barry: Your go-to motif accountant

0.0-1

Generated by Doxygen 1.9.1

Chapter 1

Main Page

Barry: your to-go motif accountant

This repository contains a C++ template library that essentially counts sufficient statistics on binary arrays. The idea of the library is that this can be used together to build exponential family models as those in Exponential Random Graph Models (ERGMs), but as a generalization that also deals with non square arrays.

Examples

Counting statistics in a graph

In the following code we create an array of size 5x5 of class `Network` (available in the namespace `netcounters`), add/remove ties, print the graph, and count common statistics used in ERGMs:

```
#include <iostream>
#include <ostream>
#include "../include/barry.hpp"
typedef std::vector< unsigned int > vuint;
int main() {
  // Creating network of size six with five ties
  netcounters::Network net(
      6, 6,
      {0, 0, 4, 4, 2, 0, 1},
      {1, 2, 0, 2, 4, 0, 1}
  );

  // How does this looks like?
  std::cout « "Current view" « std::endl;
  net.print();

  // Adding extra ties
  net += {1, 0};
  net(2, 0) = true;

  // And removing a couple
  net(0, 0) = false;
  net -= {1, 1};
  std::cout « "New view" « std::endl;
  net.print();

  // Initializing the data. The program deals with freing the memory
  net.set_data(new netcounters::NetworkData, true);
  // Creating counter object for the network and adding stats to count
  netcounters::NetStatsCounter counter(&net);
  netcounters::counter_edges(counter.counters);
```

```
  netcounters::counter_ttriads(counter.counters);
  netcounters::counter_isolates(counter.counters);
  netcounters::counter_ctriads(counter.counters);
  netcounters::counter_mutual(counter.counters);

  // Counting and printing the results
  std::vector< double > counts = counter.count_all();

  std::cout «
    "Edges             : " « counts[0] « std::endl «
    "Transitive triads : " « counts[1] « std::endl «
    "Isolates          : " « counts[2] « std::endl «
    "C triads          : " « counts[3] « std::endl «
    "Mutuals           : " « counts[4] « std::endl;

  return 0;
}
```

Compiling this program using g++

```
g++ -std=c++11 -Wall -pedantic 08-counts.cpp -o counts && ./counts
```

Yields the following output:

```
Current view
[  0,]  1  1  1  .  .  .
[  1,]  .  1  .  .  .  .
[  2,]  .  .  .  .  1  .
[  3,]  .  .  .  .  .  .
[  4,]  1  .  1  .  .  .
[  5,]  .  .  .  .  .  .
New view
[  0,]  .  1  1  .  .  .
[  1,]  1  .  .  .  .  .
[  2,]  1  .  .  .  1  .
[  3,]  .  .  .  .  .  .
[  4,]  1  .  1  .  .  .
[  5,]  .  .  .  .  .  .
Edges             : 7
Transitive triads : 3
Isolates          : 2
C triads          : 1
Mutuals           : 3
```

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 barry Namespace Reference

barry: Your go-to motif accountant

Namespaces

- counters

  Tree class and TreeIterator class.

5.1.1 Detailed Description

barry: Your go-to motif accountant

5.2 barry::counters Namespace Reference

Tree class and TreeIterator class.

Namespaces

- network
- phylo

5.2.1 Detailed Description

Tree class and TreeIterator class.

5.3 barry::counters::network Namespace Reference

5.4 barry::counters::phylo Namespace Reference

5.5 CHECK Namespace Reference

Integer constants used to specify which cell should be check.

Variables

- const int BOTH = -1
- const int NONE = 0
- const int ONE = 1
- const int TWO = 2

5.5.1 Detailed Description

Integer constants used to specify which cell should be check.

5.5.2 Variable Documentation

5.5.2.1 BOTH

```
const int CHECK::BOTH = -1
```

Definition at line 33 of file typedefs.hpp.

5.5.2.2 NONE

```
const int CHECK::NONE = 0
```

Definition at line 34 of file typedefs.hpp.

5.5.2.3 ONE

```
const int CHECK::ONE = 1
```

Definition at line 35 of file typedefs.hpp.

5.5.2.4 TWO

```
const int CHECK::TWO = 2
```

Definition at line 36 of file typedefs.hpp.

5.6 EXISTS Namespace Reference

Integer constants used to specify which cell should be check to exist or not.

Variables

- const int BOTH = -1
- const int NONE = 0
- const int ONE = 1
- const int TWO = 1
- const int UKNOWN = -1
- const int AS_ZERO = 0
- const int AS_ONE = 1

5.6.1 Detailed Description

Integer constants used to specify which cell should be check to exist or not.

5.6.2 Variable Documentation

5.6.2.1 AS_ONE

```
const int EXISTS::AS_ONE = 1
```

Definition at line 51 of file typedefs.hpp.

5.6.2.2 AS_ZERO

```
const int EXISTS::AS_ZERO = 0
```

Definition at line 50 of file typedefs.hpp.

5.6.2.3 BOTH

```
const int EXISTS::BOTH = -1
```

Definition at line 44 of file typedefs.hpp.

5.6.2.4 NONE

```
const int EXISTS::NONE = 0
```

Definition at line 45 of file typedefs.hpp.

5.6.2.5 ONE

```
const int EXISTS::ONE = 1
```

Definition at line 46 of file typedefs.hpp.

5.6.2.6 TWO

```
const int EXISTS::TWO = 1
```

Definition at line 47 of file typedefs.hpp.

5.6.2.7 UKNOWN

```
const int EXISTS::UKNOWN = -1
```

Definition at line 49 of file typedefs.hpp.

Chapter 6

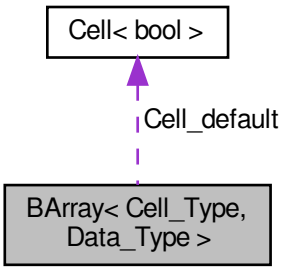
Class Documentation

6.1 BArray< Cell_Type, Data_Type > Class Template Reference

Baseline class for binary arrays.

`#include <barray-bones.hpp>`

Collaboration diagram for BArray< Cell_Type, Data_Type >:



Public Member Functions

- bool operator== (const BArray< Cell_Type, Data_Type > &Array_)
- ~BArray ()
- void set_data (Data_Type *data_, bool delete_data_=false)
- void out_of_range (uint i, uint j) const
- Cell_Type get_cell (uint i, uint j, bool check_bounds=true) const
- const Row_type< Cell_Type > * get_row (uint i, bool check_bounds=true) const
- const Col_type< Cell_Type > * get_col (uint i, bool check_bounds=true) const
- std::vector< Cell_Type > get_col_vec (uint i, bool check_bounds=true) const
- std::vector< Cell_Type > get_row_vec (uint i, bool check_bounds=true) const
- void get_col_vec (std::vector< Cell_Type > *x, uint i, bool check_bounds=true) const

- void get_row_vec (std::vector< Cell_Type > *x, uint i, bool check_bounds=true) const
- const Row_type< Cell_Type > & row (uint i, bool check_bounds=true) const
- const Col_type< Cell_Type > & col (uint i, bool check_bounds=true) const
- Entries< Cell_Type > get_entries () const

  Get the edgelist.
- void transpose ()
- void clear (bool hard=true)
- void resize (uint N_, uint M_)
- void reserve ()
- void print () const

Constructors

Parameters

N_	*Number of rows*
M_	*Number of columns*
source	*An unsigned vector ranging from 0 to N_*
target	*An unsigned int vector ranging from 0 to M_*
target	*When `true` tries to add repeated observations.*

- BArray ()

  Zero-size array.
- BArray (uint N_, uint M_)

  Empty array.
- BArray (uint N_, uint M_, const std::vector< uint > &source, const std::vector< uint > &target, const std::vector< Cell_Type > &value, bool add=true)

  Edgelist with data.
- BArray (uint N_, uint M_, const std::vector< uint > &source, const std::vector< uint > &target, bool add=true)

  Edgelist with no data (simpler)
- BArray (const BArray< Cell_Type, Data_Type > &Array_, bool copy_data=false)

  Copy constructor.
- BArray< Cell_Type, Data_Type > & operator= (const BArray< Cell_Type, Data_Type > &Array_)

  Assignment constructor.
- BArray (BArray< Cell_Type, Data_Type > &&x) noexcept

  Move operator.
- BArray< Cell_Type, Data_Type > & operator= (BArray< Cell_Type, Data_Type > &&x) noexcept

  Move assignment.

Queries

`is_empty` queries a single cell. `nrow`, `ncol`, and `nnozero` return the number of rows, columns, and non-zero cells respectively.

Parameters

i,j	*Coordinates*
check_bounds	*If `false` avoids checking bounds.*

- bool is_empty (uint i, uint j, bool check_bounds=true) const
- uint nrow () const
- uint ncol () const
- uint nnozero () const

Cell-wise insertion/deletion

Parameters

i,j	*Row,column*
check_bounds	*When `true` and out of range, the function throws an error.*
check_exists	*Wither check if the cell exists (before trying to delete/add), or, in the case of `swap_cells`, check if either of both cells exists/don't exist.*

- BArray< Cell_Type, Data_Type > & operator+= (const std::pair< uint, uint > &coords)
- BArray< Cell_Type, Data_Type > & operator-= (const std::pair< uint, uint > &coords)
- BArrayCell< Cell_Type, Data_Type > operator() (uint i, uint j, bool check_bounds=true)
- const BArrayCell_const< Cell_Type, Data_Type > operator() (uint i, uint j, bool check_bounds=true) const
- void rm_cell (uint i, uint j, bool check_bounds=true, bool check_exists=true)
- void insert_cell (uint i, uint j, const Cell< Cell_Type > &v, bool check_bounds, bool check_exists)
- void insert_cell (uint i, uint j, Cell< Cell_Type > &&v, bool check_bounds, bool check_exists)
- void insert_cell (uint i, uint j, Cell_Type v, bool check_bounds, bool check_exists)
- void swap_cells (uint i0, uint j0, uint i1, uint j1, bool check_bounds=true, int check_exists=CHECK::BOTH, int *report=nullptr)
- void toggle_cell (uint i, uint j, bool check_bounds=true, int check_exists=EXISTS::UKNOWN)
- void toggle_lock (uint i, uint j, bool check_bounds=true)

Column/row wise interchange

- void swap_rows (uint i0, uint i1, bool check_bounds=true)
- void swap_cols (uint j0, uint j1, bool check_bounds=true)
- void zero_row (uint i, bool check_bounds=true)
- void zero_col (uint j, bool check_bounds=true)

Arithmetic operators

- BArray< Cell_Type, Data_Type > & operator+= (const BArray< Cell_Type, Data_Type > &rhs)
- BArray< Cell_Type, Data_Type > & operator+= (const Cell_Type &rhs)
- BArray< Cell_Type, Data_Type > & operator-= (const BArray< Cell_Type, Data_Type > &rhs)
- BArray< Cell_Type, Data_Type > & operator-= (const Cell_Type &rhs)
- BArray< Cell_Type, Data_Type > & operator/= (const Cell_Type &rhs)
- BArray< Cell_Type, Data_Type > & operator*= (const Cell_Type &rhs)

Public Attributes

- uint N
- uint M
- uint NCells = 0u
- std::vector< Row_type< Cell_Type > > el_ij
- std::vector< Col_type< Cell_Type > > el_ji
- Data_Type * data = nullptr
- bool delete_data = false
- bool visited = false

Static Public Attributes

- static Cell< Cell_Type > Cell_default

Friends

- class BArrayCell< Cell_Type, Data_Type >
- class BArrayCell_const< Cell_Type, Data_Type >

6.1.1 Detailed Description

template<typename Cell_Type = bool, typename Data_Type = bool>
class BArray< Cell_Type, Data_Type >

Baseline class for binary arrays.

`BArray` class objects are arbitrary arrays in which non-empty cells hold data of type `Cell_Type`. The non-empty cells are stored by row and indexed using `unordered_map`s, i.e. `std::vector< std::unordered_map<unsigned int,Cell_Type> >`.

Template Parameters

Cell_Type	Type of cell (any type).
Data_Type	Data type of the array (bool default).

Definition at line 22 of file barray-bones.hpp.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 BArray() [1/6]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray< Cell_Type, Data_Type >::BArray ( )  [inline]
```

Zero-size array.

Definition at line 58 of file barray-bones.hpp.

6.1.2.2 BArray() [2/6]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray< Cell_Type, Data_Type >::BArray (
            uint N_,
            uint M_ )  [inline]
```

Empty array.

Definition at line 61 of file barray-bones.hpp.

6.1.2.3 BArray() [3/6]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray< Cell_Type, Data_Type >::BArray (
            uint N_,
            uint M_,
            const std::vector< uint > & source,
            const std::vector< uint > & target,
            const std::vector< Cell_Type > & value,
            bool add = true )
```

Edgelist with data.

6.1.2.4 BArray() [4/6]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray< Cell_Type, Data_Type >::BArray (
            uint N_,
            uint M_,
            const std::vector< uint > & source,
            const std::vector< uint > & target,
            bool add = true )
```

Edgelist with no data (simpler)

6.1.2.5 BArray() [5/6]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray< Cell_Type, Data_Type >::BArray (
            const BArray< Cell_Type, Data_Type > & Array_,
            bool copy_data = false )
```

Copy constructor.

6.1.2.6 BArray() [6/6]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray< Cell_Type, Data_Type >::BArray (
            BArray< Cell_Type, Data_Type > && x )  [noexcept]
```

Move operator.

6.1.2.7 ~BArray()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray< Cell_Type, Data_Type >::~BArray ( )
```

6.1.3 Member Function Documentation

6.1.3.1 clear()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::clear (
            bool hard = true )
```

6.1.3.2 col()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
const Col_type< Cell_Type >& BArray< Cell_Type, Data_Type >::col (
            uint i,
            bool check_bounds = true ) const
```

6.1.3.3 get_cell()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
Cell_Type BArray< Cell_Type, Data_Type >::get_cell (
            uint i,
            uint j,
            bool check_bounds = true ) const
```

6.1.3.4 get_col()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
const Col_type< Cell_Type >* BArray< Cell_Type, Data_Type >::get_col (
            uint i,
            bool check_bounds = true ) const
```

6.1.3.5 get_col_vec() [1/2]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::get_col_vec (
            std::vector< Cell_Type > * x,
            uint i,
            bool check_bounds = true ) const
```

6.1.3.6 get_col_vec() [2/2]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
std::vector< Cell_Type > BArray< Cell_Type, Data_Type >::get_col_vec (
            uint i,
            bool check_bounds = true ) const
```

6.1.3.7 get_entries()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
Entries<Cell_Type> BArray< Cell_Type, Data_Type >::get_entries ( ) const
```

Get the edgelist.

Entries is a class with three objects: Two std::vector with the row and column coordinates respectively, and one std::vector with the corresponding value of the cell.

Returns

Entries<Cell_Type>

6.1.3.8 get_row()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
const Row_type< Cell_Type >* BArray< Cell_Type, Data_Type >::get_row (
            uint i,
            bool check_bounds = true ) const
```

6.1.3.9 get_row_vec() [1/2]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::get_row_vec (
            std::vector< Cell_Type > * x,
            uint i,
            bool check_bounds = true ) const
```

6.1.3.10 get_row_vec() [2/2]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
std::vector< Cell_Type > BArray< Cell_Type, Data_Type >::get_row_vec (
            uint i,
            bool check_bounds = true ) const
```

6.1.3.11 insert_cell() [1/3]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::insert_cell (
            uint i,
            uint j,
            Cell< Cell_Type > && v,
            bool check_bounds,
            bool check_exists )
```

6.1.3.12 insert_cell() [2/3]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::insert_cell (
            uint i,
            uint j,
            Cell_Type v,
            bool check_bounds,
            bool check_exists )
```

6.1.3.13 insert_cell() [3/3]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::insert_cell (
            uint i,
            uint j,
            const Cell< Cell_Type > & v,
            bool check_bounds,
            bool check_exists )
```

6.1.3.14 is_empty()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
bool BArray< Cell_Type, Data_Type >::is_empty (
            uint i,
            uint j,
            bool check_bounds = true ) const
```

6.1.3.15 ncol()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
uint BArray< Cell_Type, Data_Type >::ncol ( ) const
```

6.1.3.16 nnozero()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
uint BArray< Cell_Type, Data_Type >::nnozero ( ) const
```

6.1.3.17 nrow()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
uint BArray< Cell_Type, Data_Type >::nrow ( ) const
```

6.1.3.18 operator()() [1/2]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArrayCell<Cell_Type,Data_Type> BArray< Cell_Type, Data_Type >::operator() (
            uint i,
            uint j,
            bool check_bounds = true )
```

6.1.3.19 operator()() [2/2]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
const BArrayCell_const<Cell_Type,Data_Type> BArray< Cell_Type, Data_Type >::operator() (
            uint i,
            uint j,
            bool check_bounds = true ) const
```

6.1.3.20 operator*=()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray<Cell_Type,Data_Type>& BArray< Cell_Type, Data_Type >::operator*= (
            const Cell_Type & rhs )
```

6.1.3.21 operator+=() [1/3]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray<Cell_Type,Data_Type>& BArray< Cell_Type, Data_Type >::operator+= (
            const BArray< Cell_Type, Data_Type > & rhs )
```

6.1.3.22 operator+=() [2/3]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray<Cell_Type,Data_Type>& BArray< Cell_Type, Data_Type >::operator+= (
            const Cell_Type & rhs )
```

6.1.3.23 operator+=() [3/3]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray<Cell_Type,Data_Type>& BArray< Cell_Type, Data_Type >::operator+= (
            const std::pair< uint, uint > & coords )
```

6.1.3.24 operator-=() [1/3]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray<Cell_Type,Data_Type>& BArray< Cell_Type, Data_Type >::operator-= (
            const BArray< Cell_Type, Data_Type > & rhs )
```

6.1.3.25 operator-=() [2/3]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray<Cell_Type,Data_Type>& BArray< Cell_Type, Data_Type >::operator-= (
            const Cell_Type & rhs )
```

6.1.3.26 operator-=() [3/3]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray<Cell_Type,Data_Type>& BArray< Cell_Type, Data_Type >::operator-= (
            const std::pair< uint, uint > & coords )
```

6.1.3.27 operator/=()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray<Cell_Type,Data_Type>& BArray< Cell_Type, Data_Type >::operator/= (
            const Cell_Type & rhs )
```

6.1.3.28 operator=() [1/2]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray<Cell_Type,Data_Type>& BArray< Cell_Type, Data_Type >::operator= (
            BArray< Cell_Type, Data_Type > && x )  [noexcept]
```

Move assignment.

6.1.3.29 operator=() [2/2]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArray<Cell_Type,Data_Type>& BArray< Cell_Type, Data_Type >::operator= (
            const BArray< Cell_Type, Data_Type > & Array_ )
```

Assignment constructor.

6.1.3.30 operator==()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
bool BArray< Cell_Type, Data_Type >::operator== (
            const BArray< Cell_Type, Data_Type > & Array_ )
```

6.1.3.31 out_of_range()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::out_of_range (
            uint i,
            uint j ) const
```

6.1.3.32 print()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::print ( ) const
```

6.1.3.33 reserve()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::reserve ( )
```

6.1.3.34 resize()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::resize (
            uint N_,
            uint M_ )
```

6.1.3.35 rm_cell()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::rm_cell (
            uint i,
            uint j,
            bool check_bounds = true,
            bool check_exists = true )
```

6.1.3.36 row()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
const Row_type< Cell_Type >& BArray< Cell_Type, Data_Type >::row (
            uint i,
            bool check_bounds = true ) const
```

6.1.3.37 set_data()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::set_data (
            Data_Type * data_,
            bool delete_data_ = false )
```

6.1.3.38 swap_cells()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::swap_cells (
            uint i0,
            uint j0,
            uint i1,
            uint j1,
            bool check_bounds = true,
            int check_exists = CHECK::BOTH,
            int * report = nullptr )
```

6.1.3.39 swap_cols()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::swap_cols (
            uint j0,
            uint j1,
            bool check_bounds = true )
```

6.1.3.40 swap_rows()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::swap_rows (
            uint i0,
            uint i1,
            bool check_bounds = true )
```

6.1.3.41 toggle_cell()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::toggle_cell (
            uint i,
            uint j,
            bool check_bounds = true,
            int check_exists = EXISTS::UKNOWN )
```

6.1.3.42 toggle_lock()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::toggle_lock (
            uint i,
            uint j,
            bool check_bounds = true )
```

6.1.3.43 transpose()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::transpose ( )
```

6.1.3.44 zero_col()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::zero_col (
            uint j,
            bool check_bounds = true )
```

6.1.3.45 zero_row()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::zero_row (
            uint i,
            bool check_bounds = true )
```

6.1.4 Friends And Related Function Documentation

6.1.4.1 BArrayCell< Cell_Type, Data_Type >

```
template<typename Cell_Type = bool, typename Data_Type = bool>
friend class BArrayCell< Cell_Type, Data_Type >  [friend]
```

Definition at line 1 of file barray-bones.hpp.

6.1.4.2 BArrayCell_const< Cell_Type, Data_Type >

```
template<typename Cell_Type = bool, typename Data_Type = bool>
friend class BArrayCell_const< Cell_Type, Data_Type >  [friend]
```

Definition at line 1 of file barray-bones.hpp.

6.1.5 Member Data Documentation

6.1.5.1 Cell_default

```
template<typename Cell_Type = bool, typename Data_Type = bool>
Cell< Cell_Type > BArray< Cell_Type, Data_Type >::Cell_default  [static]
```

Definition at line 34 of file barray-bones.hpp.

6.1.5.2 data

```
template<typename Cell_Type = bool, typename Data_Type = bool>
Data_Type* BArray< Cell_Type, Data_Type >::data = nullptr
```

Definition at line 31 of file barray-bones.hpp.

6.1.5.3 delete_data

```
template<typename Cell_Type = bool, typename Data_Type = bool>
bool BArray< Cell_Type, Data_Type >::delete_data = false
```

Definition at line 32 of file barray-bones.hpp.

6.1.5.4 el_ij

```
template<typename Cell_Type = bool, typename Data_Type = bool>
std::vector< Row_type< Cell_Type > > BArray< Cell_Type, Data_Type >::el_ij
```

Definition at line 29 of file barray-bones.hpp.

6.1.5.5 el_ji

```
template<typename Cell_Type = bool, typename Data_Type = bool>
std::vector< Col_type< Cell_Type > > BArray< Cell_Type, Data_Type >::el_ji
```

Definition at line 30 of file barray-bones.hpp.

6.1.5.6 M

```
template<typename Cell_Type = bool, typename Data_Type = bool>
uint BArray< Cell_Type, Data_Type >::M
```

Definition at line 27 of file barray-bones.hpp.

6.1.5.7 N

```
template<typename Cell_Type = bool, typename Data_Type = bool>
uint BArray< Cell_Type, Data_Type >::N
```

Definition at line 26 of file barray-bones.hpp.

6.1.5.8 NCells

```
template<typename Cell_Type = bool, typename Data_Type = bool>
uint BArray< Cell_Type, Data_Type >::NCells = 0u
```

Definition at line 28 of file barray-bones.hpp.

6.1.5.9 visited

```
template<typename Cell_Type = bool, typename Data_Type = bool>
bool BArray< Cell_Type, Data_Type >::visited = false
```

This is as a reference, if we need to iterate through the cells and we need to keep track which were visited, we use this as a reference. So that if cell.visited = true and visited = true, it means that we haven't been here yet. Ideally, any routine using this->visited should switch it at the beginning of the routine.

Definition at line 43 of file barray-bones.hpp.

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

- include/barry/barray-bones.hpp

6.2 BArrayCell< Cell_Type, Data_Type > Class Template Reference

```
#include <barraycell-bones.hpp>
```

Public Member Functions

- BArrayCell (BArray< Cell_Type, Data_Type > *Array_, uint i_, uint j_, bool check_bounds=true)
- ~BArrayCell ()
- void operator= (const Cell_Type &val)
- void operator+= (const Cell_Type &val)
- void operator-= (const Cell_Type &val)
- void operator*= (const Cell_Type &val)
- void operator/= (const Cell_Type &val)
- operator Cell_Type () const
- bool operator== (const Cell_Type &val) const

6.2.1 Detailed Description

template<typename Cell_Type = bool, typename Data_Type = bool>
class BArrayCell< Cell_Type, Data_Type >

Definition at line 7 of file barraycell-bones.hpp.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 BArrayCell()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArrayCell< Cell_Type, Data_Type >::BArrayCell (
            BArray< Cell_Type, Data_Type > * Array_,
            uint i_,
            uint j_,
            bool check_bounds = true )  [inline]
```

Definition at line 16 of file barraycell-bones.hpp.

6.2.2.2 ~BArrayCell()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArrayCell< Cell_Type, Data_Type >::~BArrayCell ( )  [inline]
```

Definition at line 28 of file barraycell-bones.hpp.

6.2.3 Member Function Documentation

6.2.3.1 operator Cell_Type()

```
template<typename Cell_Type , typename Data_Type >
BArrayCell< Cell_Type, Data_Type >::operator Cell_Type  [inline]
```

Definition at line 58 of file barraycell-meat.hpp.

6.2.3.2 operator*=()

```
template<typename Cell_Type , typename Data_Type >
void BArrayCell< Cell_Type, Data_Type >::operator*= (
            const Cell_Type & val )  [inline]
```

Definition at line 40 of file barraycell-meat.hpp.

6.2.3.3 operator+=()

```
template<typename Cell_Type , typename Data_Type >
void BArrayCell< Cell_Type, Data_Type >::operator+= (
            const Cell_Type & val )  [inline]
```

Definition at line 18 of file barraycell-meat.hpp.

6.2.3.4 operator-=()

```
template<typename Cell_Type , typename Data_Type >
void BArrayCell< Cell_Type, Data_Type >::operator-= (
            const Cell_Type & val )  [inline]
```

Definition at line 29 of file barraycell-meat.hpp.

6.2.3.5 operator/=()

```
template<typename Cell_Type , typename Data_Type >
void BArrayCell< Cell_Type, Data_Type >::operator/= (
            const Cell_Type & val )  [inline]
```

Definition at line 49 of file barraycell-meat.hpp.

6.2.3.6 operator=()

```
template<typename Cell_Type , typename Data_Type >
void BArrayCell< Cell_Type, Data_Type >::operator= (
            const Cell_Type & val )  [inline]
```

Definition at line 7 of file barraycell-meat.hpp.

6.2.3.7 operator==()

```
template<typename Cell_Type , typename Data_Type >
bool BArrayCell< Cell_Type, Data_Type >::operator== (
             const Cell_Type & val ) const  [inline]
```

Definition at line 63 of file barraycell-meat.hpp.

