

barry: Your go-to motif accountant

0.0-1

Generated by Doxygen 1.9.1

1 Main Page	1
2 Module Index	3
2.1 Modules	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Module Documentation	9
5.1 Counting	9
5.1.1 Detailed Description	9
5.2 Statistical Models	9
5.2.1 Detailed Description	10
5.3 Network counters	10
5.3.1 Detailed Description	10
5.3.2 Function Documentation	11
5.3.2.1 counter_absdiff()	11
5.3.2.2 counter_ctriads()	11
5.3.2.3 counter_degree()	11
5.3.2.4 counter_density()	11
5.3.2.5 counter_diff()	12
5.3.2.6 counter_edges()	12
5.3.2.7 counter_idegree()	12
5.3.2.8 counter_idegree15()	12
5.3.2.9 counter_isolates()	12
5.3.2.10 counter_istar2()	13
5.3.2.11 counter_mutual()	13
5.3.2.12 counter_nodecov()	13
5.3.2.13 counter_nodeicov()	13
5.3.2.14 counter_nodematch()	13
5.3.2.15 counter_nodeocov()	14
5.3.2.16 counter_odegree()	14
5.3.2.17 counter_odegree15()	14
5.3.2.18 counter_ostar2()	14
5.3.2.19 counter_ttriads()	14
5.3.2.20 NETWORK_COUNTER()	15
5.4 Phylo counters	15
5.4.1 Detailed Description	15
5.4.2 Function Documentation	16
5.4.2.1 counter_co_opt()	16
5.4.2.2 counter_cogain()	16

5.4.2.3 counter_gains()	17
5.4.2.4 counter_gains_k_offspring()	17
5.4.2.5 counter_genes_changing()	17
5.4.2.6 counter_longest()	17
5.4.2.7 counter_loss()	18
5.4.2.8 counter_maxfun()	18
5.4.2.9 counter_neofun()	18
5.4.2.10 counter_neofun_a2b()	18
5.4.2.11 counter_overall_changes()	19
5.4.2.12 counter_overall_gains()	19
5.4.2.13 counter_overall_loss()	19
5.4.2.14 counter_subfun()	19
6 Namespace Documentation	21
6.1 barry Namespace Reference	21
6.1.1 Detailed Description	21
6.2 barry::counters Namespace Reference	21
6.2.1 Detailed Description	21
6.3 barry::counters::network Namespace Reference	22
6.4 barry::counters::phylo Namespace Reference	22
6.5 CHECK Namespace Reference	22
6.5.1 Detailed Description	22
6.5.2 Variable Documentation	22
6.5.2.1 BOTH	22
6.5.2.2 NONE	22
6.5.2.3 ONE	22
6.5.2.4 TWO	23
6.6 EXISTS Namespace Reference	23
6.6.1 Detailed Description	23
6.6.2 Variable Documentation	23
6.6.2.1 AS_ONE	23
6.6.2.2 AS_ZERO	23
6.6.2.3 BOTH	24
6.6.2.4 NONE	24
6.6.2.5 ONE	24
6.6.2.6 TWO	24
6.6.2.7 UNKNOWN	24
7 Class Documentation	25
7.1 BArray< Cell_Type, Data_Type > Class Template Reference	25
7.1.1 Detailed Description	28
7.1.2 Constructor & Destructor Documentation	28
7.1.2.1 BArray() [1/6]	28

