, 45
  - internal\_TotalMass, 45
  - Mass, 44
  - NeedsRecomputing, 45
  - Source, 45
- generator
  - InitialisedData, 54
  - IsochroneTracker, 62
- GetBracket
  - YieldGrid, 118
- GetProperties
  - IsochroneTracker, 61
- GiveElementsNames
  - ElementValues, 10
- GlobalParameters, 45
  - Catalogue, 47
  - Element, 47
  - Galaxy, 47
  - GlobalParameters, 46
  - Initialise, 47
  - Meta, 47
  - Migration, 48
  - Output, 48
  - ParamMembers, 48
  - Resources, 48
  - SaveInputs, 47
  - Stellar, 48
  - Thermal, 48
  - Yield, 49
- GlobalParameters.h
  - PI, 139
- Grid
  - IsochroneTracker, 62
  - MigrationMatrix, 70
  - SimpleYield, 91
  - YieldGrid, 120
- hasSingle
  - YieldBracket, 116
- Heat
  - GasReservoir, 34
  - GasStream, 43
- Helium
  - EnumSets.h, 136
- HMag
  - IsochroneTracker.h, 148
- Hot
  - GasStream, 43
- HotBuffer
  - Ring, 88
- HotGasDataFile
  - OutputValues, 75
- HotInjection\_AGB
  - ThermalValues, 113
- HotInjection\_CCSN
  - ThermalValues, 114

- HotInjection\_NSM
  - ThermalValues, 114
- HotInjection\_SNIa
  - ThermalValues, 114
- hotInjectionFraction
  - SimpleYield, 91
  - YieldGrid, 120
- HotMass
  - GasReservoir, 34
  - GasStream, 43
- Hydrogen
  - EnumSets.h, 136
- IMag
  - IsochroneTracker.h, 148
- IMF
  - IMF\_Functor, 50
  - InitialisedData, 54
  - StarReservoir, 99
  - StellarPopulation, 105
- IMF\_Functor, 49
  - FormationCount, 50
  - IMF, 50
  - IMF\_Functor, 49
  - IMF\_MeanMass, 51
  - IMF\_Normalisation, 51
  - IMF\_Weighting, 51
  - MomentCompute, 50
  - Normalise, 50
  - operator(), 50
  - Param, 51
  - Weighting, 50
- IMF\_MeanMass
  - IMF\_Functor, 51
- IMF\_Normalisation
  - IMF\_Functor, 51
- IMF\_Slope
  - StellarValues, 109
- IMF\_Weighting
  - IMF\_Functor, 51
- ImmortalMass
  - StellarValues, 109
- ImmortalStars
  - StellarPopulation, 106
- Infall
  - Galaxy, 15
- InfallMass
  - Galaxy, 15
- InfallMass1
  - GalaxyValues, 22
- InfallMass2
  - GalaxyValues, 23
- InfallMassMerger
  - GalaxyValues, 23
- InfallTime1
  - GalaxyValues, 23
- InfallTime2
  - GalaxyValues, 23
- InfallTimeMerger
  - GalaxyValues, 23
- InflowActive
  - MigrationValues, 72
- InflowParameterA
  - MigrationValues, 72
- InflowParameterB
  - MigrationValues, 72
- Initialise
  - ElementValues, 11
  - GalaxyValues, 22
  - GlobalParameters, 47
  - MetaValues, 67
  - OutputValues, 74
  - ParamList, 78
  - ResourceValues, 83
  - StellarValues, 108
  - YieldValues, 125
- InitialisedData, 51
  - AGBYield, 54
  - CCSNYield, 54
  - distribution, 54
  - ECSNYield, 54
  - generator, 54
  - IMF, 54
  - InitialisedData, 52
  - Isochrones, 55
  - Log, 52, 53
  - LogFlush, 53
  - NormalDist, 53
  - NSMYield, 55
  - Param, 55
  - ProgressBar, 53
  - SLF, 55
  - SNIIaYield, 55
  - UniformDist, 53
  - UrgentLog, 54
- InitialiseLargeGrid
  - YieldGrid, 119
- InsertInfallingGas