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

- include/barry/barraycell-bones.hpp
- include/barry/barraycell-meat.hpp

6.3 BArrayCell_const< Cell_Type, Data_Type > Class Template Reference

```
#include <barraycell-bones.hpp>
```

Public Member Functions

- BArrayCell_const (const BArray< Cell_Type, Data_Type > *Array_, uint i_, uint j_, bool check_bounds=true)
- ~BArrayCell_const ()
- operator Cell_Type () const
- bool operator== (const Cell_Type &val) const
- bool operator!= (const Cell_Type &val) const
- bool operator< (const Cell_Type &val) const
- bool operator> (const Cell_Type &val) const
- bool operator<= (const Cell_Type &val) const
- bool operator>= (const Cell_Type &val) const

6.3.1 Detailed Description

template<typename Cell_Type = bool, typename Data_Type = bool>
class BArrayCell_const< Cell_Type, Data_Type >

Definition at line 41 of file barraycell-bones.hpp.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 BArrayCell_const()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArrayCell_const< Cell_Type, Data_Type >::BArrayCell_const (
            const BArray< Cell_Type, Data_Type > * Array_,
            uint i_,
            uint j_,
            bool check_bounds = true )  [inline]
```

Definition at line 50 of file barraycell-bones.hpp.

6.3.2.2 ~BArrayCell_const()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArrayCell_const< Cell_Type, Data_Type >::~BArrayCell_const ( )  [inline]
```

Definition at line 62 of file barraycell-bones.hpp.

6.3.3 Member Function Documentation

6.3.3.1 operator Cell_Type()

```
template<typename Cell_Type , typename Data_Type >
BArrayCell_const< Cell_Type, Data_Type >::operator Cell_Type  [inline]
```

Definition at line 68 of file barraycell-meat.hpp.

6.3.3.2 operator"!=()

```
template<typename Cell_Type , typename Data_Type >
bool BArrayCell_const< Cell_Type, Data_Type >::operator!= (
            const Cell_Type & val ) const  [inline]
```

Definition at line 78 of file barraycell-meat.hpp.

6.3.3.3 operator<()

```
template<typename Cell_Type , typename Data_Type >
bool BArrayCell_const< Cell_Type, Data_Type >::operator< (
            const Cell_Type & val ) const  [inline]
```

Definition at line 83 of file barraycell-meat.hpp.

6.3.3.4 operator<=()

```
template<typename Cell_Type , typename Data_Type >
bool BArrayCell_const< Cell_Type, Data_Type >::operator<= (
            const Cell_Type & val ) const  [inline]
```

Definition at line 93 of file barraycell-meat.hpp.

6.3.3.5 operator==()

```
template<typename Cell_Type , typename Data_Type >
bool BArrayCell_const< Cell_Type, Data_Type >::operator== (
            const Cell_Type & val ) const  [inline]
```

Definition at line 73 of file barraycell-meat.hpp.

6.3.3.6 operator>()

```
template<typename Cell_Type , typename Data_Type >
bool BArrayCell_const< Cell_Type, Data_Type >::operator> (
            const Cell_Type & val ) const  [inline]
```

Definition at line 88 of file barraycell-meat.hpp.

6.3.3.7 operator>=()

```
template<typename Cell_Type , typename Data_Type >
bool BArrayCell_const< Cell_Type, Data_Type >::operator>= (
            const Cell_Type & val ) const  [inline]
```

Definition at line 98 of file barraycell-meat.hpp.

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

- include/barry/barraycell-bones.hpp
- include/barry/barraycell-meat.hpp

6.4 Cell< Cell_Type > Class Template Reference

Entries in BArray. For now, it only has two members:

```
#include <cell-bones.hpp>
```

Public Member Functions

- Cell ()
- Cell (Cell_Type value_, bool visited_=false)
- ~Cell ()
- Cell (const Cell< Cell_Type > &arg)
- Cell< Cell_Type > & operator= (Cell< Cell_Type > &other)
- Cell (Cell< Cell_Type > &&arg) noexcept
- Cell< Cell_Type > & operator= (Cell< Cell_Type > &&other) noexcept
- void add (Cell_Type x)
- operator Cell_Type () const
- void add (double x)
- void add (unsigned int x)
- void add (int x)
- Cell ()
- Cell ()
- Cell ()

Public Attributes

- Cell_Type value
- bool visited

6.4.1 Detailed Description

template<class Cell_Type>
class Cell< Cell_Type >

Entries in BArray. For now, it only has two members:

- value: the content
- visited: boolean (just a convenient)

Definition at line 13 of file cell-bones.hpp.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 Cell() [1/7]

```
template<class Cell_Type >
Cell< Cell_Type >::Cell ( )
```

6.4.2.2 Cell() [2/7]

```
template<class Cell_Type >
Cell< Cell_Type >::Cell (
            Cell_Type value_,
            bool visited_ = false )  [inline]
```

Definition at line 18 of file cell-bones.hpp.

6.4.2.3 ~Cell()

```
template<class Cell_Type >
Cell< Cell_Type >::~Cell ( )  [inline]
```

Definition at line 20 of file cell-bones.hpp.

6.4.2.4 Cell() [3/7]

```
template<class Cell_Type >
Cell< Cell_Type >::Cell (
            const Cell< Cell_Type > & arg )  [inline]
```

Definition at line 24 of file cell-bones.hpp.

6.4.2.5 Cell() [4/7]

```
template<class Cell_Type >
Cell< Cell_Type >::Cell (
            Cell< Cell_Type > && arg )  [inline], [noexcept]
```

Definition at line 30 of file cell-bones.hpp.

6.4.2.6 Cell() [5/7]

```
Cell< double >::Cell ( )  [inline]
```

Definition at line 44 of file cell-meat.hpp.

6.4.2.7 Cell() [6/7]

```
Cell< uint >::Cell ( )  [inline]
```

Definition at line 45 of file cell-meat.hpp.

6.4.2.8 Cell() [7/7]

```
Cell< int >::Cell ( )  [inline]
```

Definition at line 46 of file cell-meat.hpp.

6.4.3 Member Function Documentation

6.4.3.1 add() [1/4]

```
template<class Cell_Type >
void Cell< Cell_Type >::add (
            Cell_Type x )
```

6.4.3.2 add() [2/4]

```
void Cell< double >::add (
            double x )  [inline]
```

Definition at line 24 of file cell-meat.hpp.

6.4.3.3 add() [3/4]

```
void Cell< int >::add (
            int x )  [inline]
```

Definition at line 34 of file cell-meat.hpp.

6.4.3.4 add() [4/4]

```
void Cell< unsigned int >::add (
            unsigned int x )  [inline]
```

Definition at line 29 of file cell-meat.hpp.

6.4.3.5 operator Cell_Type()

```
template<class Cell_Type >
Cell< Cell_Type >::operator Cell_Type ( ) const  [inline]
```

Definition at line 41 of file cell-bones.hpp.

6.4.3.6 operator=() [1/2]

```
template<typename Cell_Type >
Cell< Cell_Type > & Cell< Cell_Type >::operator= (
            Cell< Cell_Type > && other )  [noexcept]
```

Definition at line 14 of file cell-meat.hpp.

6.4.3.7 operator=() [2/2]

```
template<typename Cell_Type >
Cell< Cell_Type > & Cell< Cell_Type >::operator= (
            Cell< Cell_Type > & other )
```

Definition at line 7 of file cell-meat.hpp.

6.4.4 Member Data Documentation

6.4.4.1 value

```
template<class Cell_Type >
Cell_Type Cell< Cell_Type >::value
```

Definition at line 15 of file cell-bones.hpp.

6.4.4.2 visited

```
template<class Cell_Type >
bool Cell< Cell_Type >::visited
```

Definition at line 16 of file cell-bones.hpp.

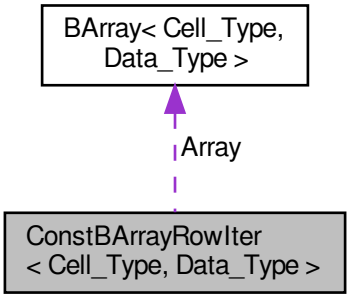
The documentation for this class was generated from the following files:

- include/barry/cell-bones.hpp
- include/barry/cell-meat.hpp

6.5 ConstBArrayRowIter< Cell_Type, Data_Type > Class Template Reference

```
#include <barray-iterator.hpp>
```

Collaboration diagram for ConstBArrayRowIter< Cell_Type, Data_Type >:



Public Member Functions

- ConstBArrayRowIter (const BArray< Cell_Type, Data_Type > *Array_)
- ~ConstBArrayRowIter ()

Public Attributes

- uint current_row
- uint current_col
- Row_type< Cell_Type >::const_iterator iter
- const BArray< Cell_Type, Data_Type > * Array

6.5.1 Detailed Description

template<typename Cell_Type, typename Data_Type>
class ConstBArrayRowIter< Cell_Type, Data_Type >

Definition at line 10 of file barray-iterator.hpp.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 ConstBArrayRowIter()

```
template<typename Cell_Type , typename Data_Type >
ConstBArrayRowIter< Cell_Type, Data_Type >::ConstBArrayRowIter (
            const BArray< Cell_Type, Data_Type > * Array_ )  [inline]
```

Definition at line 17 of file barray-iterator.hpp.

6.5.2.2 ~ConstBArrayRowIter()

```
template<typename Cell_Type , typename Data_Type >
ConstBArrayRowIter< Cell_Type, Data_Type >::~ConstBArrayRowIter ( )  [inline]
```

Definition at line 29 of file barray-iterator.hpp.

6.5.3 Member Data Documentation

6.5.3.1 Array

```
template<typename Cell_Type , typename Data_Type >
const BArray<Cell_Type,Data_Type>* ConstBArrayRowIter< Cell_Type, Data_Type >::Array
```

Definition at line 15 of file barray-iterator.hpp.

6.5.3.2 current_col

```
template<typename Cell_Type , typename Data_Type >
uint ConstBArrayRowIter< Cell_Type, Data_Type >::current_col
```

Definition at line 13 of file barray-iterator.hpp.

6.5.3.3 current_row

```
template<typename Cell_Type , typename Data_Type >
uint ConstBArrayRowIter< Cell_Type, Data_Type >::current_row
```

Definition at line 13 of file barray-iterator.hpp.

6.5.3.4 iter

```
template<typename Cell_Type , typename Data_Type >
Row_type<Cell_Type>::const_iterator ConstBArrayRowIter< Cell_Type, Data_Type >::iter
```

Definition at line 14 of file barray-iterator.hpp.

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

- include/barry/barray-iterator.hpp

6.6 Counter< Array_Type, Data_Type > Class Template Reference

A counter function based on change statistics.

```
#include <counters-bones.hpp>
```

Public Member Functions

- Counter< Array_Type, Data_Type > operator= (const Counter< Array_Type, Data_Type > &counter_)
- ~Counter ()
- double count (Array_Type &Array, uint i, uint j)
- double init (Array_Type &Array, uint i, uint j)

Creator passing a counter and an initializer

Parameters

count_fun_	*The main counter function.*
init_fun_	*The initializer function can also be used to check if the `BArray` as the needed variables (see BArray::data).*
data_	*Data to be used with the counter.*
delete_data_	*When `true`, the destructor will delete the pointer in the main data.*

- Counter ()
- Counter (Counter_fun_type< Array_Type, Data_Type > count_fun_, Counter_fun_type< Array_Type, Data_Type > init_fun_=nullptr, Data_Type *data_=nullptr, bool delete_data_=false)

- Counter (const Counter< Array_Type, Data_Type > &counter_)

Public Attributes

- Counter_fun_type< Array_Type, Data_Type > count_fun
- Counter_fun_type< Array_Type, Data_Type > init_fun
- Data_Type * data = nullptr
- bool delete_data = false

6.6.1 Detailed Description

template<typename Array_Type = BArray<>, typename Data_Type = bool>
class Counter< Array_Type, Data_Type >

A counter function based on change statistics.

This class is used by `CountStats` and `StatsCounter` as a way to count statistics using change statistics.

Definition at line 15 of file counters-bones.hpp.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 Counter() [1/3]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Counter< Array_Type, Data_Type >::Counter ( )  [inline]
```

Definition at line 34 of file counters-bones.hpp.

6.6.2.2 Counter() [2/3]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Counter< Array_Type, Data_Type >::Counter (
            Counter_fun_type< Array_Type, Data_Type > count_fun_,
            Counter_fun_type< Array_Type, Data_Type > init_fun_ = nullptr,
            Data_Type * data_ = nullptr,
            bool delete_data_ = false )  [inline]
```

Definition at line 36 of file counters-bones.hpp.

6.6.2.3 Counter() [3/3]

```
template<typename Array_Type , typename Data_Type >
Counter< Array_Type, Data_Type >::Counter (
            const Counter< Array_Type, Data_Type > & counter_ )  [inline]
```

Definition at line 7 of file counters-meat.hpp.

6.6.2.4 ~Counter()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Counter< Array_Type, Data_Type >::~Counter ( )  [inline]
```

Definition at line 48 of file counters-bones.hpp.

6.6.3 Member Function Documentation

6.6.3.1 count()

```
template<typename Array_Type , typename Data_Type >
double Counter< Array_Type, Data_Type >::count (
            Array_Type & Array,
            uint i,
            uint j )  [inline]
```

Definition at line 100 of file counters-meat.hpp.

6.6.3.2 init()

```
template<typename Array_Type , typename Data_Type >
double Counter< Array_Type, Data_Type >::init (
            Array_Type & Array,
            uint i,
            uint j )  [inline]
```

Definition at line 108 of file counters-meat.hpp.

6.6.3.3 operator=()

```
template<typename Array_Type , typename Data_Type >
Counter< Array_Type, Data_Type > Counter< Array_Type, Data_Type >::operator= (
            const Counter< Array_Type, Data_Type > & counter_ )
```

Definition at line 24 of file counters-meat.hpp.

6.6.4 Member Data Documentation

6.6.4.1 count_fun

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Counter_fun_type<Array_Type,Data_Type> Counter< Array_Type, Data_Type >::count_fun
```

Definition at line 18 of file counters-bones.hpp.

6.6.4.2 data

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Data_Type* Counter< Array_Type, Data_Type >::data = nullptr
```

Definition at line 20 of file counters-bones.hpp.

6.6.4.3 delete_data

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
bool Counter< Array_Type, Data_Type >::delete_data = false
```

Definition at line 21 of file counters-bones.hpp.

6.6.4.4 init_fun

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Counter_fun_type<Array_Type,Data_Type> Counter< Array_Type, Data_Type >::init_fun
```

Definition at line 19 of file counters-bones.hpp.

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

- include/barry/counters-bones.hpp
- include/barry/counters-meat.hpp

6.7 Counters< Array_Type, Data_Type > Class Template Reference

Vector of counters.

```
#include <counters-bones.hpp>
```

Public Member Functions

- Counters ()
- ~Counters ()
- Counters (const Counters< Array_Type, Data_Type > &counter_)
- Counters< Array_Type, Data_Type > operator= (const Counters< Array_Type, Data_Type > &counter_)
- Counter< Array_Type, Data_Type > & operator[ ] (uint idx)

  Returns a pointer to a particular counter.
- uint size () const

  Number of counters in the set.
- void add_counter (Counter< Array_Type, Data_Type > &counter)
- void add_counter (Counter< Array_Type, Data_Type > *counter)
- void add_counter (Counter_fun_type< Array_Type, Data_Type > count_fun_, Counter_fun_type< Array_Type, Data_Type > init_fun_=nullptr, Data_Type *data_=nullptr, bool delete_data_=false)
- void clear ()

6.7.1 Detailed Description

template<typename Array_Type = BArray<>, typename Data_Type = bool>
class Counters< Array_Type, Data_Type >

Vector of counters.

Various functions hold more than one counter, so this class is a helper class that allows managing multiple counters efficiently. The main data is a vector to pointers of counters.

Definition at line 69 of file counters-bones.hpp.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 Counters() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Counters< Array_Type, Data_Type >::Counters ( )  [inline]
```

Definition at line 78 of file counters-bones.hpp.

6.7.2.2 ~Counters()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Counters< Array_Type, Data_Type >::~Counters ( )  [inline]
```

Definition at line 81 of file counters-bones.hpp.

6.7.2.3 Counters() [2/2]

```
template<typename Array_Type , typename Data_Type >
Counters< Array_Type, Data_Type >::Counters (
            const Counters< Array_Type, Data_Type > & counter_ )  [inline]
```

Definition at line 48 of file counters-meat.hpp.

6.7.3 Member Function Documentation

6.7.3.1 add_counter() [1/3]

```
template<typename Array_Type , typename Data_Type >
void Counters< Array_Type, Data_Type >::add_counter (
            Counter< Array_Type, Data_Type > & counter )  [inline]
```

Definition at line 121 of file counters-meat.hpp.

6.7.3.2 add_counter() [2/3]

```
template<typename Array_Type , typename Data_Type >
void Counters< Array_Type, Data_Type >::add_counter (
            Counter< Array_Type, Data_Type > * counter )  [inline]
```

Definition at line 132 of file counters-meat.hpp.

6.7.3.3 add_counter() [3/3]

```
template<typename Array_Type , typename Data_Type >
void Counters< Array_Type, Data_Type >::add_counter (
            Counter_fun_type< Array_Type, Data_Type > count_fun_,
            Counter_fun_type< Array_Type, Data_Type > init_fun_ = nullptr,
            Data_Type * data_ = nullptr,
            bool delete_data_ = false )  [inline]
```

Definition at line 142 of file counters-meat.hpp.

6.7.3.4 clear()

```
template<typename Array_Type , typename Data_Type >
void Counters< Array_Type, Data_Type >::clear  [inline]
```

Definition at line 166 of file counters-meat.hpp.

6.7.3.5 operator=()

```
template<typename Array_Type , typename Data_Type >
Counters< Array_Type, Data_Type > Counters< Array_Type, Data_Type >::operator= (
            const Counters< Array_Type, Data_Type > & counter_ )
```

Definition at line 72 of file counters-meat.hpp.

6.7.3.6 operator[]()

```
template<typename Array_Type , typename Data_Type >
Counter< Array_Type, Data_Type > & Counters< Array_Type, Data_Type >::operator[] (
            uint idx )  [inline]
```

Returns a pointer to a particular counter.

Parameters

idx	Id of the counter

Returns

Counter<Array_Type,Data_Type>*

Definition at line 116 of file counters-meat.hpp.

6.7.3.7 size()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
uint Counters< Array_Type, Data_Type >::size ( ) const  [inline]
```

Number of counters in the set.

Returns

uint

Definition at line 101 of file counters-bones.hpp.

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

- include/barry/counters-bones.hpp
- include/barry/counters-meat.hpp

6.8 Entries< Cell_Type > Class Template Reference

A wrapper class to store `source`, `target`, `val` from a `BArray` object.

`#include <typedefs.hpp>`

Public Member Functions

- Entries ()
- Entries (uint n)
- ~Entries ()
- void resize (uint n)

Public Attributes

- std::vector< uint > source
- std::vector< uint > target
- std::vector< Cell_Type > val

6.8.1 Detailed Description

template<typename Cell_Type>
class Entries< Cell_Type >

A wrapper class to store `source`, `target`, `val` from a `BArray` object.

Template Parameters

Cell_Type	Any type

Definition at line 72 of file typedefs.hpp.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Entries() [1/2]

```
template<typename Cell_Type >
Entries< Cell_Type >::Entries ( )  [inline]
```

Definition at line 78 of file typedefs.hpp.

6.8.2.2 Entries() [2/2]

```
template<typename Cell_Type >
Entries< Cell_Type >::Entries (
            uint n )  [inline]
```

Definition at line 79 of file typedefs.hpp.

6.8.2.3 ~Entries()

```
template<typename Cell_Type >
Entries< Cell_Type >::~Entries ( )  [inline]
```

Definition at line 86 of file typedefs.hpp.

6.8.3 Member Function Documentation

6.8.3.1 resize()

```
template<typename Cell_Type >
void Entries< Cell_Type >::resize (
            uint n )  [inline]
```

Definition at line 88 of file typedefs.hpp.

6.8.4 Member Data Documentation

6.8.4.1 source

```
template<typename Cell_Type >
std::vector< uint > Entries< Cell_Type >::source
```

Definition at line 74 of file typedefs.hpp.

6.8.4.2 target

```
template<typename Cell_Type >
std::vector< uint > Entries< Cell_Type >::target
```

Definition at line 75 of file typedefs.hpp.

6.8.4.3 val

```
template<typename Cell_Type >
std::vector< Cell_Type > Entries< Cell_Type >::val
```

Definition at line 76 of file typedefs.hpp.

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

- include/barry/typedefs.hpp

6.9 Flock Class Reference

A Flock is a group of Geese.

```
#include <flock-bones.hpp>
```

Public Member Functions

- Flock ()
- ~Flock ()
- unsigned int add_data (std::vector< std::vector< unsigned int > > &annotations, std::vector< unsigned int > &geneid, std::vector< int > &parent, std::vector< bool > &duplication)
- void set_seed (const unsigned int &s)
- void init ()
- phylocounters::PhyloCounters * counters_ptr ()
- double likelihood_joint (const std::vector< double > &par, bool as_log=false, bool use_likelihood_↩ sequence=true)

Information about the model

- unsigned int nfuns () const
- unsigned int ntrees () const
- std::vector< unsigned int > nnodes () const
- std::vector< unsigned int > nleafs () const
- unsigned int nterms () const

Public Attributes

- std::vector< Geese > dat
- unsigned int nfunctions = 0u
- bool initialized = false
- std::mt19937 rengine
- phylocounters::PhyloModel support = phylocounters::PhyloModel()

6.9.1 Detailed Description

A Flock is a group of Geese.

This object buils a model with multiple trees (Geese objects), with all of these using the same PhyloModel object.

Definition at line 12 of file flock-bones.hpp.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Flock()

```
Flock::Flock ( )  [inline]
```

Definition at line 23 of file flock-bones.hpp.

6.9.2.2 ~Flock()

```
Flock::~Flock ( )  [inline]
```

Definition at line 24 of file flock-bones.hpp.

6.9.3 Member Function Documentation

6.9.3.1 add_data()

```
unsigned int Flock::add_data (
            std::vector< std::vector< unsigned int > > & annotations,
            std::vector< unsigned int > & geneid,
            std::vector< int > & parent,
            std::vector< bool > & duplication )  [inline]
```

Definition at line 6 of file flock-meet.hpp.

6.9.3.2 counters_ptr()

```
phylocounters::PhyloCounters * Flock::counters_ptr ( )  [inline]
```

Definition at line 75 of file flock-meet.hpp.

6.9.3.3 init()

```
void Flock::init ( )  [inline]
```

Definition at line 41 of file flock-meet.hpp.

6.9.3.4 likelihood_joint()

```
double Flock::likelihood_joint (
            const std::vector< double > & par,
            bool as_log = false,
            bool use_likelihood_sequence = true )  [inline]
```

Definition at line 84 of file flock-meet.hpp.

6.9.3.5 nfuns()

```
unsigned int Flock::nfuns ( ) const  [inline]
```

Definition at line 109 of file flock-meet.hpp.

6.9.3.6 nleafs()

```
std::vector< unsigned int > Flock::nleafs ( ) const  [inline]
```

Definition at line 132 of file flock-meet.hpp.

6.9.3.7 nnodes()

```
std::vector< unsigned int > Flock::nnodes ( ) const  [inline]
```

Definition at line 121 of file flock-meet.hpp.

6.9.3.8 nterms()

```
unsigned int Flock::nterms ( ) const  [inline]
```

Definition at line 144 of file flock-meet.hpp.

6.9.3.9 ntrees()

```
unsigned int Flock::ntrees ( ) const  [inline]
```

Definition at line 115 of file flock-meet.hpp.

6.9.3.10 set_seed()

```
void Flock::set_seed (
            const unsigned int & s )  [inline]
```

Definition at line 37 of file flock-meet.hpp.

6.9.4 Member Data Documentation

6.9.4.1 dat

```
std::vector< Geese > Flock::dat
```

Definition at line 15 of file flock-bones.hpp.

6.9.4.2 initialized

```
bool Flock::initialized = false
```

Definition at line 17 of file flock-bones.hpp.

6.9.4.3 nfunctions

```
unsigned int Flock::nfunctions = 0u
```

Definition at line 16 of file flock-bones.hpp.

6.9.4.4 rengine

```
std::mt19937 Flock::rengine
```

Definition at line 20 of file flock-bones.hpp.

6.9.4.5 support

```
phylocounters::PhyloModel Flock::support = phylocounters::PhyloModel()
```

Definition at line 21 of file flock-bones.hpp.

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

- include/barry/models/geese/flock-bones.hpp
- include/barry/models/geese/flock-meet.hpp

6.10 FreqTable< T > Class Template Reference

Database of statistics.

```
#include <statsdb.hpp>
```

Public Member Functions

- FreqTable ()
- ~FreqTable ()
- void add (const std::vector< T > &x)
- Counts_type as_vector () const
- MapVec_type< T, uint > get_data () const
- const MapVec_type< T, uint > * get_data_ptr () const
- void clear ()
- void reserve (unsigned int n)
- void print () const

6.10.1 Detailed Description

template<typename T = double>
class FreqTable< T >

Database of statistics.

This is mostly used in `Support`.

Definition at line 16 of file statsdb.hpp.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 FreqTable()

```
template<typename T = double>
FreqTable< T >::FreqTable ( )  [inline]
```

Definition at line 28 of file statsdb.hpp.

6.10.2.2 ~FreqTable()

```
template<typename T = double>
FreqTable< T >::~FreqTable ( )  [inline]
```

Definition at line 29 of file statsdb.hpp.

6.10.3 Member Function Documentation

6.10.3.1 add()

```
template<typename T >
void FreqTable< T >::add (
            const std::vector< T > & x )  [inline]
```

Definition at line 46 of file statsdb.hpp.

6.10.3.2 as_vector()

```
template<typename T >
Counts_type FreqTable< T >::as_vector  [inline]
```

Definition at line 60 of file statsdb.hpp.

6.10.3.3 clear()

```
template<typename T >
void FreqTable< T >::clear  [inline]
```

Definition at line 82 of file statsdb.hpp.

6.10.3.4 get_data()

```
template<typename T >
MapVec_type< T, uint > FreqTable< T >::get_data  [inline]
```

Definition at line 72 of file statsdb.hpp.

6.10.3.5 get_data_ptr()

```
template<typename T >
const MapVec_type< T, uint > * FreqTable< T >::get_data_ptr  [inline]
```

Definition at line 77 of file statsdb.hpp.

6.10.3.6 print()

```
template<typename T >
void FreqTable< T >::print  [inline]
```

Definition at line 101 of file statsdb.hpp.

6.10.3.7 reserve()

```
template<typename T >
void FreqTable< T >::reserve (
            unsigned int n )  [inline]
```

Definition at line 88 of file statsdb.hpp.