7.1.2.2 BArray() [2/6]	28
7.1.2.3 BArray() [3/6]	29
7.1.2.4 BArray() [4/6]	29
7.1.2.5 BArray() [5/6]	29
7.1.2.6 BArray() [6/6]	29
7.1.2.7 ~BArray()	30
7.1.3 Member Function Documentation	30
7.1.3.1 clear()	30
7.1.3.2 col()	30
7.1.3.3 get_cell()	30
7.1.3.4 get_col()	30
7.1.3.5 get_col_vec() [1/2]	31
7.1.3.6 get_col_vec() [2/2]	31
7.1.3.7 get_entries()	31
7.1.3.8 get_row()	31
7.1.3.9 get_row_vec() [1/2]	31
7.1.3.10 get_row_vec() [2/2]	32
7.1.3.11 insert_cell() [1/3]	32
7.1.3.12 insert_cell() [2/3]	32
7.1.3.13 insert_cell() [3/3]	32
7.1.3.14 is_empty()	32
7.1.3.15 ncol()	33
7.1.3.16 nnozero()	33
7.1.3.17 nrow()	33
7.1.3.18 operator>() [1/2]	33
7.1.3.19 operator>() [2/2]	33
7.1.3.20 operator*=()	33
7.1.3.21 operator+=() [1/3]	34
7.1.3.22 operator+=() [2/3]	34
7.1.3.23 operator+=() [3/3]	34
7.1.3.24 operator-=() [1/3]	34
7.1.3.25 operator-=() [2/3]	34
7.1.3.26 operator-=() [3/3]	34
7.1.3.27 operator/=()	35
7.1.3.28 operator=() [1/2]	35
7.1.3.29 operator=() [2/2]	35
7.1.3.30 operator==()	35
7.1.3.31 out_of_range()	35
7.1.3.32 print()	35
7.1.3.33 reserve()	36
7.1.3.34 resize()	36
7.1.3.35 rm_cell()	36

7.1.3.36 row()	36
7.1.3.37 set_data()	36
7.1.3.38 swap_cells()	37
7.1.3.39 swap_cols()	37
7.1.3.40 swap_rows()	37
7.1.3.41 toggle_cell()	37
7.1.3.42 toggle_lock()	37
7.1.3.43 transpose()	38
7.1.3.44 zero_col()	38
7.1.3.45 zero_row()	38
7.1.4 Friends And Related Function Documentation	38
7.1.4.1 BArrayCell< Cell_Type, Data_Type >	38
7.1.4.2 BArrayCell_const< Cell_Type, Data_Type >	38
7.1.5 Member Data Documentation	38
7.1.5.1 Cell_default	39
7.1.5.2 data	39
7.1.5.3 delete_data	39
7.1.5.4 el_ij	39
7.1.5.5 el_ji	39
7.1.5.6 M	39
7.1.5.7 N	40
7.1.5.8 NCells	40
7.1.5.9 visited	40
7.2 BArrayCell< Cell_Type, Data_Type > Class Template Reference	40
7.2.1 Detailed Description	41
7.2.2 Constructor & Destructor Documentation	41
7.2.2.1 BArrayCell()	41
7.2.2.2 ~BArrayCell()	41
7.2.3 Member Function Documentation	41
7.2.3.1 operator Cell_Type()	41
7.2.3.2 operator*=()	42
7.2.3.3 operator+=()	42
7.2.3.4 operator-=()	42
7.2.3.5 operator/=()	42
7.2.3.6 operator=()	42
7.2.3.7 operator==()	43
7.3 BArrayCell_const< Cell_Type, Data_Type > Class Template Reference	43
7.3.1 Detailed Description	43
7.3.2 Constructor & Destructor Documentation	43
7.3.2.1 BArrayCell_const()	44
7.3.2.2 ~BArrayCell_const()	44
7.3.3 Member Function Documentation	44

7.3.3.1 operator Cell_Type()	44
7.3.3.2 operator!=(())	44
7.3.3.3 operator<()	44
7.3.3.4 operator<=()	45
7.3.3.5 operator==(())	45
7.3.3.6 operator>()	45
7.3.3.7 operator>=()	45
7.4 Cell< Cell_Type > Class Template Reference	45
7.4.1 Detailed Description	46
7.4.2 Constructor & Destructor Documentation	46
7.4.2.1 Cell() [1/7]	46
7.4.2.2 Cell() [2/7]	47
7.4.2.3 ~Cell()	47
7.4.2.4 Cell() [3/7]	47
7.4.2.5 Cell() [4/7]	47
7.4.2.6 Cell() [5/7]	47
7.4.2.7 Cell() [6/7]	48
7.4.2.8 Cell() [7/7]	48
7.4.3 Member Function Documentation	48
7.4.3.1 add() [1/4]	48
7.4.3.2 add() [2/4]	48
7.4.3.3 add() [3/4]	48
7.4.3.4 add() [4/4]	49
7.4.3.5 operator Cell_Type()	49
7.4.3.6 operator=() [1/2]	49
7.4.3.7 operator=() [2/2]	49
7.4.4 Member Data Documentation	49
7.4.4.1 value	49
7.4.4.2 visited	50
7.5 ConstBArrayRowIter< Cell_Type, Data_Type > Class Template Reference	50
7.5.1 Detailed Description	51
7.5.2 Constructor & Destructor Documentation	51
7.5.2.1 ConstBArrayRowIter()	51
7.5.2.2 ~ConstBArrayRowIter()	51
7.5.3 Member Data Documentation	51
7.5.3.1 Array	51
7.5.3.2 current_col	51
7.5.3.3 current_row	52
7.5.3.4 iter	52
7.6 Counter< Array_Type, Data_Type > Class Template Reference	52
7.6.1 Detailed Description	53
7.6.2 Constructor & Destructor Documentation	53

