

Flourite Germanium Rush

Generated by Doxygen 1.9.1



<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 Class Documentation</b>	<b>3</b>
2.1 Algorithm_Parameters Struct Reference	3
2.1.1 Detailed Description	3
2.1.2 Member Data Documentation	3
2.1.2.1 crossover_rate	3
2.1.2.2 crossover_type_rate	4
2.1.2.3 early_stopping_generations	4
2.1.2.4 generation_step	4
2.1.2.5 mutation_max_step	4
2.1.2.6 mutation_rate	4
2.1.2.7 mutation_type_rate	4
2.1.2.8 population_size	4
2.2 CCircuit Class Reference	5
2.2.1 Detailed Description	5
2.2.2 Constructor & Destructor Documentation	6
2.2.2.1 CCircuit()	6
2.2.3 Member Function Documentation	6
2.2.3.1 Check_Velocity()	6
2.2.3.2 FillIDs()	6
2.2.3.3 mark_units()	7
2.2.3.4 ResetFlowrates()	7
2.2.3.5 ReturnFlowrates()	7
2.2.3.6 ReturnProfit()	8
2.2.3.7 SetInlet()	8
2.2.3.8 SolveCCircuit()	9
2.2.3.9 SystemFlowrates()	9
2.3 Circuit_Parameters Struct Reference	9
2.3.1 Detailed Description	10
2.4 CUnit Class Reference	10
2.4.1 Detailed Description	11
2.4.2 Member Function Documentation	11
2.4.2.1 CheckMassBalance()	11
2.4.2.2 ConcentrationCalc()	12
2.4.2.3 OutletCalc()	12
2.4.2.4 PrintCUnit()	12
2.4.2.5 ReactionComp()	12
2.4.2.6 ResidenceTime()	13
2.4.2.7 ReturnOutlets()	13
2.4.2.8 SolveCUnit()	13

2.5 Individual Struct Reference . . . . .	14
2.5.1 Detailed Description . . . . .	14
2.5.2 Member Data Documentation . . . . .	14
2.5.2.1 fitness_val . . . . .	14
2.5.2.2 vector . . . . .	14
<b>Index</b>	<b>15</b>

# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">Algorithm_Parameters</a>	
Contains parameters used to control the behavior of the genetic algorithm . . . . .	<a href="#">3</a>
<a href="#">CCircuit</a> . . . . .	<a href="#">5</a>
<a href="#">Circuit_Parameters</a> . . . . .	<a href="#">9</a>
<a href="#">CUnit</a> . . . . .	<a href="#">10</a>
<a href="#">Individual</a>	
Represents an individual in the population . . . . .	<a href="#">14</a>



## Chapter 2

# Class Documentation

### 2.1 Algorithm\_Parameters Struct Reference

Contains parameters used to control the behavior of the genetic algorithm.

```
#include <Genetic_Algorithm.h>
```

#### Public Attributes

- int [population\\_size](#)
- double [mutation\\_rate](#)
- double [crossover\\_rate](#)
- int [generation\\_step](#)
- int [mutation\\_max\\_step](#)
- int [early\\_stopping\\_generations](#)
- std::vector< double > [mutation\\_type\\_rate](#)
- std::vector< double > [crossover\\_type\\_rate](#)

#### 2.1.1 Detailed Description

Contains parameters used to control the behavior of the genetic algorithm.

#### 2.1.2 Member Data Documentation

##### 2.1.2.1 crossover\_rate

```
Algorithm_Parameters::crossover_rate
```

The chance that a crossover will occur during reproduction.

#### 2.1.2.2 crossover\_type\_rate

`Algorithm_Parameters::crossover_type_rate`

A vector that determines the relative probabilities of different types of crossovers.

#### 2.1.2.3 early\_stopping\_generations

`Algorithm_Parameters::early_stopping_generations`

The number of generations without improvement after which the algorithm should stop early.

#### 2.1.2.4 generation\_step

`Algorithm_Parameters::generation_step`

The maximum number of generations for which the algorithm should run.

#### 2.1.2.5 mutation\_max\_step

`Algorithm_Parameters::mutation_max_step`

The maximum amount by which an individual's value can change during a mutation.

#### 2.1.2.6 mutation\_rate

`Algorithm_Parameters::mutation_rate`

The chance that a mutation will occur during reproduction.

