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
```

Efficient memory usage

One of the key features of `barry` is that it will handle memory efficiently.

Code of Conduct

Please note that the `barry` project is released with a `Contributor Code of Conduct`. By contributing to this project, you agree to abide by its terms.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

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

Module Documentation

5.1 Counting

Classes

- class NetworkData

  Data class for Networks.
- class NodeData

  Data definition for the `PhyloArray` *class.*
- class Counter< Array_Type, Data_Type >

  A counter function based on change statistics.

5.1.1 Detailed Description

`barry` includes a flexible way to generate counters based on change statistics. Since most of the time we are counting many motifs in a graph, change statistics make a reasonable (and efficient) way to make such counts.

In particular, let the motif be defined as $s(y)$, with y as the binary array. The change statistic when adding cell y_{ij}, i.e. when the cell moves from being emty to have a one, is defined as

$$\delta(y_{ij}) = s^{+}_{ij}(y) - s^{-}_{ij}(y),$$

where $s^{+}_{ij}(y)$ and $s^{-}_{ij}(y)$ represent the motif statistic with and without the ij-cell. For example, in the case of networks, the change statistic for the number of edges is always 1.

To count statistics in an array, the [Counter] class will empty the array, initialize the counters, and then start counting while adding at each step a single cell, until matching the original array.

5.2 Statistical Models

Statistical models available in `barry`.

Classes

- class Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >

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

  A Flock is a group of Geese.
- class Geese

  Annotated Phylo Model.

5.2.1 Detailed Description

Statistical models available in `barry`.

5.3 Network counters

Counters for network models.

Functions

- 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.

5.3.1 Detailed Description

Counters for network models.

Parameters

counters	A pointer to a `NetCounters` object (`Counters<Network, NetCounterData>`).

5.3.2 Function Documentation

5.3.2.1 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 419 of file network.hpp.

5.3.2.2 counter_ctriads()

```
void counter_ctriads (
            NetCounters * counters )  [inline]
```

Definition at line 322 of file network.hpp.

5.3.2.3 counter_degree()

```
void counter_degree (
            NetCounters * counters,
            std::vector< uint > d )  [inline]
```

Counts number of vertices with a given out-degree.

Definition at line 690 of file network.hpp.

5.3.2.4 counter_density()

```
void counter_density (
            NetCounters * counters )  [inline]
```

Definition at line 361 of file network.hpp.

5.3.2.5 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 461 of file network.hpp.

5.3.2.6 counter_edges()

```
void counter_edges (
            NetCounters * counters )  [inline]
```

Number of edges.

Definition at line 128 of file network.hpp.

5.3.2.7 counter_idegree()

```
void counter_idegree (
            NetCounters * counters,
            std::vector< uint > d )  [inline]
```

Counts number of vertices with a given in-degree.

Definition at line 604 of file network.hpp.

5.3.2.8 counter_idegree15()

```
void counter_idegree15 (
            NetCounters * counters )  [inline]
```

Definition at line 377 of file network.hpp.

5.3.2.9 counter_isolates()

```
void counter_isolates (
            NetCounters * counters )  [inline]
```

Number of isolated vertices.

Definition at line 142 of file network.hpp.

5.3.2.10 counter_istar2()

```
void counter_istar2 (
            NetCounters * counters )  [inline]
```

Definition at line 210 of file network.hpp.

5.3.2.11 counter_mutual()

```
void counter_mutual (
            NetCounters * counters )  [inline]
```

Number of mutual ties.

Definition at line 172 of file network.hpp.

5.3.2.12 counter_nodecov()

```
void counter_nodecov (
            NetCounters * counters,
            uint attr_id )  [inline]
```

Definition at line 558 of file network.hpp.

5.3.2.13 counter_nodeicov()

```
void counter_nodeicov (
            NetCounters * counters,
            uint attr_id )  [inline]
```

Definition at line 520 of file network.hpp.

5.3.2.14 counter_nodematch()

```
void counter_nodematch (
            NetCounters * counters,
            uint attr_id )  [inline]
```

Definition at line 578 of file network.hpp.

5.3.2.15 counter_nodeocov()

```
void counter_nodeocov (
            NetCounters * counters,
            uint attr_id )  [inline]
```

Definition at line 539 of file network.hpp.

5.3.2.16 counter_odegree()

```
void counter_odegree (
            NetCounters * counters,
            std::vector< uint > d )  [inline]
```

Counts number of vertices with a given out-degree.

Definition at line 646 of file network.hpp.

5.3.2.17 counter_odegree15()

```
void counter_odegree15 (
            NetCounters * counters )  [inline]
```

Definition at line 397 of file network.hpp.

5.3.2.18 counter_ostar2()

```
void counter_ostar2 (
            NetCounters * counters )  [inline]
```

Definition at line 228 of file network.hpp.

5.3.2.19 counter_ttriads()

```
void counter_ttriads (
            NetCounters * counters )  [inline]
```

Definition at line 247 of file network.hpp.

5.3.2.20 NETWORK_COUNTER()

```
NETWORK_COUNTER (
             init_single_attr  )
```

Definition at line 503 of file network.hpp.

5.4 Phylo counters

Counters for phylogenetic modeling.

Functions

- 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)

  k genes gain 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_prop_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.

5.4.1 Detailed Description

Counters for phylogenetic modeling.

Parameters

counters	A pointer to a `PhyloCounters` object (`Counters<PhyloArray, PhyloCounterData>`).

5.4.2 Function Documentation

5.4.2.1 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:

$$x_{pa}(1-x_{pb})\sum_{i<j}\left[x_{ia}^{p}(1-x_{ib}^{p})x_{ja}^{p}x_{jb}^{p}+x_{ja}^{p}(1-x_{jb}^{p})x_{ia}^{p}x_{ib}^{p}\right]$$

This algorithm implements the change statistic.

Definition at line 1203 of file phylo.hpp.

5.4.2.2 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 833 of file phylo.hpp.

5.4.2.3 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 152 of file phylo.hpp.

5.4.2.4 counter_gains_k_offspring()

```
void counter_gains_k_offspring (
            PhyloCounters * counters,
            std::vector< uint > nfun,
            uint k = 1u,
            bool duplication = true )  [inline]
```

k genes gain function nfun

Definition at line 194 of file phylo.hpp.

5.4.2.5 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 268 of file phylo.hpp.

5.4.2.6 counter_longest()

```
void counter_longest (
            PhyloCounters * counters )  [inline]
```

Longest branch mutates (either by gain or by loss)

Definition at line 892 of file phylo.hpp.

5.4.2.7 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 633 of file phylo.hpp.

5.4.2.8 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 549 of file phylo.hpp.

5.4.2.9 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 997 of file phylo.hpp.

5.4.2.10 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 1082 of file phylo.hpp.

5.4.2.11 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 680 of file phylo.hpp.

5.4.2.12 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 112 of file phylo.hpp.

5.4.2.13 counter_overall_loss()

```
void counter_overall_loss (
            PhyloCounters * counters,
            bool duplication = true )  [inline]
```

Overall functional loss.

Definition at line 503 of file phylo.hpp.

5.4.2.14 counter_prop_genes_changing()

```
void counter_prop_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 386 of file phylo.hpp.

5.4.2.15 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 747 of file phylo.hpp.

5.5 Phylo rules

Rules for phylogenetic modeling.

Classes

- class PhyloRuleDynData

Functions

- void rule_dyn_limit_changes (PhyloSupport *support, uint pos, uint lb, uint ub, bool duplication=true)
  Overall functional gains.

5.5.1 Detailed Description

Rules for phylogenetic modeling.

Parameters

rules	A pointer to a `PhyloRules` object (`Rules<PhyloArray, PhyloRuleData>`).

5.5.2 Function Documentation

5.5.2.1 rule_dyn_limit_changes()

```
void rule_dyn_limit_changes (
            PhyloSupport * support,
            uint pos,
```

```
uint lb,
uint ub,
bool duplication = true )  [inline]
```

Overall functional gains.

Parameters

support	Support of a model.
pos	Position of the focal statistic.
lb	Lower bound
ub	Upper bound

Returns

(void) adds a rule limiting the support of the model.

Definition at line 1336 of file phylo.hpp.

Chapter 6

Namespace Documentation

6.1 barry Namespace Reference

barry: Your go-to motif accountant

Namespaces

- counters

  Tree class and TreeIterator class.

6.1.1 Detailed Description

barry: Your go-to motif accountant

6.2 barry::counters Namespace Reference

Tree class and TreeIterator class.

Namespaces

- network
- phylo

6.2.1 Detailed Description

Tree class and TreeIterator class.

6.3 barry::counters::network Namespace Reference

6.4 barry::counters::phylo Namespace Reference

6.5 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

6.5.1 Detailed Description

Integer constants used to specify which cell should be check.

6.5.2 Variable Documentation

6.5.2.1 BOTH

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

Definition at line 20 of file typedefs.hpp.

6.5.2.2 NONE

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

Definition at line 21 of file typedefs.hpp.

6.5.2.3 ONE

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

Definition at line 22 of file typedefs.hpp.

6.5.2.4 TWO

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

Definition at line 23 of file typedefs.hpp.

6.6 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

6.6.1 Detailed Description

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

6.6.2 Variable Documentation

6.6.2.1 AS_ONE

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

Definition at line 38 of file typedefs.hpp.

6.6.2.2 AS_ZERO

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

Definition at line 37 of file typedefs.hpp.

6.6.2.3 BOTH

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

Definition at line 31 of file typedefs.hpp.

6.6.2.4 NONE

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

Definition at line 32 of file typedefs.hpp.

6.6.2.5 ONE

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

Definition at line 33 of file typedefs.hpp.

6.6.2.6 TWO

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

Definition at line 34 of file typedefs.hpp.

6.6.2.7 UKNOWN

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

Definition at line 36 of file typedefs.hpp.

Chapter 7

Class Documentation

7.1 BArray< Cell_Type, Data_Type > Class Template Reference

Baseline class for binary arrays.

`#include <barray-bones.hpp>`

Public Member Functions

- bool operator== (const BArray< Cell_Type, Data_Type > &Array_)
- ~BArray ()
- void out_of_range (uint i, uint j) const
- Cell_Type get_cell (uint i, uint j, 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 char *fmt=nullptr,...) 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.

- void set_data (Data_Type *data_, bool delete_data_=false)
  Set the data object.
- Data_Type * D ()
- const Data_Type * D () const
- void flush_data ()

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 noexcept
- uint ncol () const noexcept
- uint nnozero () const noexcept
- Cell< Cell_Type > default_val () 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

- bool visited = false

Friends

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

7.1.1 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.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 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 60 of file barray-bones.hpp.

7.1.2.2 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 63 of file barray-bones.hpp.

7.1.2.3 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.

7.1.2.4 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)

7.1.2.5 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.

7.1.2.6 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.

7.1.2.7 ~BArray()

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

7.1.3 Member Function Documentation

7.1.3.1 clear()

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

7.1.3.2 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
```

7.1.3.3 D() [1/2]

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

7.1.3.4 D() [2/2]

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

7.1.3.5 default_val()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
Cell<Cell_Type> BArray< Cell_Type, Data_Type >::default_val ( ) const
```

7.1.3.6 flush_data()

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

7.1.3.7 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
```

7.1.3.8 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
```

7.1.3.9 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
```

7.1.3.10 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>

7.1.3.11 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
```

7.1.3.12 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
```

7.1.3.13 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 )
```

7.1.3.14 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 )
```

7.1.3.15 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 )
```

7.1.3.16 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
```

7.1.3.17 ncol()

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

7.1.3.18 nnozero()

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

7.1.3.19 nrow()

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

7.1.3.20 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 )
```

7.1.3.21 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
```

7.1.3.22 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 )
```

7.1.3.23 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 )
```

7.1.3.24 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 )
```

7.1.3.25 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 )
```

7.1.3.26 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 )
```

7.1.3.27 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 )
```

7.1.3.28 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 )
```

7.1.3.29 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 )
```

7.1.3.30 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.

7.1.3.31 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.

7.1.3.32 operator==()

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

7.1.3.33 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
```

7.1.3.34 print()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArray< Cell_Type, Data_Type >::print (
            const char * fmt = nullptr,
             ...  )  const
```

7.1.3.35 reserve()

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

7.1.3.36 resize()

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

7.1.3.37 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 )
```

7.1.3.38 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
```

7.1.3.39 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 )
```

Set the data object.

Parameters

*data_*	
*delete_↩ data_*	

7.1.3.40 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 )
```

7.1.3.41 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 )
```

7.1.3.42 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 )
```

7.1.3.43 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 )
```

7.1.3.44 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 )
```

7.1.3.45 transpose()

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

7.1.3.46 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 )
```

7.1.3.47 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 )
```

7.1.4 Friends And Related Function Documentation

7.1.4.1 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.

7.1.4.2 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.

7.1.5 Member Data Documentation

7.1.5.1 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 45 of file barray-bones.hpp.

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

- include/barry/barray-bones.hpp

7.2 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

7.2.1 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.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 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.

7.2.2.2 ~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.

7.2.3 Member Function Documentation

7.2.3.1 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.

7.2.3.2 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.

7.2.3.3 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.

7.2.3.4 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.

7.2.3.5 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.

7.2.3.6 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.

7.2.3.7 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

7.3 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

7.3.1 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.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 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.

7.3.2.2 ~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.

7.3.3 Member Function Documentation

7.3.3.1 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.

7.3.3.2 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.

7.3.3.3 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.

7.3.3.4 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.

7.3.3.5 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.

7.3.3.6 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.

7.3.3.7 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

7.4 BArrayDense< Cell_Type, Data_Type > Class Template Reference

Baseline class for binary arrays.

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

Public Member Functions

- bool operator== (const BArrayDense< Cell_Type, Data_Type > &Array_)
- ~BArrayDense ()
- void out_of_range (uint i, uint j) const
- Cell_Type get_cell (uint i, uint j, 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.*

- BArrayDense ()
  Zero-size array.
- BArrayDense (uint N_, uint M_)
  Empty array.
- BArrayDense (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.
- BArrayDense (uint N_, uint M_, const std::vector< uint > &source, const std::vector< uint > &target, bool add=true)
  Edgelist with no data (simpler)
- BArrayDense (const BArrayDense< Cell_Type, Data_Type > &Array_, bool copy_data=false)
  Copy constructor.
- BArrayDense< Cell_Type, Data_Type > & operator= (const BArrayDense< Cell_Type, Data_Type > &Array_)
  Assignment constructor.
- BArrayDense (BArrayDense< Cell_Type, Data_Type > &&x) noexcept
  Move operator.
- BArrayDense< Cell_Type, Data_Type > & operator= (BArrayDense< Cell_Type, Data_Type > &&x) noexcept
  Move assignment.

- void set_data (Data_Type *data_, bool delete_data_=false)
  Set the data object.
- Data_Type * D ()
- const Data_Type * D () const

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 noexcept
- uint ncol () const noexcept
- uint nnozero () const noexcept
- Cell< Cell_Type > default_val () 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.*

- BArrayDense< Cell_Type, Data_Type > & operator+= (const std::pair< uint, uint > &coords)
- BArrayDense< 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

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

Public Attributes

- bool visited = false

Friends

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

7.4.1 Detailed Description

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

Baseline class for binary arrays.

`BArrayDense` 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 barraydense-bones.hpp.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 BArrayDense() [1/6]

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

Zero-size array.

Definition at line 59 of file barraydense-bones.hpp.

7.4.2.2 BArrayDense() [2/6]

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

Empty array.

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

7.4.2.3 BArrayDense() [3/6]

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

Edgelist with data.

Definition at line 18 of file barraydense-meet.hpp.

7.4.2.4 BArrayDense() [4/6]

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

Edgelist with no data (simpler)

Definition at line 65 of file barraydense-meet.hpp.

7.4.2.5 BArrayDense() [5/6]

```
template<typename Cell_Type , typename Data_Type >
BArrayDense< Cell_Type, Data_Type >::BArrayDense (
            const BArrayDense< Cell_Type, Data_Type > & Array_,
            bool copy_data = false )  [inline]
```

Copy constructor.

Definition at line 120 of file barraydense-meet.hpp.

7.4.2.6 BArrayDense() [6/6]

```
template<typename Cell_Type , typename Data_Type >
BArrayDense< Cell_Type, Data_Type >::BArrayDense (
            BArrayDense< Cell_Type, Data_Type > && x )  [inline], [noexcept]
```

Move operator.

Definition at line 196 of file barraydense-meet.hpp.

7.4.2.7 ~BArrayDense()

```
template<typename Cell_Type , typename Data_Type >
BArrayDense< Cell_Type, Data_Type >::~BArrayDense  [inline]
```

Definition at line 273 of file barraydense-meet.hpp.

7.4.3 Member Function Documentation

7.4.3.1 clear()

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::clear (
            bool hard = true )  [inline]
```

Definition at line 1004 of file barraydense-meet.hpp.

7.4.3.2 col()

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

Definition at line 401 of file barraydense-meet.hpp.

7.4.3.3 D() [1/2]

```
template<typename Cell_Type , typename Data_Type >
Data_Type * BArrayDense< Cell_Type, Data_Type >::D  [inline]
```

Definition at line 297 of file barraydense-meet.hpp.

7.4.3.4 D() [2/2]

```
template<typename Cell_Type , typename Data_Type >
const Data_Type * BArrayDense< Cell_Type, Data_Type >::D  [inline]
```

Definition at line 302 of file barraydense-meet.hpp.

7.4.3.5 default_val()

```
template<typename Cell_Type , typename Data_Type >
Cell< Cell_Type > BArrayDense< Cell_Type, Data_Type >::default_val  [inline]
```

Definition at line 467 of file barraydense-meet.hpp.

7.4.3.6 get_cell()

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

Definition at line 317 of file barraydense-meet.hpp.

7.4.3.7 get_col_vec() [1/2]

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

Definition at line 377 of file barraydense-meet.hpp.

7.4.3.8 get_col_vec() [2/2]

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

Definition at line 362 of file barraydense-meet.hpp.

7.4.3.9 get_entries()

```
template<typename Cell_Type , typename Data_Type >
Entries< Cell_Type > BArrayDense< Cell_Type, Data_Type >::get_entries  [inline]
```

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>

Definition at line 412 of file barraydense-meet.hpp.

7.4.3.10 get_row_vec() [1/2]

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

Definition at line 346 of file barraydense-meet.hpp.

7.4.3.11 get_row_vec() [2/2]

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

Definition at line 331 of file barraydense-meet.hpp.

7.4.3.12 insert_cell() [1/3]

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::insert_cell (
            uint i,
            uint j,
            Cell< Cell_Type > && v,
            bool check_bounds,
            bool check_exists )  [inline]
```

Definition at line 601 of file barraydense-meet.hpp.

7.4.3.13 insert_cell() [2/3]

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::insert_cell (
            uint i,
            uint j,
            Cell_Type v,
            bool check_bounds,
            bool check_exists )  [inline]
```

Definition at line 649 of file barraydense-meet.hpp.