7.6.2.1 Counter() [1/3]	53
7.6.2.2 Counter() [2/3]	53
7.6.2.3 Counter() [3/3]	54
7.6.2.4 ~Counter()	54
7.6.3 Member Function Documentation	54
7.6.3.1 count()	54
7.6.3.2 init()	54
7.6.3.3 operator=()	54
7.6.4 Member Data Documentation	55
7.6.4.1 count_fun	55
7.6.4.2 data	55
7.6.4.3 delete_data	55
7.6.4.4 desc	55
7.6.4.5 init_fun	55
7.6.4.6 name	56
7.7 Counters< Array_Type, Data_Type > Class Template Reference	56
7.7.1 Detailed Description	56
7.7.2 Constructor & Destructor Documentation	57
7.7.2.1 Counters() [1/2]	57
7.7.2.2 ~Counters()	57
7.7.2.3 Counters() [2/2]	57
7.7.3 Member Function Documentation	57
7.7.3.1 add_counter() [1/3]	57
7.7.3.2 add_counter() [2/3]	58
7.7.3.3 add_counter() [3/3]	58
7.7.3.4 clear()	58
7.7.3.5 operator=()	58
7.7.3.6 operator[]()	58
7.7.3.7 size()	59
7.8 Entries< Cell_Type > Class Template Reference	59
7.8.1 Detailed Description	59
7.8.2 Constructor & Destructor Documentation	60
7.8.2.1 Entries() [1/2]	60
7.8.2.2 Entries() [2/2]	60
7.8.2.3 ~Entries()	60
7.8.3 Member Function Documentation	60
7.8.3.1 resize()	60
7.8.4 Member Data Documentation	61
7.8.4.1 source	61
7.8.4.2 target	61
7.8.4.3 val	61
7.9 Flock Class Reference	61

7.9.1 Detailed Description	62
7.9.2 Constructor & Destructor Documentation	62
7.9.2.1 Flock()	62
7.9.2.2 ~Flock()	63
7.9.3 Member Function Documentation	63
7.9.3.1 add_data()	63
7.9.3.2 counters_ptr()	63
7.9.3.3 init()	63
7.9.3.4 likelihood_joint()	64
7.9.3.5 nfuncs()	64
7.9.3.6 nleaves()	64
7.9.3.7 nnodes()	64
7.9.3.8 nterms()	65
7.9.3.9 ntrees()	65
7.9.3.10 operator()()	65
7.9.3.11 set_seed()	65
7.9.4 Member Data Documentation	66
7.9.4.1 dat	66
7.9.4.2 initialized	66
7.9.4.3 nfunctions	66
7.9.4.4 rengine	66
7.9.4.5 support	66
7.10 FreqTable< T > Class Template Reference	67
7.10.1 Detailed Description	67
7.10.2 Constructor & Destructor Documentation	67
7.10.2.1 FreqTable()	67
7.10.2.2 ~FreqTable()	67
7.10.3 Member Function Documentation	68
7.10.3.1 add()	68
7.10.3.2 as_vector()	68
7.10.3.3 clear()	68
7.10.3.4 get_data()	68
7.10.3.5 get_data_ptr()	68
7.10.3.6 print()	69
7.10.3.7 reserve()	69
7.10.3.8 size()	69
7.11 Geese Class Reference	69
7.11.1 Detailed Description	71
7.11.2 Constructor & Destructor Documentation	71
7.11.2.1 Geese() [1/4]	71
7.11.2.2 Geese() [2/4]	71
7.11.2.3 Geese() [3/4]	72