#### 2.1.2.7 mutation\_type\_rate

`Algorithm_Parameters::mutation_type_rate`

A vector that determines the relative probabilities of different types of mutations.

#### 2.1.2.8 population\_size

`Algorithm_Parameters::population_size`

Size of the population in each generation.

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

- `include/Genetic_Algorithm.h`



## 2.2 CCircuit Class Reference

```
#include <CCircuit.h>
```

### Public Member Functions

- [CCircuit](#) (int num\_units)  
*Constructor of the [CCircuit](#) class that initialises the vector of [CUnit](#) units.*
- [CCircuit](#) (int num\_units\_, const std::vector< int > &CCircuit\_vector\_)  
*Constructor of the [CCircuit](#) class with specified number of units and a circuit vector.*
- void [FillIDs](#) ()  
*Fills the IDs of the [CCircuit](#) units and initializes related data structures. This function fills the IDs of the [CCircuit](#) units and performs the following steps: Resizes the units, recycle\_mineral, and recycle\_waste vectors based on the number of units. Initializes the recycle\_mineral and recycle\_waste vectors with zeros. Assigns the appropriate IDs to each unit based on the CCircuit\_vector. Sets the IDs for the concentrate and tailing outlets of the system.*
- void [SystemFlowrates](#) (const double Fgi\_, const double Fwi\_)  
*Sets the system flow rates for mineral and waste inputs.*
- void [SolveCCircuit](#) ()  
*Solves the [CCircuit](#) by iterating through its units.*
- void [ReturnFlowrates](#) (std::vector< double > &Flowrates\_g, std::vector< double > &Flowrates\_w)  
*Returns the mineral and waste flow rates of each unit in the circuit.*
- double [ReturnProfit](#) ()  
*Calculates and returns the profit based on the current unit's data.*
- void [ResetFlowrates](#) ()  
*Resets the flow rates of all units in the circuit to zero.*
- void [SetInlet](#) ()  
*Sets the inlet flow rates for each unit in the circuit.*
- bool [Check\\_VValidity](#) (int \*circuit\_vector)  
*Function to check the validity of a circuit.*
- void [mark\\_units](#) (int unit\_num)  
*Function to mark all units accessible from the feed.*

### Public Attributes

- int **num\_units**
- double **system\_mineral\_input**
- double **system\_waste\_input**
- std::vector< double > **recycle\_mineral**
- std::vector< double > **recycle\_waste**
- std::vector< int > **CCircuit\_vector**
- std::vector< [CUnit](#) > **units**

### 2.2.1 Detailed Description

header file for the circuit

This header file defines the function that will be used to generate and validate the circuit

## 2.2.2 Constructor & Destructor Documentation

### 2.2.2.1 CCircuit()

```
CCircuit::CCircuit (
    int num_units_,
    const std::vector< int > & CCircuit_vector_ )
```

Constructor of the [CCircuit](#) class with specified number of units and a circuit vector.

This constructor initializes a [CCircuit](#) object with the given number of units and circuit vector.

#### Parameters

<i>num_units_</i>	The number of units in the circuit.
<i>CCircuit_↔ vector_</i>	The circuit vector containing information about the circuit.

## 2.2.3 Member Function Documentation

### 2.2.3.1 Check\_Validity()

```
bool CCircuit::Check_Validity (
    int * circuit_vector )
```

Function to check the validity of a circuit.

#### Parameters

<i>int</i>	* circuit_vector Pointer to a circuit vector that contains information about the circuit The function iterates through circuit to check its validity
------------	--

### 2.2.3.2 FillIDs()

```
void CCircuit::FillIDs ( )
```

Fills the IDs of the [CCircuit](#) units and initializes related data structures. This function fills the IDs of the [CCircuit](#) units and performs the following steps: Resizes the units, recycle\_mineral, and recycle\_waste vectors based on the number of units. Initializes the recycle\_mineral and recycle\_waste vectors with zeros. Assigns the appropriate IDs to each unit based on the CCircuit\_vector. Sets the IDs for the concentrate and tailing outlets of the system.

**Note**

This function assumes that the [CCircuit](#), units, and related data structures have been properly initialized.