7.4.3.14 insert_cell() [3/3]

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::insert_cell (
            uint i,
            uint j,
            const Cell< Cell_Type > & v,
            bool check_bounds,
            bool check_exists )  [inline]
```

Definition at line 553 of file barraydense-meet.hpp.

7.4.3.15 is_empty()

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

Definition at line 432 of file barraydense-meet.hpp.

7.4.3.16 ncol()

```
template<typename Cell_Type , typename Data_Type >
uint BArrayDense< Cell_Type, Data_Type >::ncol  [inline], [noexcept]
```

Definition at line 457 of file barraydense-meet.hpp.

7.4.3.17 nnozero()

```
template<typename Cell_Type , typename Data_Type >
uint BArrayDense< Cell_Type, Data_Type >::nnozero  [inline], [noexcept]
```

Definition at line 462 of file barraydense-meet.hpp.

7.4.3.18 nrow()

```
template<typename Cell_Type , typename Data_Type >
uint BArrayDense< Cell_Type, Data_Type >::nrow  [inline], [noexcept]
```

Definition at line 452 of file barraydense-meet.hpp.

7.4.3.19 operator()() [1/2]

```
template<typename Cell_Type , typename Data_Type >
BArrayDenseCell< Cell_Type, Data_Type > BArrayDense< Cell_Type, Data_Type >::operator() (
            uint i,
            uint j,
            bool check_bounds = true )  [inline]
```

Definition at line 503 of file barraydense-meet.hpp.

7.4.3.20 operator()() [2/2]

```
template<typename Cell_Type , typename Data_Type >
const BArrayDenseCell_const< Cell_Type, Data_Type > BArrayDense< Cell_Type, Data_Type >↩
::operator() (
            uint i,
            uint j,
            bool check_bounds = true ) const  [inline]
```

Definition at line 512 of file barraydense-meet.hpp.

7.4.3.21 operator*=()

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

7.4.3.22 operator+=() [1/3]

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

7.4.3.23 operator+=() [2/3]

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

7.4.3.24 operator+=() [3/3]

```
template<typename Cell_Type , typename Data_Type >
BArrayDense< Cell_Type, Data_Type > & BArrayDense< Cell_Type, Data_Type >::operator+= (
            const std::pair< uint, uint > & coords )  [inline]
```

Definition at line 472 of file barraydense-meet.hpp.

7.4.3.25 operator-=() [1/3]

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

7.4.3.26 operator-=() [2/3]

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

7.4.3.27 operator-=() [3/3]

```
template<typename Cell_Type , typename Data_Type >
BArrayDense< Cell_Type, Data_Type > & BArrayDense< Cell_Type, Data_Type >::operator-= (
            const std::pair< uint, uint > & coords )  [inline]
```

Definition at line 488 of file barraydense-meet.hpp.

7.4.3.28 operator/=()

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

7.4.3.29 operator=() [1/2]

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

Move assignment.

Definition at line 212 of file barraydense-meet.hpp.

7.4.3.30 operator=() [2/2]

```
template<typename Cell_Type , typename Data_Type >
BArrayDense< Cell_Type, Data_Type > & BArrayDense< Cell_Type, Data_Type >::operator= (
            const BArrayDense< Cell_Type, Data_Type > & Array_ )  [inline]
```

Assignment constructor.

Definition at line 156 of file barraydense-meet.hpp.

7.4.3.31 operator==()

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

Definition at line 254 of file barraydense-meet.hpp.

7.4.3.32 out_of_range()

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::out_of_range (
            uint i,
            uint j ) const  [inline]
```

Definition at line 307 of file barraydense-meet.hpp.

7.4.3.33 print()

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::print  [inline]
```

Definition at line 1072 of file barraydense-meet.hpp.

7.4.3.34 reserve()

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::reserve  [inline]
```

Definition at line 1059 of file barraydense-meet.hpp.

7.4.3.35 resize()

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::resize (
            uint N_,
            uint M_ )  [inline]
```

Definition at line 1027 of file barraydense-meet.hpp.

7.4.3.36 rm_cell()

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

Definition at line 521 of file barraydense-meet.hpp.

7.4.3.37 row()

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

Definition at line 391 of file barraydense-meet.hpp.

7.4.3.38 set_data()

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::set_data (
            Data_Type * data_,
            bool delete_data_ = false )  [inline]
```

Set the data object.

Parameters

*data_*	
*delete_↩ data_*	

Definition at line 282 of file barraydense-meet.hpp.

7.4.3.39 swap_cells()

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< 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 )  [inline]
```

Definition at line 657 of file barraydense-meet.hpp.

7.4.3.40 swap_cols()

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

Definition at line 838 of file barraydense-meet.hpp.

7.4.3.41 swap_rows()

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

Definition at line 792 of file barraydense-meet.hpp.

7.4.3.42 toggle_cell()

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

Definition at line 758 of file barraydense-meet.hpp.

7.4.3.43 toggle_lock()

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

7.4.3.44 transpose()

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::transpose  [inline]
```

Definition at line 947 of file barraydense-meet.hpp.

7.4.3.45 zero_col()

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::zero_col (
            uint j,
            bool check_bounds = true )  [inline]
```

Definition at line 925 of file barraydense-meet.hpp.

7.4.3.46 zero_row()

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::zero_row (
            uint i,
            bool check_bounds = true )  [inline]
```

Definition at line 906 of file barraydense-meet.hpp.

7.4.4 Friends And Related Function Documentation

7.4.4.1 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 barraydense-bones.hpp.

7.4.4.2 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 barraydense-bones.hpp.

7.4.5 Member Data Documentation

7.4.5.1 visited

```
template<typename Cell_Type = bool, typename Data_Type = bool>
bool BArrayDense< 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 44 of file barraydense-bones.hpp.

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

- include/barry/barraydense-bones.hpp
- include/barry/barraydense-meet.hpp

7.5 BArrayDenseCell< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

- BArrayDenseCell (BArrayDense< Cell_Type, Data_Type > *Array_, uint i_, uint j_, bool check_bounds=true)
- ~BArrayDenseCell ()
- 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

7.5.1 Detailed Description

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

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

7.5.2 Constructor & Destructor Documentation

7.5.2.1 BArrayDenseCell()

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

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

7.5.2.2 ~BArrayDenseCell()

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

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

7.5.3 Member Function Documentation

7.5.3.1 operator Cell_Type()

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

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

7.5.3.2 operator*=()

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

Definition at line 30 of file barraydensecell-meat.hpp.

7.5.3.3 operator+=()

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

Definition at line 16 of file barraydensecell-meat.hpp.

7.5.3.4 operator-=()

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

Definition at line 23 of file barraydensecell-meat.hpp.

7.5.3.5 operator/=()

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

Definition at line 37 of file barraydensecell-meat.hpp.

7.5.3.6 operator=()

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

Definition at line 9 of file barraydensecell-meat.hpp.

7.5.3.7 operator==()

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

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

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

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

7.6 BArrayDenseCell_const< Cell_Type, Data_Type > Class Template Reference

`#include <barraydensecell-bones.hpp>`

Public Member Functions

- BArrayDenseCell_const (const BArrayDense< Cell_Type, Data_Type > *Array_, uint i_, uint j_, bool check_bounds=true)
- ~BArrayDenseCell_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

7.6.1 Detailed Description

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

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

7.6.2 Constructor & Destructor Documentation

7.6.2.1 BArrayDenseCell_const()

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

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

7.6.2.2 ~BArrayDenseCell_const()

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

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

7.6.3 Member Function Documentation

7.6.3.1 operator Cell_Type()

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

Definition at line 54 of file barraydensecell-meat.hpp.

7.6.3.2 operator"!=()

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

Definition at line 64 of file barraydensecell-meat.hpp.

7.6.3.3 operator<()

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

Definition at line 69 of file barraydensecell-meat.hpp.

7.6.3.4 operator<=()

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

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

7.6.3.5 operator==()

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

Definition at line 59 of file barraydensecell-meat.hpp.

7.6.3.6 operator>()

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

Definition at line 74 of file barraydensecell-meat.hpp.

7.6.3.7 operator>=()

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

Definition at line 84 of file barraydensecell-meat.hpp.

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

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

7.7 BArrayVector< Cell_Type, Data_Type > Class Template Reference

Row or column of a BArray

`#include <barrayvector-bones.hpp>`

Public Member Functions

- BArrayVector (BArray< Cell_Type, Data_Type > *Array_, uint &dim_ uint &i_, bool check_bounds=true)
  Construct a new BArrayVector object.
- ~BArrayVector ()
- bool is_row () const noexcept
- bool is_col () const noexcept
- uint size () const noexcept
- std::vector< Cell_Type >::const_iterator begin () noexcept
- std::vector< Cell_Type >::const_iterator end () noexcept
- 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 std::vector< Cell_Type > () const
- bool operator== (const Cell_Type &val) const

7.7.1 Detailed Description

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

Row or column of a BArray

Template Parameters

Cell_Type	
Data_Type	

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

7.7.2 Constructor & Destructor Documentation

7.7.2.1 BArrayVector()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArrayVector< Cell_Type, Data_Type >::BArrayVector (
```

```
BArray< Cell_Type, Data_Type > * Array_,
uint &dim_ uint & i_,
bool check_bounds = true )  [inline]
```

Construct a new BArrayVector object.

Parameters

*Array_*	Pointer to a `BArray` object
*dim_*	Dimension. 0 means row and 1 means column.
*i_*	Element to point.
check_bounds	When `true`, check boundaries.

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

7.7.2.2 ~BArrayVector()

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

Definition at line 55 of file barrayvector-bones.hpp.

7.7.3 Member Function Documentation

7.7.3.1 begin()

```
template<typename Cell_Type , typename Data_Type >
std::vector< Cell_Type >::const_iterator BArrayVector< Cell_Type, Data_Type >::begin  [inline],
[noexcept]
```

Definition at line 52 of file barrayvector-meat.hpp.

7.7.3.2 end()

```
template<typename Cell_Type , typename Data_Type >
std::vector< Cell_Type >::const_iterator BArrayVector< Cell_Type, Data_Type >::end  [inline],
[noexcept]
```

Definition at line 66 of file barrayvector-meat.hpp.

7.7.3.3 is_col()

```
template<typename Cell_Type , typename Data_Type >
bool BArrayVector< Cell_Type, Data_Type >::is_col  [inline], [noexcept]
```

Definition at line 36 of file barrayvector-meat.hpp.

7.7.3.4 is_row()

```
template<typename Cell_Type , typename Data_Type >
bool BArrayVector< Cell_Type, Data_Type >::is_row  [inline], [noexcept]
```

Definition at line 31 of file barrayvector-meat.hpp.

7.7.3.5 operator std::vector< Cell_Type >()

```
template<typename Cell_Type , typename Data_Type >
BArrayVector< Cell_Type, Data_Type >::operator std::vector< Cell_Type >  [inline]
```

Definition at line 177 of file barrayvector-meat.hpp.

7.7.3.6 operator*=()

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

Definition at line 135 of file barrayvector-meat.hpp.

7.7.3.7 operator+=()

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

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

7.7.3.8 operator-=()

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

Definition at line 114 of file barrayvector-meat.hpp.

7.7.3.9 operator/=()

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

Definition at line 156 of file barrayvector-meat.hpp.

7.7.3.10 operator=()

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

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

7.7.3.11 operator==()

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

Definition at line 187 of file barrayvector-meat.hpp.

7.7.3.12 size()

```
template<typename Cell_Type , typename Data_Type >
uint BArrayVector< Cell_Type, Data_Type >::size  [inline], [noexcept]
```

Definition at line 41 of file barrayvector-meat.hpp.

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

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

7.8 BArrayVector_const< Cell_Type, Data_Type > Class Template Reference

`#include <barrayvector-bones.hpp>`

Public Member Functions

- BArrayVector_const (const BArray< Cell_Type, Data_Type > *Array_, uint &dim_ uint &i_, bool check_↩ bounds=true)
- ~BArrayVector_const ()
- bool is_row () const noexcept
- bool is_col () const noexcept
- uint size () const noexcept
- std::vector< Cell_Type >::const_iterator begin () noexcept
- std::vector< Cell_Type >::const_iterator end () noexcept
- operator std::vector< 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

7.8.1 Detailed Description

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

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

7.8.2 Constructor & Destructor Documentation

7.8.2.1 BArrayVector_const()

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

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

7.8.2.2 ~BArrayVector_const()

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

Definition at line 110 of file barrayvector-bones.hpp.

7.8.3 Member Function Documentation

7.8.3.1 begin()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
std::vector< Cell_Type >::const_iterator BArrayVector_const< Cell_Type, Data_Type >::begin (
)  [noexcept]
```

7.8.3.2 end()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
std::vector< Cell_Type >::const_iterator BArrayVector_const< Cell_Type, Data_Type >::end ( )
[noexcept]
```

7.8.3.3 is_col()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
bool BArrayVector_const< Cell_Type, Data_Type >::is_col ( ) const  [noexcept]
```

7.8.3.4 is_row()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
bool BArrayVector_const< Cell_Type, Data_Type >::is_row ( ) const  [noexcept]
```

7.8.3.5 operator std::vector< Cell_Type >()

```
template<typename Cell_Type , typename Data_Type >
BArrayVector_const< Cell_Type, Data_Type >::operator std::vector< Cell_Type >  [inline]
```

Definition at line 214 of file barrayvector-meat.hpp.

7.8.3.6 operator"!=()

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

Definition at line 251 of file barrayvector-meat.hpp.

7.8.3.7 operator<()

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

Definition at line 256 of file barrayvector-meat.hpp.

7.8.3.8 operator<=()

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

Definition at line 283 of file barrayvector-meat.hpp.

7.8.3.9 operator==()

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

Definition at line 224 of file barrayvector-meat.hpp.

7.8.3.10 operator>()

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

Definition at line 310 of file barrayvector-meat.hpp.

7.8.3.11 operator>=()

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

Definition at line 317 of file barrayvector-meat.hpp.

7.8.3.12 size()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
uint BArrayVector_const< Cell_Type, Data_Type >::size ( ) const  [noexcept]
```

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

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

7.9 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
- bool operator== (const Cell< Cell_Type > &rhs) const
- bool operator!= (const Cell< Cell_Type > &rhs) const
- void add (double x)
- void add (unsigned int x)
- void add (int x)
- Cell ()
- Cell ()
- Cell ()

Public Attributes

- Cell_Type value
- bool visited

7.9.1 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.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 Cell() [1/7]

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

7.9.2.2 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.

7.9.2.3 ~Cell()

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

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

7.9.2.4 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.

7.9.2.5 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.

7.9.2.6 Cell() [5/7]

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

Definition at line 62 of file cell-meat.hpp.

7.9.2.7 Cell() [6/7]

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

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

7.9.2.8 Cell() [7/7]

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

Definition at line 64 of file cell-meat.hpp.

7.9.3 Member Function Documentation

7.9.3.1 add() [1/4]

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

7.9.3.2 add() [2/4]

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

Definition at line 42 of file cell-meat.hpp.

7.9.3.3 add() [3/4]

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

Definition at line 52 of file cell-meat.hpp.

7.9.3.4 add() [4/4]

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

Definition at line 47 of file cell-meat.hpp.

7.9.3.5 operator Cell_Type()

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

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

7.9.3.6 operator"!=()

```
template<typename Cell_Type >
bool Cell< Cell_Type >::operator!= (
            const Cell< Cell_Type > & rhs ) const
```

Definition at line 31 of file cell-meat.hpp.

7.9.3.7 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.

7.9.3.8 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.

7.9.3.9 operator==()

```
template<typename Cell_Type >
bool Cell< Cell_Type >::operator== (
            const Cell< Cell_Type > & rhs ) const
```

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

7.9.4 Member Data Documentation

7.9.4.1 value

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

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

7.9.4.2 visited

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

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

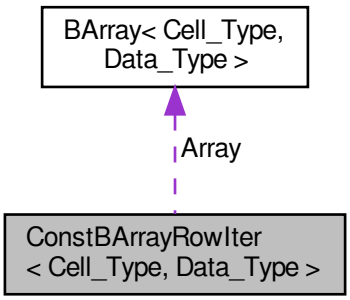
The documentation for this class was generated from the following files:

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

7.10 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

7.10.1 Detailed Description

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

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

7.10.2 Constructor & Destructor Documentation

7.10.2.1 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.

7.10.2.2 ~ConstBArrayRowIter()

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

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

7.10.3 Member Data Documentation

7.10.3.1 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.

7.10.3.2 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.

7.10.3.3 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.

7.10.3.4 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

7.11 Counter< Array_Type, Data_Type > Class Template Reference

A counter function based on change statistics.

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

Public Member Functions

- ~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, std::string name_="", std::string desc_="")
- Counter (const Counter< Array_Type, Data_Type > &counter_)
  Copy constructor.
- Counter (Counter< Array_Type, Data_Type > &&counter_) noexcept
  Move constructor.
- Counter< Array_Type, Data_Type > operator= (const Counter< Array_Type, Data_Type > &counter_)
  Copy assignment.
- Counter< Array_Type, Data_Type > & operator= (Counter< Array_Type, Data_Type > &&counter_) noexcept
  Move assignment.

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
- std::string name = ""
- std::string desc = ""

7.11.1 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 38 of file counters-bones.hpp.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 Counter() [1/4]

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

Definition at line 59 of file counters-bones.hpp.

7.11.2.2 Counter() [2/4]

```
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,
            std::string name_ = "",
            std::string desc_ = "" )  [inline]
```

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

7.11.2.3 Counter() [3/4]

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

Copy constructor.

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

7.11.2.4 Counter() [4/4]

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

Move constructor.

Definition at line 33 of file counters-meat.hpp.

7.11.2.5 ~Counter()

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

Definition at line 77 of file counters-bones.hpp.

7.11.3 Member Function Documentation

7.11.3.1 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 114 of file counters-meat.hpp.

7.11.3.2 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 126 of file counters-meat.hpp.

7.11.3.3 operator=() [1/2]

```
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_ )  [inline]
```

Copy assignment.

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

7.11.3.4 operator=() [2/2]

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

Move assignment.

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

7.11.4 Member Data Documentation

7.11.4.1 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 41 of file counters-bones.hpp.

7.11.4.2 data

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

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

7.11.4.3 delete_data

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

Definition at line 44 of file counters-bones.hpp.

7.11.4.4 desc

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::string Counter< Array_Type, Data_Type >::desc = ""
```

Definition at line 46 of file counters-bones.hpp.

7.11.4.5 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 42 of file counters-bones.hpp.

