RAMICESII

Generated by Doxygen 1.9.4

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

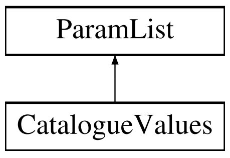
Chapter 4

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:

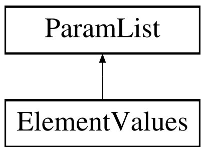
- /Users/jf20/Documents/Physics/RAMICES_II/src/Parameters/ParameterLists.h

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 Solar←Abundances 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 fo 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:

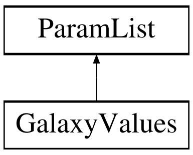
- /Users/jf20/Documents/Physics/RAMICES_II/src/Galaxy/Galaxy.h

4.4 GalaxyValues Class Reference

The galaxy suboptions contians 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 contians 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 amoutToHeat)

  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

givingGas	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

givingGas	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

amountToLose	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

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

mass	The total mass of the new reservoir
param	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 fractiongas 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

source	The value of Source 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

source	The value of Source which this object inherits
hot	The gas component which is copied into Hot
cold	The gas component which is copied into Cold

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

source	The value of Source which this object inherits
gas	The total gas mass of the new stream
hotFraction	the fraction of the gas which is put into Hot, 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

input	The object which donates the gas (unaltered)
hotFraction	The amount of the input which goes into the Hot 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

input	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

input	The object which donates the gas (unaltered)
fraction	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

amountToRemove	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

amountToRemove_Cold,the	amount of cold gas to lose
amountToRemove_Hot,the	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,&Mig

    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

- Integral MomentCompute (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:

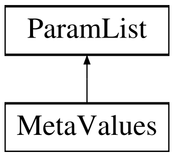
- /Users/jf20/Documents/Physics/RAMICES_II/src/Stars/RemnantPopulation.h

4.18 MetaValues Class Reference

The MetaValues contains variables associated with the base-level information about the sumulation - 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 sumulation - 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:

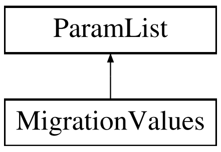
- /Users/jf20/Documents/Physics/RAMICES_II/src/Galaxy/MigrationMatrix.h

4.20 MigrationValues Class Reference

Holds values associated with how matter mvoes 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 kpc^2/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 mvoes 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 kpc^2/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.

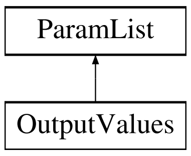
The documentation for this class was generated from the following file:

- /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:

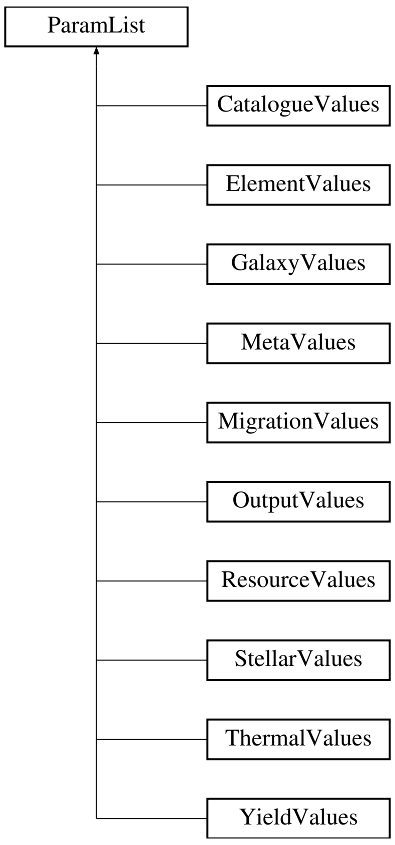
- /Users/jf20/Documents/Physics/RAMICES_II/src/Parameters/ParameterLists.h