**See also**

[CCircuit](#)

**Parameters**

<i>CCircuit_vector</i>	A vector containing the IDs of the units in the <a href="#">CCircuit</a> .
------------------------	--

**2.2.3.3 mark\_units()**

```
void CCircuit::mark_units (
    int unit_num )
```

Function to mark all units accessible from the feed.

**Parameters**

<i>int</i>	unit_num Integer value representing the initial unit This function is recursively called to mark all units that can be accessed from the feed
------------	---

**2.2.3.4 ResetFlowrates()**

```
void CCircuit::ResetFlowrates ( )
```

Resets the flow rates of all units in the circuit to zero.

This function sets the flow rates of all units in the circuit to zero. It iterates through each unit and resets various flow rate variables to zero, including Fti, Fgi, Fwi, Ftc, Fgc, Fwc, Ftt, Fgt, Fwt, Rg, Rw, and tau.

**2.2.3.5 ReturnFlowrates()**

```
void CCircuit::ReturnFlowrates (
    std::vector< double > & Flowrates_g,
    std::vector< double > & Flowrates_w )
```

Returns the mineral and waste flow rates of each unit in the circuit.

This function populates the provided vectors with the mineral and waste flow rates of each unit in the circuit. The function resizes the `Flowrates_g` and `Flowrates_w` vectors to accommodate the required number of flow rates. Then, it iterates through each unit and assigns the mineral and waste flow rates to the corresponding elements in the vectors.

**Parameters**

out	<i>Flowrates</i> ↔ _g	A vector that will be filled with the mineral flow rates of each unit.
out	<i>Flowrates</i> ↔ _w	A vector that will be filled with the waste flow rates of each unit.

**Note**

The function assumes that the `Flowrates_g` and `Flowrates_w` vectors have been properly initialized before calling this function.

**2.2.3.6 ReturnProfit()**

```
double CCircuit::ReturnProfit ( )
```

Calculates and returns the profit based on the current unit's data.

This function calculates the profit based on the current unit's data and returns the result.

**Returns**

The calculated profit based on the formula:  $\text{ProfitGerardium} * \text{Fgi} - \text{PenalizationWaste} * \text{Fwi}$ .

**2.2.3.7 SetInlet()**

```
void CCircuit::SetInlet ( )
```

Sets the inlet flow rates for each unit in the circuit.

This function sets the inlet flow rates for each unit in the circuit based on the system feed ID and the provided recycle mineral and waste vectors. It iterates through each unit and checks if the unit's index matches the system feed ID. If a match is found, the mineral and waste input flow rates ( $\text{Fgi}$  and  $\text{Fwi}$ ) as well as the total input flow rate ( $\text{Fti}$ ) are set to the sum of the corresponding recycle flow rates, system mineral input, and system waste input. For units that don't match the system feed ID, the inlet flow rates are set to the corresponding recycle flow rates.

**Note**

The function assumes that the `CCircuit_vector`, `recycle_mineral`, and `recycle_waste` vectors have been properly initialized before calling this function.

### 2.2.3.8 SolveCCircuit()

```
void CCircuit::SolveCCircuit ( )
```

Solves the [CCircuit](#) by iterating through its units.

This function solves the [CCircuit](#) by performing the following steps: Resets the flow rates. Sets the inlet. Initializes the recycle\_mineral and recycle\_waste vectors with zeros. Iterates through each unit in the circuit and solves it using SolveCUnit(). Updates the recycle\_mineral and recycle\_waste vectors based on the unit outputs.

#### Note

This function assumes that the [CCircuit](#) and its units have been properly initialized.

#### See also

[CCircuit](#), [SolveCUnit\(\)](#)

### 2.2.3.9 SystemFlowrates()

```
void CCircuit::SystemFlowrates (
    const double Fgi_,
    const double Fwi_ )
```

Sets the system flow rates for mineral and waste inputs.

This function sets the system flow rates for mineral and waste inputs based on the provided values.

#### Parameters

$Sg_{\leftrightarrow}$ —	The mineral input flow rate to be set for the system.
$Sw_{\leftrightarrow}$ —	The waste input flow rate to be set for the system.