7.11.2.4 Geese() [4 / 4]	72
7.11.2.5 ~Geese()	72
7.11.3 Member Function Documentation	72
7.11.3.1 calc_reduced_sequence()	72
7.11.3.2 calc_sequence()	72
7.11.3.3 get_probabilities()	73
7.11.3.4 inherit_support()	73
7.11.3.5 init()	73
7.11.3.6 init_node()	73
7.11.3.7 likelihood()	73
7.11.3.8 likelihood_exhaust()	74
7.11.3.9 nfuncs()	74
7.11.3.10 nleafs()	74
7.11.3.11 nnodes()	74
7.11.3.12 nterms()	74
7.11.3.13 observed_counts()	74
7.11.3.14 operator=() [1 / 2]	75
7.11.3.15 operator=() [2 / 2]	75
7.11.3.16 predict()	75
7.11.3.17 predict_backend()	75
7.11.3.18 print_observed_counts()	75
7.11.3.19 set_seed()	76
7.11.3.20 simulate()	76
7.11.3.21 update_annotations()	76
7.11.4 Member Data Documentation	76
7.11.4.1 counters	76
7.11.4.2 delete_counters	76
7.11.4.3 delete_engine	77
7.11.4.4 delete_support	77
7.11.4.5 initialized	77
7.11.4.6 map_to_nodes	77
7.11.4.7 nfunctions	77
7.11.4.8 nodes	77
7.11.4.9 reduced_sequence	78
7.11.4.10 rengine	78
7.11.4.11 sequence	78
7.11.4.12 states	78
7.11.4.13 support	78
7.12 Model< Array_Type, Data_Counter_Type, Data_Rule_Type > Class Template Reference	79
7.12.1 Detailed Description	82
7.12.2 Constructor & Destructor Documentation	82
7.12.2.1 Model() [1 / 3]	82

7.12.2.2 Model() [2/3]	82
7.12.2.3 Model() [3/3]	83
7.12.2.4 ~Model()	83
7.12.3 Member Function Documentation	83
7.12.3.1 add_array()	83
7.12.3.2 add_counter() [1/3]	84
7.12.3.3 add_counter() [2/3]	84
7.12.3.4 add_counter() [3/3]	84
7.12.3.5 add_rule() [1/3]	84
7.12.3.6 add_rule() [2/3]	84
7.12.3.7 add_rule() [3/3]	85
7.12.3.8 get_norm_const()	85
7.12.3.9 get_pset()	85
7.12.3.10 get_stats()	85
7.12.3.11 likelihood() [1/3]	86
7.12.3.12 likelihood() [2/3]	86
7.12.3.13 likelihood() [3/3]	86
7.12.3.14 likelihood_total()	86
7.12.3.15 nterms()	87
7.12.3.16 operator=()	87
7.12.3.17 print_stats()	87
7.12.3.18 sample() [1/2]	87
7.12.3.19 sample() [2/2]	87
7.12.3.20 set_counters()	88
7.12.3.21 set_keygen()	88
7.12.3.22 set_engine()	88
7.12.3.23 set_rules()	88
7.12.3.24 set_seed()	88
7.12.3.25 size()	89
7.12.3.26 size_unique()	89
7.12.3.27 store_psets()	89
7.12.4 Member Data Documentation	89
7.12.4.1 array_frequency	89
7.12.4.2 arrays2support	89
7.12.4.3 counter_fun	90
7.12.4.4 counters	90
7.12.4.5 delete_engine	90
7.12.4.6 first_calc_done	90
7.12.4.7 keygen	90
7.12.4.8 keys2support	91
7.12.4.9 n_arrays_per_stats	91
7.12.4.10 normalizing_constants	91