7.11.4.6 name

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::string Counter< Array_Type, Data_Type >::name = ""
```

Definition at line 45 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

7.12 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_)

  Copy constructor.
- Counters (Counters< Array_Type, Data_Type > &&counters_) noexcept

  Move constructor.
- Counters< Array_Type, Data_Type > operator= (const Counters< Array_Type, Data_Type > &counter_)

  Copy assignment constructor.
- Counters< Array_Type, Data_Type > & operator= (Counters< Array_Type, Data_Type > &&counter_) noexcept

  Move assignment constructor.
- Counter< Array_Type, Data_Type > & operator[ ] (uint idx)

  Returns a pointer to a particular counter.
- std::size_t size () const noexcept

  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, std::string name_="", std::string desc_="")
- void clear ()

7.12.1 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 99 of file counters-bones.hpp.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 Counters() [1/3]

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

Definition at line 143 of file counters-meat.hpp.

7.12.2.2 ~Counters()

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

Definition at line 113 of file counters-bones.hpp.

7.12.2.3 Counters() [2/3]

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

Copy constructor.

Parameters

*counter_*	

Definition at line 160 of file counters-meat.hpp.

7.12.2.4 Counters() [3/3]

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

Move constructor.

Parameters

*counters_*	

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

7.12.3 Member Function Documentation

7.12.3.1 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 276 of file counters-meat.hpp.

7.12.3.2 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 288 of file counters-meat.hpp.

7.12.3.3 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,
            std::string name_ = "",
            std::string desc_ = "" )  [inline]
```

Definition at line 299 of file counters-meat.hpp.

7.12.3.4 clear()

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

Definition at line 328 of file counters-meat.hpp.

7.12.3.5 operator=() [1/2]

```
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_ )
```

Copy assignment constructor.

Parameters

*counter_*	

Returns

Counters<Array_Type,Data_Type>

Definition at line 209 of file counters-meat.hpp.

7.12.3.6 operator=() [2/2]

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

Move assignment constructor.

Parameters

*counter_*	

Returns

Counters<Array_Type,Data_Type>&

Definition at line 248 of file counters-meat.hpp.

7.12.3.7 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 153 of file counters-meat.hpp.

7.12.3.8 size()

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

Number of counters in the set.

Returns

uint

Definition at line 159 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

7.13 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

7.13.1 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 59 of file typedefs.hpp.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 Entries() [1/2]

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

Definition at line 65 of file typedefs.hpp.

7.13.2.2 Entries() [2/2]

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

Definition at line 66 of file typedefs.hpp.

7.13.2.3 ~Entries()

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

Definition at line 73 of file typedefs.hpp.

7.13.3 Member Function Documentation

7.13.3.1 resize()

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

Definition at line 75 of file typedefs.hpp.

7.13.4 Member Data Documentation

7.13.4.1 source

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

Definition at line 61 of file typedefs.hpp.

7.13.4.2 target

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

Definition at line 62 of file typedefs.hpp.

7.13.4.3 val

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

Definition at line 63 of file typedefs.hpp.

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

- include/barry/typedefs.hpp

7.14 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)

  Add a tree to the flock.
- void set_seed (const unsigned int &s)

  Set the seed of the model.
- void init (bool verb=true)
- phylocounters::PhyloCounters * get_counters ()
- phylocounters::PhyloSupport * get_support ()
- double likelihood_joint (const std::vector< double > &par, bool as_log=false, bool use_reduced_sequence=true)

  Returns the joint likelihood of the model.
- Geese * operator() (unsigned int i, bool check_bounds=true)

  Access the i-th geese element.

Information about the model

- unsigned int nfuns () const noexcept
- unsigned int ntrees () const noexcept
- std::vector< unsigned int > nnodes () const noexcept
- std::vector< unsigned int > nleafs () const noexcept
- unsigned int nterms () const
- unsigned int support_size () const noexcept
- std::vector< std::string > colnames () const
- unsigned int parse_polytomies (bool verb=true) const noexcept

Public Attributes

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

7.14.1 Detailed Description

A Flock is a group of Geese.

This object builds a model with multiple trees (Geese objects), with all of these using the same PhyloModel object. Available counters (terms) can be found in counter-phylo.

Definition at line 14 of file flock-bones.hpp.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 Flock()

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

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

7.14.2.2 ~Flock()

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

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

7.14.3 Member Function Documentation

7.14.3.1 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]
```

Add a tree to the flock.

Parameters

annotations	see Geese::Geese.
geneid	see Geese.
parent	see Geese.
duplication	see Geese.

Returns

unsigned int The number of tree in the model (starting from zero).

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

7.14.3.2 colnames()

```
std::vector< std::string > Flock::colnames ( ) const  [inline]
```

Definition at line 155 of file flock-meet.hpp.

7.14.3.3 get_counters()

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

Definition at line 69 of file flock-meet.hpp.

7.14.3.4 get_support()

```
phylocounters::PhyloSupport * Flock::get_support ( )  [inline]
```

Definition at line 78 of file flock-meet.hpp.

7.14.3.5 init()

```
void Flock::init (
            bool verb = true )  [inline]
```

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

7.14.3.6 likelihood_joint()

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

Returns the joint likelihood of the model.

Parameters

par	Vector of model parameters.
as_log	When `true` it will return the value as log.
use_reduced_sequence	When `true` (default) will compute the likelihood using the reduced sequence, which is faster.

Returns

double

Definition at line 82 of file flock-meet.hpp.

7.14.3.7 nfuns()

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

Definition at line 107 of file flock-meet.hpp.

7.14.3.8 nleafs()

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

Definition at line 130 of file flock-meet.hpp.

7.14.3.9 nnodes()

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

Definition at line 119 of file flock-meet.hpp.

7.14.3.10 nterms()

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

Definition at line 142 of file flock-meet.hpp.

7.14.3.11 ntrees()

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

Definition at line 113 of file flock-meet.hpp.

7.14.3.12 operator()()

```
Geese * Flock::operator() (
            unsigned int i,
            bool check_bounds = true )  [inline]
```

Access the i-th geese element.

Parameters

i	Element to access
check_bounds	When true, it will check bounds.

Returns

Geese*

Definition at line 180 of file flock-meet.hpp.

7.14.3.13 parse_polytomies()

```
unsigned int Flock::parse_polytomies (
            bool verb = true ) const  [inline], [noexcept]
```

Definition at line 161 of file flock-meet.hpp.

7.14.3.14 set_seed()

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

Set the seed of the model.

Parameters

s	Passed to the `rengine.seed()` member object.

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

7.14.3.15 support_size()

```
unsigned int Flock::support_size ( ) const  [inline], [noexcept]
```

Definition at line 149 of file flock-meet.hpp.

7.14.4 Member Data Documentation

7.14.4.1 dat

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

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

7.14.4.2 initialized

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

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

7.14.4.3 model

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

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

7.14.4.4 nfunctions

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

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

7.14.4.5 rengine

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

Definition at line 22 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

7.15 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
- size_t size () const noexcept

7.15.1 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.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 FreqTable()

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

Definition at line 28 of file statsdb.hpp.

7.15.2.2 ~FreqTable()

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

Definition at line 29 of file statsdb.hpp.

7.15.3 Member Function Documentation

7.15.3.1 add()

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

Definition at line 47 of file statsdb.hpp.

7.15.3.2 as_vector()

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

Definition at line 61 of file statsdb.hpp.

7.15.3.3 clear()

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

Definition at line 83 of file statsdb.hpp.

7.15.3.4 get_data()

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

Definition at line 73 of file statsdb.hpp.

7.15.3.5 get_data_ptr()

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

Definition at line 78 of file statsdb.hpp.

7.15.3.6 print()

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

Definition at line 102 of file statsdb.hpp.

7.15.3.7 reserve()

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

Definition at line 89 of file statsdb.hpp.

7.15.3.8 size()

```
template<typename T >
size_t FreqTable< T >::size  [inline], [noexcept]
```

Definition at line 126 of file statsdb.hpp.

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

- include/barry/statsdb.hpp

7.16 Geese Class Reference

Annotated Phylo Model.

`#include <geese-bones.hpp>`

Public Member Functions

- ~Geese ()
- void init (bool verb=true)
- void inherit_support (const Geese &model_, bool delete_support_=false)
- void calc_sequence (Node *n=nullptr)
- void calc_reduced_sequence ()
- double likelihood (const std::vector< double > &par, bool as_log=false, bool use_reduced_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 ()
- void init_node (Node &n)
- void update_annotations (unsigned int nodeid, std::vector< unsigned int > newann)
- std::vector< std::vector< bool > > get_states () const

  Powerset of a gene's possible states.
- std::vector< unsigned int > get_annotated_nodes () const

  Returns the ids of the nodes with at least one annotation.

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

Parameters

verb	*When* `true` *it will print out information about the encountered polytomies.*

- unsigned int nfuns () const noexcept
  Number of functions analyzed.
- unsigned int nnodes () const noexcept
  Number of nodes (interior + leaf)
- unsigned int nleafs () const noexcept
  Number of leaf.
- unsigned int nterms () const
  Number of terms included.
- unsigned int support_size () const noexcept
  Number of unique sets of sufficient stats.
- std::vector< std::string > colnames () const
  Names of the terms in the model.
- unsigned int parse_polytomies (bool verb=true) const noexcept
  Check polytomies and return the largest.

Geese prediction

Calculate the conditional probability

Parameters

par	*Vector of parameters (terms + root).*
res_prob	*Vector indicating each nodes' state probability.*
leave_one_out	*When* `true`*, it will compute the predictions using leave-one-out, thus the prediction will be repeated nleaf times.*
only_annotated	*When* `true`*, it will make the predictions only on the induced sub-tree with annotated leafs.*
use_reduced_sequence	*Passed to the* `likelihood` *method.*
preorder	*For the tree traversal.*

When `res_prob` *is specified, the function will attach the member vector* `probabilities` *from the* `Node`*s objects. This contains the probability that the ith node has either of the possible states.*

Returns

std::vector< double > Returns the posterior probability

- std::vector< std::vector< double > > predict (const std::vector< double > &par, std::vector< std::vector< double > > *res_prob=nullptr, bool leave_one_out=false, bool only_annotated=false, bool use_reduced_sequence=true)
- std::vector< std::vector< double > > predict_backend (const std::vector< double > &par, bool use_reduced_sequence, const std::vector< uint > &preorder)
- std::vector< std::vector< double > > predict_exhaust_backend (const std::vector< double > &par, const std::vector< uint > &preorder)
- std::vector< std::vector< double > > predict_exhaust (const std::vector< double > &par)
- std::vector< std::vector< double > > predict_sim (const std::vector< double > &par, bool only_annotated=false, unsigned int nsims=10000u)

Non-const pointers to shared objects in <tt>Geese</tt>

These functions provide direct access to some member objects that are shared by the nodes within `Geese`.

Returns

`get_rengine()` returns the Pseudo-RNG engine used.
`get_counters()` returns the vector of counters used.
`get_model()` returns the `Model` object used.
`get_support()` returns the computed support of the model.

- std::mt19937 * get_rengine ()
- phylocounters::PhyloCounters * get_counters ()
- phylocounters::PhyloModel * get_model ()
- phylocounters::PhyloSupport * get_support ()

Public Attributes

- unsigned int nfunctions
- std::map< unsigned int, Node > nodes
- barry::MapVec_type< unsigned int > map_to_nodes
- std::vector< unsigned int > sequence
- std::vector< unsigned int > reduced_sequence
- bool initialized = false
- bool delete_rengine = false
- bool delete_support = false

7.16.1 Detailed Description

Annotated Phylo Model.

A list of available terms for this model can be found in the Phylo counters section.

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

7.16.2 Constructor & Destructor Documentation

7.16.2.1 Geese() [1/4]

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

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

7.16.2.2 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 20 of file geese-meat-constructors.hpp.

7.16.2.3 Geese() [3/4]

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

Definition at line 163 of file geese-meat-constructors.hpp.

7.16.2.4 Geese() [4/4]

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

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

7.16.2.5 ~Geese()

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

Definition at line 75 of file geese-meat.hpp.

7.16.3 Member Function Documentation

7.16.3.1 calc_reduced_sequence()

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

Definition at line 274 of file geese-meat.hpp.

7.16.3.2 calc_sequence()

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

Definition at line 235 of file geese-meat.hpp.

7.16.3.3 colnames()

```
std::vector< std::string > Geese::colnames ( ) const  [inline]
```

Names of the terms in the model.

Definition at line 367 of file geese-meat.hpp.

7.16.3.4 get_annotated_nodes()

```
std::vector< unsigned int > Geese::get_annotated_nodes ( ) const  [inline]
```

Returns the ids of the nodes with at least one annotation.

Definition at line 527 of file geese-meat.hpp.

7.16.3.5 get_counters()

```
phylocounters::PhyloCounters * Geese::get_counters ( )  [inline]
```

Definition at line 511 of file geese-meat.hpp.

7.16.3.6 get_model()

```
phylocounters::PhyloModel * Geese::get_model ( )  [inline]
```

Definition at line 515 of file geese-meat.hpp.

7.16.3.7 get_probabilities()

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

Definition at line 317 of file geese-meat.hpp.

7.16.3.8 get_rengine()

```
std::mt19937 * Geese::get_rengine ( )  [inline]
```

Definition at line 507 of file geese-meat.hpp.

7.16.3.9 get_states()

```
std::vector< std::vector< bool > > Geese::get_states ( ) const  [inline]
```

Powerset of a gene's possible states.

This list of vectors is used throughout Geese. It lists all possible combinations of functional states for any gene. Thus, for P functions, there will be 2^P possible combinations.

Returns

std::vector< std::vector< bool > > of length 2^P.

Definition at line 523 of file geese-meat.hpp.

7.16.3.10 get_support()

```
phylocounters::PhyloSupport * Geese::get_support ( )  [inline]
```

Definition at line 519 of file geese-meat.hpp.

7.16.3.11 inherit_support()

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

Definition at line 183 of file geese-meat.hpp.

7.16.3.12 init()

```
void Geese::init (
            bool verb = true )  [inline]
```

Definition at line 87 of file geese-meat.hpp.

7.16.3.13 init_node()

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

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

7.16.3.14 likelihood()

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

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

7.16.3.15 likelihood_exhaust()

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

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

7.16.3.16 nfuns()

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

Number of functions analyzed.

Definition at line 333 of file geese-meat.hpp.

7.16.3.17 nleafs()

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

Number of leaf.

Definition at line 341 of file geese-meat.hpp.

7.16.3.18 nnodes()

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

Number of nodes (interior + leaf)

Definition at line 337 of file geese-meat.hpp.

7.16.3.19 nterms()

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

Number of terms included.

Definition at line 351 of file geese-meat.hpp.

7.16.3.20 observed_counts()

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

Definition at line 397 of file geese-meat.hpp.

7.16.3.21 operator=() [1/2]

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

7.16.3.22 operator=() [2/2]

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

7.16.3.23 parse_polytomies()

```
unsigned int Geese::parse_polytomies (
            bool verb = true ) const  [inline], [noexcept]
```

Check polytomies and return the largest.

Definition at line 373 of file geese-meat.hpp.

7.16.3.24 predict()

```
std::vector< std::vector< double > > Geese::predict (
            const std::vector< double > & par,
            std::vector< std::vector< double > > * res_prob = nullptr,
            bool leave_one_out = false,
            bool only_annotated = false,
            bool use_reduced_sequence = true )  [inline]
```

Definition at line 212 of file geese-meat-predict.hpp.

7.16.3.25 predict_backend()

```
std::vector< std::vector< double > > Geese::predict_backend (
            const std::vector< double > & par,
            bool use_reduced_sequence,
            const std::vector< uint > & preorder )  [inline]
```

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

7.16.3.26 predict_exhaust()

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

Definition at line 5 of file geese-meat-predict_exhaust.hpp.

7.16.3.27 predict_exhaust_backend()

```
std::vector< std::vector< double > > Geese::predict_exhaust_backend (
            const std::vector< double > & par,
            const std::vector< uint > & preorder )  [inline]
```

Definition at line 47 of file geese-meat-predict_exhaust.hpp.

7.16.3.28 predict_sim()

```
std::vector< std::vector< double > > Geese::predict_sim (
            const std::vector< double > & par,
            bool only_annotated = false,
            unsigned int nsims = 10000u )  [inline]
```

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

7.16.3.29 print_observed_counts()

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

Definition at line 445 of file geese-meat.hpp.

7.16.3.30 set_seed()

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

Definition at line 4 of file geese-meat-simulate.hpp.

7.16.3.31 simulate()

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

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

7.16.3.32 support_size()

```
unsigned int Geese::support_size ( ) const  [inline], [noexcept]
```

Number of unique sets of sufficient stats.

Definition at line 358 of file geese-meat.hpp.

7.16.3.33 update_annotations()

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

Definition at line 206 of file geese-meat.hpp.

7.16.4 Member Data Documentation

7.16.4.1 delete_rengine

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

Definition at line 109 of file geese-bones.hpp.

7.16.4.2 delete_support

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

Definition at line 110 of file geese-bones.hpp.

7.16.4.3 initialized

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

Definition at line 108 of file geese-bones.hpp.

7.16.4.4 map_to_nodes

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

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

7.16.4.5 nfunctions

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

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

7.16.4.6 nodes

```
std::map< unsigned int, Node > Geese::nodes
```

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

7.16.4.7 reduced_sequence

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

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

7.16.4.8 sequence

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

Definition at line 104 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-predict_exhaust.hpp
- include/barry/models/geese/geese-meat-predict_sim.hpp
- include/barry/models/geese/geese-meat-simulate.hpp
- include/barry/models/geese/geese-meat.hpp

7.17 Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_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>
```

Public Member Functions

- void set_rengine (std::mt19937 *rengine_, bool delete_=false)
- void set_seed (unsigned int s)
- Model ()
- Model (uint size_)
- Model (const Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > &Model_)
- Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > & operator= (const Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > &Model_)
- ~Model ()
- void store_psets () noexcept
- 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)
- const std::mt19937 * get_rengine () const
- Counters< Array_Type, Data_Counter_Type > * get_counters ()
- Rules< Array_Type, Data_Rule_Type > * get_rules ()
- Rules< Array_Type, Data_Rule_Dyn_Type > * get_rules_dyn ()
- Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > * get_support ()

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_)
- void add_rule_dyn (Rule< Array_Type, Data_Rule_Dyn_Type > &rule)
- void add_rule_dyn (Rule< Array_Type, Data_Rule_Dyn_Type > *rule)
- void add_rule_dyn (Rule_fun_type< Array_Type, Data_Rule_Dyn_Type > count_fun_, Data_Rule_Dyn_Type *data_=nullptr, bool delete_data_=false)
- void set_rules_dyn (Rules< Array_Type, Data_Rule_Dyn_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 noexcept
- unsigned int size_unique () const noexcept
- unsigned int nterms () const noexcept
- unsigned int support_size () const noexcept
- std::vector< std::string > colnames () const

7.17.1 Detailed Description

template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
class Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_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 104 of file model-bones.hpp.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 Model() [1/3]

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

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

7.17.2.2 Model() [2/3]

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

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

7.17.2.3 Model() [3/3]

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

Definition at line 64 of file model-meat.hpp.

7.17.2.4 ~Model()

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

Definition at line 204 of file model-bones.hpp.

7.17.3 Member Function Documentation

7.17.3.1 add_array()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
uint Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_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 333 of file model-meat.hpp.

7.17.3.2 add_counter() [1/3]

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

Definition at line 167 of file model-meat.hpp.

7.17.3.3 add_counter() [2/3]

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

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

7.17.3.4 add_counter() [3/3]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_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 186 of file model-meat.hpp.

7.17.3.5 add_rule() [1/3]

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

Definition at line 225 of file model-meat.hpp.