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

- include/CCircuit.h
- src/CCircuit.cpp

## 2.3 Circuit\_Parameters Struct Reference

```
#include <CSimulator.h>
```

### Public Attributes

- double **tolerance**
- int **max\_iterations**

### 2.3.1 Detailed Description

header file for the circuit simulator

This header file defines the function that will be used to evaluate the circuit

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

- include/CSimulator.h

## 2.4 CUnit Class Reference

```
#include <CUnit.h>
```

### Public Member Functions

- void [ConcentrationCalc](#) ()  
*Calculates the concentrations of components in the [CUnit](#). This function calculates the concentrations of components in the [CUnit](#) by dividing the corresponding flow rates by the total flow rate.*
- void [ResidenceTime](#) ()  
*Calculates the residence time of the [CUnit](#).*
- void [ReactionComp](#) ()  
*Calculates the reaction components.*
- void [OutletCalc](#) ()  
*Calculates the outlet components.*
- int [CheckMassBalance](#) ()  
*Checks the mass balance of the unit.*
- void [SolveCUnit](#) ()  
*Solves the [CUnit](#) by performing necessary calculations. This function solves the [CUnit](#) by performing the following steps: Calculates the residence time. Performs reaction composition calculations. Calculates the outlet values. Calculates the concentration.*
- void [PrintCUnit](#) ()  
*Prints the [CUnit](#) information and performs a mass balance check. This function prints various components and flow rates at the inlet, concentrate, and tailing of the [CUnit](#). It also displays the spatial time and reaction rates. Finally, it calls the [CheckMassBalance\(\)](#) function to perform a mass balance check.*
- std::vector< double > [ReturnOutlets](#) ()  
*Returns the outlet components.*
- **CUnit** (int ID\_num\_, double Fgi\_, double Fwi\_)

## Public Attributes

- int **ID\_num**
- int **ID\_num\_conc**
- int **ID\_num\_tail**
- bool **mark**
- int **conc\_num**
- int **tails\_num**
- double **tau**
- double **Rg**
- double **Rw**
- double **Fti**
- double **Fgi**
- double **Fwi**
- double **Cgi**
- double **Cwi**
- double **Ftc**
- double **Fgc**
- double **Fwc**
- double **Cgc**
- double **Cwc**
- double **Ftt**
- double **Fgt**
- double **Fwt**
- double **Cgt**
- double **Cwt**

### 2.4.1 Detailed Description

header file for the units of the ircuit

This header file defines the function that will be used to initialize the units in the circuit

### 2.4.2 Member Function Documentation

#### 2.4.2.1 CheckMassBalance()

```
int CUnit::CheckMassBalance ( )
```

Checks the mass balance of the unit.

This method checks the mass balance of the unit by comparing the inlet and outlet components and printing the results. It performs the following checks:

- Global Total mass balance
- Global Gerardium mass balance
- Global Waste mass balance
- Mass balance at the inlet
- Mass balance at the concentrate
- Mass balance at the tailing
- Global general mass balance

#### 2.4.2.2 ConcentrationCalc()

```
void CUnit::ConcentrationCalc ( )
```

Calculates the concentrations of components in the [CUnit](#). This function calculates the concentrations of components in the [CUnit](#) by dividing the corresponding flow rates by the total flow rate.

##### Note

This function assumes that the [CUnit](#) and its related data have been properly initialized.

##### See also

[CUnit](#)

#### 2.4.2.3 OutletCalc()

```
void CUnit::OutletCalc ( )
```

Calculates the outlet components.

This method calculates the outlet components based on the reaction rates and the input components.

#### 2.4.2.4 PrintCUnit()

```
void CUnit::PrintCUnit ( )
```

Prints the [CUnit](#) information and performs a mass balance check. This function prints various components and flow rates at the inlet, concentrate, and tailing of the [CUnit](#). It also displays the spatial time and reaction rates. Finally, it calls the [CheckMassBalance\(\)](#) function to perform a mass balance check.

##### Note

This function assumes that the [CUnit](#) and its related data have been properly initialized.

##### See also

[CUnit](#), [CheckMassBalance\(\)](#)

#### 2.4.2.5 ReactionComp()

```
void CUnit::ReactionComp ( )
```

Calculates the reaction components.

This method calculates the reaction components based on the reaction rate constants for Gerardium and waste.



### 2.4.2.6 ResidenceTime()

```
void CUnit::ResidenceTime ( )
```

Calculates the residence time of the [CUnit](#).

This function calculates the residence time of the [CUnit](#) based on the given parameters: gamma, V, pg, pw, Fgi, and Fwi.