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

- include/barry/statsdb.hpp

6.11 Geese Class Reference

Annotated Phylo Model.

```
#include <geese-bones.hpp>
```

Public Member Functions

- ~Geese ()
- void init ()
- void inherit_support (const Geese &model_, bool delete_support_=false)
- void calc_sequence (Node *n=nullptr)
- void calc_likelihood_sequence ()
- double likelihood (const std::vector< double > &par, bool as_log=false, bool use_likelihood_sequence=true)
- double likelihood_exhaust (const std::vector< double > &par)
- std::vector< double > get_probabilities () const
- void set_seed (const unsigned int &s)
- std::vector< std::vector< unsigned int > > simulate (const std::vector< double > &par)
- std::vector< std::vector< double > > observed_counts ()
- void print_observed_counts ()
- std::vector< std::vector< double > > predict (const std::vector< double > &p, std::vector< std::vector< double > > *res_prob=nullptr)

  Calculate the conditional probability.
- void init_node (Node &n)
- void update_annotations (unsigned int nodeid, std::vector< unsigned int > newann)

Construct a new Geese object

The model includes a total of $N + 1$ nodes, the $+ 1$ beign the root node.

Parameters

annotations	*A vector of vectors with annotations. It should be of length k (number of functions). Each vector should be of length N (equal to the number of nodes, including interior). Possible values are 0, 1, and 9.*
geneid	*Id of the gene. It should be of length N.*
parent	*Id of the parent gene. Also of length N*

- Geese ()
- Geese (std::vector< std::vector< unsigned int > > &annotations, std::vector< unsigned int > &geneid, std::vector< int > &parent, std::vector< bool > &duplication)
- Geese (const Geese &model_, bool copy_data=true)
- Geese (Geese &&x) noexcept
- Geese & operator= (const Geese &model_)=delete
- Geese & operator= (Geese &&model_) noexcept=delete

Information about the model

- unsigned int nfuns () const
- unsigned int nnodes () const
- unsigned int nleafs () const
- unsigned int nterms () const

Public Attributes

- unsigned int nfunctions
- barry::Map< unsigned int, Node > nodes
- barry::MapVec_type< unsigned int > map_to_nodes
- std::vector< unsigned int > sequence
- std::vector< unsigned int > likelihood_sequence

- bool initialized = false
- bool delete_rengine = false
- bool delete_counters = false
- bool delete_support = false

Shared objects within a <tt>Geese</tt>

Since users may start adding counters before initializing the PhyloModel object, the object `counter` *is initialized first.*

While the member `support` *has an* `rengine`*, since* `Geese` *can sample trees, we have the option to keep it separate.*

- std::mt19937 * rengine = nullptr
- phylocounters::PhyloCounters * counters = nullptr
- phylocounters::PhyloModel * support = nullptr
- std::vector< std::vector< bool > > states

6.11.1 Detailed Description

Annotated Phylo Model.

Definition at line 67 of file geese-bones.hpp.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Geese() [1/4]

```
Geese::Geese ( )  [inline]
```

Definition at line 6 of file geese-meat-constructors.hpp.

6.11.2.2 Geese() [2/4]

```
Geese::Geese (
            std::vector< std::vector< unsigned int > > & annotations,
            std::vector< unsigned int > & geneid,
            std::vector< int > & parent,
            std::vector< bool > & duplication )  [inline]
```

Definition at line 17 of file geese-meat-constructors.hpp.

6.11.2.3 Geese() [3/4]

```
Geese::Geese (
            const Geese & model_,
            bool copy_data = true )  [inline]
```

Definition at line 157 of file geese-meat-constructors.hpp.

6.11.2.4 Geese() [4/4]

```
Geese::Geese (
            Geese && x )  [inline], [noexcept]
```

Definition at line 232 of file geese-meat-constructors.hpp.

6.11.2.5 ~Geese()

```
Geese::~Geese ( )  [inline]
```

Definition at line 71 of file geese-meat.hpp.

6.11.3 Member Function Documentation

6.11.3.1 calc_likelihood_sequence()

```
void Geese::calc_likelihood_sequence ( )  [inline]
```

Definition at line 234 of file geese-meat.hpp.

6.11.3.2 calc_sequence()

```
void Geese::calc_sequence (
            Node * n = nullptr )  [inline]
```

Definition at line 191 of file geese-meat.hpp.

6.11.3.3 get_probabilities()

```
std::vector< double > Geese::get_probabilities ( ) const  [inline]
```

Definition at line 271 of file geese-meat.hpp.

6.11.3.4 inherit_support()

```
void Geese::inherit_support (
            const Geese & model_,
            bool delete_support_ = false )  [inline]
```

Definition at line 140 of file geese-meat.hpp.

6.11.3.5 init()

```
void Geese::init ( )  [inline]
```

Definition at line 83 of file geese-meat.hpp.

6.11.3.6 init_node()

```
void Geese::init_node (
            Node & n )  [inline]
```

Definition at line 6 of file geese-meat.hpp.

6.11.3.7 likelihood()

```
double Geese::likelihood (
            const std::vector< double > & par,
            bool as_log = false,
            bool use_likelihood_sequence = true )  [inline]
```

Definition at line 6 of file geese-meat-likelihood.hpp.

6.11.3.8 likelihood_exhaust()

```
double Geese::likelihood_exhaust (
            const std::vector< double > & par )  [inline]
```

Definition at line 7 of file geese-meat-likelihood_exhaust.hpp.

6.11.3.9 nfuns()

```
unsigned int Geese::nfuns ( ) const  [inline]
```

Definition at line 287 of file geese-meat.hpp.

6.11.3.10 nleafs()

```
unsigned int Geese::nleafs ( ) const  [inline]
```

Definition at line 295 of file geese-meat.hpp.

6.11.3.11 nnodes()

```
unsigned int Geese::nnodes ( ) const  [inline]
```

Definition at line 291 of file geese-meat.hpp.

6.11.3.12 nterms()

```
unsigned int Geese::nterms ( ) const  [inline]
```

Definition at line 305 of file geese-meat.hpp.

6.11.3.13 observed_counts()

```
std::vector< std::vector< double > > Geese::observed_counts ( )  [inline]
```

Definition at line 312 of file geese-meat.hpp.

6.11.3.14 operator=() [1/2]

```
Geese& Geese::operator= (
            const Geese & model_ )  [delete]
```

6.11.3.15 operator=() [2/2]

```
Geese& Geese::operator= (
            Geese && model_ )  [delete], [noexcept]
```

6.11.3.16 predict()

```
std::vector< std::vector< double > > Geese::predict (
            const std::vector< double > & p,
            std::vector< std::vector< double > > * res_prob = nullptr )  [inline]
```

Calculate the conditional probability.

Parameters

p	Vector of parameters

Returns

std::vector< double > Returns the posterior probability

Definition at line 7 of file geese-meat-predict.hpp.

6.11.3.17 print_observed_counts()

```
void Geese::print_observed_counts ( )  [inline]
```

Definition at line 360 of file geese-meat.hpp.

6.11.3.18 set_seed()

```
void Geese::set_seed (
            const unsigned int & s )  [inline]
```

Definition at line 8 of file geese-meat-simulate.hpp.

6.11.3.19 simulate()

```
std::vector< std::vector< unsigned int > > Geese::simulate (
            const std::vector< double > & par )  [inline]
```

Definition at line 12 of file geese-meat-simulate.hpp.

6.11.3.20 update_annotations()

```
void Geese::update_annotations (
            unsigned int nodeid,
            std::vector< unsigned int > newann )  [inline]
```

Definition at line 168 of file geese-meat.hpp.

6.11.4 Member Data Documentation

6.11.4.1 counters

```
phylocounters::PhyloCounters* Geese::counters = nullptr
```

Definition at line 82 of file geese-bones.hpp.

6.11.4.2 delete_counters

```
bool Geese::delete_counters = false
```

Definition at line 99 of file geese-bones.hpp.

6.11.4.3 delete_rengine

```
bool Geese::delete_rengine = false
```

Definition at line 98 of file geese-bones.hpp.

6.11.4.4 delete_support

```
bool Geese::delete_support = false
```

Definition at line 100 of file geese-bones.hpp.

6.11.4.5 initialized

```
bool Geese::initialized = false
```

Definition at line 97 of file geese-bones.hpp.

6.11.4.6 likelihood_sequence

```
std::vector< unsigned int > Geese::likelihood_sequence
```

Definition at line 94 of file geese-bones.hpp.

6.11.4.7 map_to_nodes

```
barry::MapVec_type< unsigned int > Geese::map_to_nodes
```

Definition at line 90 of file geese-bones.hpp.

6.11.4.8 nfunctions

```
unsigned int Geese::nfunctions
```

Definition at line 88 of file geese-bones.hpp.

6.11.4.9 nodes

```
barry::Map< unsigned int, Node > Geese::nodes
```

Definition at line 89 of file geese-bones.hpp.

6.11.4.10 rengine

```
std::mt19937* Geese::rengine = nullptr
```

Definition at line 81 of file geese-bones.hpp.

6.11.4.11 sequence

```
std::vector< unsigned int > Geese::sequence
```

Definition at line 93 of file geese-bones.hpp.

6.11.4.12 states

```
std::vector< std::vector< bool > > Geese::states
```

Definition at line 84 of file geese-bones.hpp.

6.11.4.13 support

```
phylocounters::PhyloModel* Geese::support = nullptr
```

Definition at line 83 of file geese-bones.hpp.

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

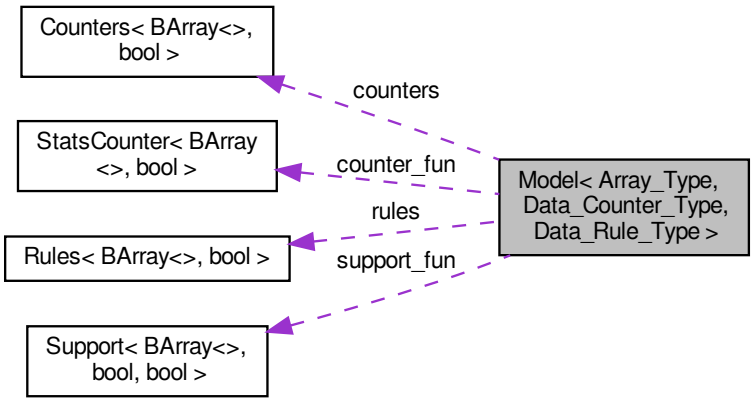
- include/barry/models/geese/geese-bones.hpp
- include/barry/models/geese/geese-meat-constructors.hpp
- include/barry/models/geese/geese-meat-likelihood.hpp
- include/barry/models/geese/geese-meat-likelihood_exhaust.hpp
- include/barry/models/geese/geese-meat-predict.hpp
- include/barry/models/geese/geese-meat-simulate.hpp
- include/barry/models/geese/geese-meat.hpp

6.12 Model< Array_Type, Data_Counter_Type, Data_Rule_Type > Class Template Reference

General framework for discrete exponential models. This class allows generating discrete exponential models in the form of a linear exponential model:

`#include <model-bones.hpp>`

Collaboration diagram for Model< Array_Type, Data_Counter_Type, Data_Rule_Type >:



Public Member Functions

- Model ()
- Model (uint size_)
- Model (const Model< Array_Type, Data_Counter_Type, Data_Rule_Type > &Model_)
- Model< Array_Type, Data_Counter_Type, Data_Rule_Type > & operator= (const Model< Array_Type, Data_Counter_Type, Data_Rule_Type > &Model_)
- ~Model ()
- void store_psets ()
- void set_keygen (std::function< std::vector< double >(const Array_Type &)> keygen_)
- uint add_array (const Array_Type &Array_, bool force_new=false)

  Adds an array to the support of not already included.
- void print_stats (uint i) const
- Array_Type sample (const Array_Type &Array_, const std::vector< double > &params={})
- Array_Type sample (const uint &i, const std::vector< double > &params)

Wrappers for the <tt>Counters</tt> member.

These will add counters to the model, which are shared by the support and the actual counter function.

- void add_counter (Counter< Array_Type, Data_Counter_Type > &counter)
- void add_counter (Counter< Array_Type, Data_Counter_Type > *counter)
- void add_counter (Counter_fun_type< Array_Type, Data_Counter_Type > count_fun_, Counter_fun_type< Array_Type, Data_Counter_Type > init_fun_=nullptr, Data_Counter_Type *data_=nullptr, bool delete_↩ data_=false)

- void set_counters (Counters< Array_Type, Data_Counter_Type > *counters_)

Wrappers for the <tt>Rules</tt> member.

These will add rules to the model, which are shared by the support and the actual counter function.

- void add_rule (Rule< Array_Type, Data_Rule_Type > &rule)
- void add_rule (Rule< Array_Type, Data_Rule_Type > *rule)
- void add_rule (Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_, Data_Rule_Type *data_=nullptr, bool delete_data_=false)
- void set_rules (Rules< Array_Type, Data_Rule_Type > *rules_)

Likelihood functions.

Calculation of likelihood functions is done reusing normalizing constants. Before recalculating the normalizing constant, the function checks whether `params` matches the last set vector of parameters used to compute it.

Parameters

params	*Vector of parameters*
as_log	*When `true`, the function returns the log-likelihood.*

- double likelihood (const std::vector< double > &params, const uint &i, bool as_log=false)
- double likelihood (const std::vector< double > &params, const Array_Type &Array_, int i=-1, bool as_log=false)
- double likelihood (const std::vector< double > &params, const std::vector< double > &target_, const uint &i, bool as_log=false)
- double likelihood_total (const std::vector< double > &params, bool as_log=false)

Extract elements by index

Parameters

i	*Index relative to the array in the model.*
params	*A new vector of model parameters to compute the normalizing constant.*
as_log	*When `true` returns the logged version of the normalizing constant.*

- double get_norm_const (const std::vector< double > &params, const uint &i, bool as_log=false)
- const std::vector< Array_Type > * get_pset (const uint &i)
- const std::vector< std::vector< double > > * get_stats (const uint &i)

Size of the model

Number of different supports included in the model

This will return the size of `stats`.

Returns

`size()` returns the number of arrays in the model.

`size_unique()` returns the number of unique arrays (according to the hasher) in the model.

`nterms()` returns the number of terms in the model.

- unsigned int size () const
- unsigned int size_unique () const
- unsigned int nterms () const

Public Attributes

- std::vector< Counts_type > stats
- std::vector< uint > n_arrays_per_stats
- MapVec_type< double, uint > keys2support

  Map of types of arrays to support sets.
- std::vector< std::vector< double > > params_last

  Vector of the previously used parameters.
- std::vector< double > normalizing_constants
- std::vector< bool > first_calc_done
- std::function< std::vector< double >const Array_Type &)> keygen = nullptr

  Function to extract features of the array to be hash.

Container space for the powerset (and its sufficient stats)

This is useful in the case of using simulations or evaluating functions that need to account for the full set of states.

- bool with_pset = false
- std::vector< std::vector< Array_Type > > pset_arrays
- std::vector< std::vector< std::vector< double > > > pset_stats
- std::vector< std::vector< double > > pset_probs

Information about the arrays used in the model

`target_stats` *holds the observed sufficient statistics for each array in the dataset.* `array_frequency` *contains the frequency with which each of the target stats (arrays) shows in the support.* `array2support` *maps array indices (0, 1, ...) to the corresponding support.*

- std::vector< std::vector< double > > target_stats
- std::vector< uint > array_frequency
- std::vector< uint > arrays2support

Functions to compute statistics

Arguments are recycled to save memory and computation.

- Counters< Array_Type, Data_Counter_Type > counters
- Rules< Array_Type, Data_Rule_Type > rules
- Support< Array_Type, Data_Counter_Type, Data_Rule_Type > support_fun
- StatsCounter< Array_Type, Data_Counter_Type > counter_fun

Random number generation

Random number generation

- std::mt19937 * rengine = nullptr
- bool delete_rengine = false
- void set_rengine (std::mt19937 *rengine_, bool delete_=false)
- void set_seed (unsigned int s)

6.12.1 Detailed Description

template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool>
class Model< Array_Type, Data_Counter_Type, Data_Rule_Type >

General framework for discrete exponential models. This class allows generating discrete exponential models in the form of a linear exponential model:

$$\frac{\exp\left(\theta^{\mathsf{t}} c(A)\right)}{\sum_{A' \in \mathcal{A}} \exp\left(\theta^{\mathsf{t}} c(A')\right)}$$

This implementation aims to reduce the number of times that the support needs to be computed. Models included here use more than a single array, and thus allow the function to recycle support sets as needed. For example, if we are looking at directed graphs all of the same size and without vertex level features, i.e. a model that only counts edges, triangles, etc. then the support needs to be fully computed only once.

Template Parameters

Array_Type	Class of `BArray` object.
Data_Counter_Type	Any type.
Data_Rule_Type	Any type.

Definition at line 95 of file model-bones.hpp.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 Model() [1/3]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::Model  [inline]
```

Definition at line 7 of file model-meat.hpp.

6.12.2.2 Model() [2/3]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::Model (
            uint size_ )  [inline]
```

Definition at line 27 of file model-meat.hpp.

6.12.2.3 Model() [3/3]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::Model (
            const Model< Array_Type, Data_Counter_Type, Data_Rule_Type > & Model_ )  [inline]
```

Definition at line 50 of file model-meat.hpp.

6.12.2.4 ~Model()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::~Model ( )  [inline]
```

Definition at line 190 of file model-bones.hpp.

6.12.3 Member Function Documentation

6.12.3.1 add_array()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
uint Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::add_array (
            const Array_Type & Array_,
            bool force_new = false )  [inline]
```

Adds an array to the support of not already included.

Parameters

*Array_*	array to be added
force_new	If `false`, it will use `keygen` to obtain a double vector and create a hash of it. If the hash has been computed earlier, the support is recycled.

Returns

The number of the array.

When computing with the powerset, we need to grow the corresponding vectors on the fly

Definition at line 229 of file model-meat.hpp.

6.12.3.2 add_counter() [1/3]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::add_counter (
            Counter< Array_Type, Data_Counter_Type > & counter )  [inline]
```

Definition at line 131 of file model-meat.hpp.

6.12.3.3 add_counter() [2/3]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::add_counter (
            Counter< Array_Type, Data_Counter_Type > * counter )  [inline]
```

Definition at line 140 of file model-meat.hpp.

6.12.3.4 add_counter() [3/3]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::add_counter (
            Counter_fun_type< Array_Type, Data_Counter_Type > count_fun_,
            Counter_fun_type< Array_Type, Data_Counter_Type > init_fun_ = nullptr,
            Data_Counter_Type * data_ = nullptr,
            bool delete_data_ = false )  [inline]
```

Definition at line 150 of file model-meat.hpp.

6.12.3.5 add_rule() [1/3]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > & rule )  [inline]
```

Definition at line 182 of file model-meat.hpp.

6.12.3.6 add_rule() [2/3]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > * rule )  [inline]
```

Definition at line 191 of file model-meat.hpp.

6.12.3.7 add_rule() [3/3]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::add_rule (
            Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_,
            Data_Rule_Type * data_ = nullptr,
            bool delete_data_ = false )  [inline]
```

Definition at line 201 of file model-meat.hpp.

6.12.3.8 get_norm_const()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::get_norm_const (
            const std::vector< double > & params,
            const uint & i,
            bool as_log = false )  [inline]
```

Definition at line 460 of file model-meat.hpp.

6.12.3.9 get_pset()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
const std::vector< Array_Type > * Model< Array_Type, Data_Counter_Type, Data_Rule_Type >↩
::get_pset (
            const uint & i )  [inline]
```

Definition at line 492 of file model-meat.hpp.

6.12.3.10 get_stats()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
const std::vector< std::vector< double > > * Model< Array_Type, Data_Counter_Type, Data_↩
Rule_Type >::get_stats (
            const uint & i )  [inline]
```

Definition at line 505 of file model-meat.hpp.

6.12.3.11 likelihood() [1/3]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::likelihood (
            const std::vector< double > & params,
            const Array_Type & Array_,
            int i = -1,
            bool as_log = false )  [inline]
```

Definition at line 346 of file model-meat.hpp.

6.12.3.12 likelihood() [2/3]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::likelihood (
            const std::vector< double > & params,
            const std::vector< double > & target_,
            const uint & i,
            bool as_log = false )  [inline]
```

Definition at line 386 of file model-meat.hpp.

6.12.3.13 likelihood() [3/3]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::likelihood (
            const std::vector< double > & params,
            const uint & i,
            bool as_log = false )  [inline]
```

Definition at line 313 of file model-meat.hpp.

6.12.3.14 likelihood_total()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::likelihood_total (
            const std::vector< double > & params,
            bool as_log = false )  [inline]
```

Definition at line 420 of file model-meat.hpp.

6.12.3.15 nterms()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
uint Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::nterms  [inline]
```

Definition at line 546 of file model-meat.hpp.

6.12.3.16 operator=()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
Model< Array_Type, Data_Counter_Type, Data_Rule_Type > & Model< Array_Type, Data_Counter_↩
Type, Data_Rule_Type >::operator= (
            const Model< Array_Type, Data_Counter_Type, Data_Rule_Type > & Model_ )  [inline]
```

Definition at line 80 of file model-meat.hpp.

6.12.3.17 print_stats()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::print_stats (
            uint i ) const  [inline]
```

Definition at line 517 of file model-meat.hpp.

6.12.3.18 sample() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
Array_Type Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::sample (
            const Array_Type & Array_,
            const std::vector< double > & params = {} )
```

6.12.3.19 sample() [2/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
Array_Type Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::sample (
            const uint & i,
            const std::vector< double > & params )  [inline]
```

Definition at line 553 of file model-meat.hpp.

6.12.3.20 set_counters()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::set_counters (
            Counters< Array_Type, Data_Counter_Type > * counters_ )  [inline]
```

Definition at line 169 of file model-meat.hpp.

6.12.3.21 set_keygen()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::set_keygen (
            std::function< std::vector< double >(const Array_Type &)> keygen_ )  [inline]
```

Definition at line 123 of file model-meat.hpp.

6.12.3.22 set_rengine()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::set_rengine (
            std::mt19937 * rengine_,
            bool delete_ = false )  [inline]
```

Definition at line 106 of file model-bones.hpp.

6.12.3.23 set_rules()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::set_rules (
            Rules< Array_Type, Data_Rule_Type > * rules_ )  [inline]
```

Definition at line 218 of file model-meat.hpp.

6.12.3.24 set_seed()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::set_seed (
            unsigned int s )  [inline]
```

Definition at line 116 of file model-bones.hpp.

6.12.3.25 size()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
uint Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::size  [inline]
```

Definition at line 536 of file model-meat.hpp.

6.12.3.26 size_unique()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
uint Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::size_unique  [inline]
```

Definition at line 541 of file model-meat.hpp.

6.12.3.27 store_psets()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::store_psets  [inline]
```

Definition at line 115 of file model-meat.hpp.

6.12.4 Member Data Documentation

6.12.4.1 array_frequency

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::vector< uint > Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::array_frequency
```

Definition at line 154 of file model-bones.hpp.

6.12.4.2 arrays2support

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::vector< uint > Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::arrays2support
```

Definition at line 155 of file model-bones.hpp.

6.12.4.3 counter_fun

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
StatsCounter<Array_Type,Data_Counter_Type> Model< Array_Type, Data_Counter_Type, Data_Rule_↩
Type >::counter_fun
```

Definition at line 172 of file model-bones.hpp.

6.12.4.4 counters

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
Counters<Array_Type,Data_Counter_Type> Model< Array_Type, Data_Counter_Type, Data_Rule_Type
>::counters
```

Definition at line 169 of file model-bones.hpp.

6.12.4.5 delete_rengine

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
bool Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::delete_rengine = false
```

Definition at line 105 of file model-bones.hpp.

6.12.4.6 first_calc_done

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::vector< bool > Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::first_calc_done
```

Definition at line 178 of file model-bones.hpp.

6.12.4.7 keygen

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::function<std::vector<double>const Array_Type &)> Model< Array_Type, Data_Counter_Type,
Data_Rule_Type >::keygen = nullptr
```

Function to extract features of the array to be hash.

Definition at line 182 of file model-bones.hpp.

6.12.4.8 keys2support

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
MapVec_type< double, uint > Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::keys2support
```

Map of types of arrays to support sets.

This is of the same length as the vector `stats`.

Definition at line 162 of file model-bones.hpp.

6.12.4.9 n_arrays_per_stats

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::vector< uint > Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::n_arrays_per_↩
stats
```

Definition at line 131 of file model-bones.hpp.

6.12.4.10 normalizing_constants

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::vector< double > Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::normalizing_↩
constants
```

Definition at line 177 of file model-bones.hpp.

6.12.4.11 params_last

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::vector< std::vector<double> > Model< Array_Type, Data_Counter_Type, Data_Rule_Type >↩
::params_last
```

Vector of the previously used parameters.

Definition at line 176 of file model-bones.hpp.

6.12.4.12 pset_arrays

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::vector< std::vector< Array_Type > > Model< Array_Type, Data_Counter_Type, Data_Rule_↩
Type >::pset_arrays
```

Definition at line 140 of file model-bones.hpp.

6.12.4.13 pset_probs

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::vector< std::vector<double> > Model< Array_Type, Data_Counter_Type, Data_Rule_Type >↩
::pset_probs
```

Definition at line 142 of file model-bones.hpp.

6.12.4.14 pset_stats

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::vector< std::vector< std::vector<double> > > Model< Array_Type, Data_Counter_Type,
Data_Rule_Type >::pset_stats
```

Definition at line 141 of file model-bones.hpp.

6.12.4.15 rengine

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::mt19937* Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::rengine = nullptr
```

Definition at line 104 of file model-bones.hpp.

6.12.4.16 rules

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
Rules<Array_Type,Data_Rule_Type> Model< Array_Type, Data_Counter_Type, Data_Rule_Type >↩
::rules
```

Definition at line 170 of file model-bones.hpp.

6.12.4.17 stats

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::vector< Counts_type > Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::stats
```

Definition at line 130 of file model-bones.hpp.

6.12.4.18 support_fun

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
Support<Array_Type,Data_Counter_Type,Data_Rule_Type> Model< Array_Type, Data_Counter_Type,
Data_Rule_Type >::support_fun
```

Definition at line 171 of file model-bones.hpp.

6.12.4.19 target_stats

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::vector< std::vector< double > > Model< Array_Type, Data_Counter_Type, Data_Rule_Type
>::target_stats
```

Definition at line 153 of file model-bones.hpp.

6.12.4.20 with_pset

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
bool Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::with_pset = false
```

Definition at line 139 of file model-bones.hpp.

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

- include/barry/model-bones.hpp
- include/barry/model-meat.hpp

6.13 NetCounterData Class Reference

Data class used to store arbitrary uint or double vectors.