7.17.3.6 add_rule() [2/3]

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

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

7.17.3.7 add_rule() [3/3]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_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 244 of file model-meat.hpp.

7.17.3.8 add_rule_dyn() [1/3]

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

Definition at line 279 of file model-meat.hpp.

7.17.3.9 add_rule_dyn() [2/3]

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

Definition at line 288 of file model-meat.hpp.

7.17.3.10 add_rule_dyn() [3/3]

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

Definition at line 298 of file model-meat.hpp.

7.17.3.11 colnames()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
std::vector< std::string > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_↩
Dyn_Type >::colnames  [inline]
```

Definition at line 689 of file model-meat.hpp.

7.17.3.12 get_counters()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
Counters< Array_Type, Data_Counter_Type > * Model< Array_Type, Data_Counter_Type, Data_Rule↩
_Type, Data_Rule_Dyn_Type >::get_counters  [inline]
```

Definition at line 743 of file model-meat.hpp.

7.17.3.13 get_norm_const()

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

Definition at line 585 of file model-meat.hpp.

7.17.3.14 get_pset()

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

Definition at line 617 of file model-meat.hpp.

7.17.3.15 get_rengine()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
const std::mt19937 * Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type
>::get_rengine  [inline]
```

Definition at line 737 of file model-meat.hpp.

7.17.3.16 get_rules()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
Rules< Array_Type, Data_Rule_Type > * Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_rules  [inline]
```

Definition at line 749 of file model-meat.hpp.

7.17.3.17 get_rules_dyn()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
Rules< Array_Type, Data_Rule_Dyn_Type > * Model< Array_Type, Data_Counter_Type, Data_Rule_↩
Type, Data_Rule_Dyn_Type >::get_rules_dyn  [inline]
```

Definition at line 755 of file model-meat.hpp.

7.17.3.18 get_stats()

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

Definition at line 630 of file model-meat.hpp.

7.17.3.19 get_support()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > * Model< Array↩
_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_support  [inline]
```

Definition at line 761 of file model-meat.hpp.

7.17.3.20 likelihood() [1/3]

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

Definition at line 463 of file model-meat.hpp.

7.17.3.21 likelihood() [2/3]

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

Definition at line 507 of file model-meat.hpp.

7.17.3.22 likelihood() [3/3]

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

Definition at line 426 of file model-meat.hpp.

7.17.3.23 likelihood_total()

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

Definition at line 545 of file model-meat.hpp.

7.17.3.24 nterms()

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

Definition at line 671 of file model-meat.hpp.

7.17.3.25 operator=()

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

Definition at line 102 of file model-meat.hpp.

7.17.3.26 print_stats()

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

Definition at line 642 of file model-meat.hpp.

7.17.3.27 sample() [1/2]

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

7.17.3.28 sample() [2/2]

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

Definition at line 701 of file model-meat.hpp.

7.17.3.29 set_counters()

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

Definition at line 205 of file model-meat.hpp.

7.17.3.30 set_keygen()

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

Definition at line 159 of file model-meat.hpp.

7.17.3.31 set_rengine()

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

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

7.17.3.32 set_rules()

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

Definition at line 261 of file model-meat.hpp.

7.17.3.33 set_rules_dyn()

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

Definition at line 315 of file model-meat.hpp.

7.17.3.34 set_seed()

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

Definition at line 186 of file model-bones.hpp.

7.17.3.35 size()

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

Definition at line 661 of file model-meat.hpp.

7.17.3.36 size_unique()

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

Definition at line 666 of file model-meat.hpp.

7.17.3.37 store_psets()

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

Definition at line 151 of file model-meat.hpp.

7.17.3.38 support_size()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
uint Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::support_size
[inline], [noexcept]
```

Definition at line 678 of file model-meat.hpp.

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

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

7.18 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

7.18.1 Detailed Description

Data class used to store arbitrary uint or double vectors.

Definition at line 61 of file network.hpp.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 NetCounterData() [1/2]

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

Definition at line 67 of file network.hpp.

7.18.2.2 NetCounterData() [2/2]

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

Definition at line 68 of file network.hpp.

7.18.2.3 ~NetCounterData()

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

Definition at line 73 of file network.hpp.

7.18.3 Member Data Documentation

7.18.3.1 indices

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

Definition at line 64 of file network.hpp.

7.18.3.2 numbers

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

Definition at line 65 of file network.hpp.

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

- include/barry/counters/network.hpp

7.19 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

7.19.1 Detailed Description

Data class for Networks.

Details on the available counters for `NetworkData` can be found in the Network counters section.

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

Definition at line 24 of file network.hpp.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 NetworkData() [1/3]

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

Definition at line 30 of file network.hpp.

7.19.2.2 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 38 of file network.hpp.

7.19.2.3 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 50 of file network.hpp.

7.19.2.4 ~NetworkData()

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

Definition at line 56 of file network.hpp.

7.19.3 Member Data Documentation

7.19.3.1 directed

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

Definition at line 27 of file network.hpp.

7.19.3.2 vertex_attr

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

Definition at line 28 of file network.hpp.

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

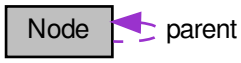
- include/barry/counters/network.hpp

7.20 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
- unsigned int noffspring () const noexcept
- bool is_leaf () const noexcept

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.

7.20.1 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.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 Node() [1/5]

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

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

7.20.2.2 Node() [2/5]

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

Definition at line 56 of file geese-node-bones.hpp.

7.20.2.3 Node() [3/5]

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

Definition at line 62 of file geese-node-bones.hpp.

7.20.2.4 Node() [4/5]

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

Definition at line 69 of file geese-node-bones.hpp.

7.20.2.5 Node() [5/5]

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

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

7.20.2.6 ~Node()

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

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

7.20.3 Member Function Documentation

7.20.3.1 get_parent()

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

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

7.20.3.2 is_leaf()

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

Definition at line 109 of file geese-node-bones.hpp.

7.20.3.3 noffspring()

```
unsigned int Node::noffspring ( ) const  [inline], [noexcept]
```

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

7.20.4 Member Data Documentation

7.20.4.1 annotations

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

Observed annotations (only defined for Geese)

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

7.20.4.2 array

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

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

7.20.4.3 arrays

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

Arrays given all possible states.

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

7.20.4.4 duplication

```
bool Node::duplication
```

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

7.20.4.5 id

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

Id of the node (as specified in the input)

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

7.20.4.6 narray

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

ID of the array in the model.

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

7.20.4.7 offspring

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

Offspring nodes.

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

7.20.4.8 ord

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

Order in which the node was created.

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

7.20.4.9 parent

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

Parent node.

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

7.20.4.10 probability

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

The probability of observing each state.

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

7.20.4.11 subtree_prob

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

Induced subtree probabilities.

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

7.20.4.12 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

7.21 NodeData Class Reference

Data definition for the `PhyloArray` class.

`#include <phylo.hpp>`

Public Member Functions

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

Public Attributes

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

7.21.1 Detailed Description

Data definition for the `PhyloArray` class.

Details about the available counters for `PhyloArray` objects can be found in the Phylo counters section.

This holds basic information about a given node.

Definition at line 23 of file phylo.hpp.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 NodeData()

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

Definition at line 43 of file phylo.hpp.

7.21.3 Member Data Documentation

7.21.3.1 blengths

```
std::vector< double > NodeData::blengths = {}
```

Branch length.

Definition at line 29 of file phylo.hpp.

7.21.3.2 duplication

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

Definition at line 39 of file phylo.hpp.

7.21.3.3 states

```
std::vector< bool > NodeData::states = {}
```

State of the parent node.

Definition at line 34 of file phylo.hpp.

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

- include/barry/counters/phylo.hpp

7.22 PhyloRuleDynData Class Reference

```
#include <phylo.hpp>
```

Public Member Functions

- PhyloRuleDynData (const std::vector< double > *counts_, uint pos_, uint lb_, uint ub_, bool duplication_)
- ~PhyloRuleDynData ()

Public Attributes

- const std::vector< double > * counts
- uint pos
- uint lb
- uint ub
- bool duplication

7.22.1 Detailed Description

Definition at line 1308 of file phylo.hpp.

7.22.2 Constructor & Destructor Documentation

7.22.2.1 PhyloRuleDynData()

```
PhyloRuleDynData::PhyloRuleDynData (
            const std::vector< double > * counts_,
            uint pos_,
            uint lb_,
            uint ub_,
            bool duplication_ )  [inline]
```

Definition at line 1315 of file phylo.hpp.

7.22.2.2 ~PhyloRuleDynData()

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

Definition at line 1324 of file phylo.hpp.

7.22.3 Member Data Documentation

7.22.3.1 counts

```
const std::vector< double >* PhyloRuleDynData::counts
```

Definition at line 1310 of file phylo.hpp.

7.22.3.2 duplication

```
bool PhyloRuleDynData::duplication
```

Definition at line 1314 of file phylo.hpp.

7.22.3.3 lb

```
uint PhyloRuleDynData::lb
```

Definition at line 1312 of file phylo.hpp.

7.22.3.4 pos

```
uint PhyloRuleDynData::pos
```

Definition at line 1311 of file phylo.hpp.

7.22.3.5 ub

```
uint PhyloRuleDynData::ub
```

Definition at line 1313 of file phylo.hpp.

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

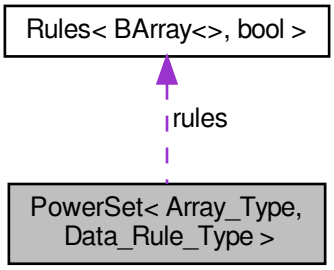
- include/barry/counters/phylo.hpp

7.23 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 ()
- std::size_t size () const noexcept
- 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

7.23.1 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.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 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.

7.23.2.2 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.

7.23.2.3 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.

7.23.2.4 ~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.

7.23.3 Member Function Documentation

7.23.3.1 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.

7.23.3.2 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.

7.23.3.3 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.

7.23.3.4 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.

7.23.3.5 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.

7.23.3.6 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.

7.23.3.7 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.

7.23.3.8 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.

7.23.3.9 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.

7.23.3.10 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.

7.23.3.11 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.

7.23.3.12 size()

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

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

7.23.4 Member Data Documentation

7.23.4.1 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.

7.23.4.2 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.

7.23.4.3 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.

7.23.4.4 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.

7.23.4.5 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.

7.23.4.6 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.

7.23.4.7 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.

7.23.4.8 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

7.24 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 ()
- Data_Type * D ()

  Read/Write access to the data.
- bool operator() (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)

7.24.1 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.

7.24.2 Constructor & Destructor Documentation

7.24.2.1 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.

7.24.2.2 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.

7.24.2.3 ~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.

7.24.3 Member Function Documentation

7.24.3.1 D()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Data_Type* Rule< Array_Type, Data_Type >::D ( )
```

Read/Write access to the data.

7.24.3.2 operator()()

```
template<typename Array_Type , typename Data_Type >
bool Rule< Array_Type, Data_Type >::operator() (
            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

7.25 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 noexcept
- bool operator() (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)

7.25.1 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 69 of file rules-bones.hpp.

7.25.2 Constructor & Destructor Documentation

7.25.2.1 Rules() [1/2]

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

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

7.25.2.2 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.

7.25.2.3 ~Rules()

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

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

7.25.3 Member Function Documentation

7.25.3.1 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.

7.25.3.2 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.

7.25.3.3 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.

7.25.3.4 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.

7.25.3.5 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.

7.25.3.6 operator()()

```
template<typename Array_Type , typename Data_Type >
bool Rules< Array_Type, Data_Type >::operator() (
```

```
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.

7.25.3.7 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.

7.25.3.8 size()

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

Definition at line 86 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

7.26 StatsCounter< Array_Type, Data_Type > Class Template Reference

Count stats for a single Array.

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

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 ()
- Counters< Array_Type, Data_Type > * get_counters ()

7.26.1 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.

7.26.2 Constructor & Destructor Documentation

7.26.2.1 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 36 of file statscounter-bones.hpp.

7.26.2.2 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 51 of file statscounter-bones.hpp.

7.26.2.3 ~StatsCounter()

```
template<typename Array_Type , typename Data_Type >
StatsCounter< Array_Type, Data_Type >::~StatsCounter  [inline]
```

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

7.26.3 Member Function Documentation

7.26.3.1 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.

7.26.3.2 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.

7.26.3.3 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.

7.26.3.4 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.

7.26.3.5 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.

7.26.3.6 get_counters()

```
template<typename Array_Type , typename Data_Type >
Counters< Array_Type, Data_Type > * StatsCounter< Array_Type, Data_Type >::get_counters  [inline]
```

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

7.26.3.7 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.

7.26.3.8 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.

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

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

7.27 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference

Compute the support of sufficient statistics.

`#include <support-bones.hpp>`

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, unsigned int max_num_elements_=0u)
  Computes the entire support.
- Counts_type get_counts () const
- const MapVec_type * get_counts_ptr () const
- std::vector< double > * get_current_stats ()
  List current statistics.
- void print () const
- const FreqTable & get_data () const
- Counters< Array_Type, Data_Counter_Type > * get_counters ()

Vector of couter functions.

- Rules< Array_Type, Data_Rule_Type > * get_rules ()

  Vector of static rules (cells to iterate).

- Rules< Array_Type, Data_Rule_Dyn_Type > * get_rules_dyn ()

  Vector of dynamic rules (to include/exclude a realizaton).

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_)
- void add_rule_dyn (Rule< Array_Type, Data_Rule_Dyn_Type > *f_)
- void add_rule_dyn (Rule< Array_Type, Data_Rule_Dyn_Type > f_)
- void set_rules_dyn (Rules< Array_Type, Data_Rule_Dyn_Type > *rules_)

Public Attributes

- uint N
- uint M
- bool delete_counters = true
- bool delete_rules = true
- bool delete_rules_dyn = true
- uint max_num_elements = BARRY_MAX_NUM_ELEMENTS
- 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

7.27.1 Detailed Description

template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
class Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_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.

The members `rule` and `rule_dyn` allow constraining the support. The first will establish which cells of the array will be used to iterate, for example, in the case of social networks, self-loops are not allowed, so the entire diagonal would be fixed to zero, reducing the size of the support.

In the case of `rule_dyn`, the function will stablish dynamically whether the current state will be included in the counts or not. For example, this set of rules can be used to constrain the support to networks that have a prescribed degree sequence.

Definition at line 35 of file support-bones.hpp.

7.27.2 Constructor & Destructor Documentation

7.27.2.1 Support() [1/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Support (
            const Array_Type & Array_ )  [inline]
```

Constructor passing a reference Array.

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

7.27.2.2 Support() [2/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Support (
            uint N_,
            uint M_ )  [inline]
```

Constructor specifying the dimensions of the array (empty).

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

7.27.2.3 Support() [3/3]

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

Definition at line 85 of file support-bones.hpp.

7.27.2.4 ~Support()

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

Definition at line 92 of file support-bones.hpp.

7.27.3 Member Function Documentation

7.27.3.1 add_counter() [1/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_counter
(
            Counter< Array_Type, Data_Counter_Type > * f_ )  [inline]
```

Definition at line 219 of file support-meat.hpp.

7.27.3.2 add_counter() [2/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_counter
(
            Counter< Array_Type, Data_Counter_Type > f_ )  [inline]
```

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

7.27.3.3 add_rule() [1/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > * f_ )  [inline]
```

Definition at line 256 of file support-meat.hpp.

7.27.3.4 add_rule() [2/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > f_ )  [inline]
```

Definition at line 266 of file support-meat.hpp.

7.27.3.5 add_rule_dyn() [1/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_↩
dyn (
            Rule< Array_Type, Data_Rule_Dyn_Type > * f_ )  [inline]
```

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

7.27.3.6 add_rule_dyn() [2/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_↩
dyn (
            Rule< Array_Type, Data_Rule_Dyn_Type > f_ )  [inline]
```

Definition at line 301 of file support-meat.hpp.

7.27.3.7 calc()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::calc (
            std::vector< Array_Type > * array_bank = nullptr,
            std::vector< std::vector< double > > * stats_bank = nullptr,
            unsigned int max_num_elements_ = 0u )  [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 193 of file support-meat.hpp.

7.27.3.8 get_counters()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
Counters< Array_Type, Data_Counter_Type > * Support< Array_Type, Data_Counter_Type, Data_↩
Rule_Type, Data_Rule_Dyn_Type >::get_counters  [inline]
```

Vector of couter functions.

Definition at line 364 of file support-meat.hpp.

7.27.3.9 get_counts()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
Counts_type Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >↩
::get_counts  [inline]
```

Definition at line 328 of file support-meat.hpp.

7.27.3.10 get_counts_ptr()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
const MapVec_type * Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type
>::get_counts_ptr  [inline]
```

Definition at line 335 of file support-meat.hpp.

7.27.3.11 get_current_stats()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
std::vector< double > * Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_↩
Dyn_Type >::get_current_stats  [inline]
```

List current statistics.

Definition at line 342 of file support-meat.hpp.

7.27.3.12 get_data()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
const FreqTable & Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type
>::get_data  [inline]
```

Definition at line 359 of file support-meat.hpp.

7.27.3.13 get_rules()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
Rules< Array_Type, Data_Rule_Type > * Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_rules  [inline]
```

Vector of static rules (cells to iterate).

Definition at line 369 of file support-meat.hpp.

7.27.3.14 get_rules_dyn()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
Rules< Array_Type, Data_Rule_Dyn_Type > * Support< Array_Type, Data_Counter_Type, Data_Rule↩
_Type, Data_Rule_Dyn_Type >::get_rules_dyn  [inline]
```

Vector of dynamic rules (to include/exclude a realizaton).

Definition at line 374 of file support-meat.hpp.

7.27.3.15 init_support()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_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.

7.27.3.16 print()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::print
[inline]
```

Definition at line 347 of file support-meat.hpp.

7.27.3.17 reset_array() [1/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::reset_array
[inline]
```

Definition at line 84 of file support-meat.hpp.

7.27.3.18 reset_array() [2/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::reset_array
(
            const Array_Type & Array_ )  [inline]
```

Definition at line 91 of file support-meat.hpp.

7.27.3.19 set_counters()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_↩
counters (
            Counters< Array_Type, Data_Counter_Type > * counters_ )  [inline]
```

Definition at line 239 of file support-meat.hpp.

7.27.3.20 set_rules()

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

Definition at line 276 of file support-meat.hpp.

7.27.3.21 set_rules_dyn()

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

Definition at line 311 of file support-meat.hpp.

7.27.4 Member Data Documentation

7.27.4.1 change_stats

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector< double > > Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::change_stats
```

Definition at line 65 of file support-bones.hpp.

7.27.4.2 coordinates_free

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::pair<uint,uint> > Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::coordinates_free
```

Definition at line 63 of file support-bones.hpp.

7.27.4.3 coordinates_locked

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::pair<uint,uint> > Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::coordinates_locked
```

Definition at line 64 of file support-bones.hpp.

7.27.4.4 current_stats

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< double > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn↩
_Type >::current_stats
```

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

7.27.4.5 delete_counters

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↩
counters = true
```

Definition at line 56 of file support-bones.hpp.