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]
```

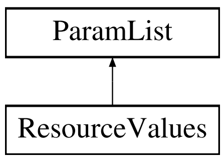
The documentation for this class was generated from the following file:

- /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 messgae.
- 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 messgae.

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 &logarithmic↩ StreamCold, 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 origin↩ Radius, 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 SNIa_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:

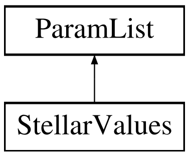
- /Users/jf20/Documents/Physics/RAMICES_II/src/Stars/StellarPopulation.h

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")

  Maxmimum 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 fo 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")
```

Maxmimum 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:

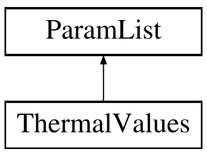
- /Users/jf20/Documents/Physics/RAMICES_II/src/Parameters/ParameterLists.h

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-sn1a")
```

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
```

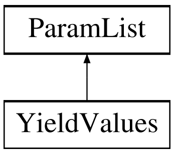
The documentation for this class was generated from the following file:

- /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 SNIa_ActiveFraction

```
Argument<double> YieldValues::SNIa_ActiveFraction = Argument<double>(0.1,"sn1a-fraction")
```

4.38.3.14 SNIa_DelayTime

```
Argument<double> YieldValues::SNIa_DelayTime = Argument<double>(0.2,"sn1a-delay")
```

Time before SNIa can turn on, in Gyr.

4.38.3.15 SNIa_LongFraction

```
Argument<double> YieldValues::SNIa_LongFraction = Argument<double>(0.99,"sn1a-fraction-long")
```

4.38.3.16 SNIa_LongScale

```
Argument<double> YieldValues::SNIa_LongScale = Argument<double>(1.5,"sn1a-long-decay")
```

4.38.3.17 SNIa_ShortScale

```
Argument<double> YieldValues::SNIa_ShortScale = Argument<double>(0.1,"sn1a-short-decay")
```

4.38.3.18 SNIa_TypicalMass

```
Argument<double> YieldValues::SNIa_TypicalMass = Argument<double>(1.37,"sn1a-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.

```
#pragma once
#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>

enum ParallelJob {RingStep, Compounding, Scattering, AssignIsochrones,Synthesis,Selection};

class Galaxy
{
    public:
        Galaxy(InitialisedData & Data);
        void Evolve();
        void SynthesiseObservations();
        std::vector<Ring> Rings;
    private:

        std::vector<std::thread> Threads;
        std::vector<MigrationMatrix> Migrator;
        GasReservoir CGM;
        const GlobalParameters & Param;

        void LaunchParallelOperation(int time,int nOperations,ParallelJob type);

        //Infall Stuff
        double GasScaleLength(double t);
        double InfallMass(double t);
        void InsertInfallingGas(int ring, double amount);
        void Infall(double t);

        //Star Formation

        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);

        static std::string MassHeaders();

        InitialisedData & Data;

        std::vector<double> RingMasses;

        void ComputeVisibilityFunction();
        void SelectionFunction(int ringstart, int ringend, int threadID);
        void StellarSynthesis(int ringstart, int ringend,int threadID);
        std::vector<std::string> SynthesisOutput;
        std::vector<double> SynthesisProgress;

        double DimmestStar;
        double BrightestStar;
        int ParallelBars = 0;

};
```

5.3 /Users/jf20/Documents/Physics/RAMICES_II/src/Galaxy/Migration↵ Matrix.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.

```
#pragma once
#include <vector>
#include "../Parameters/InitialisedData.h"
#include <iomanip>
class MigrationMatrix
{
    public:
        MigrationMatrix(InitialisedData & Data);

         std::vector<std::vector<double» Grid;

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

         void Compound(const MigrationMatrix & newTime);
         void Print();
     private:
         int NRings;
         const GlobalParameters & Param;
         std::vector<std::vector<double» DiagonalMultiply(const std::vector<std::vector<double» & a, const
     std::vector<std::vector<double» & b, int diagonalDistance);

};
```

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.