```
#include <network.hpp>
```

Public Member Functions

- NetCounterData ()
- NetCounterData (const std::vector< uint > indices_, const std::vector< double > numbers_)
- ~NetCounterData ()

Public Attributes

- std::vector< uint > indices
- std::vector< double > numbers

6.13.1 Detailed Description

Data class used to store arbitrary uint or double vectors.

Definition at line 50 of file network.hpp.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 NetCounterData() [1/2]

```
NetCounterData::NetCounterData ( )  [inline]
```

Definition at line 56 of file network.hpp.

6.13.2.2 NetCounterData() [2/2]

```
NetCounterData::NetCounterData (
            const std::vector< uint > indices_,
            const std::vector< double > numbers_ )  [inline]
```

Definition at line 57 of file network.hpp.

6.13.2.3 ~NetCounterData()

```
NetCounterData::~NetCounterData ( )  [inline]
```

Definition at line 62 of file network.hpp.

6.13.3 Member Data Documentation

6.13.3.1 indices

```
std::vector< uint > NetCounterData::indices
```

Definition at line 53 of file network.hpp.

6.13.3.2 numbers

```
std::vector< double > NetCounterData::numbers
```

Definition at line 54 of file network.hpp.

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

- include/barry/counters/network.hpp

6.14 NetworkData Class Reference

Data class for Networks.

```
#include <network.hpp>
```

Public Member Functions

- NetworkData ()
- NetworkData (std::vector< double > vertex_attr_, bool directed_=true)

  Constructor using a single attribute.
- NetworkData (std::vector< std::vector< double > > vertex_attr_, bool directed_=true)

  Constructor using multiple attributes.
- ~NetworkData ()

Public Attributes

- bool directed = true
- std::vector< std::vector< double > > vertex_attr

6.14.1 Detailed Description

Data class for Networks.

This holds information about whether the graph is directed or not, and, if defined, vectors of node (vertex) attributes (`vertex_attr`).

Definition at line 15 of file network.hpp.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 NetworkData() [1/3]

```
NetworkData::NetworkData ( )  [inline]
```

Definition at line 21 of file network.hpp.

6.14.2.2 NetworkData() [2/3]

```
NetworkData::NetworkData (
            std::vector< double > vertex_attr_,
            bool directed_ = true )  [inline]
```

Constructor using a single attribute.

Parameters

*vertex_attr_*	Double vector of length equal to the number of vertices in the data.
*directed_*	When `true` the graph as treated as directed.

Definition at line 28 of file network.hpp.

6.14.2.3 NetworkData() [3/3]

```
NetworkData::NetworkData (
            std::vector< std::vector< double > > vertex_attr_,
            bool directed_ = true )  [inline]
```

Constructor using multiple attributes.

Parameters

*vertex_↩ attr_*	Vector of double vectors. The size equals to the number of attributes to be created. Each individual vector should be of length equal to the number of vertices.
*directed_*	When `true` the graph as treated as directed.

Definition at line 39 of file network.hpp.

6.14.2.4 ~NetworkData()

```
NetworkData::~NetworkData ( )  [inline]
```

Definition at line 45 of file network.hpp.

6.14.3 Member Data Documentation

6.14.3.1 directed

```
bool NetworkData::directed = true
```

Definition at line 18 of file network.hpp.

6.14.3.2 vertex_attr

```
std::vector< std::vector< double > > NetworkData::vertex_attr
```

Definition at line 19 of file network.hpp.

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

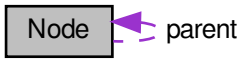
- include/barry/counters/network.hpp

6.15 Node Class Reference

A single node for the model.

```
#include <geese-node-bones.hpp>
```

Collaboration diagram for Node:



Public Member Functions

- ~Node ()
- int get_parent () const
- bool is_leaf () const

Construct a new Node object

- Node ()
- Node (unsigned int id_, unsigned int ord_, bool duplication_)
- Node (unsigned int id_, unsigned int ord_, std::vector< unsigned int > annotations_, bool duplication_)
- Node (Node &&x) noexcept
- Node (const Node &x)

Public Attributes

- unsigned int id

  Id of the node (as specified in the input)
- unsigned int ord

  Order in which the node was created.
- phylocounters::PhyloArray array
- std::vector< unsigned int > annotations

  Observed annotations (only defined for Geese)
- bool duplication
- std::vector< phylocounters::PhyloArray > arrays = {}

  Arrays given all possible states.
- Node * parent = nullptr

  Parent node.
- std::vector< Node * > offspring = {}

  Offspring nodes.
- std::vector< unsigned int > narray = {}

  ID of the array in the model.
- bool visited = false
- std::vector< double > subtree_prob

  Induced subtree probabilities.
- std::vector< double > probability

  The probability of observing each state.

6.15.1 Detailed Description

A single node for the model.

Each node contains all the information to compute the conditional probability of the pruning algorithm at that node.

Definition at line 11 of file geese-node-bones.hpp.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 Node() [1/5]

```
Node::Node ( )  [inline]
```

Definition at line 36 of file geese-node-bones.hpp.

6.15.2.2 Node() [2/5]

```
Node::Node (
            unsigned int id_,
            unsigned int ord_,
            bool duplication_ )  [inline]
```

Definition at line 55 of file geese-node-bones.hpp.

6.15.2.3 Node() [3/5]

```
Node::Node (
            unsigned int id_,
            unsigned int ord_,
            std::vector< unsigned int > annotations_,
            bool duplication_ )  [inline]
```

Definition at line 61 of file geese-node-bones.hpp.

6.15.2.4 Node() [4/5]

```
Node::Node (
            Node && x )  [inline], [noexcept]
```

Definition at line 68 of file geese-node-bones.hpp.

6.15.2.5 Node() [5/5]

```
Node::Node (
            const Node & x )  [inline]
```

Definition at line 82 of file geese-node-bones.hpp.

6.15.2.6 ~Node()

```
Node::~Node ( )  [inline]
```

Definition at line 47 of file geese-node-bones.hpp.

6.15.3 Member Function Documentation

6.15.3.1 get_parent()

```
int Node::get_parent ( ) const  [inline]
```

Definition at line 96 of file geese-node-bones.hpp.

6.15.3.2 is_leaf()

```
bool Node::is_leaf ( ) const  [inline]
```

Definition at line 103 of file geese-node-bones.hpp.

6.15.4 Member Data Documentation

6.15.4.1 annotations

```
std::vector< unsigned int > Node::annotations
```

Observed annotations (only defined for Geese)

Definition at line 18 of file geese-node-bones.hpp.

6.15.4.2 array

```
phylocounters::PhyloArray Node::array
```

Definition at line 17 of file geese-node-bones.hpp.

6.15.4.3 arrays

```
std::vector< phylocounters::PhyloArray > Node::arrays = {}
```

Arrays given all possible states.

Definition at line 21 of file geese-node-bones.hpp.

6.15.4.4 duplication

```
bool Node::duplication
```

Definition at line 19 of file geese-node-bones.hpp.

6.15.4.5 id

```
unsigned int Node::id
```

Id of the node (as specified in the input)

Definition at line 14 of file geese-node-bones.hpp.

6.15.4.6 narray

```
std::vector< unsigned int > Node::narray = {}
```

ID of the array in the model.

Definition at line 24 of file geese-node-bones.hpp.

6.15.4.7 offspring

```
std::vector< Node* > Node::offspring = {}
```

Offspring nodes.

Definition at line 23 of file geese-node-bones.hpp.

6.15.4.8 ord

```
unsigned int Node::ord
```

Order in which the node was created.

Definition at line 15 of file geese-node-bones.hpp.

6.15.4.9 parent

```
Node* Node::parent = nullptr
```

Parent node.

Definition at line 22 of file geese-node-bones.hpp.

6.15.4.10 probability

```
std::vector< double > Node::probability
```

The probability of observing each state.

Definition at line 28 of file geese-node-bones.hpp.

6.15.4.11 subtree_prob

```
std::vector< double > Node::subtree_prob
```

Induced subtree probabilities.

Definition at line 27 of file geese-node-bones.hpp.

6.15.4.12 visited

```
bool Node::visited = false
```

Definition at line 25 of file geese-node-bones.hpp.

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

- include/barry/models/geese/geese-node-bones.hpp

6.16 NodeData Class Reference

Data definition for the `PhyloArray` class.

`#include <phylo.hpp>`

Public Member Functions

- NodeData ()
- NodeData (const std::vector< double > &blengths_, const std::vector< bool > &states_, bool duplication_=true)
- ~NodeData ()

Public Attributes

- std::vector< double > blengths
- std::vector< bool > states
- bool duplication = true

6.16.1 Detailed Description

Data definition for the `PhyloArray` class.

This holds basic information about a given node.

Definition at line 16 of file phylo.hpp.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 NodeData() [1/2]

```
NodeData::NodeData ( )  [inline]
```

Definition at line 34 of file phylo.hpp.

6.16.2.2 NodeData() [2/2]

```
NodeData::NodeData (
            const std::vector< double > & blengths_,
            const std::vector< bool > & states_,
            bool duplication_ = true )  [inline]
```

Definition at line 36 of file phylo.hpp.

6.16.2.3 ~NodeData()

```
NodeData::~NodeData ( )  [inline]
```

Definition at line 42 of file phylo.hpp.

6.16.3 Member Data Documentation

6.16.3.1 blengths

```
std::vector< double > NodeData::blengths
```

Branch length.

Definition at line 22 of file phylo.hpp.

6.16.3.2 duplication

```
bool NodeData::duplication = true
```

Definition at line 32 of file phylo.hpp.

6.16.3.3 states

```
std::vector< bool > NodeData::states
```

State of the parent node.

Definition at line 27 of file phylo.hpp.

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

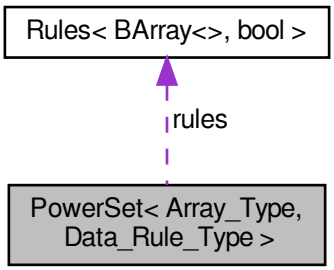
- include/barry/counters/phylo.hpp

6.17 PowerSet< Array_Type, Data_Rule_Type > Class Template Reference

Powerset of a binary array.

`#include <powerset-bones.hpp>`

Collaboration diagram for PowerSet< Array_Type, Data_Rule_Type >:



Public Member Functions

- void init_support ()
- void calc ()
- void reset (uint N_, uint M_)

Construct and destroy a PowerSet object

- PowerSet ()
- PowerSet (uint N_, uint M_)
- PowerSet (const Array_Type &array)
- ~PowerSet ()

Wrappers for the <tt>Rules</tt> member.

These will add rules to the model, which are shared by the support and the actual counter function.

- void add_rule (Rule< Array_Type, Data_Rule_Type > &rule)
- void add_rule (Rule< Array_Type, Data_Rule_Type > *rule)
- void add_rule (Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_, Data_Rule_Type *data_=nullptr, bool delete_data_=false)

Getter functions

- const std::vector< Array_Type > * get_data_ptr () const
- std::vector< Array_Type > get_data () const
- std::vector< Array_Type >::iterator begin ()
- std::vector< Array_Type >::iterator end ()
- uint size () const
- const Array_Type & operator[] (const unsigned int &i) const

Public Attributes

- Array_Type EmptyArray
- std::vector< Array_Type > data
- Rules< Array_Type, Data_Rule_Type > * rules
- uint N
- uint M
- bool rules_deleted = false
- std::vector< std::pair< uint, uint > > coordinates_free
- std::vector< std::pair< uint, uint > > coordinates_locked

6.17.1 Detailed Description

template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
class PowerSet< Array_Type, Data_Rule_Type >

Powerset of a binary array.

Template Parameters

Array_Type	
Data_Rule_Type	

Definition at line 17 of file powerset-bones.hpp.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 PowerSet() [1/3]

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
PowerSet< Array_Type, Data_Rule_Type >::PowerSet ( )  [inline]
```

Definition at line 39 of file powerset-bones.hpp.

6.17.2.2 PowerSet() [2/3]

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
PowerSet< Array_Type, Data_Rule_Type >::PowerSet (
            uint N_,
            uint M_ )  [inline]
```

Definition at line 41 of file powerset-bones.hpp.

6.17.2.3 PowerSet() [3/3]

```
template<typename Array_Type , typename Data_Rule_Type >
PowerSet< Array_Type, Data_Rule_Type >::PowerSet (
            const Array_Type & array )  [inline]
```

Definition at line 7 of file powerset-meat.hpp.

6.17.2.4 ~PowerSet()

```
template<typename Array_Type , typename Data_Rule_Type >
PowerSet< Array_Type, Data_Rule_Type >::~PowerSet  [inline]
```

Definition at line 15 of file powerset-meat.hpp.

6.17.3 Member Function Documentation

6.17.3.1 add_rule() [1/3]

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > & rule )  [inline]
```

Definition at line 113 of file powerset-meat.hpp.

6.17.3.2 add_rule() [2/3]

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > * rule )  [inline]
```

Definition at line 122 of file powerset-meat.hpp.

6.17.3.3 add_rule() [3/3]

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::add_rule (
            Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_,
            Data_Rule_Type * data_ = nullptr,
            bool delete_data_ = false )  [inline]
```

Definition at line 132 of file powerset-meat.hpp.

6.17.3.4 begin()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type >::iterator PowerSet< Array_Type, Data_Rule_Type >::begin ( )  [inline]
```

Definition at line 73 of file powerset-bones.hpp.

6.17.3.5 calc()

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::calc  [inline]
```

Definition at line 88 of file powerset-meat.hpp.

6.17.3.6 end()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type >::iterator PowerSet< Array_Type, Data_Rule_Type >::end ( )  [inline]
```

Definition at line 74 of file powerset-bones.hpp.

6.17.3.7 get_data()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type > PowerSet< Array_Type, Data_Rule_Type >::get_data ( ) const  [inline]
```

Definition at line 72 of file powerset-bones.hpp.

6.17.3.8 get_data_ptr()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
const std::vector< Array_Type >* PowerSet< Array_Type, Data_Rule_Type >::get_data_ptr ( )
const  [inline]
```

Definition at line 71 of file powerset-bones.hpp.

6.17.3.9 init_support()

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::init_support  [inline]
```

Definition at line 21 of file powerset-meat.hpp.

6.17.3.10 operator[]()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
const Array_Type& PowerSet< Array_Type, Data_Rule_Type >::operator[] (
            const unsigned int & i ) const  [inline]
```

Definition at line 76 of file powerset-bones.hpp.

6.17.3.11 reset()

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::reset (
            uint N_,
            uint M_ )  [inline]
```

Definition at line 101 of file powerset-meat.hpp.

6.17.3.12 size()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
uint PowerSet< Array_Type, Data_Rule_Type >::size ( ) const  [inline]
```

Definition at line 75 of file powerset-bones.hpp.

6.17.4 Member Data Documentation

6.17.4.1 coordinates_free

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< std::pair<uint,uint> > PowerSet< Array_Type, Data_Rule_Type >::coordinates_free
```

Definition at line 31 of file powerset-bones.hpp.

6.17.4.2 coordinates_locked

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< std::pair<uint,uint> > PowerSet< Array_Type, Data_Rule_Type >::coordinates_↩
locked
```

Definition at line 32 of file powerset-bones.hpp.

6.17.4.3 data

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type > PowerSet< Array_Type, Data_Rule_Type >::data
```

Definition at line 24 of file powerset-bones.hpp.

6.17.4.4 EmptyArray

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
Array_Type PowerSet< Array_Type, Data_Rule_Type >::EmptyArray
```

Definition at line 23 of file powerset-bones.hpp.

6.17.4.5 M

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
uint PowerSet< Array_Type, Data_Rule_Type >::M
```

Definition at line 27 of file powerset-bones.hpp.

6.17.4.6 N

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
uint PowerSet< Array_Type, Data_Rule_Type >::N
```

Definition at line 27 of file powerset-bones.hpp.

6.17.4.7 rules

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
Rules<Array_Type,Data_Rule_Type>* PowerSet< Array_Type, Data_Rule_Type >::rules
```

Definition at line 25 of file powerset-bones.hpp.

6.17.4.8 rules_deleted

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
bool PowerSet< Array_Type, Data_Rule_Type >::rules_deleted = false
```

Definition at line 28 of file powerset-bones.hpp.

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

- include/barry/powerset-bones.hpp
- include/barry/powerset-meat.hpp

6.18 Rule< Array_Type, Data_Type > Class Template Reference

Rule for determining if a cell should be included in a sequence.

```
#include <rules-bones.hpp>
```

Public Member Functions

- ~Rule ()
- bool locked (const Array_Type &a, uint i, uint j)

Construct a new Rule object

Construct a new Rule object

Parameters

fun_	*A function of type `Rule_fun_type`.*
dat_	*Data pointer to be passed to `fun_`*
delete_↩ dat_	*When `true`, the `Rule` destructor will delete the pointer, if defined.*

- Rule ()
- Rule (Rule_fun_type< Array_Type, Data_Type > fun_, Data_Type *dat_=nullptr, bool delete_dat_=false)

6.18.1 Detailed Description

template<typename Array_Type = BArray<>, typename Data_Type = bool>
class Rule< Array_Type, Data_Type >

Rule for determining if a cell should be included in a sequence.

Rules can be used together with `Support` and `PowerSet` to determine which cells should be included when enumerating all possible realizations of a binary array.

Template Parameters

Array_Type	An object of class `BArray`.
Data_Type	Any type.

Definition at line 23 of file rules-bones.hpp.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 Rule() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Rule< Array_Type, Data_Type >::Rule ( )  [inline]
```

Definition at line 42 of file rules-bones.hpp.

6.18.2.2 Rule() [2/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Rule< Array_Type, Data_Type >::Rule (
            Rule_fun_type< Array_Type, Data_Type > fun_,
            Data_Type * dat_ = nullptr,
            bool delete_dat_ = false )  [inline]
```

Definition at line 43 of file rules-bones.hpp.

6.18.2.3 ~Rule()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Rule< Array_Type, Data_Type >::~Rule ( )  [inline]
```

Definition at line 50 of file rules-bones.hpp.

6.18.3 Member Function Documentation

6.18.3.1 locked()

```
template<typename Array_Type , typename Data_Type >
bool Rule< Array_Type, Data_Type >::locked (
            const Array_Type & a,
            uint i,
            uint j )  [inline]
```

Definition at line 63 of file rules-meat.hpp.

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

- include/barry/rules-bones.hpp
- include/barry/rules-meat.hpp

6.19 Rules< Array_Type, Data_Type > Class Template Reference

Vector of objects of class Rule.

```
#include <rules-bones.hpp>
```

Public Member Functions

- Rules ()
- Rules (const Rules< Array_Type, Data_Type > &rules_)
- Rules< Array_Type, Data_Type > operator= (const Rules< Array_Type, Data_Type > &rules_)
- ~Rules ()
- uint size () const
- bool locked (const Array_Type &a, uint i, uint j)

  Check whether a given cell is free or locked.
- void clear ()
- void get_seq (const Array_Type &a, std::vector< std::pair< uint, uint > > *free, std::vector< std::pair< uint, uint > > *locked=nullptr)

  Computes the sequence of free and locked cells in an BArray.

Rule adding

Parameters

rule	

- void add_rule (Rule< Array_Type, Data_Type > &rule)
- void add_rule (Rule< Array_Type, Data_Type > *rule)

- void add_rule (Rule_fun_type< Array_Type, Data_Type > rule_, Data_Type *data_=nullptr, bool delete_data_=false)

6.19.1 Detailed Description

template<typename Array_Type, typename Data_Type>
class Rules< Array_Type, Data_Type >

Vector of objects of class Rule.

Template Parameters

Array_Type	An object of class `BArray`
Data_Type	Any type.

Definition at line 67 of file rules-bones.hpp.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 Rules() [1/2]

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::Rules ( )  [inline]
```

Definition at line 74 of file rules-bones.hpp.

6.19.2.2 Rules() [2/2]

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::Rules (
            const Rules< Array_Type, Data_Type > & rules_ )  [inline]
```

Definition at line 10 of file rules-meat.hpp.

6.19.2.3 ~Rules()

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::~Rules ( )  [inline]
```

Definition at line 79 of file rules-bones.hpp.

6.19.3 Member Function Documentation

6.19.3.1 add_rule() [1/3]

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::add_rule (
            Rule< Array_Type, Data_Type > & rule )  [inline]
```

Definition at line 68 of file rules-meat.hpp.

6.19.3.2 add_rule() [2/3]

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::add_rule (
            Rule< Array_Type, Data_Type > * rule )  [inline]
```

Definition at line 79 of file rules-meat.hpp.

6.19.3.3 add_rule() [3/3]

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::add_rule (
            Rule_fun_type< Array_Type, Data_Type > rule_,
            Data_Type * data_ = nullptr,
            bool delete_data_ = false )  [inline]
```

Definition at line 89 of file rules-meat.hpp.

6.19.3.4 clear()

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::clear  [inline]
```

Definition at line 127 of file rules-meat.hpp.

6.19.3.5 get_seq()

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::get_seq (
            const Array_Type & a,
            std::vector< std::pair< uint, uint > > * free,
            std::vector< std::pair< uint, uint > > * locked = nullptr )  [inline]
```

Computes the sequence of free and locked cells in an BArray.

Parameters

a	An object of class BArray.
free	Pointer to a vector of pairs (i, j) listing the free cells.
locked	(optional) Pointer to a vector of pairs (i, j) listing the locked cells.

Returns

Nothing.

Definition at line 139 of file rules-meat.hpp.

6.19.3.6 locked()

```
template<typename Array_Type , typename Data_Type >
bool Rules< Array_Type, Data_Type >::locked (
            const Array_Type & a,
            uint i,
            uint j )  [inline]
```

Check whether a given cell is free or locked.

Parameters

a	A BArray object
i	row position
j	col position

Returns

true If the cell is locked

false If the cell is free

Definition at line 111 of file rules-meat.hpp.

6.19.3.7 operator=()

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type > Rules< Array_Type, Data_Type >::operator= (
            const Rules< Array_Type, Data_Type > & rules_ )
```

Definition at line 35 of file rules-meat.hpp.

6.19.3.8 size()

```
template<typename Array_Type , typename Data_Type >
uint Rules< Array_Type, Data_Type >::size ( ) const  [inline]
```

Definition at line 84 of file rules-bones.hpp.

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

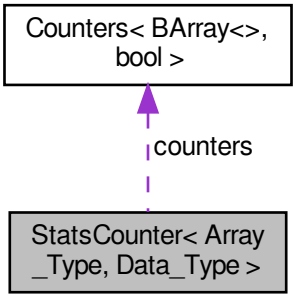
- include/barry/rules-bones.hpp
- include/barry/rules-meat.hpp

6.20 StatsCounter< Array_Type, Data_Type > Class Template Reference

Count stats for a single Array.

`#include <statscounter-bones.hpp>`

Collaboration diagram for StatsCounter< Array_Type, Data_Type >:



Public Member Functions

- StatsCounter (const Array_Type *Array_)

  Creator of a `StatsCounter`
- StatsCounter ()

  Can be created without setting the array.
- ~StatsCounter ()
- void reset_array (const Array_Type *Array_)

  Changes the reference array for the counting.
- void add_counter (Counter< Array_Type, Data_Type > *f_)
- void add_counter (Counter< Array_Type, Data_Type > f_)
- void set_counters (Counters< Array_Type, Data_Type > *counters_)
- void count_init (uint i, uint j)

  Counter functions This function recurses through the entries of `Array` and at each step of adding a new cell it uses the functions to list the statistics.
- void count_current (uint i, uint j)
- std::vector< double > count_all ()

Public Attributes

- const Array_Type * Array
- Array_Type EmptyArray
- std::vector< double > current_stats
- Counters< Array_Type, Data_Type > * counters
- bool counter_deleted = false

6.20.1 Detailed Description

template<typename Array_Type = BArray<>, typename Data_Type = bool>
class StatsCounter< Array_Type, Data_Type >

Count stats for a single Array.

Users can a list of functions that can be used with this. The baseline set of arguments is a pointer to a binary array and a dataset to add the counts to.

Definition at line 16 of file statscounter-bones.hpp.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 StatsCounter() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
StatsCounter< Array_Type, Data_Type >::StatsCounter (
            const Array_Type * Array_ )  [inline]
```

Creator of a StatsCounter

Parameters

*Array_*	A const pointer to a BArray.

Definition at line 34 of file statscounter-bones.hpp.

6.20.2.2 StatsCounter() [2/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
StatsCounter< Array_Type, Data_Type >::StatsCounter ( )  [inline]
```

Can be created without setting the array.

Definition at line 49 of file statscounter-bones.hpp.

6.20.2.3 ~StatsCounter()

```
template<typename Array_Type , typename Data_Type >
StatsCounter< Array_Type, Data_Type >::~StatsCounter  [inline]
```

Definition at line 7 of file statscounter-meat.hpp.

6.20.3 Member Function Documentation

6.20.3.1 add_counter() [1/2]

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::add_counter (
            Counter< Array_Type, Data_Type > * f_ )  [inline]
```

Definition at line 25 of file statscounter-meat.hpp.

6.20.3.2 add_counter() [2/2]

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::add_counter (
            Counter< Array_Type, Data_Type > f_ )  [inline]
```

Definition at line 35 of file statscounter-meat.hpp.

6.20.3.3 count_all()

```
template<typename Array_Type , typename Data_Type >
std::vector< double > StatsCounter< Array_Type, Data_Type >::count_all  [inline]
```

Definition at line 99 of file statscounter-meat.hpp.

6.20.3.4 count_current()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::count_current (
            uint i,
            uint j )  [inline]
```

Definition at line 81 of file statscounter-meat.hpp.

6.20.3.5 count_init()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::count_init (
            uint i,
            uint j )  [inline]
```

Counter functions This function recurses through the entries of `Array` and at each step of adding a new cell it uses the functions to list the statistics.

Definition at line 61 of file statscounter-meat.hpp.

6.20.3.6 reset_array()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::reset_array (
            const Array_Type * Array_ )  [inline]
```

Changes the reference array for the counting.

Parameters

*Array_*	A pointer to an array of class `Array_Type`.

Definition at line 14 of file statscounter-meat.hpp.

6.20.3.7 set_counters()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::set_counters (
            Counters< Array_Type, Data_Type > * counters_ )  [inline]
```

Definition at line 46 of file statscounter-meat.hpp.

6.20.4 Member Data Documentation

6.20.4.1 Array

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
const Array_Type* StatsCounter< Array_Type, Data_Type >::Array
```

Definition at line 21 of file statscounter-bones.hpp.

6.20.4.2 counter_deleted

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
bool StatsCounter< Array_Type, Data_Type >::counter_deleted = false
```

Definition at line 27 of file statscounter-bones.hpp.

6.20.4.3 counters

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Counters<Array_Type,Data_Type>* StatsCounter< Array_Type, Data_Type >::counters
```

Definition at line 26 of file statscounter-bones.hpp.

6.20.4.4 current_stats

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::vector< double > StatsCounter< Array_Type, Data_Type >::current_stats
```

Definition at line 23 of file statscounter-bones.hpp.

6.20.4.5 EmptyArray

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Array_Type StatsCounter< Array_Type, Data_Type >::EmptyArray
```

Definition at line 22 of file statscounter-bones.hpp.