#### Note

This function assumes that the [CUnit](#) and its related data have been properly initialized.

#### See also

[CUnit](#)

### 2.4.2.7 ReturnOutlets()

```
std::vector< double > CUnit::ReturnOutlets ( )
```

Returns the outlet components.

This method returns a vector containing the outlet components: Gerardium component (Fgc), waste component (Fwc), total Gerardium component (Fgt), and total waste component (Fwt).

#### Returns

Vector containing the outlet components.

### 2.4.2.8 SolveCUnit()

```
void CUnit::SolveCUnit ( )
```

Solves the [CUnit](#) by performing necessary calculations. This function solves the [CUnit](#) by performing the following steps: Calculates the residence time. Performs reaction composition calculations. Calculates the outlet values. Calculates the concentration.

#### Note

This function assumes that the [CUnit](#) and its related data have been properly initialized.

#### See also

[CUnit](#), [ResidenceTime\(\)](#), [ReactionComp\(\)](#), [OutletCalc\(\)](#), [ConcentrationCalc\(\)](#)

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

- include/CUnit.h
- src/CUnit.cpp

## 2.5 Individual Struct Reference

Represents an individual in the population.

```
#include <Genetic_Algorithm.h>
```

### Public Attributes

- `std::vector< int >` [vector](#)
- `double` [fitness\\_val](#)

### 2.5.1 Detailed Description

Represents an individual in the population.

### 2.5.2 Member Data Documentation

#### 2.5.2.1 `fitness_val`

```
Individual::fitness_val
```

The fitness value of the individual, calculated using a provided fitness function.

#### 2.5.2.2 `vector`

```
Individual::vector
```

A vector representing the state of the individual.

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

- `include/Genetic_Algorithm.h`

# Index

- Algorithm\_Parameters, 3
  - crossover\_rate, 3
  - crossover\_type\_rate, 3
  - early\_stopping\_generations, 4
  - generation\_step, 4
  - mutation\_max\_step, 4
  - mutation\_rate, 4
  - mutation\_type\_rate, 4
  - population\_size, 4
- CCircuit, 5
  - CCircuit, 6
  - Check\_Validity, 6
  - FillIDs, 6
  - mark\_units, 7
  - ResetFlowrates, 7
  - ReturnFlowrates, 7
  - ReturnProfit, 8
  - SetInlet, 8
  - SolveCCircuit, 8
  - SystemFlowrates, 9
- Check\_Validity
  - CCircuit, 6
- CheckMassBalance
  - CUnit, 11
- Circuit\_Parameters, 9
- ConcentrationCalc
  - CUnit, 11
- crossover\_rate
  - Algorithm\_Parameters, 3
- crossover\_type\_rate
  - Algorithm\_Parameters, 3
- CUnit, 10
  - CheckMassBalance, 11
  - ConcentrationCalc, 11
  - OutletCalc, 12
  - PrintCUnit, 12
  - ReactionComp, 12
  - ResidenceTime, 12
  - ReturnOutlets, 13
  - SolveCUnit, 13
- early\_stopping\_generations
  - Algorithm\_Parameters, 4
- FillIDs
  - CCircuit, 6
- fitness\_val
  - Individual, 14
- generation\_step
- Algorithm\_Parameters, 4
- Individual, 14
  - fitness\_val, 14
  - vector, 14
- mark\_units
  - CCircuit, 7
- mutation\_max\_step
  - Algorithm\_Parameters, 4
- mutation\_rate
  - Algorithm\_Parameters, 4
- mutation\_type\_rate
  - Algorithm\_Parameters, 4
- OutletCalc
  - CUnit, 12
- population\_size
  - Algorithm\_Parameters, 4
- PrintCUnit
  - CUnit, 12
- ReactionComp
  - CUnit, 12
- ResetFlowrates
  - CCircuit, 7
- ResidenceTime
  - CUnit, 12
- ReturnFlowrates
  - CCircuit, 7
- ReturnOutlets
  - CUnit, 13
- ReturnProfit
  - CCircuit, 8
- SetInlet
  - CCircuit, 8
- SolveCCircuit
  - CCircuit, 8
- SolveCUnit
  - CUnit, 13
- SystemFlowrates
  - CCircuit, 9
- vector
  - Individual, 14