```
#pragma once
#include <vector>
#include <iomanip>
#include "../Parameters/InitialisedData.h"
#include "../Gas/GasReservoir.h"
#include "../Stars/StarReservoir.h"
#include "../Stars/IMF.h"
#include "../Stars/SLF.h"
class Ring
{
    public:
        Ring(int radiusIndex, double mass, InitialisedData & data);

        double Mass();
        const double Radius;
        const double Width;
        double Area;
        //Relic Reservoir
        StarReservoir Stars;
        GasReservoir Gas;
        GasReservoir CGMBuffer;
        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);
    private:

        //~ std::vector<GasReservoir> PreviousEnrichment;
        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;
};
```

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.

```
#pragma once
#include "../Parameters/GlobalParameters.h"

class Gas
{
    public:


        Gas();

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

        double Mass();

        double Mass() const;

        double & operator[](ElementID id);

        const double & operator[](ElementID id) const;

        static Gas Primordial(double mass);

        static Gas Empty();

     private:
        std::vector<double> Species;

        void CheckMass();

        bool NeedsRecomputing;
        double internal_Mass;

 };

```

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.

```
#pragma once
#include <vector>
#include "../Parameters/GlobalParameters.h"
#include "GasStream.h"
#include <sstream>
class GasReservoir
{
    public:

        GasReservoir();

        GasReservoir(const GlobalParameters & param);

        GasStream & operator[](SourceProcess source);

        const GasStream & operator[](SourceProcess source) const;


        double Mass();

        double ColdMass();

        double HotMass();

        void Absorb(const GasReservoir & givingGas);



        void Absorb(const GasStream & givingGas);

        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);


        void Wipe();

        void Deplete(double amountToLose_Cold, double amountToLose_Hot);

        void Heat(double amoutToHeat);

        void PassiveCool(double dt, bool isCGM);

        void TransferFrom(GasReservoir & givingGas, double massToMove);

        void TransferColdFrom(GasReservoir & givingGas, double massToMove);

        void TransferHotFrom(GasReservoir & givingGas, double massToMove);

        void TransferAndHeat(GasReservoir & givingGas, double massToMove);
        GasStream AccretionStream(double amountToLose);

        static GasReservoir Primordial(double mass, const GlobalParameters & param);

        double ColdGasMetallicity() const;


        const std::vector<GasStream> & Composition() const;
    private:

        std::vector<GasStream> Components;


        const GlobalParameters & Param;


};


```

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.

```
#pragma once
#include "../Parameters/GlobalParameters.h"
#include "Gas.h"
class GasStream
{
    public:
        SourceProcess Source;

        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();

        GasStream(SourceProcess source);

        GasStream(SourceProcess source,const Gas & hot, const Gas & cold);

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

        double Mass();

        double Mass() const;
        double HotMass();

        double ColdMass();

        double HotMass() const;

        double ColdMass() const;

        void Deplete(double amountToRemove);

        void Heat(double amountToHeat);

        void Cool(double amountToCool);
        //~ void DepleteFraction(double fraction)
        //~ {
                //~ Deplete(fraction* Mass());
        //~ }

        void Deplete(double amountToRemove_Cold, double amountToRemove_Hot);

        void Absorb(const GasStream & input);

        void Absorb(const GasStream & input,double fraction);

        void Absorb(const Gas & input, double hotFraction);

        void Dirty();


    private:
        bool NeedsRecomputing;

        double internal_HotMass;

        double internal_ColdMass;

        double internal_TotalMass;

        void ComputeMasses();

        Gas internal_Hot;

         Gas internal_Cold;
};

```

5.13 /Users/jf20/Documents/Physics/RAMICES_II/src/Parameters/Enum↩ Sets.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	
SNIa	
NSM	
AGB	
YieldCount	

5.14 EnumSets.h

Go to the documentation of this file.

```
#pragma once

enum ElementID
     {Hydrogen,Helium,Metals,Iron,Oxygen,Magnesium,Carbon,Silicon,Calcium,Manganese,Chromium,Cobalt,Europium,

    ElementCount

     };

enum SourceProcess {Accreted, Stellar, Remnant,

     ProcessCount

     };

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};
```

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.