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

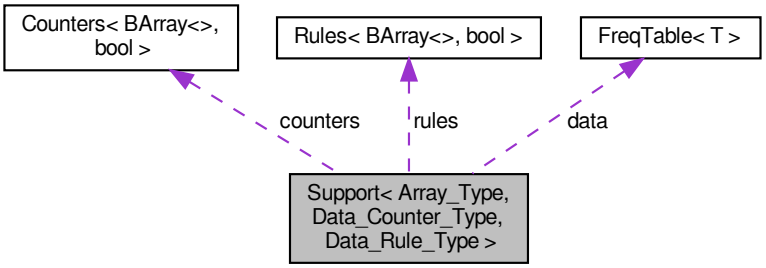
- include/barry/statscounter-bones.hpp
- include/barry/statscounter-meat.hpp

6.21 Support< Array_Type, Data_Counter_Type, Data_Rule_Type > Class Template Reference

Compute the support of sufficient statistics.

`#include <support-bones.hpp>`

Collaboration diagram for Support< Array_Type, Data_Counter_Type, Data_Rule_Type >:



Public Member Functions

- Support (const Array_Type &Array_)
  Constructor passing a reference Array.
- Support (uint N_, uint M_)
  Constructor specifying the dimensions of the array (empty).
- Support ()
- ~Support ()
- void init_support (std::vector< Array_Type > *array_bank=nullptr, std::vector< std::vector< double > > *stats_bank=nullptr)
- void calc (std::vector< Array_Type > *array_bank=nullptr, std::vector< std::vector< double > > *stats_bank=nullptr)
  Computes the entire support.
- Counts_type get_counts () const
- const MapVec_type * get_counts_ptr () const
- void print () const

Resets the support calculator

If needed, the counters of a support object can be reused.

Parameters

Array_	*New array over which the support will be computed.*

- void reset_array ()
- void reset_array (const Array_Type &Array_)

Manage counters

Parameters

f_	*A counter to be added.*
counters↩ _	*A vector of counters to be added.*

- void add_counter (Counter< Array_Type, Data_Counter_Type > *f_)
- void add_counter (Counter< Array_Type, Data_Counter_Type > f_)
- void set_counters (Counters< Array_Type, Data_Counter_Type > *counters_)

Manage rules

Parameters

f_	*A rule to be added.*
counters↩ _	*A vector of rules to be added.*

- void add_rule (Rule< Array_Type, Data_Rule_Type > *f_)
- void add_rule (Rule< Array_Type, Data_Rule_Type > f_)
- void set_rules (Rules< Array_Type, Data_Rule_Type > *rules_)

Public Attributes

- Array_Type EmptyArray

  Reference array to generate the support.
- FreqTable data
- Counters< Array_Type, Data_Counter_Type > * counters
- Rules< Array_Type, Data_Rule_Type > * rules
- uint N
- uint M
- bool counter_deleted = false
- bool rules_deleted = false
- std::vector< double > current_stats
- std::vector< std::pair< uint, uint > > coordinates_free
- std::vector< std::pair< uint, uint > > coordinates_locked
- std::vector< std::vector< double > > change_stats

6.21.1 Detailed Description

template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool>
class Support< Array_Type, Data_Counter_Type, Data_Rule_Type >

Compute the support of sufficient statistics.

Given an array and a set of counters, this object iterates throughout the support set of the Array while at the same time computing the support of the sufficient statitics.

Definition at line 24 of file support-bones.hpp.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 Support() [1/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::Support (
            const Array_Type & Array_ )  [inline]
```

Constructor passing a reference Array.

Definition at line 55 of file support-bones.hpp.

6.21.2.2 Support() [2/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::Support (
            uint N_,
            uint M_ )  [inline]
```

Constructor specifying the dimensions of the array (empty).

Definition at line 63 of file support-bones.hpp.

6.21.2.3 Support() [3/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::Support ( )  [inline]
```

Definition at line 69 of file support-bones.hpp.

6.21.2.4 ~Support()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::~Support ( )  [inline]
```

Definition at line 75 of file support-bones.hpp.

6.21.3 Member Function Documentation

6.21.3.1 add_counter() [1/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::add_counter (
            Counter< Array_Type, Data_Counter_Type > * f_ )  [inline]
```

Definition at line 171 of file support-meat.hpp.

6.21.3.2 add_counter() [2/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::add_counter (
            Counter< Array_Type, Data_Counter_Type > f_ )  [inline]
```

Definition at line 181 of file support-meat.hpp.

6.21.3.3 add_rule() [1/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > * f_ )  [inline]
```

Definition at line 208 of file support-meat.hpp.

6.21.3.4 add_rule() [2/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > f_ )  [inline]
```

Definition at line 218 of file support-meat.hpp.

6.21.3.5 calc()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::calc (
            std::vector< Array_Type > * array_bank = nullptr,
            std::vector< std::vector< double > > * stats_bank = nullptr )  [inline]
```

Computes the entire support.

Not to be used by the user. Sets the starting point in the array (column-major).

Parameters

array_bank	If specified, the counter will add to the vector each possible state of the array, as it counts.
stats_bank	If specified, the counter will add to the vector each possible set of statistics, as it counts.

Definition at line 153 of file support-meat.hpp.

6.21.3.6 get_counts()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
Counts_type Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::get_counts  [inline]
```

Definition at line 245 of file support-meat.hpp.

6.21.3.7 get_counts_ptr()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
const MapVec_type * Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::get_counts_ptr
[inline]
```

Definition at line 252 of file support-meat.hpp.

6.21.3.8 init_support()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::init_support (
            std::vector< Array_Type > * array_bank = nullptr,
            std::vector< std::vector< double > > * stats_bank = nullptr )  [inline]
```

Definition at line 7 of file support-meat.hpp.

6.21.3.9 print()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::print  [inline]
```

Definition at line 259 of file support-meat.hpp.

6.21.3.10 reset_array() [1/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::reset_array  [inline]
```

Definition at line 69 of file support-meat.hpp.

6.21.3.11 reset_array() [2/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::reset_array (
            const Array_Type & Array_ )  [inline]
```

Definition at line 76 of file support-meat.hpp.

6.21.3.12 set_counters()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::set_counters (
            Counters< Array_Type, Data_Counter_Type > * counters_ )  [inline]
```

Definition at line 191 of file support-meat.hpp.

6.21.3.13 set_rules()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::set_rules (
            Rules< Array_Type, Data_Rule_Type > * rules_ )  [inline]
```

Definition at line 228 of file support-meat.hpp.

6.21.4 Member Data Documentation

6.21.4.1 change_stats

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::vector< std::vector< double > > Support< Array_Type, Data_Counter_Type, Data_Rule_Type
>::change_stats
```

Definition at line 51 of file support-bones.hpp.

6.21.4.2 coordinates_free

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::vector< std::pair<uint,uint> > Support< Array_Type, Data_Counter_Type, Data_Rule_Type
>::coordinates_free
```

Definition at line 49 of file support-bones.hpp.

6.21.4.3 coordinates_locked

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::vector< std::pair<uint,uint> > Support< Array_Type, Data_Counter_Type, Data_Rule_Type
>::coordinates_locked
```

Definition at line 50 of file support-bones.hpp.

6.21.4.4 counter_deleted

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::counter_deleted = false
```

Definition at line 44 of file support-bones.hpp.

6.21.4.5 counters

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
Counters<Array_Type,Data_Counter_Type>* Support< Array_Type, Data_Counter_Type, Data_Rule_↩
Type >::counters
```

Definition at line 40 of file support-bones.hpp.

6.21.4.6 current_stats

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
std::vector< double > Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::current_stats
```

Definition at line 48 of file support-bones.hpp.

6.21.4.7 data

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
FreqTable Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::data
```

Definition at line 39 of file support-bones.hpp.

6.21.4.8 EmptyArray

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
Array_Type Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::EmptyArray
```

Reference array to generate the support.

Definition at line 38 of file support-bones.hpp.

6.21.4.9 M

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::M
```

Definition at line 43 of file support-bones.hpp.

6.21.4.10 N

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::N
```

Definition at line 43 of file support-bones.hpp.

6.21.4.11 rules

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
Rules<Array_Type,Data_Rule_Type>* Support< Array_Type, Data_Counter_Type, Data_Rule_Type >↩
::rules
```

Definition at line 41 of file support-bones.hpp.

6.21.4.12 rules_deleted

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::rules_deleted = false
```

Definition at line 45 of file support-bones.hpp.

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

- include/barry/support-bones.hpp
- include/barry/support-meat.hpp

6.22 vecHasher< T > Struct Template Reference

```
#include <typedefs.hpp>
```

Public Member Functions

- std::size_t operator() (std::vector< T > const &dat) const noexcept

6.22.1 Detailed Description

template<typename T>
struct vecHasher< T >

Definition at line 99 of file typedefs.hpp.

6.22.2 Member Function Documentation

6.22.2.1 operator()()

```
template<typename T >
std::size_t vecHasher< T >::operator() (
            std::vector< T > const & dat ) const  [inline], [noexcept]
```

Definition at line 100 of file typedefs.hpp.

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

- include/barry/typedefs.hpp

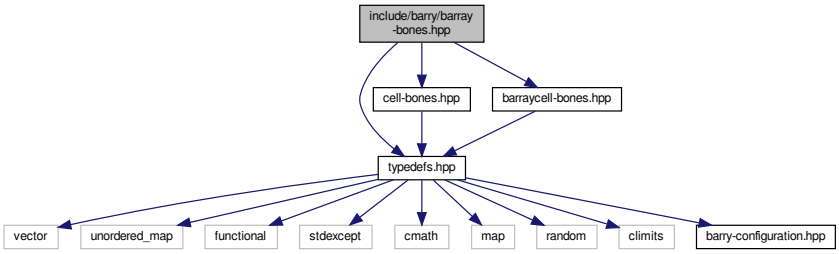
Chapter 7

File Documentation

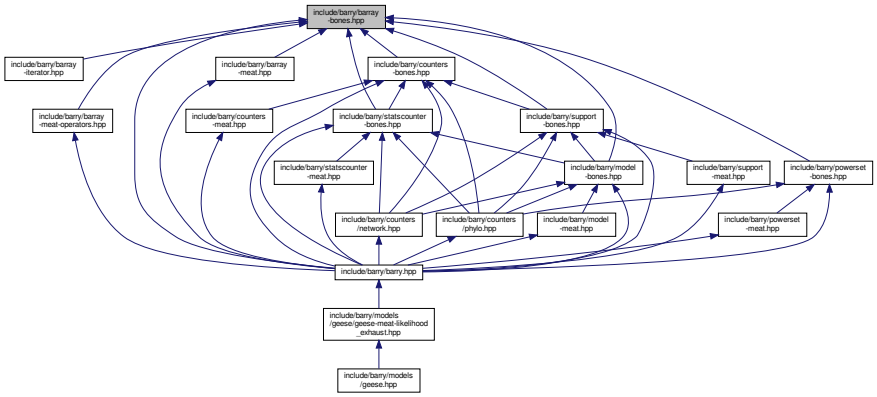
7.1 include/barry/barray-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "cell-bones.hpp"
#include "barraycell-bones.hpp"
```

Include dependency graph for barray-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class BArray< Cell_Type, Data_Type >

  Baseline class for binary arrays.

Macros

- #define BARRAY_BONES_HPP 1

7.1.1 Macro Definition Documentation

7.1.1.1 BARRAY_BONES_HPP

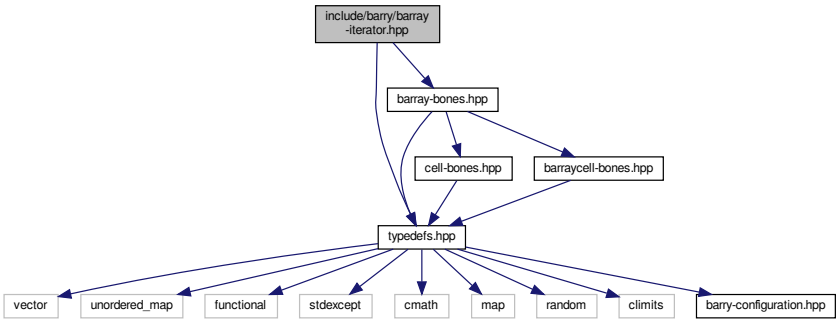
```
#define BARRAY_BONES_HPP 1
```

Definition at line 8 of file barray-bones.hpp.

7.2 include/barry/barray-iterator.hpp File Reference

```
#include "typedefs.hpp"
#include "barray-bones.hpp"
```

Include dependency graph for barray-iterator.hpp:



Classes

- class ConstBArrayRowIter< Cell_Type, Data_Type >

Macros

- #define BARRAY_ITERATOR_HPP 1

7.2.1 Macro Definition Documentation

7.2.1.1 BARRAY_ITERATOR_HPP

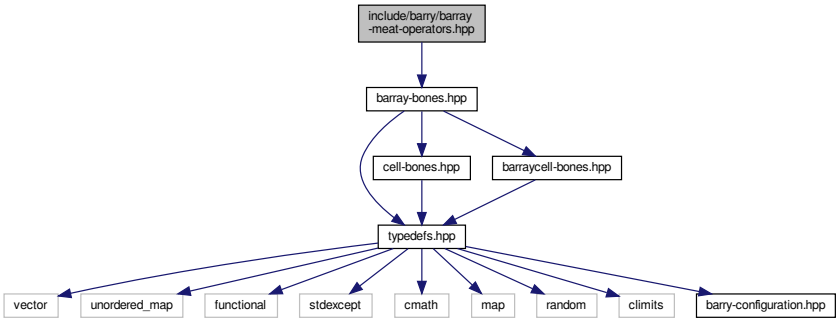
```
#define BARRAY_ITERATOR_HPP 1
```

Definition at line 7 of file barray-iterator.hpp.

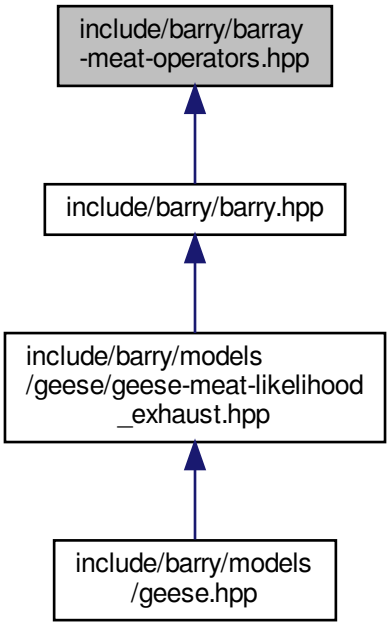
7.3 include/barry/barray-meat-operators.hpp File Reference

```
#include "barray-bones.hpp"
```

Include dependency graph for barray-meat-operators.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- template<typename Cell_Type , typename Data_Type >
  void checkdim_ (const BArray< Cell_Type, Data_Type > &lhs, const BArray< Cell_Type, Data_Type > &rhs)

7.3.1 Function Documentation

7.3.1.1 checkdim_()

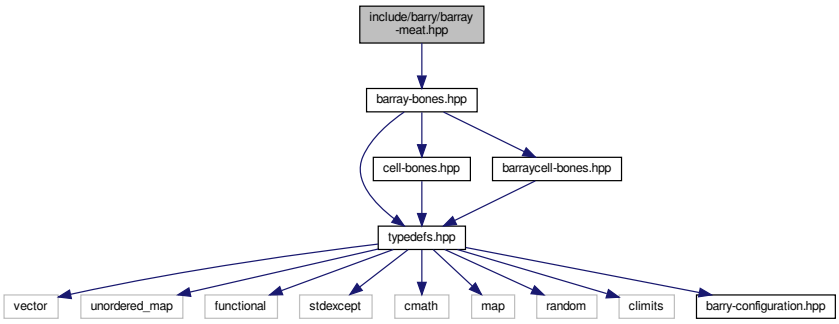
```
template<typename Cell_Type , typename Data_Type >
void checkdim_ (
            const BArray< Cell_Type, Data_Type > & lhs,
            const BArray< Cell_Type, Data_Type > & rhs )  [inline]
```

Definition at line 8 of file barray-meat-operators.hpp.

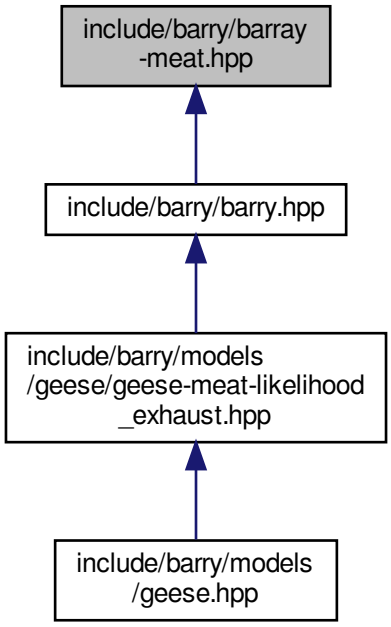
7.4 include/barry/barray-meat.hpp File Reference

```
#include "barray-bones.hpp"
```

Include dependency graph for barray-meat.hpp:



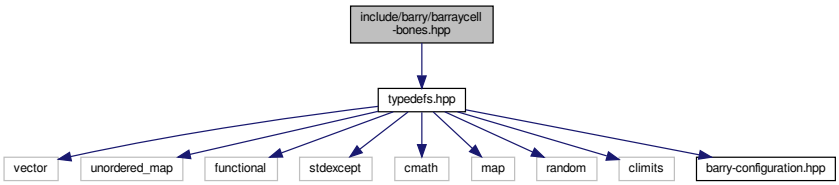
This graph shows which files directly or indirectly include this file:



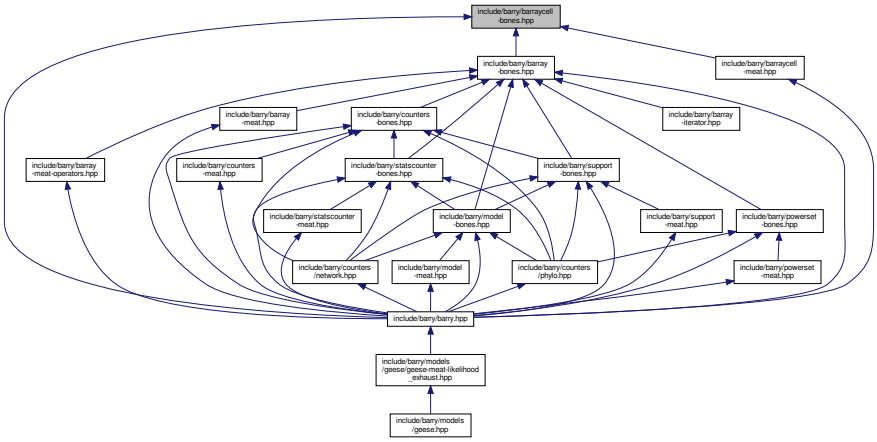
7.5 include/barry/barraycell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barraycell-bones.hpp:



This graph shows which files directly or indirectly include this file:



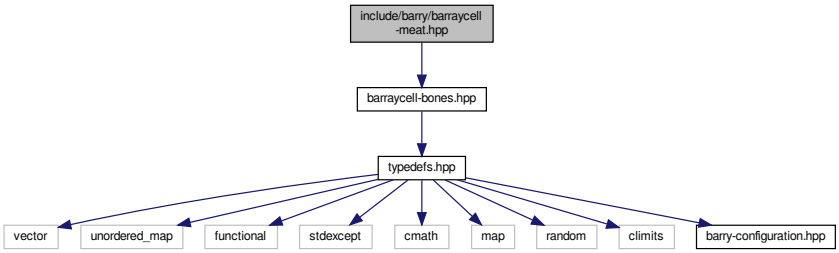
Classes

- class BArrayCell< Cell_Type, Data_Type >
- class BArrayCell_const< Cell_Type, Data_Type >

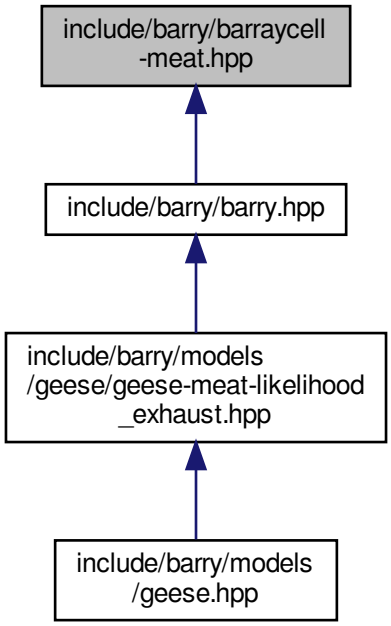
7.6 include/barry/barraycell-meat.hpp File Reference

```
#include "barraycell-bones.hpp"
```

Include dependency graph for barraycell-meat.hpp:

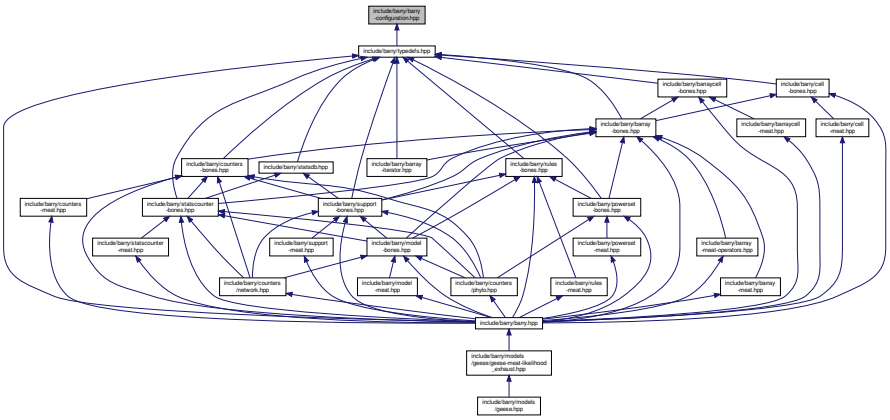


This graph shows which files directly or indirectly include this file:



7.7 include/barry/barry-configuration.hpp File Reference

This graph shows which files directly or indirectly include this file:



Configuration MACROS

These are mostly related to performance. The definitions follow:

- BARRY_USE_UNORDERED_MAP If specified, then barry is compiled using std::unordered_map. Otherwise it will use std::map for the arrays.
- BARRY_USE_SAFE_EXP When specified, it will multiply all likelihoods in Model by (1/-100)/(1/-100) so that numerical overflows are avoided.
- BARRY_CHECK_FINITE When specified, it will introduce a macro

- #define BARRY_SAFE_EXP -100.0
- #define BARRY_ISFINITE(a)
- template<typename Ta , typename Tb >
  using Map = std::map< Ta, Tb >

7.7.1 Macro Definition Documentation

7.7.1.1 BARRY_ISFINITE

```
#define BARRY_ISFINITE(
              a )
```

Definition at line 35 of file barry-configuration.hpp.

7.7.1.2 BARRY_SAFE_EXP

```
#define BARRY_SAFE_EXP -100.0
```

Definition at line 28 of file barry-configuration.hpp.

7.7.2 Typedef Documentation

7.7.2.1 Map

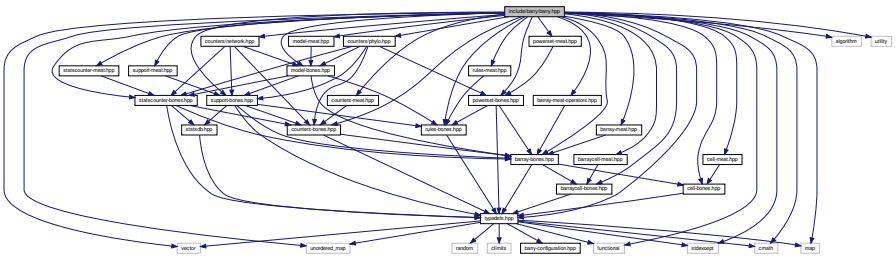
```
template<typename Ta , typename Tb >
using Map = std::map<Ta,Tb>
```

Definition at line 22 of file barry-configuration.hpp.

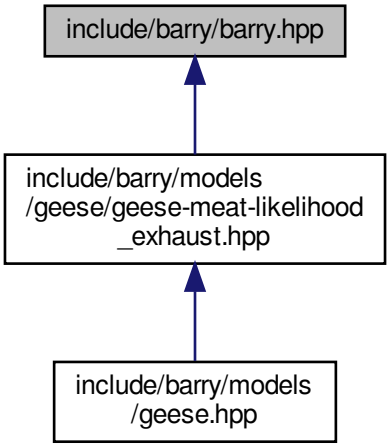
7.8 include/barry/barry.hpp File Reference

```
#include <vector>
#include <unordered_map>
#include <functional>
#include <stdexcept>
#include <cmath>
#include <map>
#include <algorithm>
#include <utility>
#include "typedefs.hpp"
#include "cell-bones.hpp"
#include "cell-meat.hpp"
#include "barray-bones.hpp"
#include "barraycell-bones.hpp"
#include "barray-meat.hpp"
#include "barraycell-meat.hpp"
#include "barray-meat-operators.hpp"
#include "counters-bones.hpp"
#include "counters-meat.hpp"
#include "statscounter-bones.hpp"
#include "statscounter-meat.hpp"
#include "support-bones.hpp"
#include "support-meat.hpp"
#include "powerset-bones.hpp"
#include "powerset-meat.hpp"
#include "model-bones.hpp"
#include "model-meat.hpp"
#include "rules-bones.hpp"
#include "rules-meat.hpp"
#include "counters/network.hpp"
#include "counters/phylo.hpp"
```

Include dependency graph for barry.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- barry

  barry: Your go-to motif accountant
- barry::counters

  Tree class and TreeIterator class.
- barry::counters::network
- barry::counters::phylo

Macros

- #define COUNTER_FUNCTION(a)
- #define COUNTER_LAMBDA(a)
- #define RULE_FUNCTION(a)
- #define RULE_LAMBDA(a)

7.8.1 Macro Definition Documentation

7.8.1.1 COUNTER_FUNCTION

```
#define COUNTER_FUNCTION(
              a )
```

Value:

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
inline double (a) (const Array_Type & Array, uint i, uint j, Data_Type * data)\
```

Definition at line 64 of file barry.hpp.

7.8.1.2 COUNTER_LAMBDA

```
#define COUNTER_LAMBDA(
              a )
```

Value:

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
Counter_fun_type<Array_Type, Data_Type> a = \
[](const Array_Type & Array, uint i, uint j, Data_Type * data)
```

Definition at line 67 of file barry.hpp.