7.27.4.6 delete_rules

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↩
rules = true
```

Definition at line 57 of file support-bones.hpp.

7.27.4.7 delete_rules_dyn

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↩
rules_dyn = true
```

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

7.27.4.8 M

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::M
```

Definition at line 55 of file support-bones.hpp.

7.27.4.9 max_num_elements

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::max_num_↩
elements = BARRY_MAX_NUM_ELEMENTS
```

Definition at line 59 of file support-bones.hpp.

7.27.4.10 N

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::N
```

Definition at line 55 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

7.28 vecHasher< T > Struct Template Reference

```
#include <typedefs.hpp>
```

Public Member Functions

- std::size_t operator() (std::vector< T > const &dat) const noexcept

7.28.1 Detailed Description

template<typename T>
struct vecHasher< T >

Definition at line 86 of file typedefs.hpp.

7.28.2 Member Function Documentation

7.28.2.1 operator()()

```
template<typename T >
std::size_t vecHasher< T >::operator() (
            std::vector< T > const & dat ) const  [inline], [noexcept]
```

Definition at line 87 of file typedefs.hpp.

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

- include/barry/typedefs.hpp

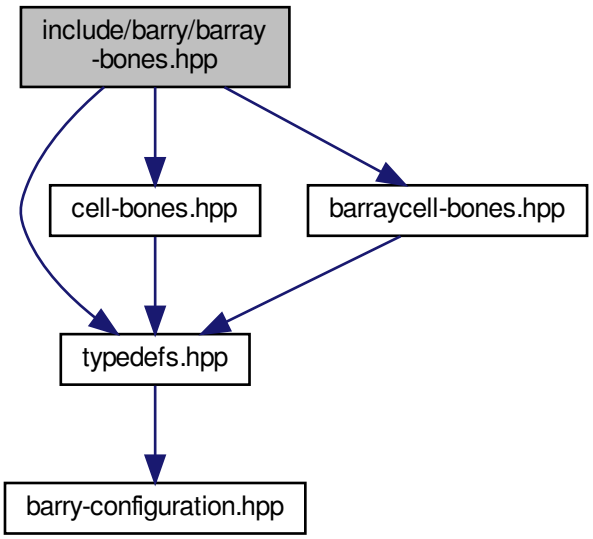
Chapter 8

File Documentation

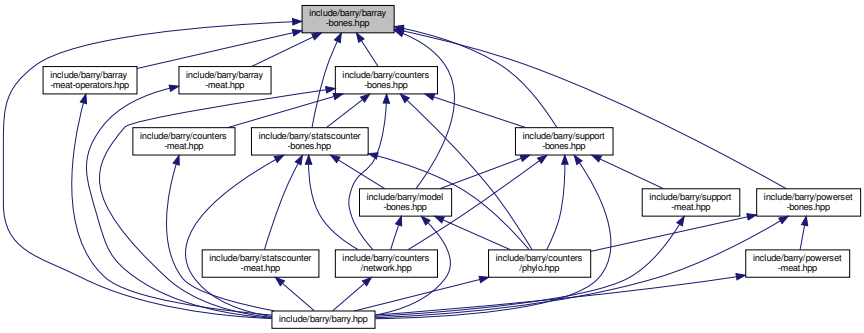
8.1 include/barry/barray-bones.hpp File Reference

```
#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

8.1.1 Macro Definition Documentation

8.1.1.1 BARRAY_BONES_HPP

```
#define BARRAY_BONES_HPP 1
```

Definition at line 8 of file barray-bones.hpp.

8.2 include/barry/barray-iterator.hpp File Reference

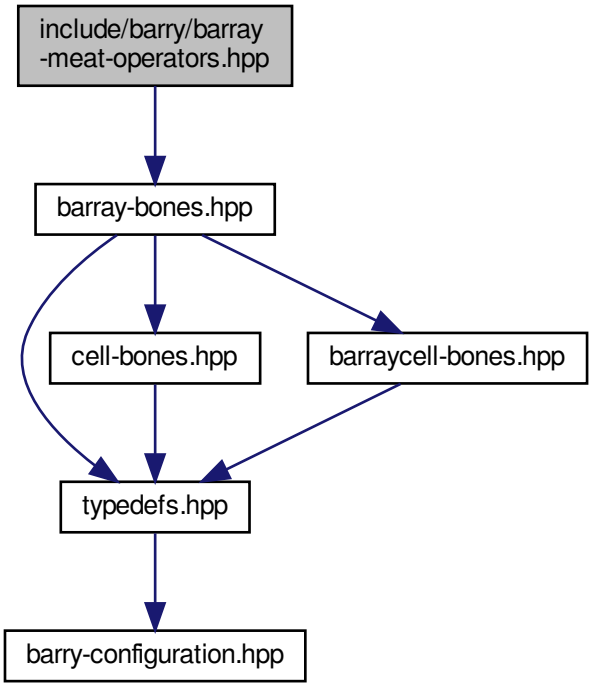
Classes

- class ConstBArrayRowIter< Cell_Type, Data_Type >

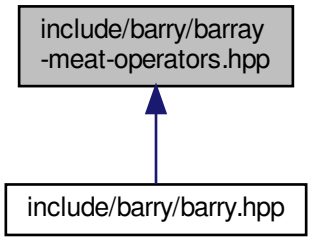
8.3 include/barry/barray-meat-operators.hpp File Reference

```
#include "barray-bones.hpp"
```

Include dependency graph for barray-meat-operators.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define BARRY_BARRAY_MEAT_OPERATORS_HPP 1
- #define ROW(a) this->el_ij[a]
- #define COL(a) this->el_ji[a]

Functions

- template<typename Cell_Type , typename Data_Type >
  void checkdim_ (const BArray< Cell_Type, Data_Type > &lhs, const BArray< Cell_Type, Data_Type > &rhs)

8.3.1 Macro Definition Documentation

8.3.1.1 BARRY_BARRAY_MEAT_OPERATORS_HPP

```
#define BARRY_BARRAY_MEAT_OPERATORS_HPP 1
```

Definition at line 5 of file barray-meat-operators.hpp.

8.3.1.2 COL

```
#define COL(
              a ) this->el_ji[a]
```

Definition at line 8 of file barray-meat-operators.hpp.

8.3.1.3 ROW

```
#define ROW(
              a ) this->el_ij[a]
```

Definition at line 7 of file barray-meat-operators.hpp.

8.3.2 Function Documentation

8.3.2.1 checkdim_()

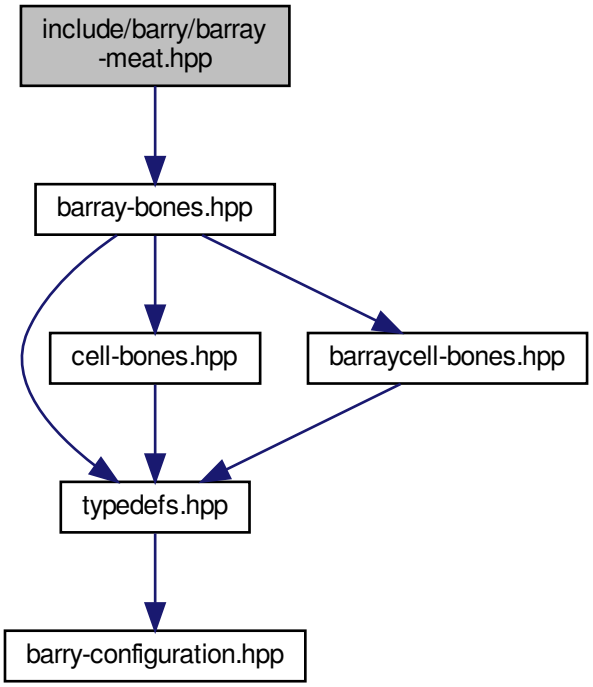
```
template<typename Cell_Type , typename Data_Type >
void checkdim_ (
            const BArray< Cell_Type, Data_Type > & lhs,
            const BArray< Cell_Type, Data_Type > & rhs )  [inline]
```

Definition at line 11 of file barray-meat-operators.hpp.

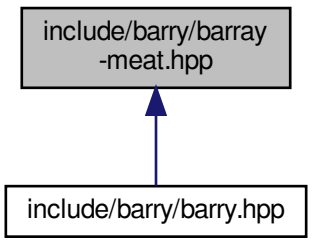
8.4 include/barry/barray-meat.hpp File Reference

```
#include "barray-bones.hpp"
```

Include dependency graph for barray-meat.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define ROW(a) this->el_ij[a]
- #define COL(a) this->el_ji[a]

8.4.1 Macro Definition Documentation

8.4.1.1 COL

```
#define COL(
              a ) this->el_ji[a]
```

Definition at line 8 of file barray-meat.hpp.

8.4.1.2 ROW

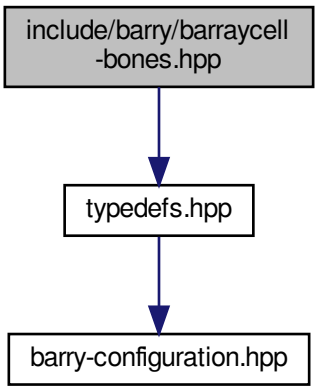
```
#define ROW(
              a ) this->el_ij[a]
```

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

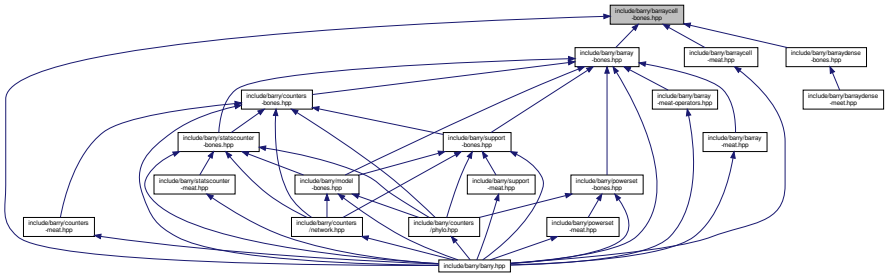
8.5 include/barry/barraycell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barraycell-bones.hpp:



This graph shows which files directly or indirectly include this file:



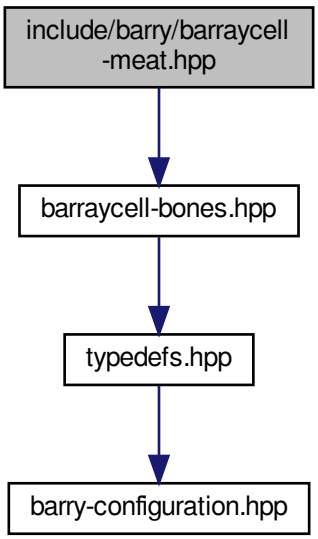
Classes

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

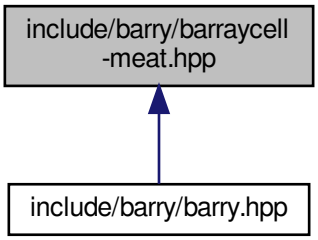
8.6 include/barry/barraycell-meat.hpp File Reference

```
#include "barraycell-bones.hpp"
```

Include dependency graph for barraycell-meat.hpp:



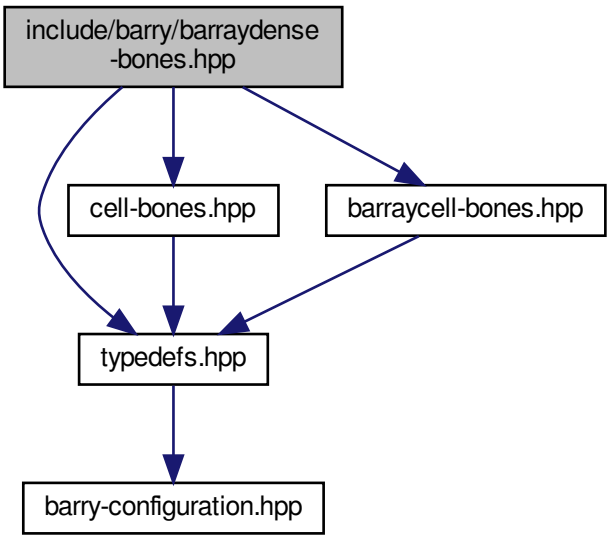
This graph shows which files directly or indirectly include this file:



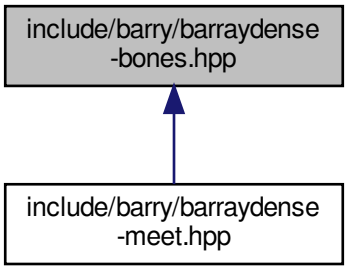
8.7 include/barry/barraydense-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "cell-bones.hpp"
#include "barraycell-bones.hpp"
```

Include dependency graph for barraydense-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

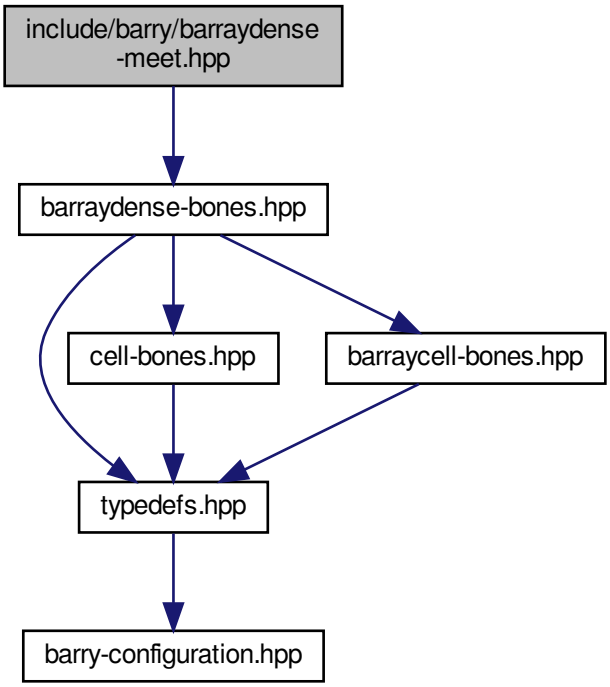
- class BArrayDense< Cell_Type, Data_Type >

  Baseline class for binary arrays.

8.8 include/barry/barraydense-meet.hpp File Reference

```
#include "barraydense-bones.hpp"
```

Include dependency graph for barraydense-meet.hpp:



Macros

- #define BARRY_BARRAYDENSE_MEAT_HPP
- #define ROW(a) this->el_ij[a]
- #define COL(a) this->el_ji[a]
- #define POS(a, b) (b)*N + (a)
- #define ZERO_CELL Cell< Cell_Type >(static_cast< Cell_Type >(0.0))

8.8.1 Macro Definition Documentation

8.8.1.1 BARRY_BARRAYDENSE_MEAT_HPP

```
#define BARRY_BARRAYDENSE_MEAT_HPP
```

Definition at line 5 of file barraydense-meet.hpp.

8.8.1.2 COL

```
#define COL(
              a ) this->el_ji[a]
```

Definition at line 8 of file barraydense-meet.hpp.

8.8.1.3 POS

```
#define POS(
              a,
              b ) (b)*N + (a)
```

Definition at line 9 of file barraydense-meet.hpp.

8.8.1.4 ROW

```
#define ROW(
              a ) this->el_ij[a]
```

Definition at line 7 of file barraydense-meet.hpp.

8.8.1.5 ZERO_CELL

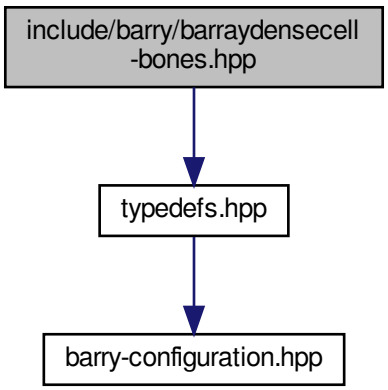
```
#define ZERO_CELL Cell< Cell_Type >(static_cast< Cell_Type >(0.0))
```

Definition at line 14 of file barraydense-meet.hpp.

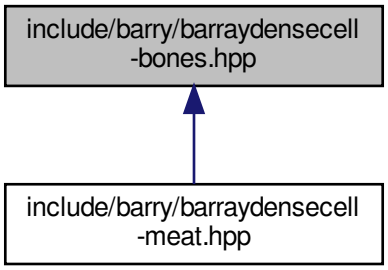
8.9 include/barry/barraydensecell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barraydensecell-bones.hpp:



This graph shows which files directly or indirectly include this file:



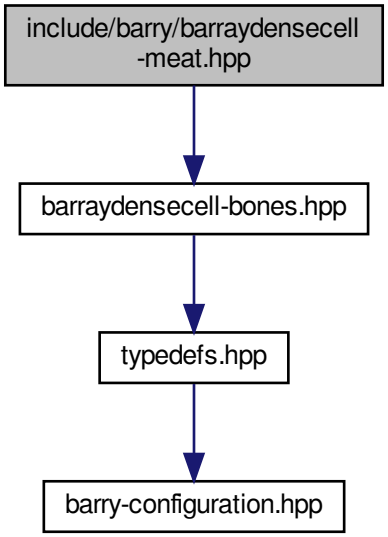
Classes

- class BArrayDenseCell< Cell_Type, Data_Type >
- class BArrayDenseCell_const< Cell_Type, Data_Type >

8.10 include/barry/barraydensecell-meat.hpp File Reference

```
#include "barraydensecell-bones.hpp"
```

Include dependency graph for barraydensecell-meat.hpp:



Macros

- #define BARRY_BARRAYDENSECELL_MEAT_HPP 1
- #define POS(a, b) (a) + (b) ∗ Array->N

8.10.1 Macro Definition Documentation

8.10.1.1 BARRY_BARRAYDENSECELL_MEAT_HPP