  - Galaxy, 15
- Integral, 55
  - FirstMoment, 56
  - ZerothMoment, 56
- internal\_Cold
  - GasStream, 44
- internal\_ColdMass
  - GasStream, 44
- internal\_Hot
  - GasStream, 44
- internal\_HotMass
  - GasStream, 45
- internal\_Mass
  - Gas, 29
- internal\_MassCounter
  - StellarPopulation, 106
- internal\_TotalMass
  - GasStream, 45
- Interpolate



- Interpolator, [56](#)
  - YieldBracket, [115](#)
- Interpolator, [56](#)
  - Interpolate, [56](#)
  - LinearFactor, [57](#)
  - LowerID, [57](#)
  - UpperID, [57](#)
- Iron
  - EnumSets.h, [137](#)
- IsDepleted
  - StellarPopulation, [106](#)
- isEnclosed
  - YieldBracket, [116](#)
- IsLifetimeMonotonic
  - StellarPopulation, [106](#)
- Isochrone
  - IsoMass, [64](#)
- IsochroneCube, [57](#)
  - Count, [58](#)
  - Data, [58](#)
  - Value, [58](#)
  - Weighting, [58](#)
- IsochroneDirectory
  - ResourceValues, [83](#)
- IsochroneEntry, [58](#)
  - Countify, [59](#)
  - IsochroneEntry, [59](#)
  - operator[], [59](#)
  - Properties, [59](#)
- IsochroneMagnitudeResolution
  - CatalogueValues, [8](#)
- IsochroneProperties
  - IsochroneTracker.h, [148](#)
- IsochroneRepository
  - ResourceValues, [83](#)
- Isochrones
  - InitialisedData, [55](#)
- IsochroneTimeStep
  - CatalogueValues, [8](#)
- IsochroneTracker, [60](#)
  - CapturedTs, [62](#)
  - CapturedZs, [62](#)
  - Construct, [60](#)
  - DeltaLogT, [62](#)
  - distribution, [62](#)
  - ExtractSample, [61](#)
  - generator, [62](#)
  - GetProperties, [61](#)
  - Grid, [62](#)
  - IsochroneTracker, [60](#)
  - IsoLog, [61](#)
  - isTimeLogUniform, [62](#)
  - NormalSample, [61](#)
  - Param, [63](#)
  - ParseFile, [61](#)
  - UniformSample, [61](#)
  - UnsortedGrid, [63](#)
- IsochroneTracker.h
  - BMag, [148](#)
  - BolometricMag, [148](#)
  - HMag, [148](#)
  - IMag, [148](#)
  - IsochroneProperties, [148](#)
  - JMag, [148](#)
  - KMag, [148](#)
  - Logg, [148](#)
  - logL, [148](#)
  - PropertyCount, [148](#)
  - PropertyNames, [149](#)
  - RMag, [148](#)
  - TEff, [148](#)
  - UMag, [148](#)
  - VMag, [148](#)
- IsoLog
  - IsochroneTracker, [61](#)
- IsoMass, [63](#)
  - BirthIndex, [64](#)
  - Count, [64](#)
  - DeathIndex, [64](#)
  - Isochrone, [64](#)
  - IsoMass, [64](#)
  - MassIndex, [64](#)
  - Metallicity, [65](#)
- isTimeLogUniform
  - IsochroneTracker, [62](#)
- JMag
  - IsochroneTracker.h, [148](#)
- KillStars
  - Ring, [86](#)
- KMag
  - IsochroneTracker.h, [148](#)
- LaunchParallelOperation
  - Galaxy, [15](#)
- LifeTime
  - SLF\_Functor, [92](#)
- LifeTimeFile
  - ResourceValues, [84](#)
- Limongi
  - EnumSets.h, [137](#)
- LinearFactor
  - Interpolator, [57](#)
- LoadLimongiYields
  - YieldGrid, [119](#)
- LoadMaederYields
  - YieldGrid, [119](#)
- LoadMarigoYields
  - YieldGrid, [119](#)
- LoadOrfeoYields
  - YieldGrid, [119](#)