7.8.1.3 RULE_FUNCTION

```
#define RULE_FUNCTION(
              a )
```

Value:

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
inline bool (a) (const Array_Type & Array, uint i, uint j, Data_Type * data)\
```

Definition at line 71 of file barry.hpp.

7.8.1.4 RULE_LAMBDA

```
#define RULE_LAMBDA(
              a )
```

Value:

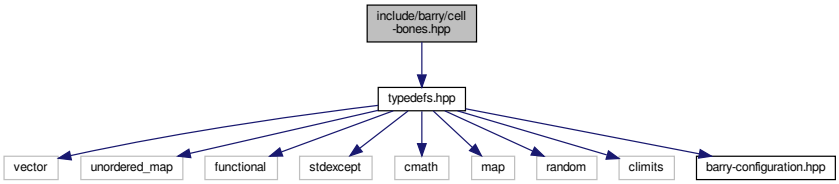
```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
Rule_fun_type<Array_Type, Data_Type> a = \
[](const Array_Type & Array, uint i, uint j, Data_Type * data)
```

Definition at line 74 of file barry.hpp.

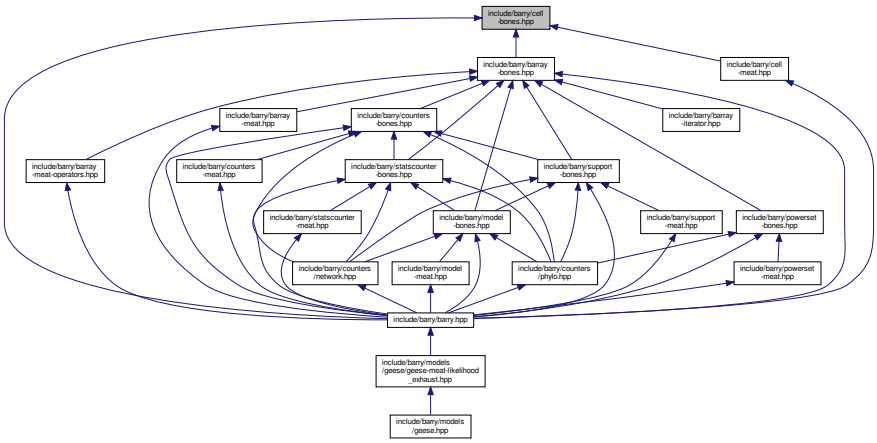
7.9 include/barry/cell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for cell-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

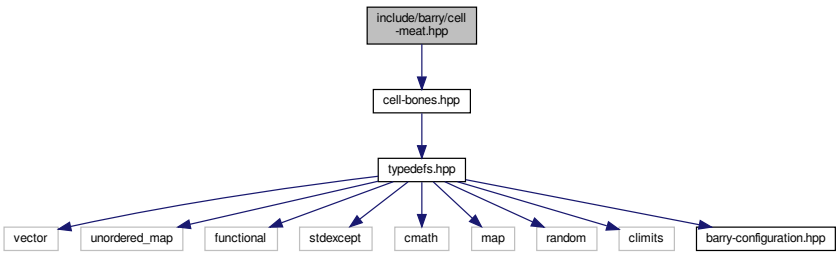
- class Cell< Cell_Type >

  Entries in BArray. For now, it only has two members:

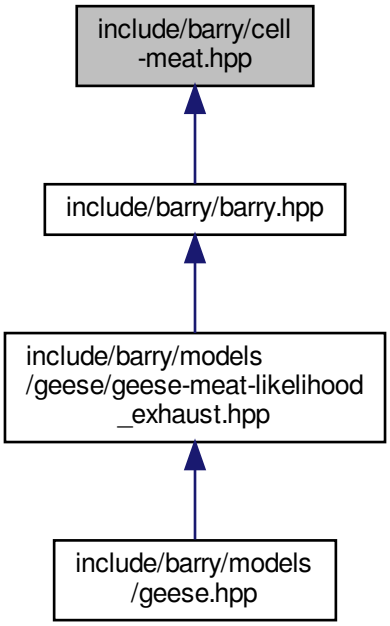
7.10 include/barry/cell-meat.hpp File Reference

```
#include "cell-bones.hpp"
```

Include dependency graph for cell-meat.hpp:



This graph shows which files directly or indirectly include this file:

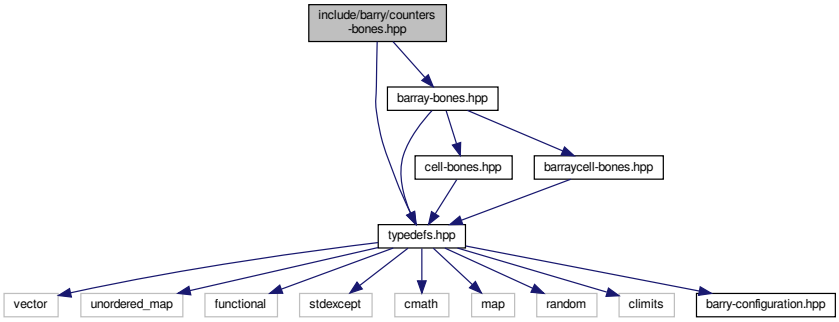


7.11 include/barry/col-bones.hpp File Reference

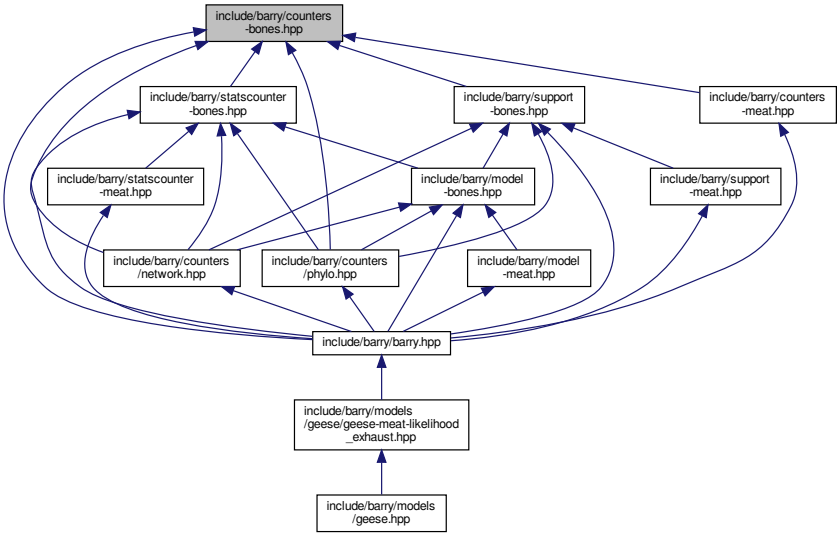
7.12 include/barry/counters-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "barray-bones.hpp"
```

Include dependency graph for counters-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

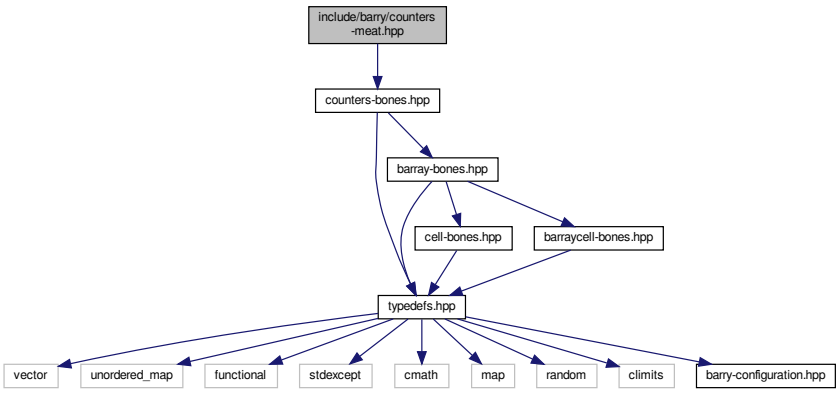
- class Counter< Array_Type, Data_Type >

  A counter function based on change statistics.
- class Counters< Array_Type, Data_Type >

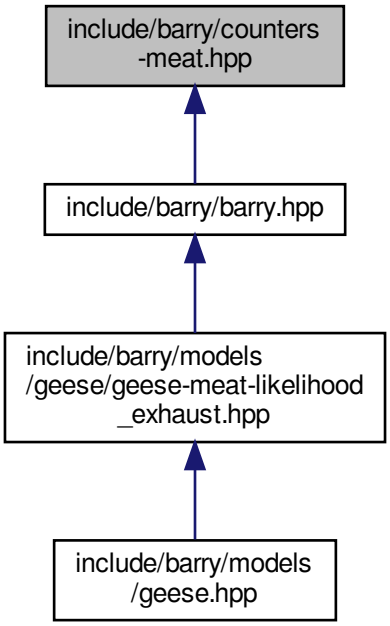
  Vector of counters.

7.13 include/barry/counters-meat.hpp File Reference

`#include "counters-bones.hpp"`
Include dependency graph for counters-meat.hpp:



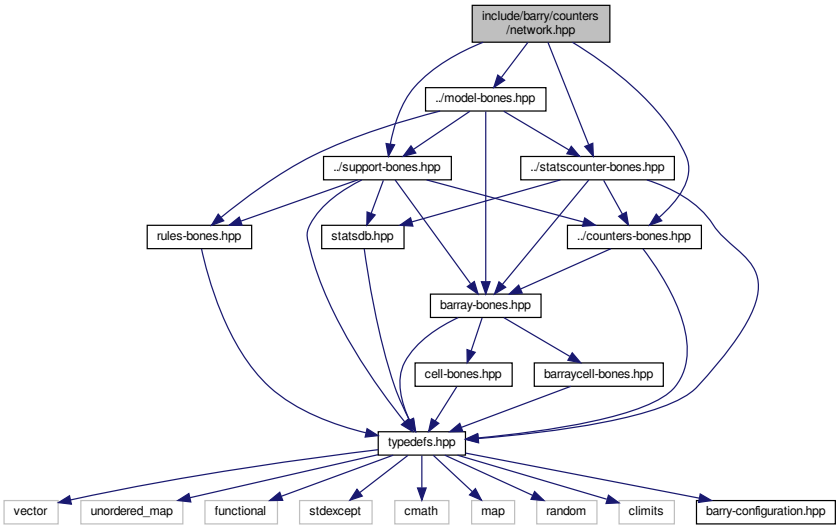
This graph shows which files directly or indirectly include this file:



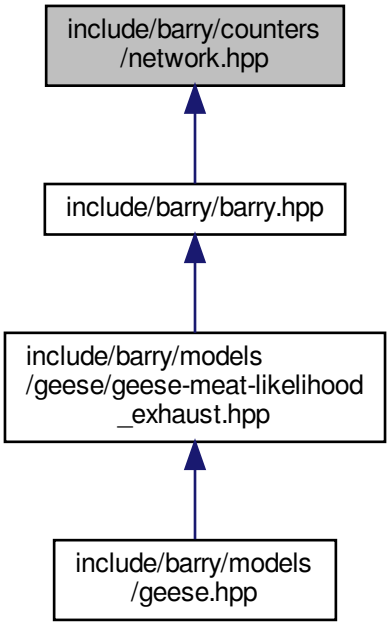
7.14 include/barry/counters/network.hpp File Reference

```
#include "../counters-bones.hpp"
#include "../support-bones.hpp"
#include "../statscounter-bones.hpp"
#include "../model-bones.hpp"
```

Include dependency graph for network.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class NetworkData

Data class for Networks.

- class NetCounterData

  Data class used to store arbitrary uint or double vectors.

Macros

- #define NET_C_DATA_IDX(i) (data->indices[i])
- #define NET_C_DATA_NUM(i) (data->numbers[i])

Macros for defining counters

- #define NETWORK_COUNTER(a)
- #define NETWORK_COUNTER_LAMBDA(a)

Macros for defining rules

- #define NETWORK_RULE(a)
- #define NETWORK_RULE_LAMBDA(a)

Typedefs

Convenient typedefs for network objects.

- typedef BArray< double, NetworkData > Network
- typedef Counter< Network, NetCounterData > NetCounter
- typedef Counters< Network, NetCounterData > NetCounters
- typedef Support< Network, NetCounterData > NetSupport
- typedef StatsCounter< Network, NetCounterData > NetStatsCounter
- typedef Model< Network, NetCounterData > NetModel
- typedef Rule< Network, bool > NetRule
- typedef Rules< Network, bool > NetRules

Functions

Counters for network models

Parameters

counters	*A pointer to a* `NetCounters` *object (*`Counters<Network, NetCounterData>`*).*

- void counter_edges (NetCounters *counters)

  Number of edges.
- void counter_isolates (NetCounters *counters)

  Number of isolated vertices.
- void counter_mutual (NetCounters *counters)

  Number of mutual ties.
- void counter_istar2 (NetCounters *counters)
- void counter_ostar2 (NetCounters *counters)
- void counter_ttriads (NetCounters *counters)
- void counter_ctriads (NetCounters *counters)
- void counter_density (NetCounters *counters)
- void counter_idegree15 (NetCounters *counters)

- void counter_odegree15 (NetCounters *counters)
- void counter_absdiff (NetCounters *counters, uint attr_id, double alpha=1.0)
  Sum of absolute attribute difference between ego and alter.
- void counter_diff (NetCounters *counters, uint attr_id, double alpha=1.0, double tail_head=true)
  Sum of attribute difference between ego and alter to pow(alpha)
- NETWORK_COUNTER (init_single_attr)
- void counter_nodeicov (NetCounters *counters, uint attr_id)
- void counter_nodeocov (NetCounters *counters, uint attr_id)
- void counter_nodecov (NetCounters *counters, uint attr_id)
- void counter_nodematch (NetCounters *counters, uint attr_id)
- void counter_idegree (NetCounters *counters, std::vector< uint > d)
  Counts number of vertices with a given in-degree.
- void counter_odegree (NetCounters *counters, std::vector< uint > d)
  Counts number of vertices with a given out-degree.
- void counter_degree (NetCounters *counters, std::vector< uint > d)
  Counts number of vertices with a given out-degree.

Rules for network models

Parameters

rules	*A pointer to a* `NetRules` *object (*`Rules<Network, bool>`*).*

- void rules_zerodiag (NetRules *rules)
  Number of edges.

7.14.1 Macro Definition Documentation

7.14.1.1 NET_C_DATA_IDX

```
#define NET_C_DATA_IDX(
              i ) (data->indices[i])
```

Definition at line 68 of file network.hpp.

7.14.1.2 NET_C_DATA_NUM

```
#define NET_C_DATA_NUM(
              i ) (data->numbers[i])
```

Definition at line 69 of file network.hpp.

7.14.1.3 NETWORK_COUNTER

```
#define NETWORK_COUNTER(
                a )
```

Value:

```
inline double (a) \
(const Network & Array, uint i, uint j, NetCounterData * data)
```

Function for definition of a network counter function

Definition at line 89 of file network.hpp.

7.14.1.4 NETWORK_COUNTER_LAMBDA

```
#define NETWORK_COUNTER_LAMBDA(
                a )
```

Value:

```
  Counter_fun_type<Network, NetCounterData> a = \
  [](const Network & Array, uint i, uint j, NetCounterData * data)
```

Lambda function for definition of a network counter function

Definition at line 92 of file network.hpp.

7.14.1.5 NETWORK_RULE

```
#define NETWORK_RULE(
                a )
```

Value:

```
inline bool (a) \
(const Network & Array, uint i, uint j, bool * data)
```

Function for definition of a network counter function

Definition at line 101 of file network.hpp.

7.14.1.6 NETWORK_RULE_LAMBDA

```
#define NETWORK_RULE_LAMBDA(
                a )
```

Value:

```
Rule_fun_type<Network, bool> a = \
[](const Network & Array, uint i, uint j, bool * data)
```

Lambda function for definition of a network counter function

Definition at line 104 of file network.hpp.

7.14.2 Typedef Documentation

7.14.2.1 NetCounter

```
typedef Counter<Network, NetCounterData > NetCounter
```

Definition at line 76 of file network.hpp.

7.14.2.2 NetCounters

```
typedef Counters< Network, NetCounterData> NetCounters
```

Definition at line 77 of file network.hpp.

7.14.2.3 NetModel

```
typedef Model<Network, NetCounterData> NetModel
```

Definition at line 80 of file network.hpp.

7.14.2.4 NetRule

```
typedef Rule<Network,bool> NetRule
```

Definition at line 81 of file network.hpp.

7.14.2.5 NetRules

```
typedef Rules<Network,bool> NetRules
```

Definition at line 82 of file network.hpp.

7.14.2.6 NetStatsCounter

```
typedef StatsCounter<Network, NetCounterData> NetStatsCounter
```

Definition at line 79 of file network.hpp.

7.14.2.7 NetSupport

```
typedef Support<Network, NetCounterData > NetSupport
```

Definition at line 78 of file network.hpp.

7.14.2.8 Network

```
typedef BArray<double, NetworkData> Network
```

Definition at line 75 of file network.hpp.

7.14.3 Function Documentation

7.14.3.1 counter_absdiff()

```
void counter_absdiff (
            NetCounters * counters,
            uint attr_id,
            double alpha = 1.0 )  [inline]
```

Sum of absolute attribute difference between ego and alter.

Definition at line 405 of file network.hpp.

7.14.3.2 counter_ctriads()

```
void counter_ctriads (
            NetCounters * counters )  [inline]
```

Definition at line 308 of file network.hpp.

7.14.3.3 counter_degree()

```
void counter_degree (
            NetCounters * counters,
            std::vector< uint > d )  [inline]
```

Counts number of vertices with a given out-degree.

Definition at line 676 of file network.hpp.

7.14.3.4 counter_density()

```
void counter_density (
            NetCounters * counters )  [inline]
```

Definition at line 347 of file network.hpp.

7.14.3.5 counter_diff()

```
void counter_diff (
            NetCounters * counters,
            uint attr_id,
            double alpha = 1.0,
            double tail_head = true )  [inline]
```

Sum of attribute difference between ego and alter to pow(alpha)

Definition at line 447 of file network.hpp.

7.14.3.6 counter_edges()

```
void counter_edges (
            NetCounters * counters )  [inline]
```

Number of edges.

Definition at line 114 of file network.hpp.

7.14.3.7 counter_idegree()

```
void counter_idegree (
            NetCounters * counters,
            std::vector< uint > d )  [inline]
```

Counts number of vertices with a given in-degree.

Definition at line 590 of file network.hpp.

7.14.3.8 counter_idegree15()

```
void counter_idegree15 (
            NetCounters * counters )  [inline]
```

Definition at line 363 of file network.hpp.

7.14.3.9 counter_isolates()

```
void counter_isolates (
            NetCounters * counters )  [inline]
```

Number of isolated vertices.

Definition at line 128 of file network.hpp.

7.14.3.10 counter_istar2()

```
void counter_istar2 (
            NetCounters * counters )  [inline]
```

Definition at line 196 of file network.hpp.

7.14.3.11 counter_mutual()

```
void counter_mutual (
            NetCounters * counters )  [inline]
```

Number of mutual ties.

Definition at line 158 of file network.hpp.

7.14.3.12 counter_nodecov()

```
void counter_nodecov (
            NetCounters * counters,
            uint attr_id )  [inline]
```

Definition at line 544 of file network.hpp.

7.14.3.13 counter_nodeicov()

```
void counter_nodeicov (
            NetCounters * counters,
            uint attr_id )  [inline]
```

Definition at line 506 of file network.hpp.

7.14.3.14 counter_nodematch()

```
void counter_nodematch (
            NetCounters * counters,
            uint attr_id )  [inline]
```

Definition at line 564 of file network.hpp.

7.14.3.15 counter_nodeocov()

```
void counter_nodeocov (
            NetCounters * counters,
            uint attr_id )  [inline]
```

Definition at line 525 of file network.hpp.

7.14.3.16 counter_odegree()

```
void counter_odegree (
            NetCounters * counters,
            std::vector< uint > d )  [inline]
```

Counts number of vertices with a given out-degree.

Definition at line 632 of file network.hpp.

7.14.3.17 counter_odegree15()

```
void counter_odegree15 (
            NetCounters * counters )  [inline]
```

Definition at line 383 of file network.hpp.

7.14.3.18 counter_ostar2()

```
void counter_ostar2 (
            NetCounters * counters )  [inline]
```

Definition at line 214 of file network.hpp.

7.14.3.19 counter_ttriads()

```
void counter_ttriads (
            NetCounters * counters )  [inline]
```

Definition at line 233 of file network.hpp.

7.14.3.20 NETWORK_COUNTER()

```
NETWORK_COUNTER (
            init_single_attr  )
```

Definition at line 489 of file network.hpp.

7.14.3.21 rules_zerodiag()

```
void rules_zerodiag (
            NetRules * rules )  [inline]
```

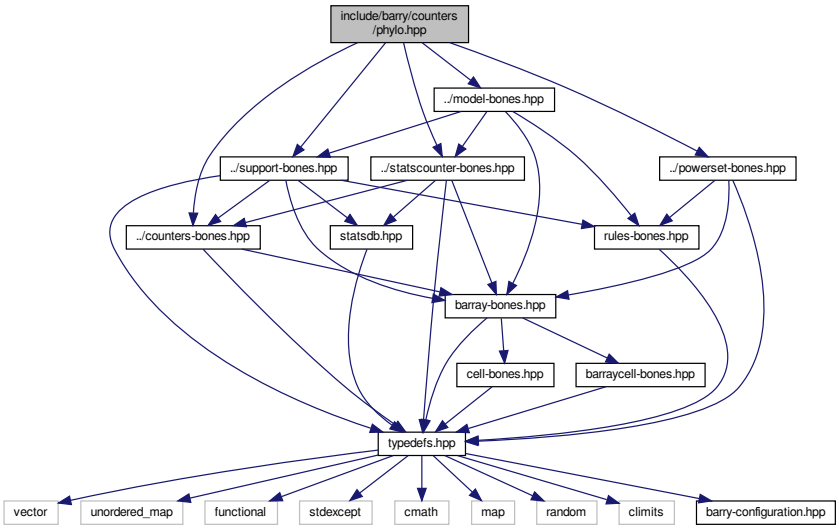
Number of edges.

Definition at line 727 of file network.hpp.

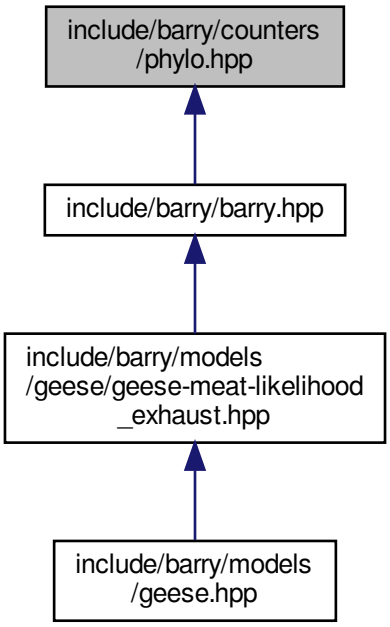
7.15 include/barry/counters/phylo.hpp File Reference

```
#include "../counters-bones.hpp"
#include "../support-bones.hpp"
#include "../statscounter-bones.hpp"
#include "../model-bones.hpp"
#include "../powerset-bones.hpp"
```

Include dependency graph for phylo.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class NodeData

  Data definition for the `PhyloArray` class.

Macros

- #define PHYLO_C_DATA_IDX(i) (data.operator[](i))
- #define PHYLO_COUNTER(a)

  Extension of a simple counter.
- #define PHYLO_COUNTER_LAMBDA(a)
- #define PHYLO_CHECK_MISSING()

Typedefs

- typedef std::vector< uint > PhyloCounterData
- typedef std::vector< std::pair< uint, uint > > PhyloRuleData

Convenient typedefs for Node objects.

- typedef BArray< uint, NodeData > PhyloArray
- typedef Counter< PhyloArray, PhyloCounterData > PhyloCounter
- typedef Counters< PhyloArray, PhyloCounterData > PhyloCounters
- typedef Rule< PhyloArray, PhyloRuleData > PhyloRule
- typedef Rules< PhyloArray, PhyloRuleData > PhyloRules
- typedef Support< PhyloArray, PhyloCounterData, PhyloRuleData > PhyloSupport
- typedef StatsCounter< PhyloArray, PhyloCounterData > PhyloStatsCounter
- typedef Model< PhyloArray, PhyloCounterData, PhyloRuleData > PhyloModel
- typedef PowerSet< PhyloArray, PhyloRuleData > PhyloPowerSet

Functions

Counters for phylogenetic modeling.

Parameters

counters	*A pointer to a* `PhyloCounters` *object (*`Counters<PhyloArray, PhyloCounterData>`*).*

- void counter_overall_gains (PhyloCounters *counters, bool duplication=true)
  Overall functional gains.
- void counter_gains (PhyloCounters *counters, std::vector< uint > nfun, bool duplication=true)
  Functional gains for a specific function (`nfun`*).*
- void counter_gains_k_offspring (PhyloCounters *counters, std::vector< uint > nfun, uint k=1u, bool duplication=true)
  Functional gains for a specific function (`nfun`*).*
- void counter_genes_changing (PhyloCounters *counters, bool duplication=true)
  Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)
- void counter_overall_loss (PhyloCounters *counters, bool duplication=true)
  Overall functional loss.
- void counter_maxfuns (PhyloCounters *counters, uint lb, uint ub, bool duplication=true)
  Cap the number of functions per gene.
- void counter_loss (PhyloCounters *counters, std::vector< uint > nfun, bool duplication=true)
  Total count of losses for an specific function.
- void counter_overall_changes (PhyloCounters *counters, bool duplication=true)
  Total number of changes. Use this statistic to account for "preservation".
- void counter_subfun (PhyloCounters *counters, uint nfunA, uint nfunB, bool duplication=true)
  Total count of Sub-functionalization events.
- void counter_cogain (PhyloCounters *counters, uint nfunA, uint nfunB, bool duplication=true)
  Co-evolution (joint gain or loss)
- void counter_longest (PhyloCounters *counters)
  Longest branch mutates (either by gain or by loss)
- void counter_neofun (PhyloCounters *counters, uint nfunA, uint nfunB, bool duplication=true)
  Total number of neofunctionalization events.
- void counter_neofun_a2b (PhyloCounters *counters, uint nfunA, uint nfunB, bool duplication=true)
  Total number of neofunctionalization events.
- void counter_co_opt (PhyloCounters *counters, uint nfunA, uint nfunB, bool duplication=true)
  Function co-opting.

7.15.1 Macro Definition Documentation

7.15.1.1 PHYLO_C_DATA_IDX

```
#define PHYLO_C_DATA_IDX(
              i ) (data.operator[](i))
```

Definition at line 49 of file phylo.hpp.

7.15.1.2 PHYLO_CHECK_MISSING

```
#define PHYLO_CHECK_MISSING( )
```

Value:

```
    if (Array.data == nullptr) \
    throw std::logic_error("The array data is nullptr."); \
    if (data == nullptr) \
    throw std::logic_error("The counter data is nullptr.")
```

Definition at line 84 of file phylo.hpp.