```
#pragma once
#include <string>
#include <vector>
#include "JSL.h"
#include <sstream>
#include "ParameterLists.h"
#include "List.h"
#include "EnumSets.h"

#define PI 3.14159265358979323846

//Options classes are sets of initialisations of Argument objetcs, such that they can be seprated &
    categorised



class GlobalParameters
{

    public:
        MetaValues Meta;

        OutputValues Output;

        ResourceValues Resources;

        ElementValues Element;

        StellarValues Stellar;

        YieldValues Yield;

        ThermalValues Thermal;

        MigrationValues Migration;

        CatalogueValues Catalogue;

        GalaxyValues Galaxy;

        std::vector<ParamList *> ParamMembers =
    {&Meta,&Output,&Resources,&Element,&Stellar,&Thermal,&Galaxy,&Yield,&Migration,&Catalogue};

        GlobalParameters();

        void Initialise(int argc, char* argv[]);

        void SaveInputs();

};
```

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.

```
#pragma once
class YieldGrid;
class SimpleYield;
#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"

class InitialisedData
```

```
{
    public:
        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;
        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);
    private:
        std::default_random_engine generator;
        std::normal_distribution<double> distribution;


};
```

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.

```
#pragma once
#include "JSL.h"
#include <vector>
#include <string>
#include <sstream>
using JSL::Argument;

class ParamList
{
    public:
        void Configure(int argc, char * argv[]);

        void virtual Initialise(std::string resourceRoot){};

        void StreamContentsTo(std::stringstream & stream);
    protected:

        std::vector<JSL::ArgumentInterface *> argPointers;

};
```

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 sumulation - 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 mvoes throughout the disc.
- class CatalogueValues
- class GalaxyValues

  The galaxy suboptions contians 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.

```
#pragma once
#include "JSL.h"
#include "List.h"
#include "EnumSets.h"
using JSL::Argument;


class MetaValues : public ParamList
{

    public:
        Argument<int> Verbosity = Argument<int>(1, "verbose");

        Argument<int> ParallelThreads = Argument<int>(3,"thread");

        Argument<double> TimeStep  = Argument<double>(0.01,"timestep");

        //~ //!The total duration of the chemical simulation
        Argument<double> SimulationDuration = Argument<double>(10.0,"duration");

        Argument<int> ProgressHashes = Argument<int>(32,"progress-hashes");


        int SimulationSteps;

        MetaValues()
        {
            argPointers = {&Verbosity,&ParallelThreads,&TimeStep,&SimulationDuration,&ProgressHashes};
        };

        virtual void Initialise(std::string resourceRoot);
};


class OutputValues : public ParamList
{
    public:
        Argument<std::string> Root =  Argument<std::string>("Output/","output");

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

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

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

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

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



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

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

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

        std::string LogarithmicColdGasFile;
        std::string AbsoluteColdGasFile;
        std::string LogarithmicHotGasFile;
        std::string AbsoluteHotGasFile;

        OutputValues()
        {
            argPointers = {&Root, &GalaxyMassFile,&StarFile, &ChemicalPrefactor,&ColdGasDataFile,&HotGasDataFile,&YieldSubdir};
        };

        virtual void Initialise(std::string resourceRoot);
};

class ResourceValues : public ParamList
{
    public:
        Argument<std::string> WelcomeFile = Argument<std::string>("welcome.dat","welcome-file");

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

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


        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");
        ResourceValues()
        {
            argPointers = {&WelcomeFile, &ResourceRoot,&YieldRoot,&IsochroneDirectory,&LifeTimeFile,&IsochroneRepository};
        };

        virtual void Initialise(std::string resourceRoot);

};

class ElementValues : public ParamList
{

    public:

        std::vector<std::string> ElementNames;

        // The values of Z associated with each element
        std::vector<int> ProtonCounts;

        std::vector<double> SolarAbundances;

        Argument<std::string> SolarAbundanceFile = Argument<std::string>("ChemicalData/SolarAbundances_Maria.dat","solar-values-file");

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

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



        ElementValues()
        {
                argPointers = {&SolarAbundanceFile, &SolarAbundanceFileDataColumn, &SolarAbundanceFileNameColumn};
        };

        virtual void Initialise(std::string resourceRoot);

        void GiveElementsNames();