- Log
  - InitialisedData, [52](#), [53](#)
- LogarithmicColdGasFile
  - OutputValues, [75](#)
- LogarithmicHotGasFile

- OutputValues, 76
- LogFlush
  - InitialisedData, 53
- Logg
  - IsochroneTracker.h, 148
- logL
  - IsochroneTracker.h, 148
- LogZDelta
  - StellarValues, 109
- LogZGrid
  - StellarValues, 109
- LogZResolution
  - StellarValues, 109
- LongSNIaBuffer
  - RemnantPopulation, 81
- LowerID
  - Interpolator, 57
- LowerRidge
  - YieldBracket, 116
- Maeder
  - EnumSets.h, 137
- Magnesium
  - EnumSets.h, 137
- MakeStars
  - Ring, 86
- Manganese
  - EnumSets.h, 137
- Marigo
  - EnumSets.h, 137
- MarkovDispersionStrength
  - MigrationValues, 72
- Mass
  - Galaxy, 16
  - Gas, 28
  - GasReservoir, 34
  - GasStream, 44
  - RemnantOutput, 79
  - RemnantPopulation, 81
  - Ring, 86
  - StellarPopulation, 103
  - YieldPoint, 122
- MassDeltas
  - StellarValues, 110
- MassGrid
  - StellarValues, 110
- MassHeaders
  - Galaxy, 16
- MassIndex
  - IsoMass, 64
- MassOffset
  - YieldGrid, 120
- MassOverhang
  - YieldValues, 126
- MassReport, 65
  - BH, 65
  - NS, 65
  - Total, 65
  - WD, 66
- MassResolution
  - StellarValues, 110
- MaxLogZ
  - StellarValues, 110
- MaxMv
  - Ring, 88
- MaxScaleLength
  - GalaxyValues, 23
- MaxSFRFraction
  - GalaxyValues, 24
- MaxStealFraction
  - MigrationValues, 72
- MaxStellarMass
  - StellarValues, 110
- MergerDelayTime
  - GalaxyValues, 24
- MergerNS
  - EnumSets.h, 137
- MergerTurnOnWidth
  - GalaxyValues, 24
- Meta
  - GlobalParameters, 47
- Metallicity
  - IsoMass, 65
  - StellarPopulation, 106
- MetallicityInterpolation
  - YieldGrid, 119
- Metals
  - EnumSets.h, 136
- MetaValues, 66
  - Initialise, 67
  - MetaValues, 67
  - ParallelThreads, 67
  - ProgressHashes, 67
  - SimulationDuration, 68
  - SimulationSteps, 68
  - TimeStep, 68
  - Verbosity, 68
- MetCheck
  - Ring, 86
- Migration
  - GlobalParameters, 48
- MigrationMatrix, 68
  - Compound, 69
  - Create, 69
  - DiagonalMultiply, 69
  - Grid, 70
  - MigrationMatrix, 69
  - NRings, 70
  - Param, 70
  - Print, 70
- MigrationValues, 70
  - DispersionOrder, 71
  - DispersionTruncation, 72
  - InflowActive, 72
  - InflowParameterA, 72
  - InflowParameterB, 72
  - MarkovDispersionStrength, 72

- MaxStealFraction, [72](#)
  - MigrationValues, [71](#)
- Migrator
  - Galaxy, [19](#)
- MinLogZ
  - StellarValues, [110](#)
- MinMv
  - Ring, [88](#)
- MinScaleLength
  - GalaxyValues, [24](#)
- MinStellarMass
  - StellarValues, [111](#)
- Mixed
  - EnumSets.h, [137](#)
- MomentCompute
  - IMF\_Functor, [50](#)
- MonotonicDeathScan
  - StellarPopulation, [103](#)
- NeedsRecomputing
  - Gas, [29](#)
  - GasStream, [45](#)
- NeutronStar
  - EnumSets.h, [137](#)
- NormalDist
  - InitialisedData, [53](#)
- Normalise
  - IMF\_Functor, [50](#)
- NormalSample
  - IsochroneTracker, [61](#)