7.15.1.3 PHYLO_COUNTER

```
#define PHYLO_COUNTER(
              a )
```

Value:

```
    inline double (a) (const PhyloArray & Array, uint i, \
    uint j, PhyloCounterData * data)
```

Extension of a simple counter.

It allows specifying extra arguments, in particular, the corresponding sets of rows to which this statistic may be relevant. This could be important in the case of, for example, counting correlation type statistics between function 1 and 2, and between function 1 and 3.

Definition at line 78 of file phylo.hpp.

7.15.1.4 PHYLO_COUNTER_LAMBDA

```
#define PHYLO_COUNTER_LAMBDA(
              a )
```

Value:

```
    Counter_fun_type<PhyloArray, PhyloCounterData> a = \
    [](const PhyloArray & Array, uint i, uint j, PhyloCounterData * data)
```

Definition at line 81 of file phylo.hpp.

7.15.2 Typedef Documentation

7.15.2.1 PhyloArray

```
typedef BArray<uint, NodeData> PhyloArray
```

Definition at line 56 of file phylo.hpp.

7.15.2.2 PhyloCounter

```
typedef Counter<PhyloArray, PhyloCounterData > PhyloCounter
```

Definition at line 57 of file phylo.hpp.

7.15.2.3 PhyloCounterData

```
typedef std::vector< uint > PhyloCounterData
```

Definition at line 46 of file phylo.hpp.

7.15.2.4 PhyloCounters

```
typedef Counters< PhyloArray, PhyloCounterData> PhyloCounters
```

Definition at line 58 of file phylo.hpp.

7.15.2.5 PhyloModel

```
typedef Model<PhyloArray, PhyloCounterData, PhyloRuleData> PhyloModel
```

Definition at line 63 of file phylo.hpp.

7.15.2.6 PhyloPowerSet

```
typedef PowerSet<PhyloArray, PhyloRuleData> PhyloPowerSet
```

Definition at line 64 of file phylo.hpp.

7.15.2.7 PhyloRule

```
typedef Rule<PhyloArray,PhyloRuleData> PhyloRule
```

Definition at line 59 of file phylo.hpp.

7.15.2.8 PhyloRuleData

```
typedef std::vector< std::pair< uint, uint > > PhyloRuleData
```

Definition at line 47 of file phylo.hpp.

7.15.2.9 PhyloRules

```
typedef Rules<PhyloArray,PhyloRuleData> PhyloRules
```

Definition at line 60 of file phylo.hpp.

7.15.2.10 PhyloStatsCounter

```
typedef StatsCounter<PhyloArray, PhyloCounterData> PhyloStatsCounter
```

Definition at line 62 of file phylo.hpp.

7.15.2.11 PhyloSupport

```
typedef Support<PhyloArray, PhyloCounterData, PhyloRuleData> PhyloSupport
```

Definition at line 61 of file phylo.hpp.

7.15.3 Function Documentation

7.15.3.1 counter_co_opt()

```
void counter_co_opt (
            PhyloCounters * counters,
            uint nfunA,
            uint nfunB,
            bool duplication = true )  [inline]
```

Function co-opting.

Function co-opting of functions A and B happens when, for example, function B is gained as a new featured leveraging what function A already does; without losing function A. The sufficient statistic is defined as follows:

$$x_{pa}(1-x_{pb})\sum_{i<j}\left[x^p_{ia}(1-x^p_{ib})x^p_{ja}x^p_{jb}+x^p_{ja}(1-x^p_{jb})x^p_{ia}x^p_{ib}\right]$$

This algorithm implements the change statistic.

Definition at line 931 of file phylo.hpp.

7.15.3.2 counter_cogain()

```
void counter_cogain (
            PhyloCounters * counters,
            uint nfunA,
            uint nfunB,
            bool duplication = true )  [inline]
```

Co-evolution (joint gain or loss)

Needs to specify pairs of functions (`nfunA`, `nfunB`).

Definition at line 618 of file phylo.hpp.

7.15.3.3 counter_gains()

```
void counter_gains (
            PhyloCounters * counters,
            std::vector< uint > nfun,
            bool duplication = true )  [inline]
```

Functional gains for a specific function (`nfun`).

Definition at line 132 of file phylo.hpp.

7.15.3.4 counter_gains_k_offspring()

```
void counter_gains_k_offspring (
            PhyloCounters * counters,
            std::vector< uint > nfun,
            uint k = 1u,
            bool duplication = true )  [inline]
```

Functional gains for a specific function (`nfun`).

Definition at line 167 of file phylo.hpp.

7.15.3.5 counter_genes_changing()

```
void counter_genes_changing (
            PhyloCounters * counters,
            bool duplication = true )  [inline]
```

Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)

Definition at line 230 of file phylo.hpp.

7.15.3.6 counter_longest()

```
void counter_longest (
            PhyloCounters * counters )  [inline]
```

Longest branch mutates (either by gain or by loss)

Definition at line 664 of file phylo.hpp.

7.15.3.7 counter_loss()

```
void counter_loss (
            PhyloCounters * counters,
            std::vector< uint > nfun,
            bool duplication = true )  [inline]
```

Total count of losses for an specific function.

Definition at line 444 of file phylo.hpp.

7.15.3.8 counter_maxfuns()

```
void counter_maxfuns (
            PhyloCounters * counters,
            uint lb,
            uint ub,
            bool duplication = true )  [inline]
```

Cap the number of functions per gene.

Definition at line 371 of file phylo.hpp.

7.15.3.9 counter_neofun()

```
void counter_neofun (
            PhyloCounters * counters,
            uint nfunA,
            uint nfunB,
            bool duplication = true )  [inline]
```

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 750 of file phylo.hpp.

7.15.3.10 counter_neofun_a2b()

```
void counter_neofun_a2b (
            PhyloCounters * counters,
            uint nfunA,
            uint nfunB,
            bool duplication = true )  [inline]
```

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 823 of file phylo.hpp.

7.15.3.11 counter_overall_changes()

```
void counter_overall_changes (
            PhyloCounters * counters,
            bool duplication = true )  [inline]
```

Total number of changes. Use this statistic to account for "preservation".

Definition at line 486 of file phylo.hpp.

7.15.3.12 counter_overall_gains()

```
void counter_overall_gains (
            PhyloCounters * counters,
            bool duplication = true )  [inline]
```

Overall functional gains.

Total number of gains (irrespective of the function).

Definition at line 99 of file phylo.hpp.

7.15.3.13 counter_overall_loss()

```
void counter_overall_loss (
            PhyloCounters * counters,
            bool duplication = true )  [inline]
```

Overall functional loss.

Definition at line 330 of file phylo.hpp.

7.15.3.14 counter_subfun()

```
void counter_subfun (
            PhyloCounters * counters,
            uint nfunA,
            uint nfunB,
            bool duplication = true )  [inline]
```

Total count of Sub-functionalization events.

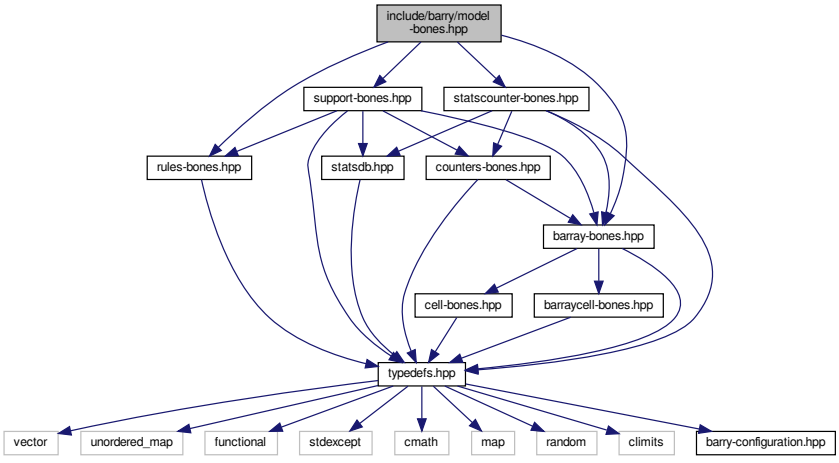
It requires to specify data = {funA, funB}

Definition at line 545 of file phylo.hpp.

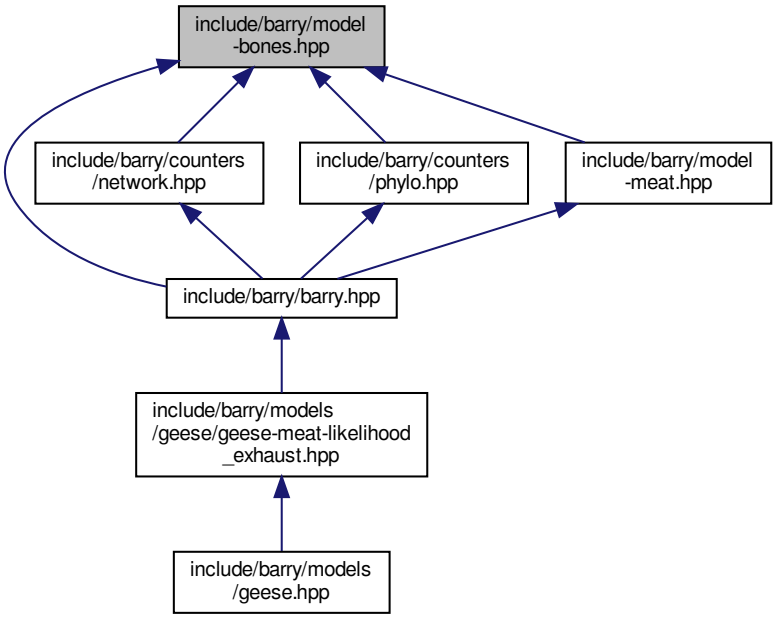
7.16 include/barry/model-bones.hpp File Reference

```
#include "barray-bones.hpp"
#include "support-bones.hpp"
#include "statscounter-bones.hpp"
#include "rules-bones.hpp"
```

Include dependency graph for model-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Model< Array_Type, Data_Counter_Type, Data_Rule_Type >

  General framework for discrete exponential models. This class allows generating discrete exponential models in the form of a linear exponential model:

Functions

- double update_normalizing_constant (const std::vector< double > &params, const Counts_type &support)
- double likelihood_ (const std::vector< double > &target_stats, const std::vector< double > &params, const double normalizing_constant, bool log_=false)
- template<typename Array_Type >
  std::vector< double > keygen_default (const Array_Type &Array_)

  Array Hasher class (used for computing support)

7.16.1 Function Documentation

7.16.1.1 keygen_default()

```
template<typename Array_Type >
std::vector< double > keygen_default (
            const Array_Type & Array_ )  [inline]
```

Array Hasher class (used for computing support)

Definition at line 63 of file model-bones.hpp.

7.16.1.2 likelihood_()

```
double likelihood_ (
            const std::vector< double > & target_stats,
            const std::vector< double > & params,
            const double normalizing_constant,
            bool log_ = false )  [inline]
```

Definition at line 35 of file model-bones.hpp.

7.16.1.3 update_normalizing_constant()

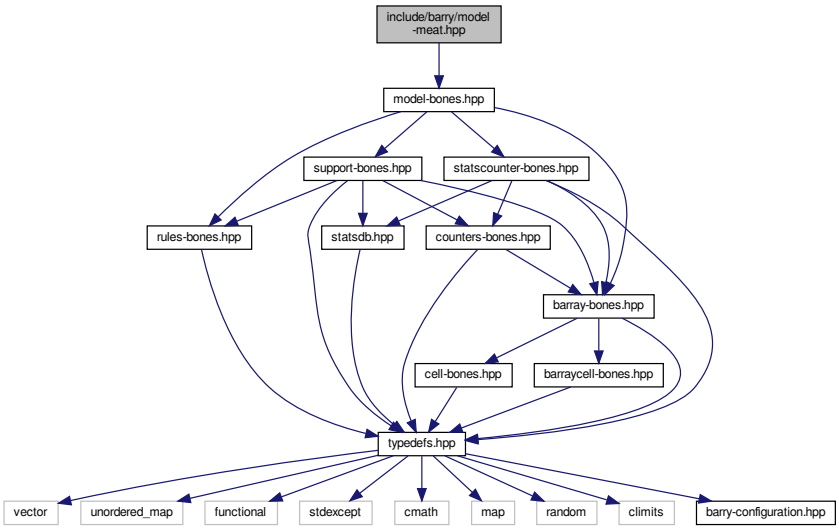
```
double update_normalizing_constant (
            const std::vector< double > & params,
            const Counts_type & support )  [inline]
```

Definition at line 11 of file model-bones.hpp.

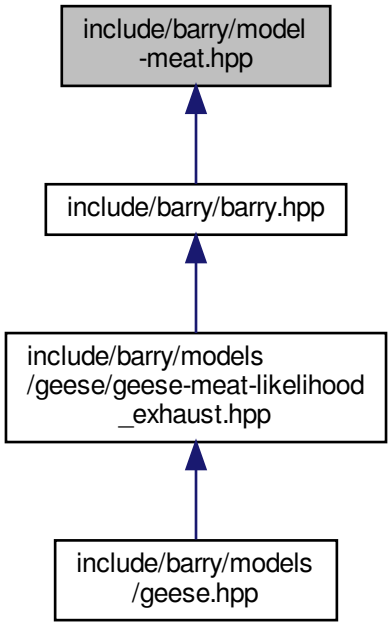
7.17 include/barry/model-meat.hpp File Reference

```
#include "model-bones.hpp"
```

Include dependency graph for model-meat.hpp:



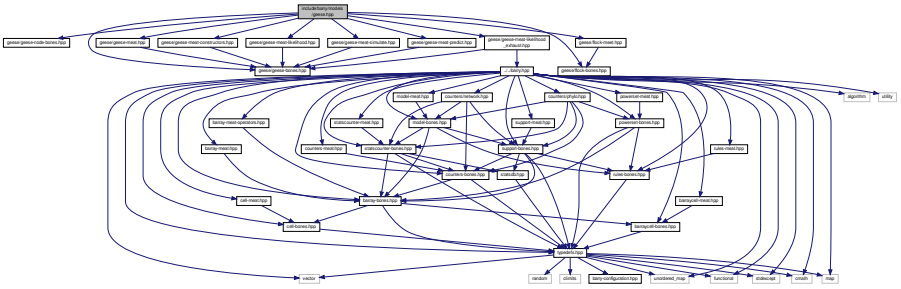
This graph shows which files directly or indirectly include this file:



7.18 include/barry/models/geese.hpp File Reference

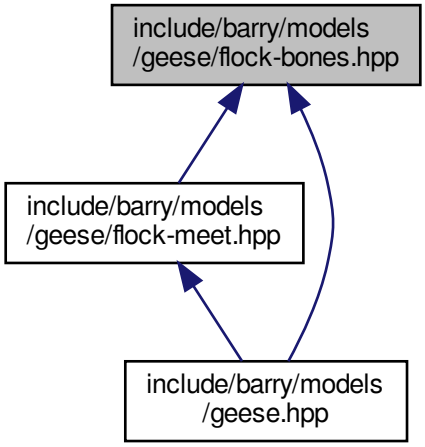
```
#include "geese/geese-node-bones.hpp"
#include "geese/geese-bones.hpp"
#include "geese/geese-meat.hpp"
#include "geese/geese-meat-constructors.hpp"
#include "geese/geese-meat-likelihood.hpp"
#include "geese/geese-meat-likelihood_exhaust.hpp"
#include "geese/geese-meat-simulate.hpp"
#include "geese/geese-meat-predict.hpp"
#include "geese/flock-bones.hpp"
#include "geese/flock-meet.hpp"
```

Include dependency graph for geese.hpp:



7.19 include/barry/models/geese/flock-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

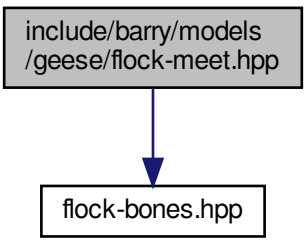
- class Flock

  A Flock is a group of Geese.

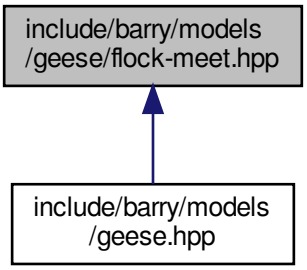
7.20 include/barry/models/geese/flock-meet.hpp File Reference

```
#include "flock-bones.hpp"
```

Include dependency graph for flock-meet.hpp:

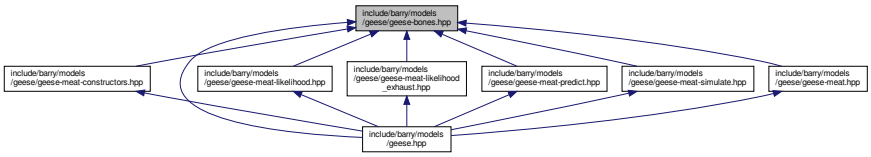


This graph shows which files directly or indirectly include this file:



7.21 include/barry/models/geese/geese-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class Geese

  Annotated Phylo Model.

Macros

- #define INITIALIZED()

Functions

- template<typename Ta , typename Tb >
  std::vector< Ta > vector_caster (const std::vector< Tb > &x)
- RULE_FUNCTION (rule_empty_free)
- std::vector< double > keygen_full (const phylocounters::PhyloArray &array)
- bool vec_diff (const std::vector< unsigned int > &s, const std::vector< unsigned int > &a)

7.21.1 Macro Definition Documentation

7.21.1.1 INITIALIZED

```
#define INITIALIZED( )
```

Value:

```
    if (!this->initialized) \
    throw std::logic_error("The model has not been initialized yet.");
```

Definition at line 18 of file geese-bones.hpp.

7.21.2 Function Documentation

7.21.2.1 keygen_full()

```
std::vector< double > keygen_full (
            const phylocounters::PhyloArray & array )  [inline]
```

Definition at line 31 of file geese-bones.hpp.

7.21.2.2 RULE_FUNCTION()

```
RULE_FUNCTION (
            rule_empty_free  )
```

Definition at line 22 of file geese-bones.hpp.

7.21.2.3 vec_diff()

```
bool vec_diff (
            const std::vector< unsigned int > & s,
            const std::vector< unsigned int > & a )  [inline]
```

Definition at line 51 of file geese-bones.hpp.

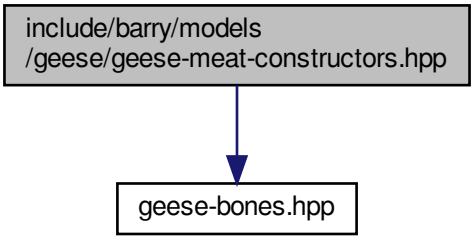
7.21.2.4 vector_caster()

```
template<typename Ta , typename Tb >
std::vector< Ta > vector_caster (
             const std::vector< Tb > & x )  [inline]
```

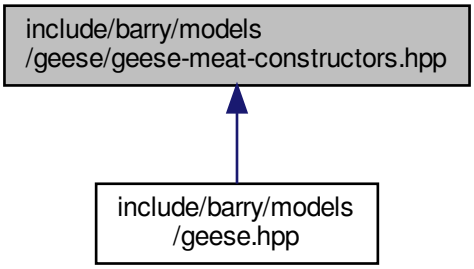
Definition at line 10 of file geese-bones.hpp.

7.22 include/barry/models/geese/geese-meat-constructors.hpp File Reference

`#include "geese-bones.hpp"`
Include dependency graph for geese-meat-constructors.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define GEESE_MEAT_CONSTRUCTORS_HPP 1

7.22.1 Macro Definition Documentation

7.22.1.1 GEESE_MEAT_CONSTRUCTORS_HPP

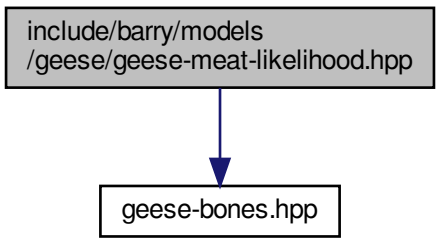
```
#define GEESE_MEAT_CONSTRUCTORS_HPP 1
```

Definition at line 4 of file geese-meat-constructors.hpp.

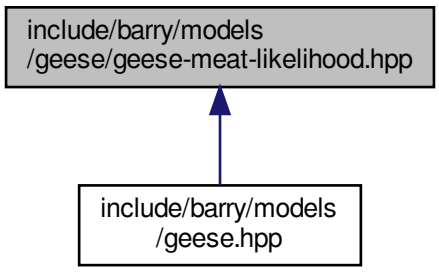
7.23 include/barry/models/geese/geese-meat-likelihood.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat-likelihood.hpp:



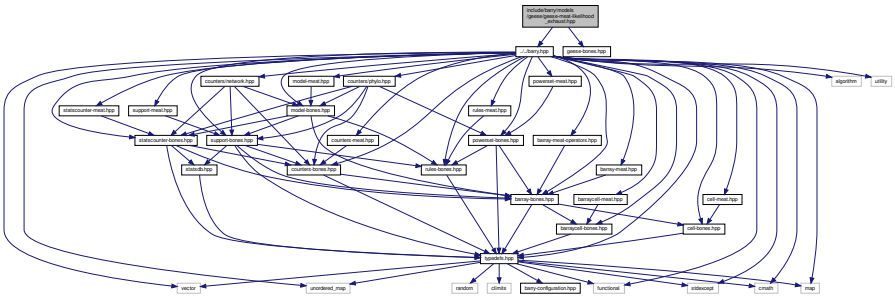
This graph shows which files directly or indirectly include this file:



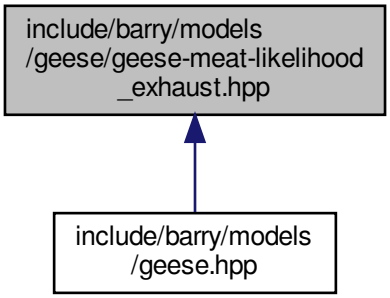
7.24 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference

```
#include "../../barry.hpp"
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat-likelihood_exhaust.hpp:



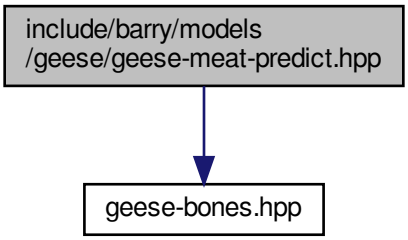
This graph shows which files directly or indirectly include this file:



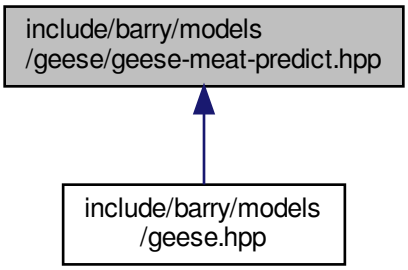
7.25 include/barry/models/geese/geese-meat-predict.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat-predict.hpp:



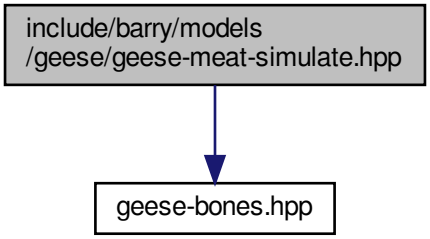
This graph shows which files directly or indirectly include this file:



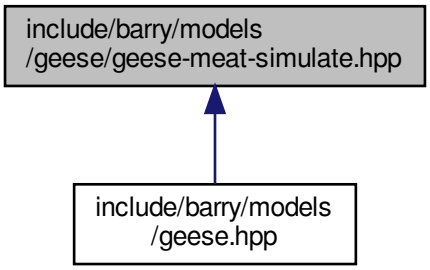
7.26 include/barry/models/geese/geese-meat-simulate.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat-simulate.hpp:



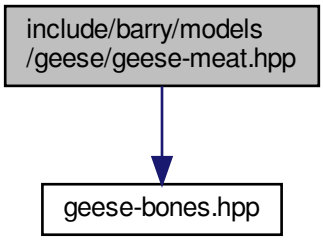
This graph shows which files directly or indirectly include this file:



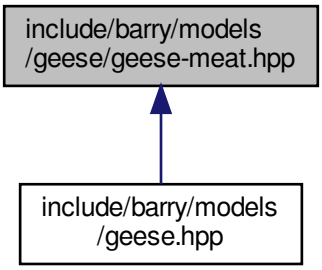
7.27 include/barry/models/geese/geese-meat.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat.hpp:

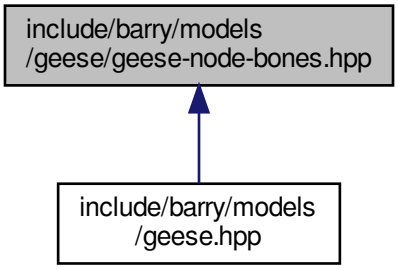


This graph shows which files directly or indirectly include this file:



7.28 include/barry/models/geese/geese-node-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class Node

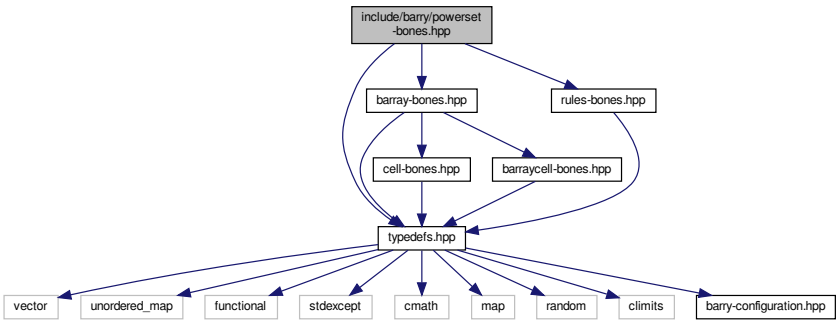
  A single node for the model.

7.29 include/barry/powerset-bones.hpp File Reference

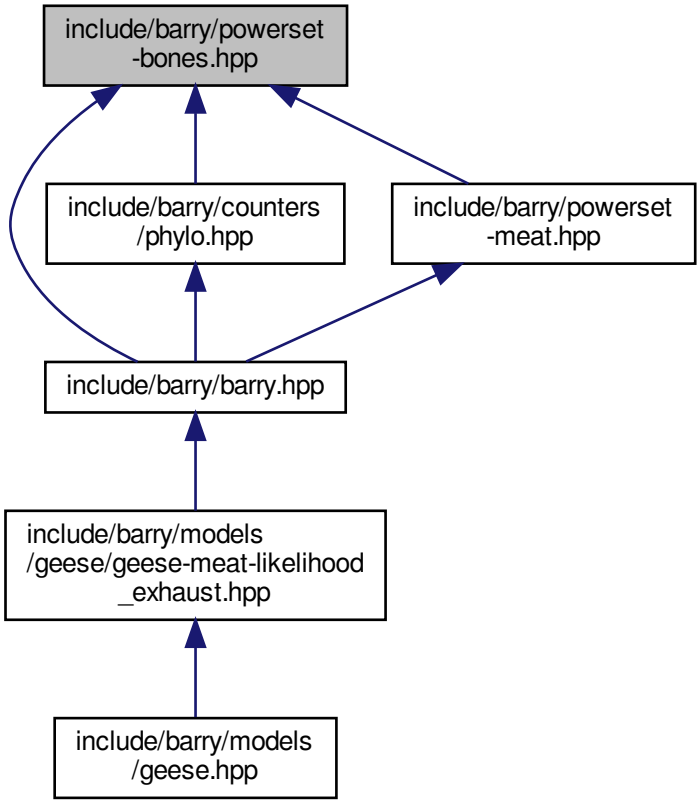
```
#include "typedefs.hpp"
#include "barray-bones.hpp"
```