7.12.4.11 params_last	91
7.12.4.12 pset_arrays	92
7.12.4.13 pset_probs	92
7.12.4.14 pset_stats	92
7.12.4.15 rengine	92
7.12.4.16 rules	92
7.12.4.17 stats	93
7.12.4.18 support_fun	93
7.12.4.19 target_stats	93
7.12.4.20 with_pset	93
7.13 NetCounterData Class Reference	93
7.13.1 Detailed Description	94
7.13.2 Constructor & Destructor Documentation	94
7.13.2.1 NetCounterData() [1/2]	94
7.13.2.2 NetCounterData() [2/2]	94
7.13.2.3 ~NetCounterData()	94
7.13.3 Member Data Documentation	95
7.13.3.1 indices	95
7.13.3.2 numbers	95
7.14 NetworkData Class Reference	95
7.14.1 Detailed Description	96
7.14.2 Constructor & Destructor Documentation	96
7.14.2.1 NetworkData() [1/3]	96
7.14.2.2 NetworkData() [2/3]	96
7.14.2.3 NetworkData() [3/3]	96
7.14.2.4 ~NetworkData()	97
7.14.3 Member Data Documentation	97
7.14.3.1 directed	97
7.14.3.2 vertex_attr	97
7.15 Node Class Reference	97
7.15.1 Detailed Description	98
7.15.2 Constructor & Destructor Documentation	98
7.15.2.1 Node() [1/5]	99
7.15.2.2 Node() [2/5]	99
7.15.2.3 Node() [3/5]	99
7.15.2.4 Node() [4/5]	99
7.15.2.5 Node() [5/5]	99
7.15.2.6 ~Node()	100
7.15.3 Member Function Documentation	100
7.15.3.1 get_parent()	100
7.15.3.2 is_leaf()	100
7.15.4 Member Data Documentation	100

7.15.4.1 annotations	100
7.15.4.2 array	100
7.15.4.3 arrays	101
7.15.4.4 duplication	101
7.15.4.5 id	101
7.15.4.6 narray	101
7.15.4.7 offspring	101
7.15.4.8 ord	102
7.15.4.9 parent	102
7.15.4.10 probability	102
7.15.4.11 subtree_prob	102
7.15.4.12 visited	102
7.16 NodeData Class Reference	103
7.16.1 Detailed Description	103
7.16.2 Constructor & Destructor Documentation	103
7.16.2.1 NodeData() [1/2]	103
7.16.2.2 NodeData() [2/2]	103
7.16.2.3 ~NodeData()	104
7.16.3 Member Data Documentation	104
7.16.3.1 blengths	104
7.16.3.2 duplication	104
7.16.3.3 states	104
7.17 PowerSet< Array_Type, Data_Rule_Type > Class Template Reference	105
7.17.1 Detailed Description	106
7.17.2 Constructor & Destructor Documentation	106
7.17.2.1 PowerSet() [1/3]	106
7.17.2.2 PowerSet() [2/3]	106
7.17.2.3 PowerSet() [3/3]	107
7.17.2.4 ~PowerSet()	107
7.17.3 Member Function Documentation	107
7.17.3.1 add_rule() [1/3]	107
7.17.3.2 add_rule() [2/3]	107
7.17.3.3 add_rule() [3/3]	107
7.17.3.4 begin()	108
7.17.3.5 calc()	108
7.17.3.6 end()	108
7.17.3.7 get_data()	108
7.17.3.8 get_data_ptr()	108
7.17.3.9 init_support()	109
7.17.3.10 operator[]()	109
7.17.3.11 reset()	109
7.17.3.12 size()	109

7.17.4 Member Data Documentation	109
7.17.4.1 coordinates_free	109
7.17.4.2 coordinates_locked	110
7.17.4.3 data	110
7.17.4.4 EmptyArray	110
7.17.4.5 M	110
7.17.4.6 N	110
7.17.4.7 rules	111
7.17.4.8 rules_deleted	111
7.18 Rule< Array_Type, Data_Type > Class Template Reference	111
7.18.1 Detailed Description	111
7.18.2 Constructor & Destructor Documentation	112
7.18.2.1 Rule() [1/2]	112
7.18.2.2 Rule() [2/2]	112
7.18.2.3 ~Rule()	112
7.18.3 Member Function Documentation	113
7.18.3.1 locked()	113
7.19 Rules< Array_Type, Data_Type > Class Template Reference	113
7.19.1 Detailed Description	114
7.19.2 Constructor & Destructor Documentation	114
7.19.2.1 Rules() [1/2]	114
7.19.2.2 Rules() [2/2]	114
7.19.2.3 ~Rules()	114
7.19.3 Member Function Documentation	115
7.19.3.1 add_rule() [1/3]	115
7.19.3.2 add_rule() [2/3]	115
7.19.3.3 add_rule() [3/3]	115
7.19.3.4 clear()	115
7.19.3.5 get_seq()	115
7.19.3.6 locked()	116
7.19.3.7 operator=()	116
7.19.3.8 size()	117
7.20 StatsCounter< Array_Type, Data_Type > Class Template Reference	117
7.20.1 Detailed Description	118
7.20.2 Constructor & Destructor Documentation	118
7.20.2.1 StatsCounter() [1/2]	118
7.20.2.2 StatsCounter() [2/2]	118
7.20.2.3 ~StatsCounter()	119
7.20.3 Member Function Documentation	119
7.20.3.1 add_counter() [1/2]	119
7.20.3.2 add_counter() [2/2]	119
7.20.3.3 count_all()	119