- NotComputed
  - SLF\_Functor, [93](#)
- NRings
  - MigrationMatrix, [70](#)
- NS
  - MassReport, [65](#)
- NSM
  - EnumSets.h, [138](#)
  - StarEvents, [95](#)
- NSM\_ActiveFraction
  - YieldValues, [126](#)
- NSM\_DelayTime
  - YieldValues, [126](#)
- NSM\_Initialise
  - SimpleYield, [90](#)
- NSM\_Scale
  - YieldValues, [126](#)
- NSM\_TypicalMass
  - YieldValues, [126](#)
- NSMBuffer
  - RemnantPopulation, [81](#)
- NSMYield
  - InitialisedData, [55](#)
  - RemnantPopulation, [81](#)
- NStarsFormed
  - StarEvents, [96](#)
- NumericalResolution
  - ThermalValues, [114](#)
- Observations
  - StarReservoir, [98](#)
- operator()
  - IMF\_Functor, [50](#)
  - SimpleYield, [90](#)
  - SLF\_Functor, [93](#)
  - YieldGrid, [119](#)
- operator[]
  - Gas, [28](#)
  - GasReservoir, [34](#), [35](#)
  - IsochroneEntry, [59](#)
  - StellarPopulation, [103](#)
- Orfeo
  - EnumSets.h, [137](#)
- Output
  - GlobalParameters, [48](#)
- OutputValues, [73](#)
  - AbsoluteColdGasFile, [74](#)
  - AbsoluteHotGasFile, [74](#)
  - ChemicalPrefactor, [74](#)
  - ColdGasDataFile, [75](#)
  - Config, [75](#)
  - EventRateFile, [75](#)
  - GalaxyMassFile, [75](#)
  - HotGasDataFile, [75](#)
  - Initialise, [74](#)
  - LogarithmicColdGasFile, [75](#)
  - LogarithmicHotGasFile, [76](#)
  - OutputValues, [74](#)
  - Root, [76](#)
  - StarFile, [76](#)
  - YieldSubdir, [76](#)
- Oxygen
  - EnumSets.h, [137](#)
- ParallelBars
  - Galaxy, [19](#)
- ParallelJob
  - Galaxy.h, [129](#)
- ParallelThreads
  - MetaValues, [67](#)
- Param
  - Galaxy, [19](#)
  - GasReservoir, [36](#)
  - IMF\_Functor, [51](#)
  - InitialisedData, [55](#)
  - IsochroneTracker, [63](#)
  - MigrationMatrix, [70](#)
  - RemnantPopulation, [82](#)
  - Ring, [88](#)
  - SimpleYield, [91](#)
  - SLF\_Functor, [93](#)
  - StarReservoir, [99](#)
  - StellarPopulation, [106](#)
  - YieldGrid, [121](#)
- ParamList, [77](#)
  - argPointers, [78](#)
  - Configure, [78](#)
  - Initialise, [78](#)

- StreamContentsTo, 78
- ParamMembers
  - GlobalParameters, 48
- ParentArea
  - StarReservoir, 99
- ParentRing
  - StarReservoir, 99
- ParseFile
  - IsochroneTracker, 61
- PassiveCool
  - GasReservoir, 35
- PI
  - GlobalParameters.h, 139
- Points
  - YieldRidge, 123
- Population
  - StarReservoir, 99
- PopulationIndex
  - StarReservoir, 100
- PrecomputedGrid
  - SLF\_Functor, 94
- PrecomputeGrid
  - SLF\_Functor, 93
- PredictLifetime
  - SLF\_Functor, 93
- PredictSurfaceDensity
  - Galaxy, 16
- PrepareIMF
  - StellarPopulation, 103
- Primordial
  - Gas, 29
  - GasReservoir, 35
- PrimordialHotFraction
  - GalaxyValues, 24
- PrimordialMass
  - GalaxyValues, 24
- Print
  - MigrationMatrix, 70
- PrintStatus
  - StarReservoir, 98
- Process
  - SimpleYield, 91
  - YieldGrid, 121
- ProcessCount
  - EnumSets.h, 138
- ProcessNames
  - YieldValues, 126