```
#include "rules-bones.hpp"
```

Include dependency graph for powerset-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

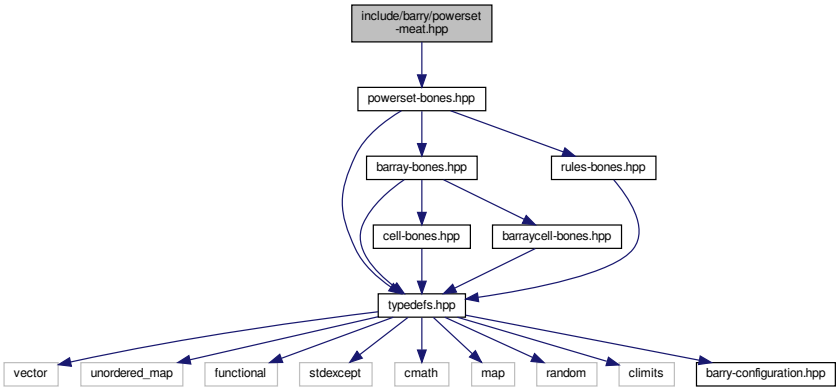
- class PowerSet< Array_Type, Data_Rule_Type >

  Powerset of a binary array.

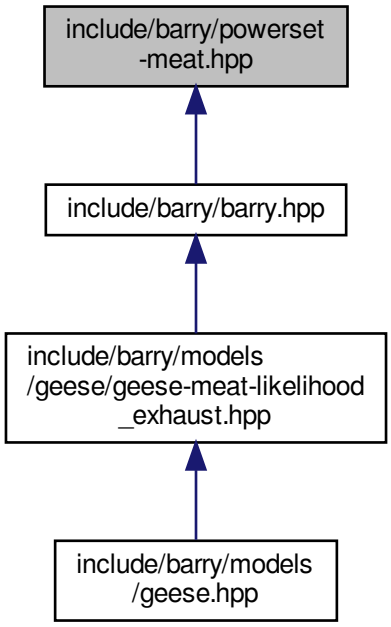
7.30 include/barry/powerset-meat.hpp File Reference

```
#include "powerset-bones.hpp"
```

Include dependency graph for powerset-meat.hpp:



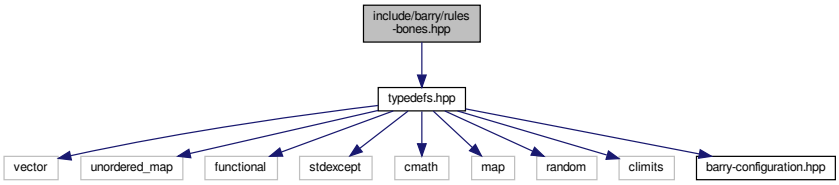
This graph shows which files directly or indirectly include this file:



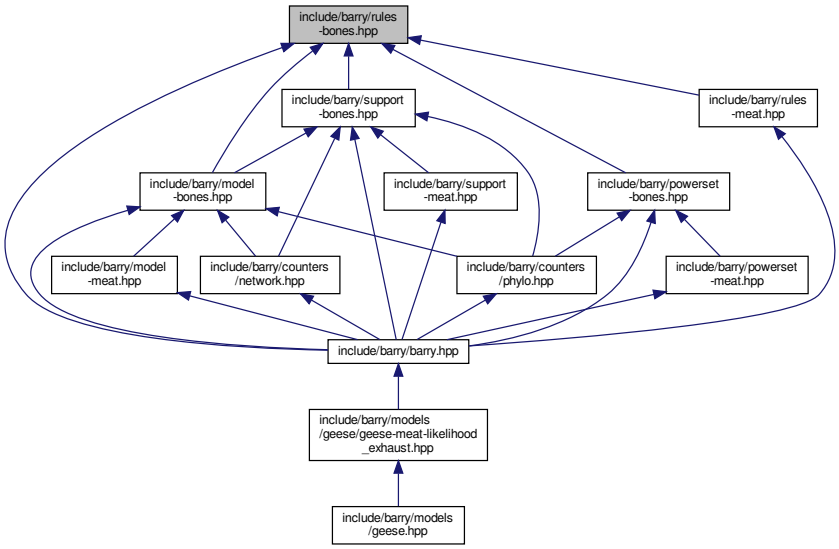
7.31 include/barry/rules-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for rules-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Rule< Array_Type, Data_Type >

  Rule for determining if a cell should be included in a sequence.
- class Rules< Array_Type, Data_Type >

  Vector of objects of class Rule.

Functions

- template<typename Array_Type , typename Data_Type >
  bool rule_fun_default (const Array_Type *array, uint i, uint j, Data_Type *dat)

7.31.1 Function Documentation

7.31.1.1 rule_fun_default()

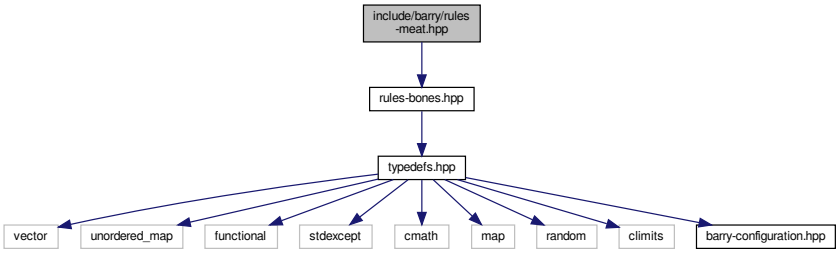
```
template<typename Array_Type , typename Data_Type >
bool rule_fun_default (
            const Array_Type * array,
            uint i,
            uint j,
            Data_Type * dat )
```

Definition at line 10 of file rules-bones.hpp.

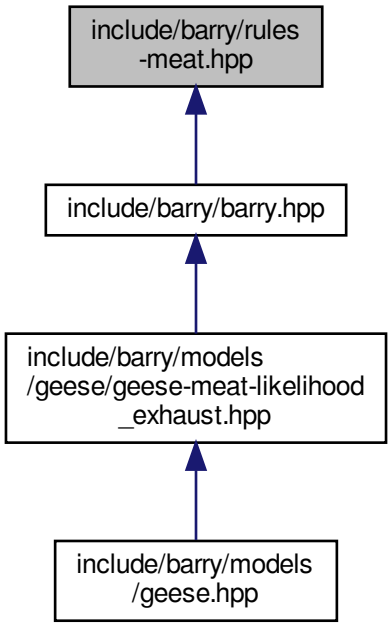
7.32 include/barry/rules-meat.hpp File Reference

```
#include "rules-bones.hpp"
```

Include dependency graph for rules-meat.hpp:



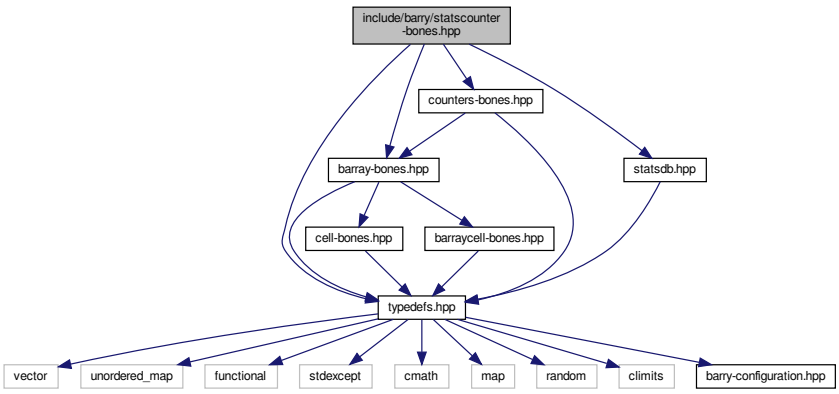
This graph shows which files directly or indirectly include this file:



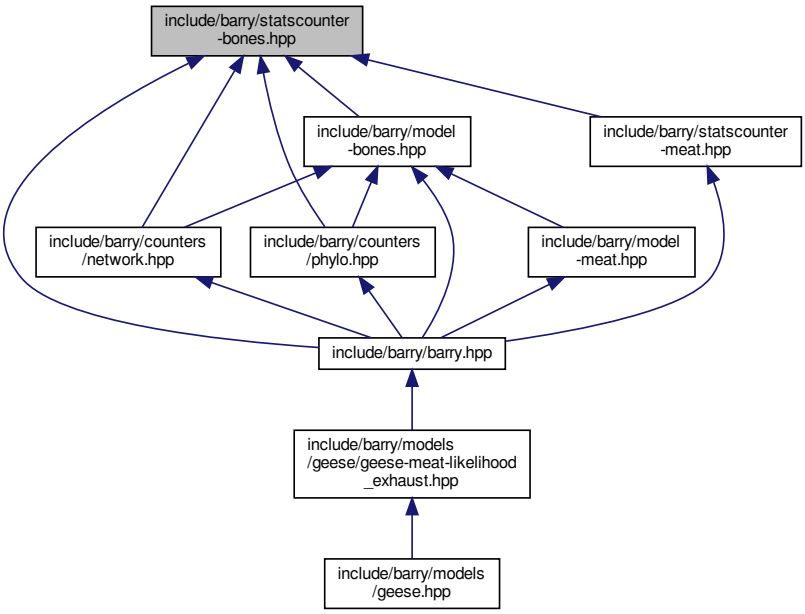
7.33 include/barry/statscounter-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "barray-bones.hpp"
#include "statsdb.hpp"
#include "counters-bones.hpp"
```

Include dependency graph for statscounter-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

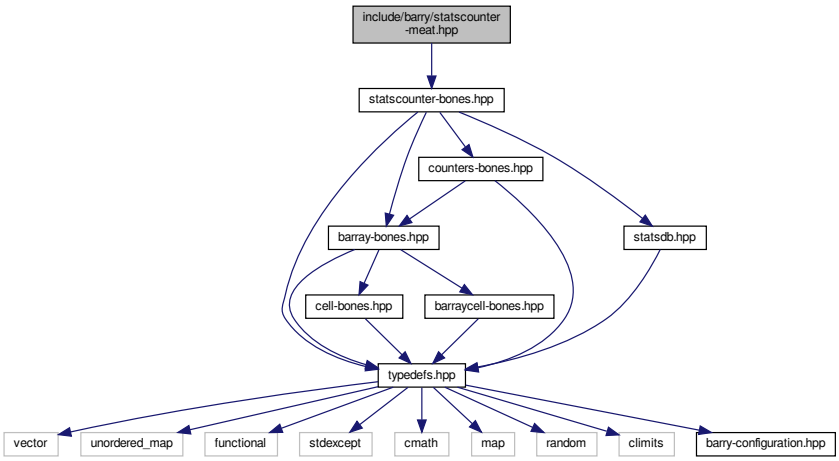
- class StatsCounter< Array_Type, Data_Type >

  Count stats for a single Array.

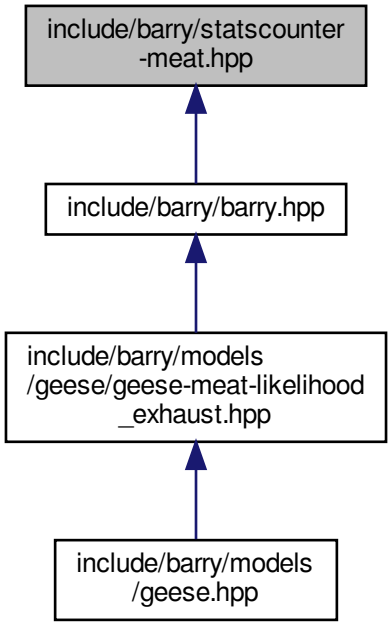
7.34 include/barry/statscounter-meat.hpp File Reference

```
#include "statscounter-bones.hpp"
```

Include dependency graph for statscounter-meat.hpp:



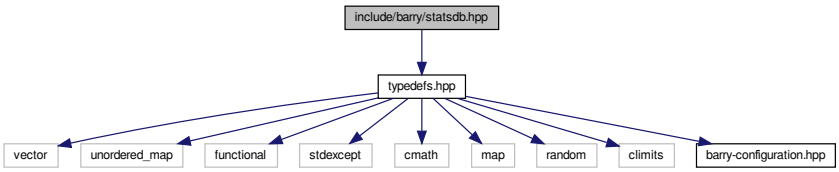
This graph shows which files directly or indirectly include this file:



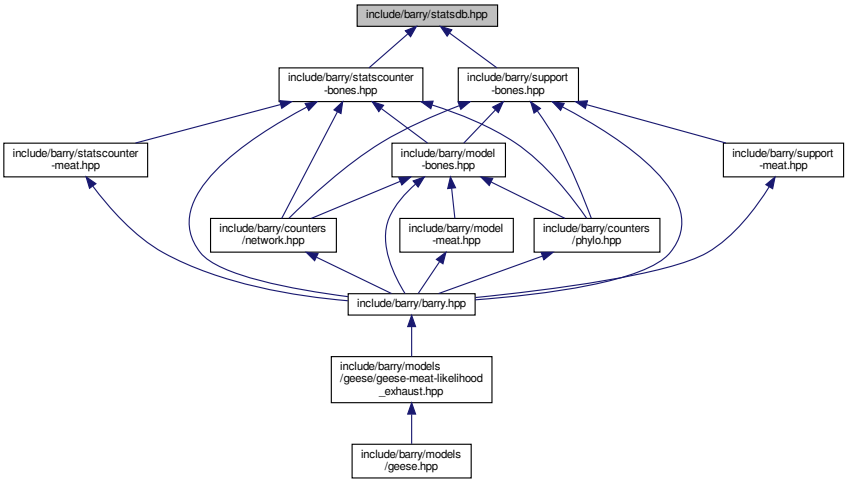
7.35 include/barry/statsdb.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for statsdb.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class FreqTable< T >

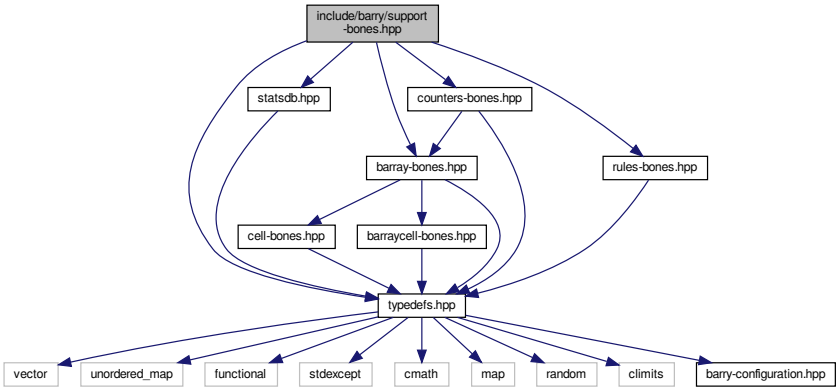
  Database of statistics.

7.36 include/barry/support-bones.hpp File Reference

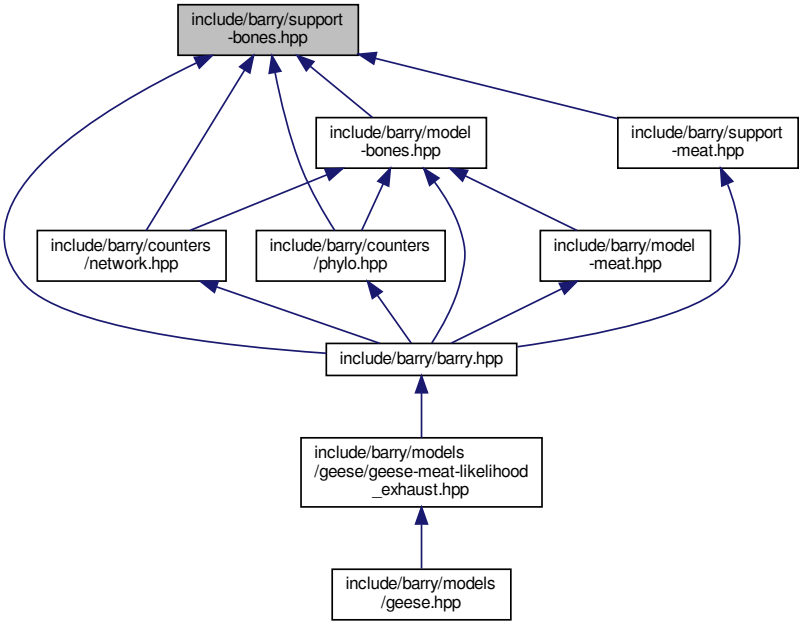
```
#include "typedefs.hpp"
#include "barray-bones.hpp"
#include "statsdb.hpp"
#include "counters-bones.hpp"
```

```
#include "rules-bones.hpp"
```

Include dependency graph for support-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

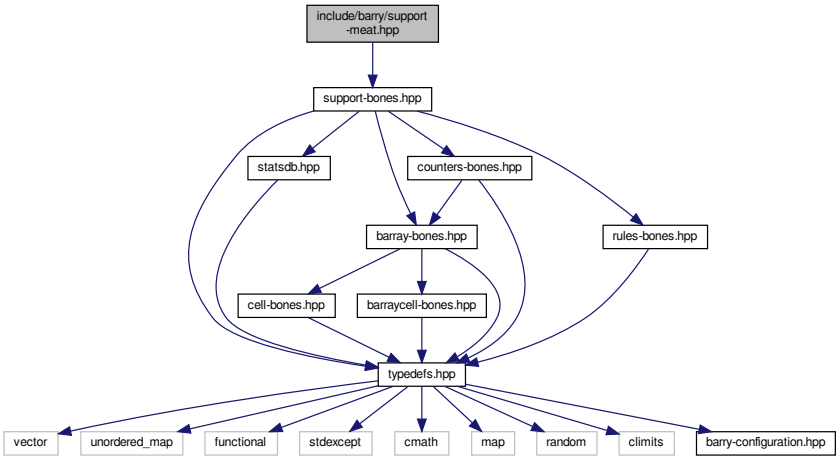
- class Support< Array_Type, Data_Counter_Type, Data_Rule_Type >

  Compute the support of sufficient statistics.

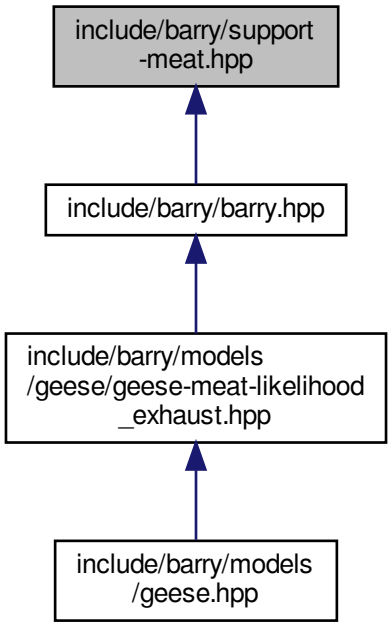
7.37 include/barry/support-meat.hpp File Reference

```
#include "support-bones.hpp"
```

Include dependency graph for support-meat.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define BARRY_SUPPORT_MEAT_HPP 1

7.37.1 Macro Definition Documentation

7.37.1.1 BARRY_SUPPORT_MEAT_HPP

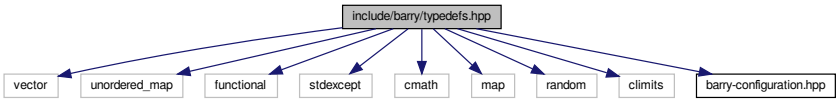
```
#define BARRY_SUPPORT_MEAT_HPP 1
```

Definition at line 4 of file support-meat.hpp.

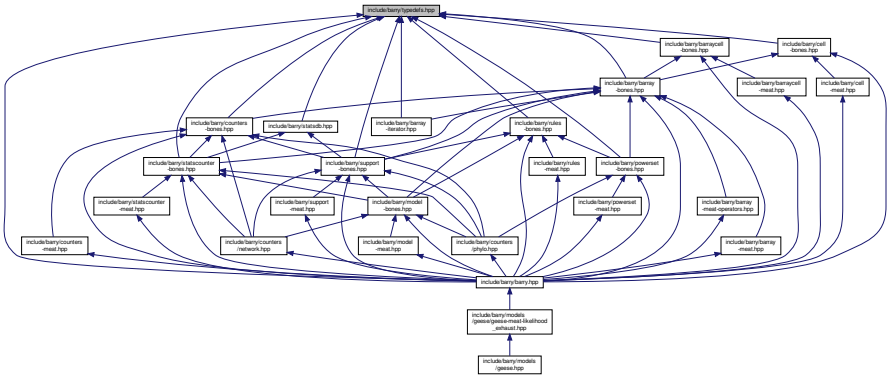
7.38 include/barry/typedefs.hpp File Reference

```
#include <vector>
#include <unordered_map>
#include <functional>
#include <stdexcept>
#include <cmath>
#include <map>
#include <random>
#include <climits>
#include "barry-configuration.hpp"
```

Include dependency graph for typedefs.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Entries< Cell_Type >

  A wrapper class to store `source`, `target`, `val` *from a* `BArray` *object.*
- struct vecHasher< T >

Namespaces

- CHECK

  Integer constants used to specify which cell should be check.
- EXISTS

  Integer constants used to specify which cell should be check to exist or not.

Macros

- #define ROW(a) this->el_ij[a]
- #define COL(a) this->el_ji[a]
- #define A_ROW(a) Array.el_ij[a]
- #define A_COL(a) Array.el_ji[a]

Typedefs

- typedef unsigned int uint
- typedef std::vector< std::pair< std::vector< double >, uint > > Counts_type
- template<typename Cell_Type >
  using Row_type = Map< uint, Cell< Cell_Type > >
- template<typename Cell_Type >
  using Col_type = Map< uint, Cell< Cell_Type > * >
- template<typename Ta = double, typename Tb = uint>
  using MapVec_type = std::unordered_map< std::vector< Ta >, Tb, vecHasher< Ta > >
- template<typename Array_Type , typename Data_Type >
  using Counter_fun_type = std::function< double(const Array_Type &, uint, uint, Data_Type *)>

  Counter and rule functions.
- template<typename Array_Type , typename Data_Type >
  using Rule_fun_type = std::function< bool(const Array_Type &, uint, uint, Data_Type *)>

Functions

- template<typename T >
  T vec_inner_prod (const std::vector< T > &a, const std::vector< T > &b)
- template<typename T >
  bool vec_equal (const std::vector< T > &a, const std::vector< T > &b)

  Compares if -a- and -b- are equal.
- template<typename T >
  bool vec_equal_approx (const std::vector< T > &a, const std::vector< T > &b, double eps=1e-10)

Variables

- const int CHECK::BOTH = -1
- const int CHECK::NONE = 0
- const int CHECK::ONE = 1
- const int CHECK::TWO = 2
- const int EXISTS::BOTH = -1
- const int EXISTS::NONE = 0
- const int EXISTS::ONE = 1
- const int EXISTS::TWO = 1
- const int EXISTS::UKNOWN = -1
- const int EXISTS::AS_ZERO = 0
- const int EXISTS::AS_ONE = 1

7.38.1 Macro Definition Documentation

7.38.1.1 A_COL

```
#define A_COL(
              a ) Array.el_ji[a]
```

Definition at line 125 of file typedefs.hpp.

7.38.1.2 A_ROW

```
#define A_ROW(
              a ) Array.el_ij[a]
```

Definition at line 124 of file typedefs.hpp.

7.38.1.3 COL

```
#define COL(
              a ) this->el_ji[a]
```

Definition at line 25 of file typedefs.hpp.

7.38.1.4 ROW

```
#define ROW(
              a ) this->el_ij[a]
```

Definition at line 24 of file typedefs.hpp.

7.38.2 Typedef Documentation

7.38.2.1 Col_type

```
template<typename Cell_Type >
using Col_type = Map< uint, Cell<Cell_Type>* >
```

Definition at line 64 of file typedefs.hpp.

7.38.2.2 Counter_fun_type

```
template<typename Array_Type , typename Data_Type >
using Counter_fun_type = std::function<double(const Array_Type &, uint, uint, Data_Type *)>
```

Counter and rule functions.

Parameters

Array_Type	a BArray
unit,uint	Focal cell
Data_Type	Data associated with the function, for example, id of the attribute in the Array.

Returns

`Counter_fun_type` a double (the change statistic)

`Rule_fun_type` a bool. True if the cell is blocked.

Definition at line 138 of file typedefs.hpp.

7.38.2.3 Counts_type

```
typedef std::vector< std::pair< std::vector<double>, uint > > Counts_type
```

Definition at line 57 of file typedefs.hpp.

7.38.2.4 MapVec_type

```
template<typename Ta = double, typename Tb = uint>
using MapVec_type = std::unordered_map< std::vector< Ta >, Tb, vecHasher<Ta> >
```

Definition at line 118 of file typedefs.hpp.

7.38.2.5 Row_type

```
template<typename Cell_Type >
using Row_type = Map< uint, Cell<Cell_Type> >
```

Definition at line 61 of file typedefs.hpp.

7.38.2.6 Rule_fun_type

```
template<typename Array_Type , typename Data_Type >
using Rule_fun_type = std::function<bool(const Array_Type &, uint, uint, Data_Type *)>
```

Definition at line 141 of file typedefs.hpp.

7.38.2.7 uint

```
typedef unsigned int uint
```

Definition at line 21 of file typedefs.hpp.

7.38.3 Function Documentation

7.38.3.1 vec_equal()

```
template<typename T >
bool vec_equal (
            const std::vector< T > & a,
            const std::vector< T > & b )  [inline]
```

Compares if -a- and -b- are equal.

Parameters

a,b	Two vectors of the same length

Returns

`true` if all elements are equal.

Definition at line 152 of file typedefs.hpp.

7.38.3.2 vec_equal_approx()

```
template<typename T >
bool vec_equal_approx (
             const std::vector< T > & a,
             const std::vector< T > & b,
             double eps = 1e-10 )  [inline]
```

Definition at line 170 of file typedefs.hpp.

7.38.3.3 vec_inner_prod()

```
template<typename T >
T vec_inner_prod (
             const std::vector< T > & a,
             const std::vector< T > & b )  [inline]
```

Definition at line 190 of file typedefs.hpp.

7.39 README.md File Reference

Index