};

class StellarValues : public ParamList
{

    public:
        Argument<double> MaxStellarMass = Argument<double>(100,"mass-max");

        Argument<double> MinStellarMass = Argument<double>(0.1,"mass-min");

        Argument<double> ImmortalMass = Argument<double>(0.3,"mass-immortal");

        Argument<int> MassResolution = Argument<int>(199,"mass-resolution");

        std::vector<double> MassGrid;

        std::vector<double> MassDeltas;



        Argument<double> MinLogZ = Argument<double>(-6,"logz-min");

        Argument<double> MaxLogZ = Argument<double>(-0.1,"logz-max");

        Argument<int> LogZResolution = Argument<int>(100,"logz-resolution");

        std::vector<double> LogZGrid;
        double LogZDelta;

        Argument<double> EjectionFraction = Argument<double>(0.45,"eject");

        Argument<double> FeedbackFactor = Argument<double>(0.5,"mass-load");



        Argument<double> SchmidtMainPower = Argument<double>(1.4,"schmidt-main");

        Argument<double> SchmidtLowPower = Argument<double>(4.0,"schmidt-low");

        Argument<double> SchmidtDensityCut = Argument<double>(0,"schmidt-cut");

        Argument<double> SchmidtPrefactor = Argument<double>(2,"schmidt-factor");

        Argument<double> IMF_Slope = Argument<double>(2.3,"imf-slope");



        StellarValues()
        {
            argPointers = {&MaxStellarMass, &MinStellarMass, &ImmortalMass, &MassResolution, &MinLogZ, &MaxLogZ, &LogZResolution, &EjectionFraction,&SchmidtMainPower, &SchmidtLowPower, &SchmidtDensityCut, &SchmidtPrefactor, &FeedbackFactor};
        }

        void Initialise(std::string resourceRoot);
};


class YieldValues : public ParamList
{
    public:
        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");

        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");

        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");

    YieldValues()
    {
        argPointers = {&SNIa_DelayTime, &SNIa_ShortScale, &SNIa_LongScale, &NSM_DelayTime, &
    SNIa_ActiveFraction, &SNIa_LongFraction, &CCSN_MassCut,&NSM_ActiveFraction,
    &NSM_Scale,&SNIa_TypicalMass,&NSM_TypicalMass, &TargetNi56Yield, &ECSN_Fraction};
    }
        void Initialise(std::string resourceRoot);

};

class ThermalValues : public ParamList
{


    public:
        Argument<double> HotInjection_CCSN = Argument<double>(0.7,"fh-ccsn");

        Argument<double> HotInjection_NSM = Argument<double>(0.4,"fh-nsm");

        Argument<double> HotInjection_AGB = Argument<double>(0.7,"fh-agb");

        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");

        Argument<double> GasCoolingTimeScale = Argument<double>(1,"cool");

        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");
        ThermalValues()
        {
            argPointers = {&HotInjection_CCSN, &HotInjection_NSM, &HotInjection_SNIa,
    &GasCoolingTimeScale, &HotInjection_AGB, &NumericalResolution, &DormantHotFraction,&CoolingPower,
    &FeedbackEjectFactor,&ChimneyFactor};
        }
};


class MigrationValues: public ParamList
{

    public:
        Argument<bool> InflowActive = Argument<bool>(true,"inflow-on");
        Argument<double> InflowParameterA = Argument<double>(0.33,"inflow-a");

        Argument<double> InflowParameterB = Argument<double>(0.53,"inflow-b");

        Argument<double> MaxStealFraction = Argument<double>(0.95,"max-steal");

        Argument<double> MarkovDispersionStrength = Argument<double>(0.2,"mixing-strength");

        Argument<int> DispersionOrder = Argument<int>(3,"mixing-order");

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

        MigrationValues()
        {
            argPointers =
    {&InflowParameterA,&InflowParameterB,&MaxStealFraction,&MarkovDispersionStrength,&DispersionOrder,
    &DispersionTruncation, &InflowActive};
```

```
        }


};

class CatalogueValues: public ParamList
{
    public:

        Argument<bool> SynthesisActive = Argument<bool>(true,"stellar-synthesis");

        Argument<double> IsochroneTimeStep = Argument<double>(0.1,"isochrone-dt");

        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");

        CatalogueValues()
        {
            argPointers =
   {&SynthesisActive,&IsochroneTimeStep,&IsochroneMagnitudeResolution,&SolarRadius,&RadialResolution,&AzimuthalResolution,
        }

};

class GalaxyValues : public ParamList
{


    public:
        Argument<int> RingCount = Argument<int>(100,"rings");


        Argument<double> Radius = Argument<double>(20.0,"radius");

        Argument<bool> UsingVariableRingWidth = Argument<bool>(false,"variable-ring-width");

        Argument<double> Ring0Width = Argument<double>(0.05,"inner-ring-width");

        Argument<bool> CGMAbsorbing = Argument<bool>(true,"cgm-absorb");

        std::vector<double> RingRadius;
        std::vector<double> RingWidth;

        Argument<double> PrimordialMass = Argument<double>(2,"M0");


        Argument<double> PrimordialHotFraction = Argument<double>(0,"primordial-hot");

        Argument<double> CGM_Mass = Argument<double>(200,"cgm-mass");

        Argument<double> MinScaleLength = Argument<double>(0.75,"scale-length-min");

        Argument<double> MaxScaleLength = Argument<double>(3.75,"scale-length-max");

        Argument<double> ScaleLengthDelay = Argument<double>(1.0,"scale-length-delay");

        Argument<double> ScaleLengthTimeScale = Argument<double>(2.0,"scale-length-time");

        Argument<double> ScaleLengthFinalTime = Argument<double>(12.0,"scale-length-final");

        Argument<double> InfallMass1 = Argument<double>(50,"M1");

        Argument<double> InfallMass2 = Argument<double>(100,"M2");

        Argument<double> InfallTime1 = Argument<double>(0.4,"b1");

        Argument<double> InfallTime2 = Argument<double>(6.0,"b2");

        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");

        GalaxyValues()
        {
            argPointers = {&RingCount, &PrimordialMass, &PrimordialHotFraction, &CGM_Mass, &Radius,
     &MinScaleLength, &MaxScaleLength, &ScaleLengthDelay, &ScaleLengthTimeScale, &ScaleLengthFinalTime,
     &InfallMass1, &InfallMass2, &InfallTime1, &InfallTime2,  &MaxSFRFraction, &Ring0Width,
     &UsingVariableRingWidth, &CGMAbsorbing, &InfallMassMerger, & InfallTimeMerger, &
     MergerDelayTime,&MergerTurnOnWidth};
        }
        void Initialise(std::string resourceRoot);
};
```

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.

```
#pragma once
#include "../Parameters/GlobalParameters.h"



struct Integral
{
    double ZerothMoment;
    double FirstMoment;
};
class IMF_Functor
{
    public:
        IMF_Functor(const GlobalParameters & param);

        double operator()(double mass);

        double FormationCount(double formationMass) const;
        double Weighting(int i) const;

    private:
        const GlobalParameters & Param;

        double IMF_Normalisation;
        double IMF_MeanMass;
        Integral MomentCompute(double start,double stop, int resolution);
        void Normalise();
        double IMF(double mass);
        std::vector<double> IMF_Weighting;
};

```

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.

```
#pragma once
#include "../Parameters/GlobalParameters.h"
#include <filesystem>
#include <random>


enum IsochroneProperties {logL,BolometricMag, UMag,BMag,VMag, RMag,IMag,JMag, HMag,KMag, TEff, Logg,
      PropertyCount};
const std::vector<std::string> PropertyNames = {"logL", "BolometricMag", "UMag","BMag","VMag",
      "RMag","IMag","JMag","HMag","KMag","TEff","Logg"};
struct IsochroneEntry
{
    std::vector<double> Properties;
    IsochroneEntry()
    {
        Properties = std::vector<double>(PropertyCount,999.0);
    }
    double & operator[](IsochroneProperties p)
    {
        return Properties[p];
    }
    const double & operator[](IsochroneProperties p) const
    {
        return Properties[p];
    }
    int Countify()
    {
        return Properties.size();
    }
};
struct IsochroneCube
{
    std::vector<double> Weighting;
    std::vector<IsochroneEntry *> Data;
    int Count() const
    {
        return Data.size();
    }
    double Value(int entry, IsochroneProperties p) const
    {
        return Data[entry]->Properties[p];
    }