- ProcessTypes
  - YieldValues, 127
- ProgressBar
  - InitialisedData, 53
- ProgressHashes
  - MetaValues, 67
- Properties
  - IsochroneEntry, 59
- PropertyCount
  - IsochroneTracker.h, 148
- PropertyNames
  - IsochroneTracker.h, 149
- ProtonCounts
  - ElementValues, 11
- PurityEnforce
  - YieldGrid, 120
- RadialResolution
  - CatalogueValues, 8
- Radius
  - GalaxyValues, 25
  - Ring, 89
- RadiusIndex
  - Ring, 89
- RecoverMatter
  - StellarPopulation, 104
- Relic
  - StellarPopulation, 104
- RelicMass
  - Galaxy, 16
- Remnant
  - EnumSets.h, 138
- RemnantInject
  - SimpleYield, 91
- RemnantLocation
  - YieldGrid, 121
- RemnantOutput, 79
  - Mass, 79
  - Type, 79
- RemnantPopulation, 79
  - BlackHoleMass, 81
  - Decay, 80
  - DormantNSMass, 81
  - DormantWDMass, 81
  - Feed, 80
  - LongSNIaBuffer, 81
  - Mass, 81
  - NSMBuffer, 81
  - NSMYield, 81
  - Param, 82
  - RemnantPopulation, 80
  - ShortSNIaBuffer, 82
  - SNIaYield, 82
- Remnants
  - StarReservoir, 100
- RemnantType
  - EnumSets.h, 137
- ResourceRoot
  - ResourceValues, 84
- Resources
  - GlobalParameters, 48
- ResourceValues, 82
  - Initialise, 83
  - IsochroneDirectory, 83
  - IsochroneRepository, 83
  - LifeTimeFile, 84
  - ResourceRoot, 84
  - ResourceValues, 83
  - WelcomeFile, 84
  - YieldRoot, 84

- RidgeStorage
  - YieldGrid, [121](#)
- Ring, [84](#)
  - Area, [87](#)
  - CGMBuffer, [88](#)
  - ColdBuffer, [88](#)
  - ComputeSelectionFunction, [86](#)
  - Cool, [86](#)
  - Data, [88](#)
  - Gas, [88](#)
  - HotBuffer, [88](#)
  - KillStars, [86](#)
  - MakeStars, [86](#)
  - Mass, [86](#)
  - MaxMv, [88](#)
  - MetCheck, [86](#)
  - MinMv, [88](#)
  - Param, [88](#)
  - Radius, [89](#)
  - RadiusIndex, [89](#)
  - Ring, [85](#)
  - SaveChemicalHistory, [86](#)
  - SelectionEffect, [87](#)
  - SelectionGrid, [89](#)
  - Stars, [89](#)
  - Synthesis, [87](#)
  - TimeStep, [87](#)
  - UpdateMemory, [87](#)
  - Width, [89](#)
- Ring0Width
  - GalaxyValues, [25](#)
- RingCount
  - GalaxyValues, [25](#)
- RingEvolve
  - Galaxy, [16](#)
- RingMasses
  - Galaxy, [19](#)
- RingRadius
  - GalaxyValues, [25](#)
- Rings
  - Galaxy, [19](#)
- RingStep
  - Galaxy.h, [130](#)
- RingWidth
  - GalaxyValues, [25](#)
- RMag
  - IsochroneTracker.h, [148](#)
- Root
  - OutputValues, [76](#)
- SampleCount
  - CatalogueValues, [8](#)
- Save
  - StarEvents, [95](#)
- SaveChemicalHistory
  - Ring, [86](#)
- SaveEventRate
  - StarReservoir, [98](#)
- SaveGrid
  - YieldGrid, [120](#)
- SaveInputs
  - GlobalParameters, [47](#)
- SaveState
  - Galaxy, [16](#)
- SaveState\_Enrichment
  - Galaxy, [17](#)
- SaveState\_Events
  - Galaxy, [17](#)
- SaveState\_Mass
  - Galaxy, [17](#)
- ScaleLengthDelay
  - GalaxyValues, [25](#)
- ScaleLengthFinalTime
  - GalaxyValues, [26](#)
- ScaleLengthTimeScale
  - GalaxyValues, [26](#)