7.20.3.4 count_current()	119
7.20.3.5 count_init()	120
7.20.3.6 reset_array()	120
7.20.3.7 set_counters()	120
7.20.4 Member Data Documentation	120
7.20.4.1 Array	120
7.20.4.2 counter_deleted	121
7.20.4.3 counters	121
7.20.4.4 current_stats	121
7.20.4.5 EmptyArray	121
7.21 Support< Array_Type, Data_Counter_Type, Data_Rule_Type > Class Template Reference	122
7.21.1 Detailed Description	123
7.21.2 Constructor & Destructor Documentation	124
7.21.2.1 Support() [1/3]	124
7.21.2.2 Support() [2/3]	124
7.21.2.3 Support() [3/3]	124
7.21.2.4 ~Support()	124
7.21.3 Member Function Documentation	125
7.21.3.1 add_counter() [1/2]	125
7.21.3.2 add_counter() [2/2]	125
7.21.3.3 add_rule() [1/2]	125
7.21.3.4 add_rule() [2/2]	125
7.21.3.5 calc()	125
7.21.3.6 get_counts()	126
7.21.3.7 get_counts_ptr()	126
7.21.3.8 init_support()	126
7.21.3.9 print()	126
7.21.3.10 reset_array() [1/2]	127
7.21.3.11 reset_array() [2/2]	127
7.21.3.12 set_counters()	127
7.21.3.13 set_rules()	127
7.21.4 Member Data Documentation	127
7.21.4.1 change_stats	127
7.21.4.2 coordinates_free	128
7.21.4.3 coordinates_locked	128
7.21.4.4 counter_deleted	128
7.21.4.5 counters	128
7.21.4.6 current_stats	128
7.21.4.7 data	129
7.21.4.8 EmptyArray	129
7.21.4.9 M	129
7.21.4.10 max_num_elements	129

7.21.4.11 N	129
7.21.4.12 rules	130
7.21.4.13 rules_deleted	130
7.22 vecHasher< T > Struct Template Reference	130
7.22.1 Detailed Description	130
7.22.2 Member Function Documentation	130
7.22.2.1 operator>()	130
8 File Documentation	131
8.1 include/barry/barray-bones.hpp File Reference	131
8.1.1 Macro Definition Documentation	132
8.1.1.1 BARRAY_BONES_HPP	132
8.2 include/barry/barray-iterator.hpp File Reference	132
8.2.1 Macro Definition Documentation	133
8.2.1.1 BARRAY_ITERATOR_HPP	133
8.3 include/barry/barray-meat-operators.hpp File Reference	133
8.3.1 Function Documentation	134
8.3.1.1 checkdim_()	134
8.4 include/barry/barray-meat.hpp File Reference	135
8.5 include/barry/barraycell-bones.hpp File Reference	135
8.6 include/barry/barraycell-meat.hpp File Reference	136
8.7 include/barry/barry-configuration.hpp File Reference	137
8.7.1 Macro Definition Documentation	138
8.7.1.1 BARRY_CHECK_SUPPORT	138
8.7.1.2 BARRY_ISFINITE	138
8.7.1.3 BARRY_MAX_NUM_ELEMENTS	138
8.7.1.4 BARRY_SAFE_EXP	139
8.7.2 Typedef Documentation	139
8.7.2.1 Map	139
8.8 include/barry/barry.hpp File Reference	139
8.8.1 Macro Definition Documentation	141
8.8.1.1 COUNTER_FUNCTION	141
8.8.1.2 COUNTER_LAMBDA	141
8.8.1.3 RULE_FUNCTION	141
8.8.1.4 RULE_LAMBDA	141
8.9 include/barry/cell-bones.hpp File Reference	142
8.10 include/barry/cell-meat.hpp File Reference	142
8.11 include/barry/col-bones.hpp File Reference	143
8.12 include/barry/counters-bones.hpp File Reference	143
8.13 include/barry/counters-meat.hpp File Reference	144
8.14 include/barry/counters/network.hpp File Reference	145
8.14.1 Macro Definition Documentation	148