```
#define BARRY_BARRAYDENSECELL_MEAT_HPP 1
```

Definition at line 4 of file barraydensecell-meat.hpp.

8.10.1.2 POS

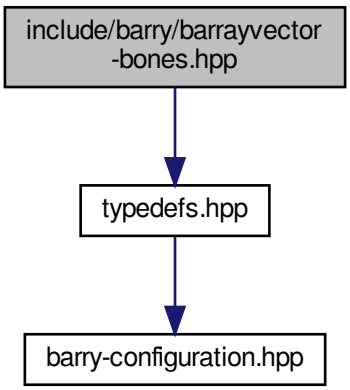
```
#define POS(
            a,
            b ) (a) + (b) * Array->N
```

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

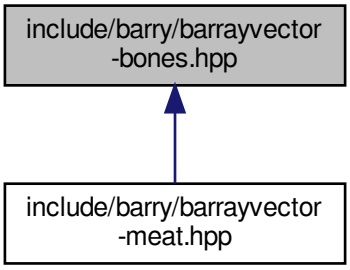
8.11 include/barry/barrayvector-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barrayvector-bones.hpp:



This graph shows which files directly or indirectly include this file:



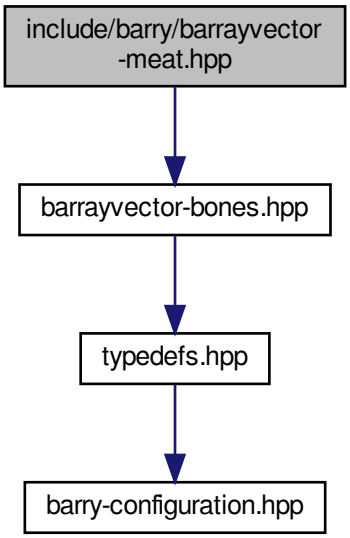
Classes

- class BArrayVector< Cell_Type, Data_Type >

  Row or column of a `BArray`
- class BArrayVector_const< Cell_Type, Data_Type >

8.12 include/barry/barrayvector-meat.hpp File Reference

`#include "barrayvector-bones.hpp"`
Include dependency graph for barrayvector-meat.hpp:



Macros

- #define BARRY_BARRAYVECTOR_MEAT_HPP 1

8.12.1 Macro Definition Documentation

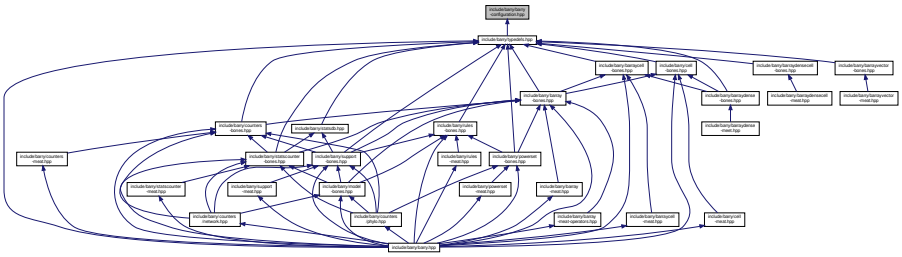
8.12.1.1 BARRY_BARRAYVECTOR_MEAT_HPP

```
#define BARRY_BARRAYVECTOR_MEAT_HPP 1
```

Definition at line 4 of file barrayvector-meat.hpp.

8.13 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_USE_ISFINITE` When specified, it will introduce a macro that checks whether the likelihood is finite or not.
- `printf_barry` If not specified, will be defined as `printf`.

- #define BARRY_SAFE_EXP -100.0
- #define BARRY_ISFINITE(a)
- #define BARRY_CHECK_SUPPORT(x, maxs)
- #define printf_barry printf
- #define BARRY_MAX_NUM_ELEMENTS static_cast< unsigned int >(UINT_MAX/2u)
- template<typename Ta , typename Tb >
  using Map = std::map< Ta, Tb >

8.13.1 Macro Definition Documentation

8.13.1.1 BARRY_CHECK_SUPPORT

```
#define BARRY_CHECK_SUPPORT(
              x,
              maxs )
```

Definition at line 45 of file barry-configuration.hpp.

8.13.1.2 BARRY_ISFINITE

```
#define BARRY_ISFINITE(
              a )
```

Definition at line 38 of file barry-configuration.hpp.

8.13.1.3 BARRY_MAX_NUM_ELEMENTS

```
#define BARRY_MAX_NUM_ELEMENTS static_cast< unsigned int >(UINT_MAX/2u)
```

Definition at line 53 of file barry-configuration.hpp.

8.13.1.4 BARRY_SAFE_EXP

```
#define BARRY_SAFE_EXP -100.0
```

Definition at line 31 of file barry-configuration.hpp.

8.13.1.5 printf_barry

```
#define printf_barry printf
```

Definition at line 49 of file barry-configuration.hpp.

8.13.2 Typedef Documentation

8.13.2.1 Map

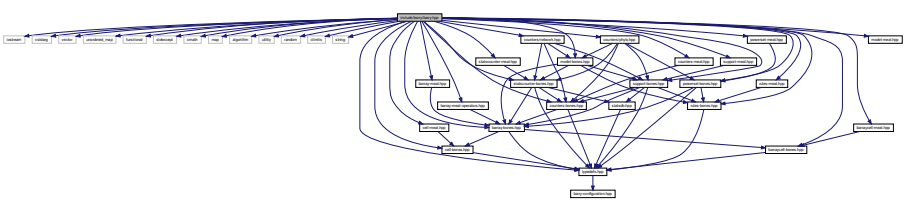
```
template<typename Ta , typename Tb >
using Map = std::map<Ta,Tb>
```

Definition at line 25 of file barry-configuration.hpp.

8.14 include/barry/barry.hpp File Reference

```
#include <iostream>
#include <cstdarg>
#include <vector>
#include <unordered_map>
#include <functional>
#include <stdexcept>
#include <cmath>
#include <map>
#include <algorithm>
#include <utility>
#include <random>
#include <climits>
#include <string>
#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:



Namespaces

- barry

  barry: Your go-to motif accountant
- barry::counters

  Tree class and TreeIterator class.
- barry::counters::network
- barry::counters::phylo

Macros

- #define BARRY_HPP
- #define BARRY_VERSION 0.1
- #define COUNTER_FUNCTION(a)
- #define COUNTER_LAMBDA(a)
- #define RULE_FUNCTION(a)
- #define RULE_LAMBDA(a)

8.14.1 Macro Definition Documentation

8.14.1.1 BARRY_HPP

```
#define BARRY_HPP
```

Definition at line 20 of file barry.hpp.

8.14.1.2 BARRY_VERSION

```
#define BARRY_VERSION 0.1
```

Definition at line 22 of file barry.hpp.

8.14.1.3 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 73 of file barry.hpp.

8.14.1.4 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 76 of file barry.hpp.

8.14.1.5 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 80 of file barry.hpp.

8.14.1.6 RULE_LAMBDA

```
#define RULE_LAMBDA(
                a )
```

Value:

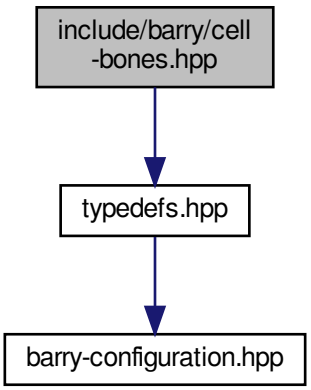
```
    template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
    Rule_fun_type<Array_Type, Data_Type> a = \
    [](const Array_Type & Array, uint i, uint j, Data_Type * data)
```

Definition at line 83 of file barry.hpp.

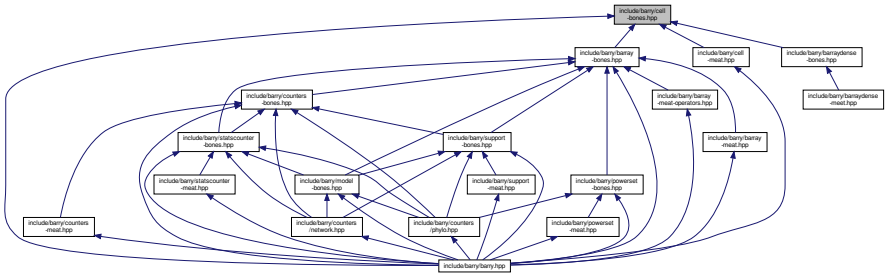
8.15 include/barry/cell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for cell-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

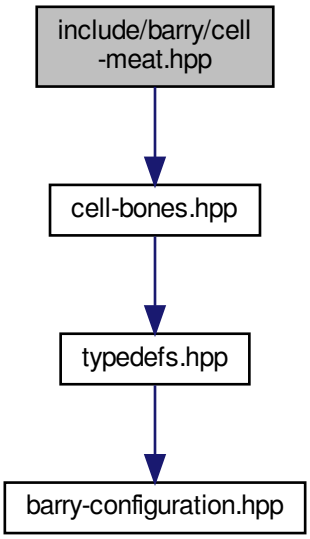
- class Cell< Cell_Type >

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

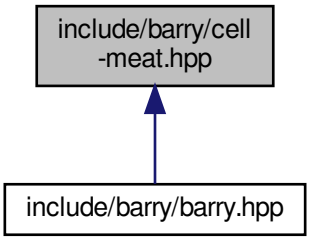
8.16 include/barry/cell-meat.hpp File Reference

```
#include "cell-bones.hpp"
```

Include dependency graph for cell-meat.hpp:



This graph shows which files directly or indirectly include this file:

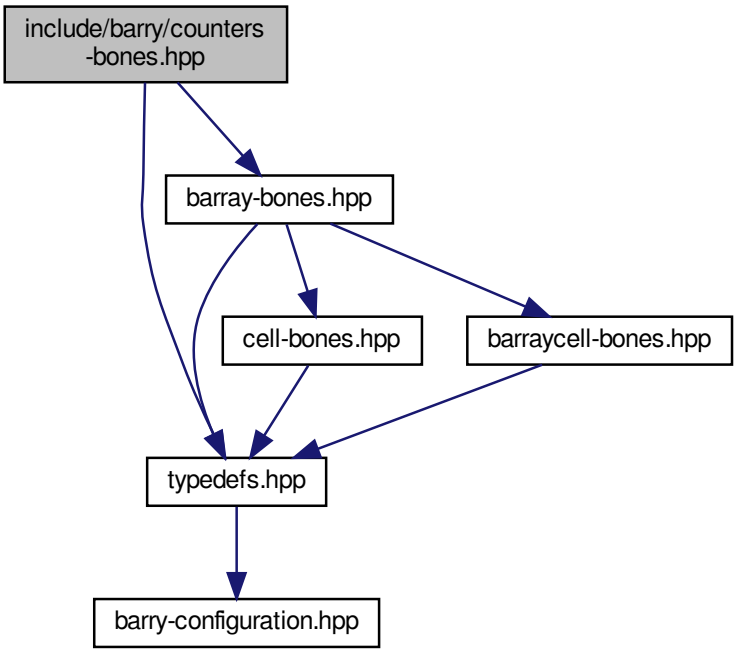


8.17 include/barry/col-bones.hpp File Reference

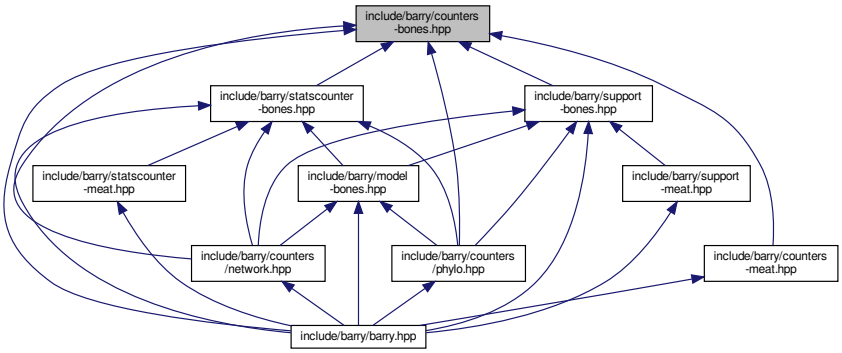
8.18 include/barry/counters-bones.hpp File Reference

```
#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 >

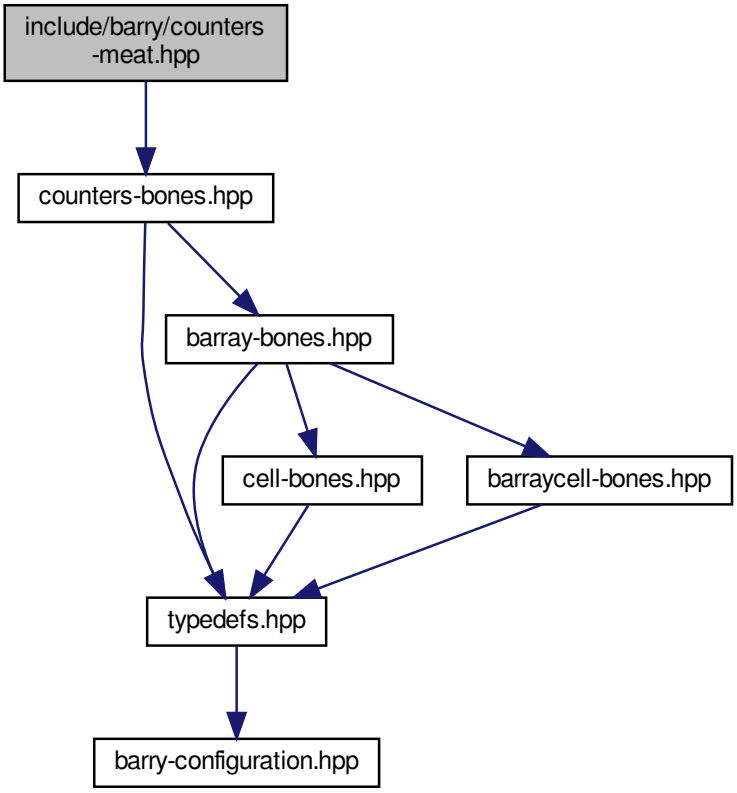
  A counter function based on change statistics.
- class Counters< Array_Type, Data_Type >

  Vector of counters.

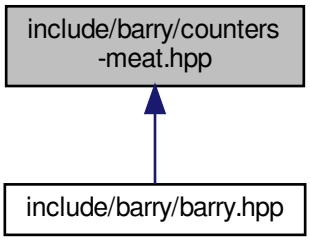
8.19 include/barry/counters-meat.hpp File Reference

```
#include "counters-bones.hpp"
```

Include dependency graph for counters-meat.hpp:



This graph shows which files directly or indirectly include this file:

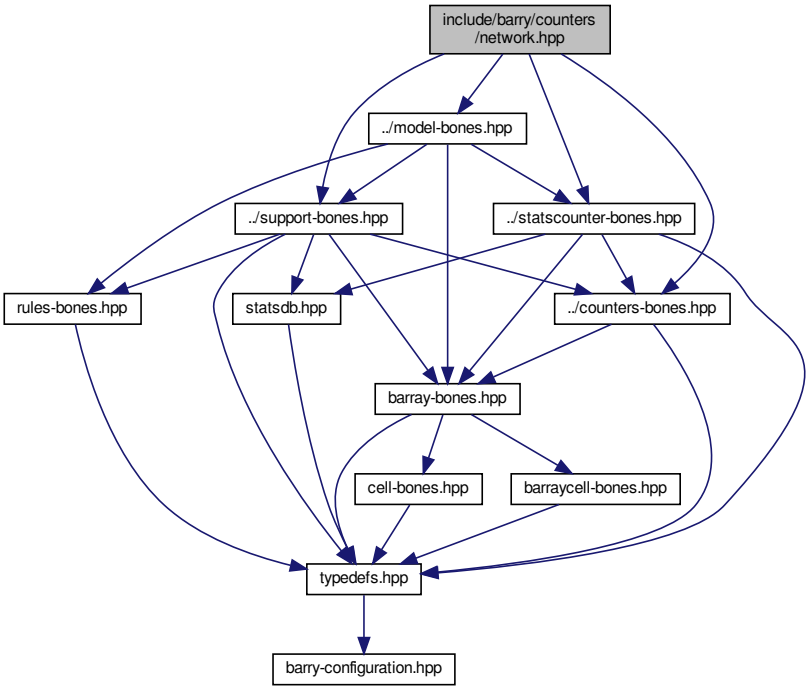


8.20 include/barry/counters/network.hpp File Reference

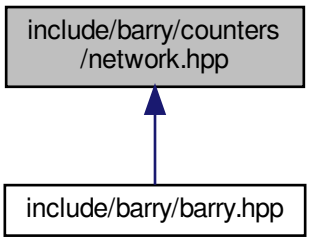
```
#include "../counters-bones.hpp"
#include "../support-bones.hpp"
```

```
#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

- 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.

8.20.1 Macro Definition Documentation

8.20.1.1 NET_C_DATA_IDX

```
#define NET_C_DATA_IDX(
              i ) (data->indices[i])
```

Definition at line 79 of file network.hpp.

8.20.1.2 NET_C_DATA_NUM

```
#define NET_C_DATA_NUM(
              i ) (data->numbers[i])
```

Definition at line 80 of file network.hpp.

8.20.1.3 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 101 of file network.hpp.

8.20.1.4 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 104 of file network.hpp.

8.20.1.5 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 113 of file network.hpp.

8.20.1.6 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 116 of file network.hpp.

8.20.2 Typedef Documentation

8.20.2.1 NetCounter

```
typedef Counter<Network, NetCounterData > NetCounter
```

Definition at line 88 of file network.hpp.

8.20.2.2 NetCounters

```
typedef Counters< Network, NetCounterData> NetCounters
```

Definition at line 89 of file network.hpp.

8.20.2.3 NetModel

```
typedef Model<Network, NetCounterData> NetModel
```

Definition at line 92 of file network.hpp.

8.20.2.4 NetRule

```
typedef Rule<Network,bool> NetRule
```

Definition at line 93 of file network.hpp.

8.20.2.5 NetRules

```
typedef Rules<Network,bool> NetRules
```

Definition at line 94 of file network.hpp.

8.20.2.6 NetStatsCounter

```
typedef StatsCounter<Network, NetCounterData> NetStatsCounter
```

Definition at line 91 of file network.hpp.

8.20.2.7 NetSupport

```
typedef Support<Network, NetCounterData > NetSupport
```

Definition at line 90 of file network.hpp.

8.20.2.8 Network

```
typedef BArray<double, NetworkData> Network
```

Definition at line 87 of file network.hpp.

8.20.3 Function Documentation

8.20.3.1 rules_zerodiag()

```
void rules_zerodiag (
            NetRules * rules )  [inline]
```

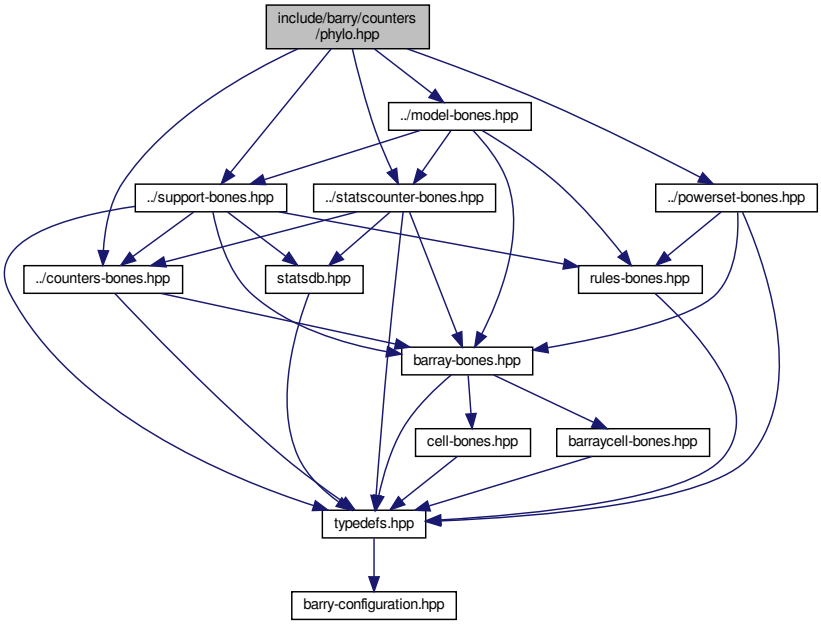
Number of edges.