- ScatterGas
  - Galaxy, [17](#)
- Scattering
  - Galaxy.h, [130](#)
- ScatterYields
  - Galaxy, [17](#)
- SchmidtDensityCut
  - StellarValues, [111](#)
- SchmidtLowPower
  - StellarValues, [111](#)
- SchmidtMainPower
  - StellarValues, [111](#)
- SchmidtPrefactor
  - StellarValues, [111](#)
- Selection
  - Galaxy.h, [130](#)
- SelectionEffect
  - Ring, [87](#)
- SelectionFunction
  - Galaxy, [17](#)
- SelectionGrid
  - Ring, [89](#)
- SFR\_GasLoss
  - StarReservoir, [98](#)
- ShortSNIaBuffer
  - RemnantPopulation, [82](#)
- Silicon
  - EnumSets.h, [137](#)
- SimpleYield, [89](#)
  - Grid, [91](#)
  - hotInjectionFraction, [91](#)
  - NSM\_Initialise, [90](#)
  - operator(), [90](#)
  - Param, [91](#)
  - Process, [91](#)
  - RemnantInject, [91](#)
  - SimpleYield, [90](#)
  - SNIa\_Initialise, [91](#)
- SimulationDuration
  - MetaValues, [68](#)
- SimulationSteps

- MetaValues, 68
- SLF
  - InitialisedData, 55
  - StarReservoir, 100
  - StellarPopulation, 106
- SLF\_Functor, 92
  - LifeTime, 92
  - NotComputed, 93
  - operator(), 93
  - Param, 93
  - PrecomputedGrid, 94
  - PrecomputeGrid, 93
  - PredictLifetime, 93
  - SLF\_Functor, 92
  - ValueInquiry, 93
- SNII
  - EnumSets.h, 138
  - StarEvents, 96
- SNII\_ActiveFraction
  - YieldValues, 127
- SNII\_DelayTime
  - YieldValues, 127
- SNII\_Initialise
  - SimpleYield, 91
- SNII\_LongFraction
  - YieldValues, 127
- SNII\_LongScale
  - YieldValues, 127
- SNII\_ShortScale
  - YieldValues, 127
- SNII\_TypicalMass
  - YieldValues, 127
- SNIIYield
  - InitialisedData, 55
  - RemnantPopulation, 82
- SolarAbundanceFile
  - ElementValues, 11
- SolarAbundanceFileDataColumn
  - ElementValues, 11
- SolarAbundanceFileNameColumn
  - ElementValues, 12
- SolarAbundances
  - ElementValues, 12
- SolarRadius
  - CatalogueValues, 9
- Source
  - GasStream, 45
  - YieldRidge, 123
- SourceCount
  - EnumSets.h, 137
- SourceID
  - EnumSets.h, 137
- SourcePriority
  - YieldGrid, 121
- SourceProcess
  - EnumSets.h, 138
- Species
  - Gas, 29
- StarEvents, 94
  - AddHeaders, 95
  - AGBDeaths, 95
  - CCSN, 95
  - ECSN, 95
  - Efficiency, 95
  - NSM, 95
  - NStarsFormed, 96
  - Save, 95
  - SNII, 96
  - StarEvents, 94
  - StarMassFormed, 96
- StarFile
  - OutputValues, 76
- StarMass
  - Galaxy, 18
- StarMassFormed
  - StarEvents, 96
- StarReservoir, 96
  - AliveMass, 97
  - AssignMagnitudes, 97
  - Data, 99
  - DeadMass, 97
  - Death, 98
  - EventRate, 99
  - Form, 98
  - IMF, 99
  - Observations, 98
  - Param, 99
  - ParentArea, 99
  - ParentRing, 99
  - Population, 99
  - PopulationIndex, 100
  - PrintStatus, 98
  - Remnants, 100
  - SaveEventRate, 98
  - SFR\_GasLoss, 98
  - SLF, 100
  - StarReservoir, 97
  - Temp\_Mass, 100
  - YieldOutput, 100
  - YieldsFrom, 98
- Stars
  - Ring, 89
- Stellar
  - EnumSets.h, 138