8.14.1.1 NET_C_DATA_IDX	148
8.14.1.2 NET_C_DATA_NUM	148
8.14.1.3 NETWORK_COUNTER	148
8.14.1.4 NETWORK_COUNTER_LAMBDA	148
8.14.1.5 NETWORK_RULE	149
8.14.1.6 NETWORK_RULE_LAMBDA	149
8.14.2 Typedef Documentation	149
8.14.2.1 NetCounter	149
8.14.2.2 NetCounters	149
8.14.2.3 NetModel	150
8.14.2.4 NetRule	150
8.14.2.5 NetRules	150
8.14.2.6 NetStatsCounter	150
8.14.2.7 NetSupport	150
8.14.2.8 Network	150
8.14.3 Function Documentation	151
8.14.3.1 rules_zerodiag()	151
8.15 include/barry/counters/phylo.hpp File Reference	151
8.15.1 Macro Definition Documentation	153
8.15.1.1 PHYLO_C_DATA_IDX	153
8.15.1.2 PHYLO_CHECK_MISSING	154
8.15.1.3 PHYLO_COUNTER	154
8.15.1.4 PHYLO_COUNTER_LAMBDA	154
8.15.2 Typedef Documentation	154
8.15.2.1 PhyloArray	154
8.15.2.2 PhyloCounter	155
8.15.2.3 PhyloCounterData	155
8.15.2.4 PhyloCounters	155
8.15.2.5 PhyloModel	155
8.15.2.6 PhyloPowerSet	155
8.15.2.7 PhyloRule	155
8.15.2.8 PhyloRuleData	156
8.15.2.9 PhyloRules	156
8.15.2.10 PhyloStatsCounter	156
8.15.2.11 PhyloSupport	156
8.15.3 Function Documentation	156
8.15.3.1 get_last_name()	156
8.16 include/barry/model-bones.hpp File Reference	157
8.16.1 Function Documentation	158
8.16.1.1 keygen_default()	158
8.16.1.2 likelihood_()	158
8.16.1.3 update_normalizing_constant()	158

8.17 include/barry/model-meat.hpp File Reference	159
8.18 include/barry/models/geese.hpp File Reference	160
8.19 include/barry/models/geese/flock-bones.hpp File Reference	160
8.20 include/barry/models/geese/flock-meet.hpp File Reference	161
8.21 include/barry/models/geese/geese-bones.hpp File Reference	161
8.21.1 Macro Definition Documentation	162
8.21.1.1 INITIALIZED	162
8.21.2 Function Documentation	162
8.21.2.1 keygen_full()	162
8.21.2.2 RULE_FUNCTION()	163
8.21.2.3 vec_diff()	163
8.21.2.4 vector_caster()	163
8.22 include/barry/models/geese/geese-meat-constructors.hpp File Reference	163
8.22.1 Macro Definition Documentation	164
8.22.1.1 GEESE_MEAT_CONSTRUCTORS_HPP	164
8.23 include/barry/models/geese/geese-meat-likelihood.hpp File Reference	164
8.24 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference	165
8.25 include/barry/models/geese/geese-meat-predict.hpp File Reference	166
8.26 include/barry/models/geese/geese-meat-simulate.hpp File Reference	167
8.27 include/barry/models/geese/geese-meat.hpp File Reference	168
8.28 include/barry/models/geese/geese-node-bones.hpp File Reference	169
8.29 include/barry/powerset-bones.hpp File Reference	169
8.30 include/barry/powerset-meat.hpp File Reference	171
8.31 include/barry/rules-bones.hpp File Reference	171
8.31.1 Function Documentation	172
8.31.1.1 rule_fun_default()	173
8.32 include/barry/rules-meat.hpp File Reference	173
8.33 include/barry/statscounter-bones.hpp File Reference	174
8.34 include/barry/statscounter-meat.hpp File Reference	175
8.35 include/barry/statsdb.hpp File Reference	175
8.36 include/barry/support-bones.hpp File Reference	176
8.37 include/barry/support-meat.hpp File Reference	178
8.37.1 Macro Definition Documentation	179
8.37.1.1 BARRY_SUPPORT_MEAT_HPP	179
8.38 include/barry/typedefs.hpp File Reference	179
8.38.1 Macro Definition Documentation	181
8.38.1.1 A_COL	181
8.38.1.2 A_ROW	181
8.38.1.3 COL	181
8.38.1.4 ROW	181
8.38.2 Typedef Documentation	182
8.38.2.1 Col_type	182