};
class IsochroneTracker
{


    public:
        IsochroneTracker(const GlobalParameters & param);
        void Construct();

        IsochroneCube GetProperties(int mass, double z, double age);

    private:
        const GlobalParameters Param;
        void IsoLog(std::string val);
        void ParseFile(std::string file);
        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;

        double NormalSample(double mu, double sigma);
        double UniformSample(double lowerBound, double upperBound);

        void ExtractSample(IsochroneCube & output, int sampleMass, double sampleZ, double sampleAge);
};
```

5.27 /Users/jf20/Documents/Physics/RAMICES_II/src/Stars/Remnant↩ Population.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.

```
#pragma once
#include "../Yields/YieldGrid.h"
#include "../Yields/SimpleYield.h"
#include "../Parameters/InitialisedData.h"
#include "StarEvents.h"
struct MassReport
{
    double Total;
    double WD;
     double NS;
     double BH;
};

class RemnantPopulation
{
     public:
         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:
         //the MASS of remnants created at each time
         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;
};
```

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.

```
#pragma once
#include "../Parameters/GlobalParameters.h"
#include <algorithm>
class SLF_Functor
{
    public:
        SLF_Functor(const GlobalParameters & Param);

        int operator() (int mass, double metallicity);
         double PredictLifetime(double mass, double logmetallicity);
     private:

         double ValueInquiry(int m, int z);
         const GlobalParameters & Param;

         double LifeTime(int mass, int metallicity);
         void PrecomputeGrid();
         std::vector<std::vector<double>> PrecomputedGrid;
         const double NotComputed = -1;

};
```

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.

```
#pragma once
#include <sstream>
class StarEvents
{
    public:
        double StarMassFormed;
        double NStarsFormed;
        double CCSN;
        double AGBDeaths;
        double NSM;
        double SNIa;
        double ECSN;
        double Efficiency;
        StarEvents();

        void AddHeaders(std::stringstream & output);
        void Save(std::stringstream & output,double timestep);
};
```

5.33 /Users/jf20/Documents/Physics/RAMICES_II/src/Stars/Star↩Reservoir.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.

```
#pragma once

#include <vector>
#include "../Parameters/InitialisedData.h"
#include "../Gas/GasReservoir.h"
#include "RemnantPopulation.h"
#include "IsochroneTracker.h"
#include "StellarPopulation.h"
#include "IsochroneTracker.h"
#include <sstream>
#include "IMF.h"
#include "SLF.h"
#include "StarEvents.h"
class StarReservoir
{
    public:
        StarReservoir(int parentRing, InitialisedData & data);
        std::vector<StellarPopulation> Population;
        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();
            std::vector<GasReservoir> YieldOutput;
    private:

        RemnantPopulation Remnants;

        double SFR_GasLoss(double coldMass, double hotMass,double ejectFactor);
        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;
};
```

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.

```
#pragma once
#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"



class IsoMass
{
    public:
        int MassIndex;
        double Count;
        double Metallicity;
        int BirthIndex;
        int DeathIndex;
        IsochroneCube Isochrone;
        IsoMass();
        IsoMass(double n, int m, double z, int birth, int death);
};


class StellarPopulation
{
    public:
        StellarPopulation(InitialisedData & data, int parentRing);

        void PrepareIMF();
        int BirthRadius;
        double Metallicity;
        int BirthIndex;
        int FormStars(double formingMass, int timeIndex, GasReservoir & formingGas);
        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::vector<IsoMass> Distribution;
        IsoMass ImmortalStars;

        std::vector<GasStream> BirthGas;

        std::string CatalogueHeaders();
        std::string CatalogueEntry(std::vector<int> popEntry, int m, double currentRadius, double
    birthRadius) const;
        double Age;
    private:
        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;

        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);

        Gas TempGas;
};
```

5.37 /Users/jf20/Documents/Physics/RAMICES_II/src/Yields/Simple↩ Yield.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.