  - GlobalParameters, 48
- StellarInject
  - YieldGrid, 120
- StellarPopulation, 100
  - Active, 102
  - AGBYield, 104
  - Age, 104
  - BirthGas, 104
  - BirthIndex, 105
  - BirthRadius, 105
  - CatalogueEntry, 102
  - CatalogueHeaders, 102

- CCSNYield, [105](#)
- Data, [105](#)
- Death, [102](#)
- DepletionIndex, [105](#)
- Distribution, [105](#)
- ECSNYield, [105](#)
- FormStars, [102](#)
- FullDeathScan, [103](#)
- IMF, [105](#)
- ImmortalStars, [106](#)
- internal\_MassCounter, [106](#)
- IsDepleted, [106](#)
- IsLifetimeMonotonic, [106](#)
- Mass, [103](#)
- Metallicity, [106](#)
- MonotonicDeathScan, [103](#)
- operator[], [103](#)
- Param, [106](#)
- PrepareIMF, [103](#)
- RecoverMatter, [104](#)
- Relic, [104](#)
- SLF, [106](#)
- StellarPopulation, [102](#)
- TempGas, [106](#)
- StellarSynthesis
  - Galaxy, [18](#)
- StellarValues, [107](#)
  - EjectionFraction, [109](#)
  - FeedbackFactor, [109](#)
  - IMF\_Slope, [109](#)
  - ImmortalMass, [109](#)
  - Initialise, [108](#)
  - LogZDelta, [109](#)
  - LogZGrid, [109](#)
  - LogZResolution, [109](#)
  - MassDeltas, [110](#)
  - MassGrid, [110](#)
  - MassResolution, [110](#)
  - MaxLogZ, [110](#)
  - MaxStellarMass, [110](#)
  - MinLogZ, [110](#)
  - MinStellarMass, [111](#)
  - SchmidtDensityCut, [111](#)
  - SchmidtLowPower, [111](#)
  - SchmidtMainPower, [111](#)
  - SchmidtPrefactor, [111](#)
  - StellarValues, [108](#)
- StreamContentsTo
  - ParamList, [78](#)
- Synthesis
  - Galaxy.h, [130](#)
  - Ring, [87](#)
- SynthesisActive
  - CatalogueValues, [9](#)
- SynthesiseObservations
  - Galaxy, [18](#)
- SynthesisOutput
  - Galaxy, [19](#)
- SynthesisProgress
  - Galaxy, [19](#)
- TargetNi56Yield
  - YieldValues, [127](#)
- TEff
  - IsochroneTracker.h, [148](#)
- Temp\_Mass
  - StarReservoir, [100](#)
- TempGas
  - StellarPopulation, [106](#)
- Thermal
  - GlobalParameters, [48](#)
- ThermalValues, [112](#)
  - ChimneyFactor, [113](#)
  - CoolingPower, [113](#)
  - DormantHotFraction, [113](#)
  - FeedbackEjectFactor, [113](#)
  - GasCoolingTimeScale, [113](#)
  - HotInjection\_AGB, [113](#)
  - HotInjection\_CCSN, [114](#)
  - HotInjection\_NSM, [114](#)
  - HotInjection\_SNIa, [114](#)
  - NumericalResolution, [114](#)
  - ThermalValues, [112](#)
- Threads
  - Galaxy, [20](#)
- TimeStep
  - MetaValues, [68](#)
  - Ring, [87](#)
- Total
  - MassReport, [65](#)
- TransferAndHeat
  - GasReservoir, [35](#)
- TransferColdFrom
  - GasReservoir, [35](#)
- TransferFrom
  - GasReservoir, [36](#)
- TransferHotFrom
  - GasReservoir, [36](#)
- Type
  - RemnantOutput, [79](#)
- UMag
  - IsochroneTracker.h, [148](#)
- UniformDist
  - InitialisedData, [53](#)
- UniformSample
  - IsochroneTracker, [61](#)
- Unknown
  - EnumSets.h, [137](#)
- UnsortedGrid
  - IsochroneTracker, [63](#)
- UpdateMemory
  - Ring, [87](#)