Definition at line 742 of file network.hpp.

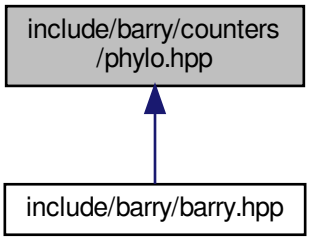
8.21 include/barry/counters/phylo.hpp File Reference

```
#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.
- class PhyloRuleDynData

Macros

- #define PHYLO_COUNTER_LAMBDA(a)

  Extension of a simple counter.
- #define PHYLO_RULE_DYN_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 Rule< PhyloArray, PhyloRuleDynData > PhyloRuleDyn
- typedef Rules< PhyloArray, PhyloRuleDynData > PhyloRulesDyn
- typedef Support< PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport
- typedef StatsCounter< PhyloArray, PhyloCounterData > PhyloStatsCounter
- typedef Model< PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel
- typedef PowerSet< PhyloArray, PhyloRuleData > PhyloPowerSet

Functions

- std::string get_last_name (bool d)
- 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)

  k genes gain 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_prop_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.
- void rule_dyn_limit_changes (PhyloSupport *support, uint pos, uint lb, uint ub, bool duplication=true)

  Overall functional gains.

8.21.1 Macro Definition Documentation

8.21.1.1 PHYLO_CHECK_MISSING

```
#define PHYLO_CHECK_MISSING( )
```

Value:

```
    if (Array.D() == nullptr) \
    throw std::logic_error("The array data is nullptr."); \
    if (data == nullptr) \
    throw std::logic_error("The counter/rule data is nullptr.")
```

Definition at line 94 of file phylo.hpp.

8.21.1.2 PHYLO_COUNTER_LAMBDA

```
#define PHYLO_COUNTER_LAMBDA(
              a )
```

Value:

```
Counter_fun_type<PhyloArray, PhyloCounterData> 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 88 of file phylo.hpp.

8.21.1.3 PHYLO_RULE_DYN_LAMBDA

```
#define PHYLO_RULE_DYN_LAMBDA(
              a )
```

Value:

```
Rule_fun_type<PhyloArray, PhyloRuleDynData> a = \
[](const PhyloArray & Array, uint i, uint j, PhyloRuleDynData * data)
```

Definition at line 91 of file phylo.hpp.

8.21.2 Typedef Documentation

8.21.2.1 PhyloArray

```
typedef BArray<uint, NodeData> PhyloArray
```

Definition at line 61 of file phylo.hpp.

8.21.2.2 PhyloCounter

```
typedef Counter<PhyloArray, PhyloCounterData > PhyloCounter
```

Definition at line 62 of file phylo.hpp.

8.21.2.3 PhyloCounterData

```
typedef std::vector< uint > PhyloCounterData
```

Definition at line 53 of file phylo.hpp.

8.21.2.4 PhyloCounters

```
typedef Counters< PhyloArray, PhyloCounterData> PhyloCounters
```

Definition at line 63 of file phylo.hpp.

8.21.2.5 PhyloModel

```
typedef Model<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel
```

Definition at line 73 of file phylo.hpp.

8.21.2.6 PhyloPowerSet

```
typedef PowerSet<PhyloArray, PhyloRuleData> PhyloPowerSet
```

Definition at line 74 of file phylo.hpp.

8.21.2.7 PhyloRule

```
typedef Rule<PhyloArray,PhyloRuleData> PhyloRule
```

Definition at line 65 of file phylo.hpp.

8.21.2.8 PhyloRuleData

```
typedef std::vector< std::pair< uint, uint > > PhyloRuleData
```

Definition at line 54 of file phylo.hpp.

8.21.2.9 PhyloRuleDyn

```
typedef Rule<PhyloArray,PhyloRuleDynData> PhyloRuleDyn
```

Definition at line 68 of file phylo.hpp.

8.21.2.10 PhyloRules

```
typedef Rules<PhyloArray,PhyloRuleData> PhyloRules
```

Definition at line 66 of file phylo.hpp.

8.21.2.11 PhyloRulesDyn

```
typedef Rules<PhyloArray,PhyloRuleDynData> PhyloRulesDyn
```

Definition at line 69 of file phylo.hpp.

8.21.2.12 PhyloStatsCounter

```
typedef StatsCounter<PhyloArray, PhyloCounterData> PhyloStatsCounter
```

Definition at line 72 of file phylo.hpp.

8.21.2.13 PhyloSupport

```
typedef Support<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport
```

Definition at line 71 of file phylo.hpp.

8.21.3 Function Documentation

8.21.3.1 get_last_name()

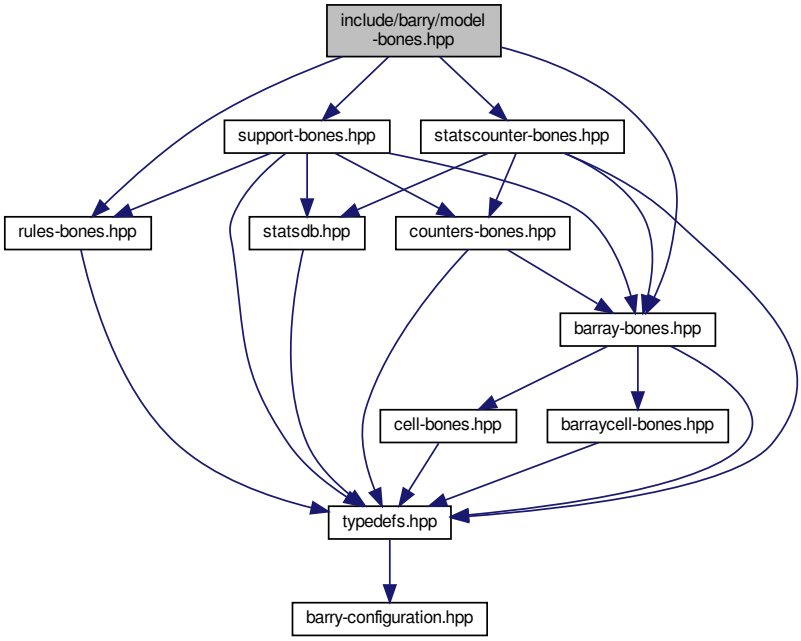
```
std::string get_last_name (
            bool d )  [inline]
```

Definition at line 99 of file phylo.hpp.

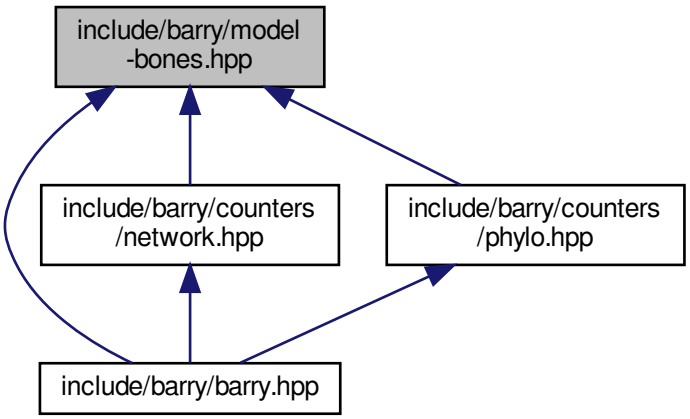
8.22 include/barry/model-bones.hpp File Reference

```
#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, Data_Rule_Dyn_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)

8.22.1 Function Documentation

8.22.1.1 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 69 of file model-bones.hpp.

8.22.1.2 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 40 of file model-bones.hpp.

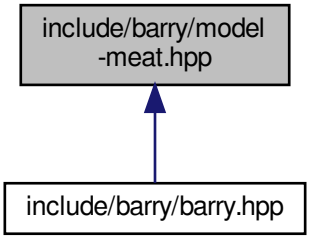
8.22.1.3 update_normalizing_constant()

```
double update_normalizing_constant (
            const std::vector< double > & params,
            const Counts_type & support )  [inline]
```

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

8.23 include/barry/model-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



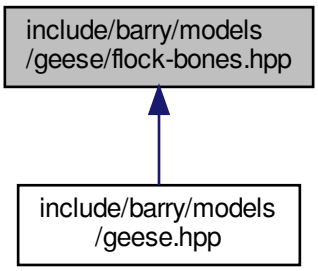
8.24 include/barry/models/geese.hpp File Reference

```
#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/geese-meat-predict_exhaust.hpp"
#include "geese/geese-meat-predict_sim.hpp"
#include "geese/flock-bones.hpp"
#include "geese/flock-meet.hpp"
```

Include dependency graph for geese.hpp:

8.25 include/barry/models/geese/flock-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



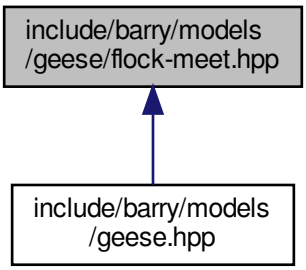
Classes

- class Flock

  A Flock is a group of Geese.

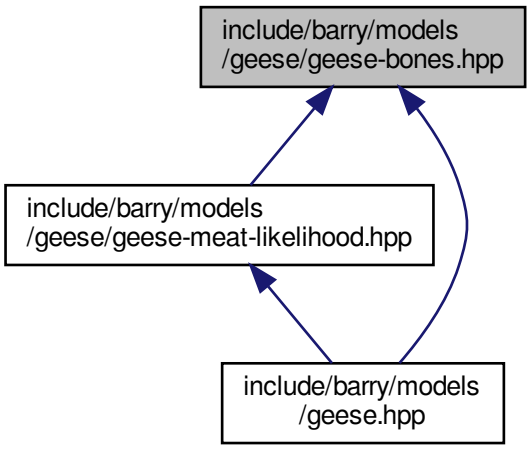
8.26 include/barry/models/geese/flock-meet.hpp File Reference

This graph shows which files directly or indirectly include this file:



8.27 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)

8.27.1 Macro Definition Documentation

8.27.1.1 INITIALIZED

```
#define INITIALIZED( )
```

Value:

```
    if (!this->initialized) \
    throw std::logic_error("The model has not been initialized yet.");
```

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

8.27.2 Function Documentation

8.27.2.1 keygen_full()

```
std::vector< double > keygen_full (
             const phylocounters::PhyloArray & array )  [inline]
```

Definition at line 35 of file geese-bones.hpp.

8.27.2.2 RULE_FUNCTION()

```
RULE_FUNCTION (
             rule_empty_free  )
```

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

8.27.2.3 vec_diff()

```
bool vec_diff (
             const std::vector< unsigned int > & s,
             const std::vector< unsigned int > & a )  [inline]
```

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

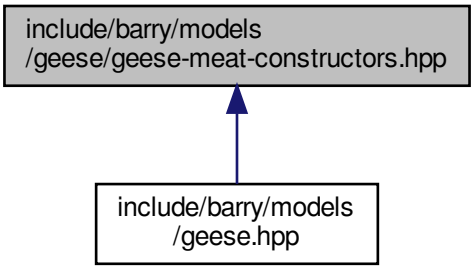
8.27.2.4 vector_caster()

```
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.

8.28 include/barry/models/geese/geese-meat-constructors.hpp File Reference

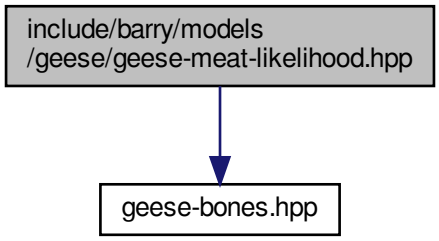
This graph shows which files directly or indirectly include this file:



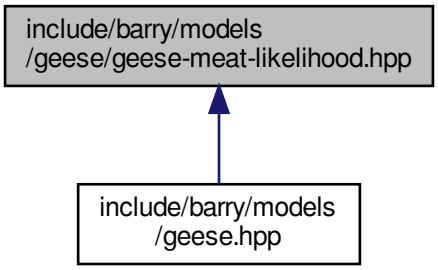
8.29 include/barry/models/geese/geese-meat-likelihood.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat-likelihood.hpp:

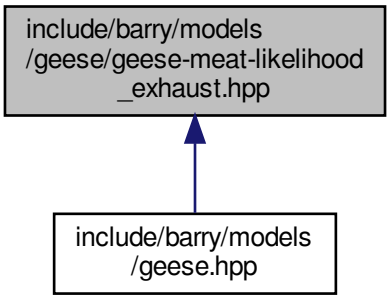


This graph shows which files directly or indirectly include this file:



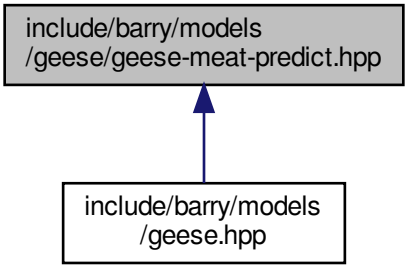
8.30 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference

This graph shows which files directly or indirectly include this file:



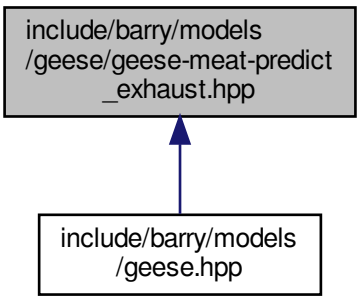
8.31 include/barry/models/geese/geese-meat-predict.hpp File Reference

This graph shows which files directly or indirectly include this file:



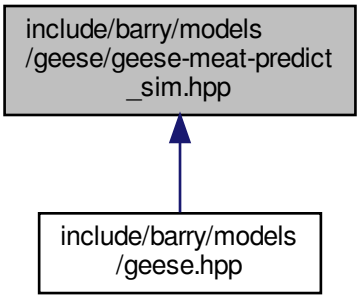
8.32 include/barry/models/geese/geese-meat-predict_exhaust.hpp File Reference

This graph shows which files directly or indirectly include this file:



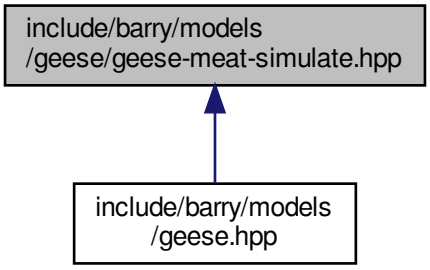
8.33 include/barry/models/geese/geese-meat-predict_sim.hpp File Reference

This graph shows which files directly or indirectly include this file:



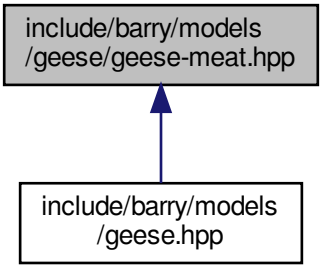
8.34 include/barry/models/geese/geese-meat-simulate.hpp File Reference

This graph shows which files directly or indirectly include this file:



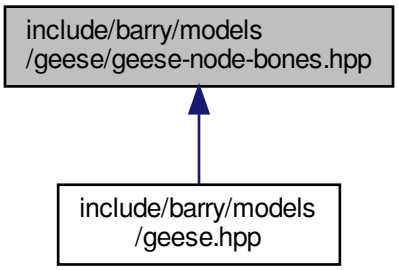
8.35 include/barry/models/geese/geese-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



8.36 include/barry/models/geese/geese-node-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

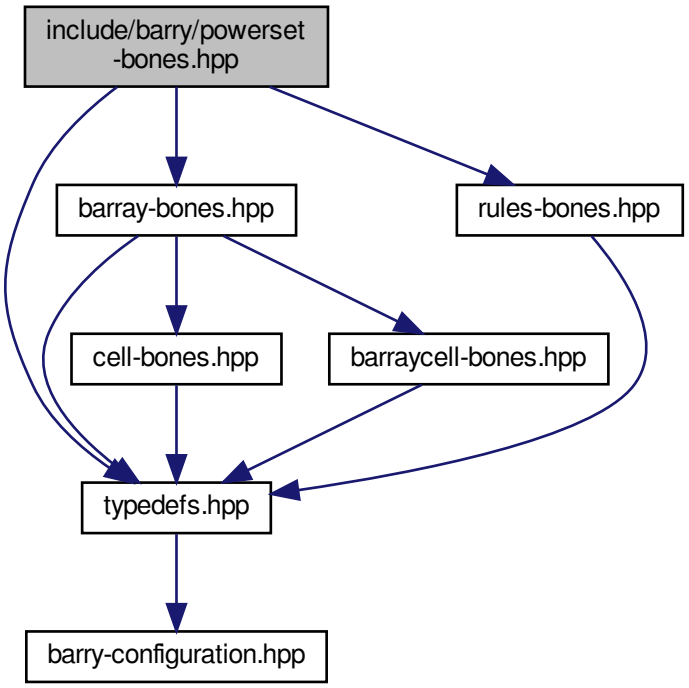
- class Node

  A single node for the model.