```
#pragma once
#include "../Parameters/GlobalParameters.h"
#include "../Gas/GasReservoir.h"

class SimpleYield
{
    public:
        const SourceProcess Process;
        SimpleYield(const GlobalParameters & param, YieldProcess Process);

         void operator()(GasReservoir & scatteringReservoir, double nObjects)  const;

     private:

         double hotInjectionFraction;
         void NSM_Initialise();
         void SNIa_Initialise();

         void RemnantInject( GasReservoir & scatteringReservoir, double Nstars) const;
         std::vector<double> Grid;
         const GlobalParameters & Param;
};
```

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.

```
#pragma once

//~ #include "../Stars/RemnantPopulation.h"
#include "../Parameters/GlobalParameters.h"
#include "../Gas/GasReservoir.h"
#include "YieldRidge.h"
//~ #include <ostream>

struct RemnantOutput
{
    RemnantType Type;
    double Mass;
};
struct Interpolator
{
    int UpperID;
    int LowerID;
    double LinearFactor;
    double Interpolate(double lower, double upper)
    {
        double val = lower + LinearFactor*(upper - lower);
        if (val < 0 && (lower > 0 || upper > 0))
        {
            val = 0;
        }
        return val;
    }
};

class YieldGrid
{
    public:
        const SourceProcess Process;
        YieldGrid(const GlobalParameters & param, YieldProcess Process);

        RemnantOutput operator()(GasReservoir & scatteringReservoir, double Nstars, int mass, double z,
    const std::vector<GasStream> & birthGas) const;

    private:
        const GlobalParameters & Param;
        std::vector<std::vector<std::vector<double»> Grid;

        double hotInjectionFraction;

        void CCSN_Initialise();

        void AGB_Initialise();
        void ECSN_Initialise();
        void InitialiseLargeGrid(int mSize, int zSize);
        //~ GasStream TempStream;
        //allows the grid size to be truncated for CCSN etc.
        int MassOffset;
        RemnantOutput StellarInject( GasReservoir & scatteringReservoir, double Nstars, int mass, double
    z, const std::vector<GasStream> & birthGas) const;

        void LoadOrfeoYields();
        void LoadMarigoYields();
        void LoadLimongiYields();
        void LoadMaederYields();
        std::vector<std::vector<YieldRidge» RidgeStorage;
        int RemnantLocation;

        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;

        // Creation properties

        void CreateGrid();
        YieldBracket GetBracket(int id, double mass, double z, bool overhang);
        //~ std::vector<int> SourcePriority;
        std::vector<std::vector<int» SourcePriority;
        void SaveGrid(std::string name);
        void PurityEnforce();
};
```

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.

```
#pragma once

#include "../Parameters/GlobalParameters.h"
struct YieldPoint
{
    double Mass;
    double Yield;
    YieldPoint(double m, double y): Mass(m), Yield(y)
    {

     }
     YieldPoint()
     {
         YieldPoint(0,0);
     }
 };



 class YieldRidge
 {
     public:
         SourceID Source;
         double Z;
         std::vector<YieldPoint> Points;
         YieldRidge();
         YieldRidge(SourceID source, double z, int nPoints);

 };

 struct YieldBracket
 {
     bool isEnclosed;
     bool hasSingle;
     YieldRidge UpperRidge;
     YieldRidge LowerRidge;

     YieldBracket()
     {
         isEnclosed = false;
         hasSingle = false;
     }
     YieldBracket(YieldRidge r1, YieldRidge r2): UpperRidge(r1), LowerRidge(r2)
     {
         isEnclosed = true;
         hasSingle = false;
     }
     YieldBracket(YieldRidge r1): UpperRidge(r1), LowerRidge(r1)
     {
         isEnclosed = true;
         hasSingle = true;
     }

     double Interpolate(double mass, double z);
 };
```

Index