- UpperID
  - Interpolator, [57](#)
- UpperRidge
  - YieldBracket, [116](#)

- UrgentLog
  - InitialisedData, [54](#)
- UsingVariableRingWidth
  - GalaxyValues, [26](#)
- Value
  - IsochroneCube, [58](#)
- ValueInquiry
  - SLF\_Functor, [93](#)
- Verbosity
  - MetaValues, [68](#)
- VerticalHeightPower
  - CatalogueValues, [9](#)
- VerticalHeightScaling
  - CatalogueValues, [9](#)
- VerticalHeightStart
  - CatalogueValues, [9](#)
- VMag
  - IsochroneTracker.h, [148](#)
- WD
  - MassReport, [66](#)
- Weighting
  - IMF\_Functor, [50](#)
  - IsochroneCube, [58](#)
- WelcomeFile
  - ResourceValues, [84](#)
- Width
  - Ring, [89](#)
- Wipe
  - GasReservoir, [36](#)
- Yield
  - GlobalParameters, [49](#)
  - YieldPoint, [122](#)
- YieldBracket, [114](#)
  - hasSingle, [116](#)
  - Interpolate, [115](#)
  - isEnclosed, [116](#)
  - LowerRidge, [116](#)
  - UpperRidge, [116](#)
  - YieldBracket, [115](#)
- YieldCount
  - EnumSets.h, [138](#)
- YieldGrid, [116](#)
  - AGB\_Initialise, [117](#)
  - CCSN\_Initialise, [118](#)
  - CreateGrid, [118](#)
  - ECSN\_Initialise, [118](#)
  - ElementDestruction, [118](#)
  - ElementProduction, [118](#)
  - GetBracket, [118](#)
  - Grid, [120](#)
  - hotInjectionFraction, [120](#)
  - InitialiseLargeGrid, [119](#)
  - LoadLimongiYields, [119](#)
  - LoadMaederYields, [119](#)
  - LoadMarigoYields, [119](#)
  - LoadOrfeoYields, [119](#)
  - MassOffset, [120](#)
  - MetallicityInterpolation, [119](#)
  - operator(), [119](#)
  - Param, [121](#)
  - Process, [121](#)
  - PurityEnforce, [120](#)
  - RemnantLocation, [121](#)
  - RidgeStorage, [121](#)
  - SaveGrid, [120](#)
  - SourcePriority, [121](#)
  - StellarInject, [120](#)
  - YieldGrid, [117](#)
- YieldOutput
  - StarReservoir, [100](#)
- YieldPoint, [121](#)
  - Mass, [122](#)
  - Yield, [122](#)
  - YieldPoint, [122](#)
- YieldProcess
  - EnumSets.h, [138](#)
- YieldRidge, [123](#)
  - Points, [123](#)
  - Source, [123](#)
  - YieldRidge, [123](#)
  - Z, [124](#)
- YieldRoot
  - ResourceValues, [84](#)
- YieldsFrom
  - StarReservoir, [98](#)
- YieldSubdir
  - OutputValues, [76](#)
- YieldValues, [124](#)
  - CCSN\_MassCut, [125](#)
  - CODwarf\_MassCut, [125](#)
  - Collapse\_MassCut, [125](#)
  - ECSN\_Fraction, [125](#)
  - ECSN\_MassCut, [126](#)
  - Initialise, [125](#)
  - MassOverhang, [126](#)
  - NSM\_ActiveFraction, [126](#)
  - NSM\_DelayTime, [126](#)
  - NSM\_Scale, [126](#)
  - NSM\_TypicalMass, [126](#)
  - ProcessNames, [126](#)
  - ProcessTypes, [127](#)
  - SNII\_ActiveFraction, [127](#)
  - SNII\_DelayTime, [127](#)
  - SNII\_LongFraction, [127](#)
  - SNII\_LongScale, [127](#)
  - SNII\_ShortScale, [127](#)
  - SNII\_TypicalMass, [127](#)
  - TargetNi56Yield, [127](#)
  - YieldValues, [125](#)
- Z
  - YieldRidge, [124](#)
- ZerothMoment
  - Integral, [56](#)