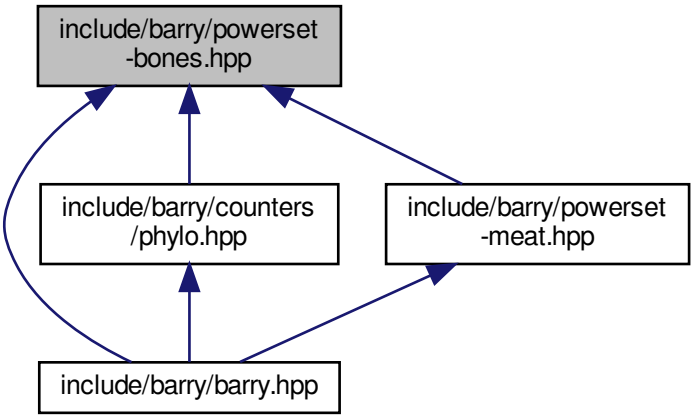
8.37 include/barry/powerset-bones.hpp File Reference

```
#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

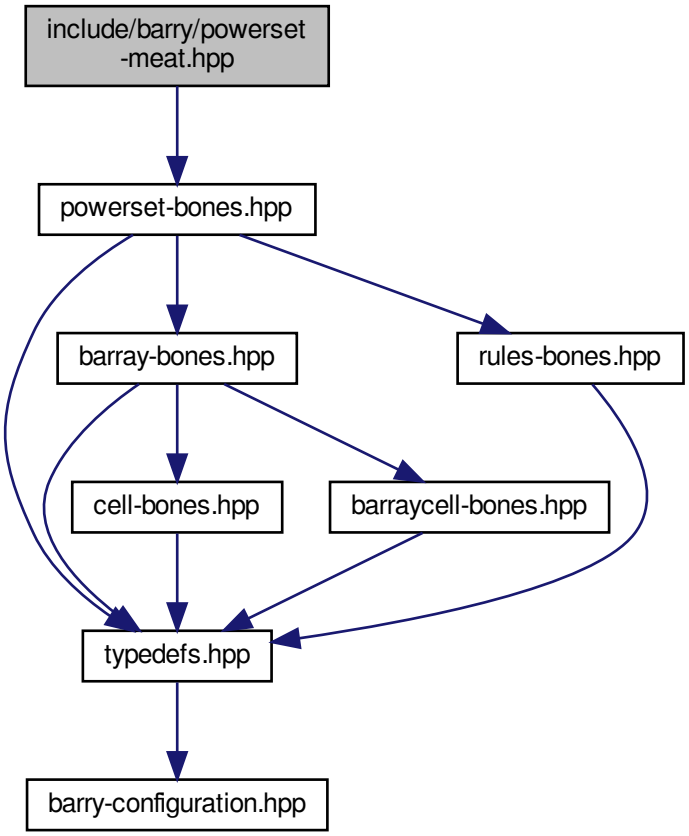
- class PowerSet< Array_Type, Data_Rule_Type >

  Powerset of a binary array.

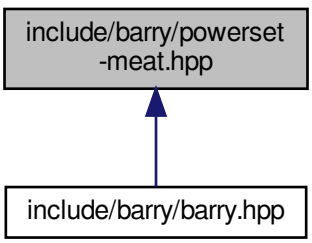
8.38 include/barry/powerset-meat.hpp File Reference

```
#include "powerset-bones.hpp"
```

Include dependency graph for powerset-meat.hpp:



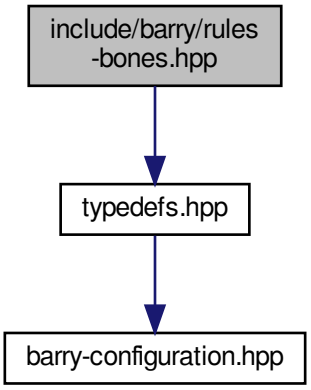
This graph shows which files directly or indirectly include this file:



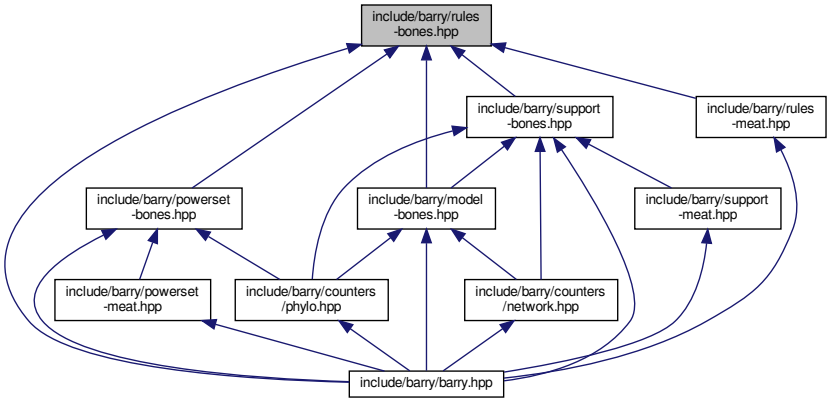
8.39 include/barry/rules-bones.hpp File Reference

```
#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)

8.39.1 Function Documentation

8.39.1.1 rule_fun_default()

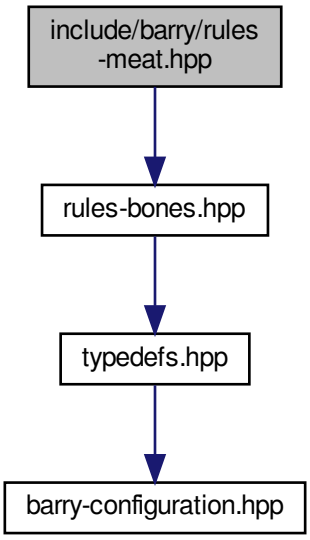
```
template<typename Array_Type , typename Data_Type >
bool rule_fun_default (
            const Array_Type * array,
            uint i,
            uint j,
            Data_Type * dat )
```

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

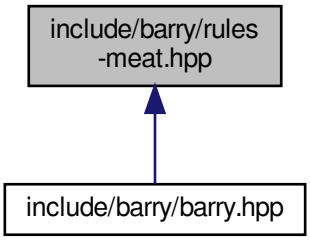
8.40 include/barry/rules-meat.hpp File Reference

```
#include "rules-bones.hpp"
```

Include dependency graph for rules-meat.hpp:



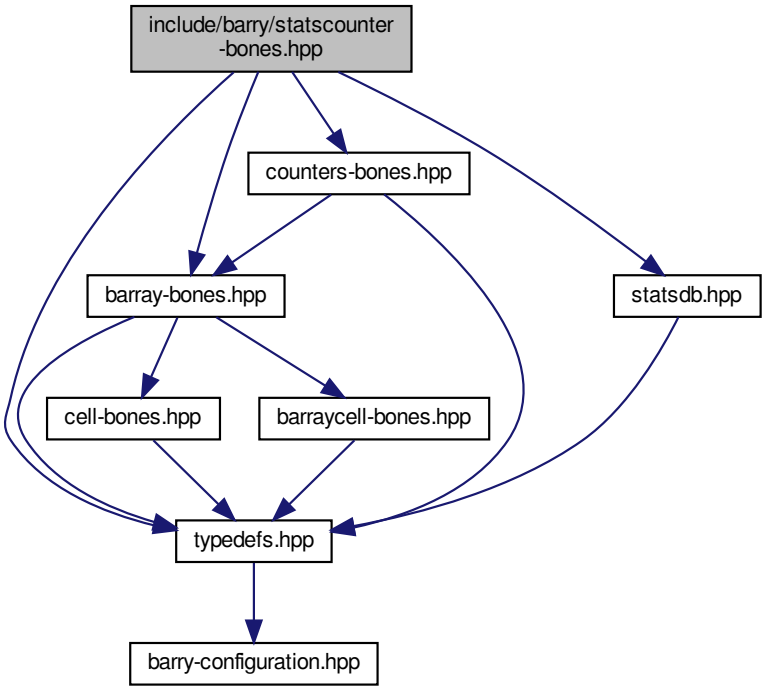
This graph shows which files directly or indirectly include this file:



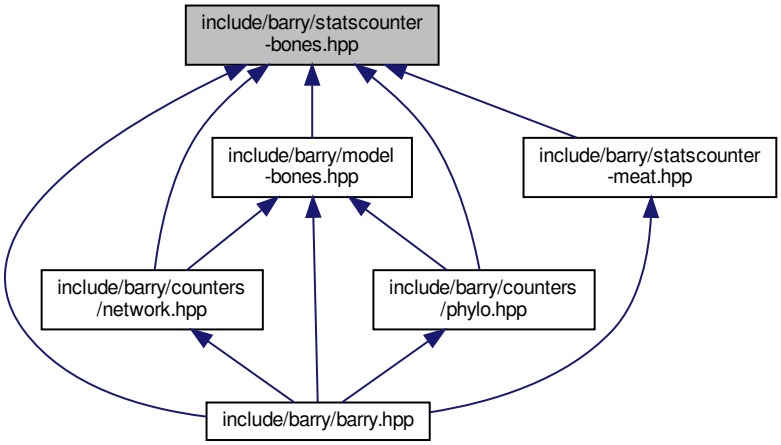
8.41 include/barry/statscounter-bones.hpp File Reference

```
#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

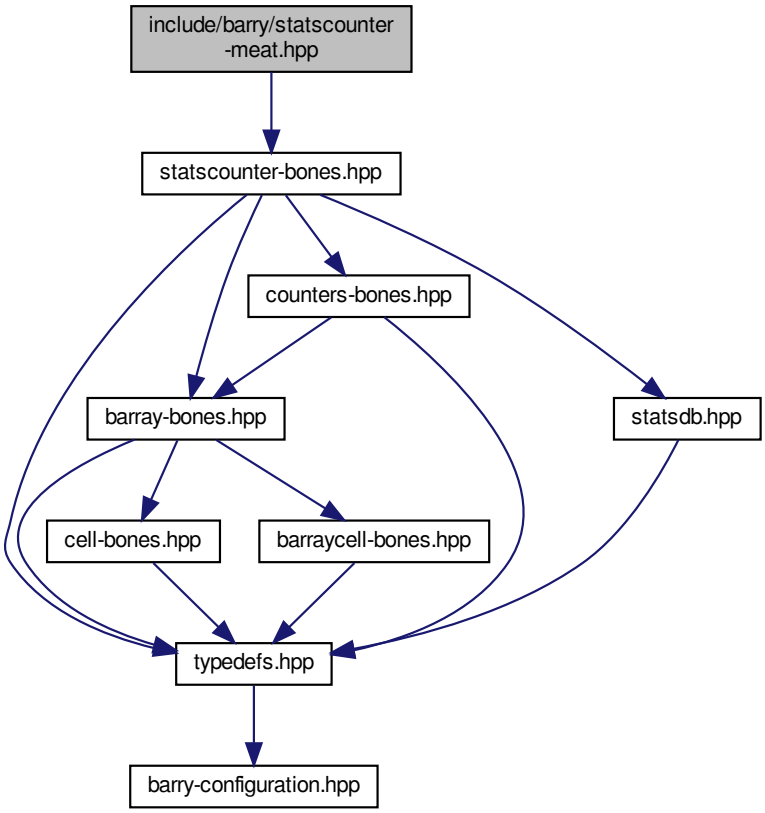
- class StatsCounter< Array_Type, Data_Type >

  Count stats for a single Array.

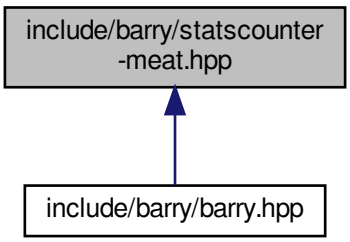
8.42 include/barry/statscounter-meat.hpp File Reference

```
#include "statscounter-bones.hpp"
```

Include dependency graph for statscounter-meat.hpp:



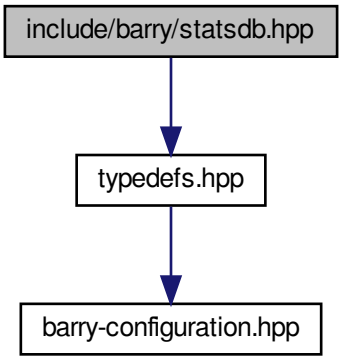
This graph shows which files directly or indirectly include this file:



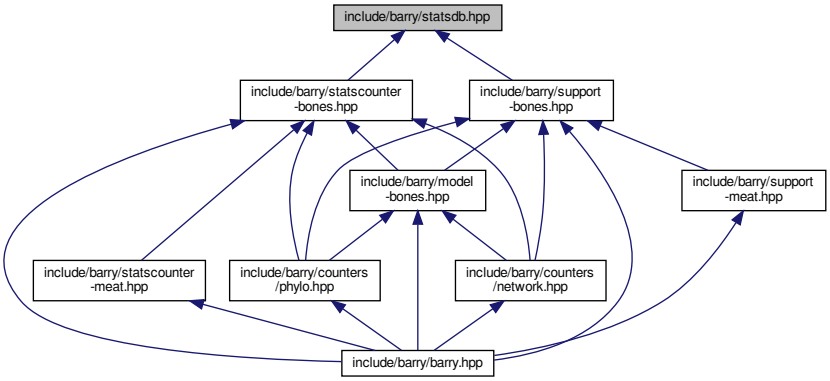
8.43 include/barry/statsdb.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for statsdb.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class FreqTable< T >

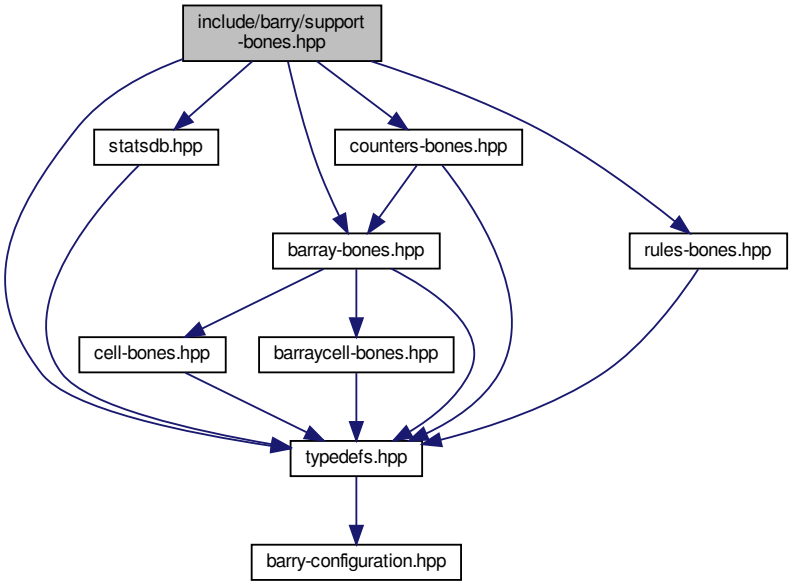
  Database of statistics.

8.44 include/barry/support-bones.hpp File Reference

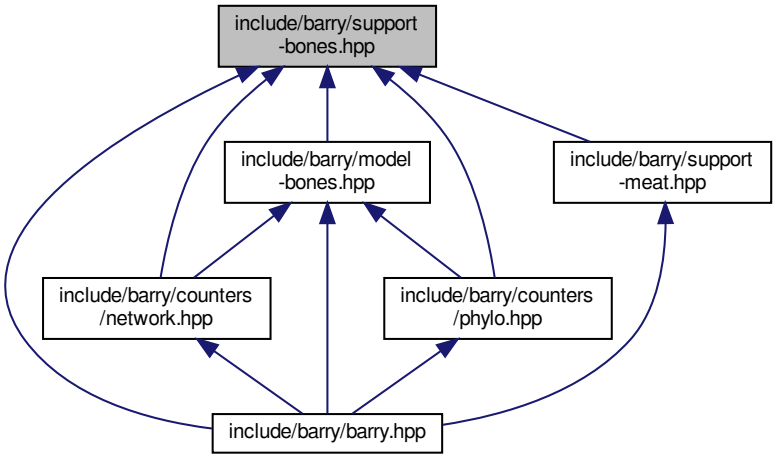
```
#include "typedefs.hpp"
#include "barray-bones.hpp"
#include "statsdb.hpp"
#include "counters-bones.hpp"
```

```
#include "rules-bones.hpp"
```

Include dependency graph for support-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

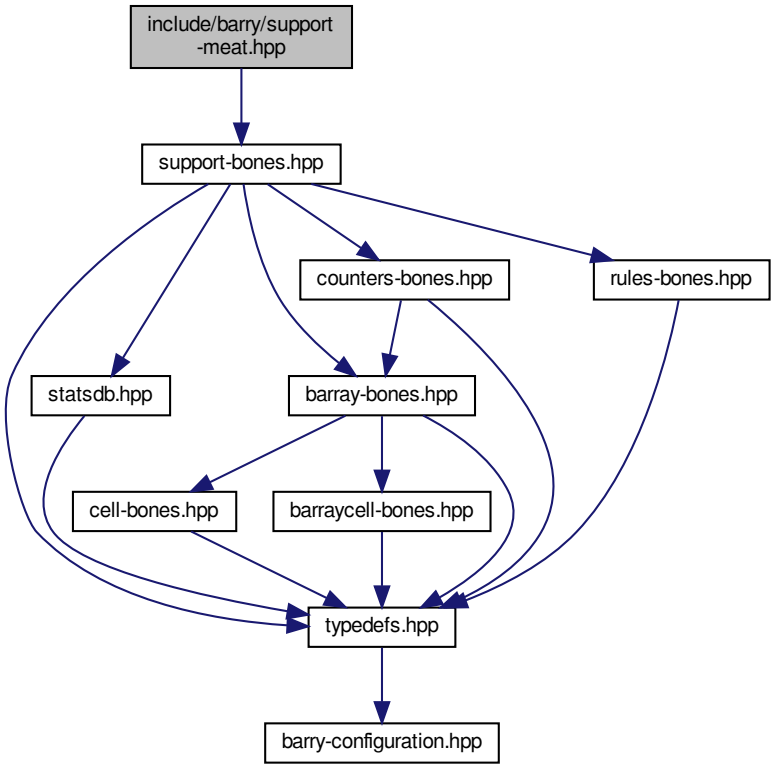
- class Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >

  Compute the support of sufficient statistics.

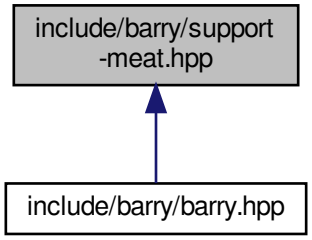
8.45 include/barry/support-meat.hpp File Reference

```
#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

8.45.1 Macro Definition Documentation

8.45.1.1 BARRY_SUPPORT_MEAT_HPP

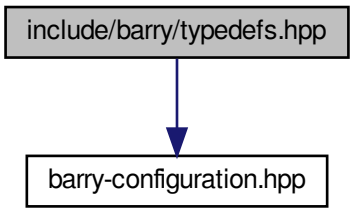
```
#define BARRY_SUPPORT_MEAT_HPP 1
```

Definition at line 4 of file support-meat.hpp.

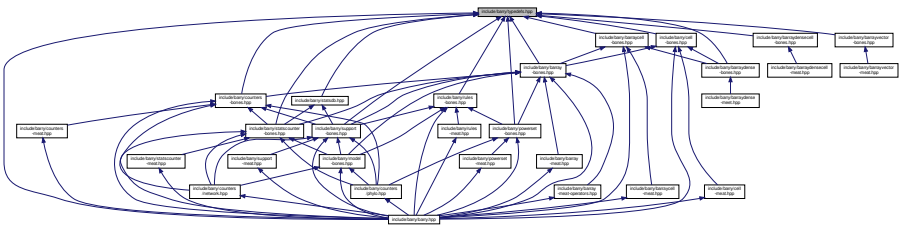
8.46 include/barry/typedefs.hpp File Reference

```
#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.

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

8.46.1 Typedef Documentation

8.46.1.1 Col_type

```
template<typename Cell_Type >
using Col_type = Map< uint, Cell<Cell_Type>* >
```

Definition at line 51 of file typedefs.hpp.

8.46.1.2 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 123 of file typedefs.hpp.

8.46.1.3 Counts_type

```
typedef std::vector< std::pair< std::vector<double>, uint > > Counts_type
```

Definition at line 44 of file typedefs.hpp.

8.46.1.4 MapVec_type

```
template<typename Ta = double, typename Tb = uint>
using MapVec_type = std::unordered_map< std::vector< Ta >, Tb, vecHasher<Ta> >
```

Definition at line 105 of file typedefs.hpp.

8.46.1.5 Row_type

```
template<typename Cell_Type >
using Row_type = Map< uint, Cell<Cell_Type> >
```

Definition at line 48 of file typedefs.hpp.

8.46.1.6 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 126 of file typedefs.hpp.

8.46.1.7 uint

```
typedef unsigned int uint
```

Definition at line 10 of file typedefs.hpp.

8.46.2 Function Documentation

8.46.2.1 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 137 of file typedefs.hpp.

8.46.2.2 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 155 of file typedefs.hpp.

8.46.2.3 vec_inner_prod()

```
template<typename T >
T vec_inner_prod (
             const std::vector< T > & a,
             const std::vector< T > & b )  [inline]
```

Definition at line 175 of file typedefs.hpp.

8.47 README.md File Reference

Index