8.38.2.2 Counter_fun_type	182
8.38.2.3 Counts_type	182
8.38.2.4 MapVec_type	182
8.38.2.5 Row_type	183
8.38.2.6 Rule_fun_type	183
8.38.2.7 uint	183
8.38.3 Function Documentation	183
8.38.3.1 vec_equal()	183
8.38.3.2 vec_equal_approx()	184
8.38.3.3 vec_inner_prod()	184
8.39 README.md File Reference	184
Index	185

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 freeing 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

```

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:

Counting	9
Statistical Models	9
Network counters	10
Phylo counters	15

Chapter 3

Class Index

3.1 Class List

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

BArray< Cell_Type, Data_Type >	
Baseline class for binary arrays	25
BArrayCell< Cell_Type, Data_Type >	40
BArrayCell_const< Cell_Type, Data_Type >	43
Cell< Cell_Type >	
Entries in BArray . For now, it only has two members:	45
ConstBArrayRowIter< Cell_Type, Data_Type >	50
Counter< Array_Type, Data_Type >	
A counter function based on change statistics	52
Counters< Array_Type, Data_Type >	
Vector of counters	56
Entries< Cell_Type >	
A wrapper class to store <code>source</code> , <code>target</code> , <code>val</code> from a BArray object	59
Flock	
A Flock is a group of Geese	61
FreqTable< T >	
Database of statistics	67
Geese	
Annotated Phylo Model	69
Model< Array_Type, Data_Counter_Type, Data_Rule_Type >	
General framework for discrete exponential models. This class allows generating discrete exponential models in the form of a linear exponential model:	79
NetCounterData	
Data class used to store arbitrary uint or double vectors	93
NetworkData	
Data class for Networks	95
Node	
A single node for the model	97
NodeData	
Data definition for the PhyloArray class	103
PowerSet< Array_Type, Data_Rule_Type >	
Powerset of a binary array	105
Rule< Array_Type, Data_Type >	
Rule for determining if a cell should be included in a sequence	111
Rules< Array_Type, Data_Type >	
Vector of objects of class Rule	113

StatsCounter< Array_Type, Data_Type >	
Count stats for a single Array	117
Support< Array_Type, Data_Counter_Type, Data_Rule_Type >	
Compute the support of sufficient statistics	122
vecHasher< T >	130

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

include/barry/barray-bones.hpp	131
include/barry/barray-iterator.hpp	132
include/barry/barray-meat-operators.hpp	133
include/barry/barray-meat.hpp	135
include/barry/barraycell-bones.hpp	135
include/barry/barraycell-meat.hpp	136
include/barry/barry-configuration.hpp	137
include/barry/barry.hpp	139
include/barry/cell-bones.hpp	142
include/barry/cell-meat.hpp	142
include/barry/col-bones.hpp	143
include/barry/counters-bones.hpp	143
include/barry/counters-meat.hpp	144
include/barry/model-bones.hpp	157
include/barry/model-meat.hpp	159
include/barry/powerset-bones.hpp	169
include/barry/powerset-meat.hpp	171
include/barry/rules-bones.hpp	171
include/barry/rules-meat.hpp	173
include/barry/statscounter-bones.hpp	174
include/barry/statscounter-meat.hpp	175
include/barry/statsdb.hpp	175
include/barry/support-bones.hpp	176
include/barry/support-meat.hpp	178
include/barry/typedefs.hpp	179
include/barry/counters/network.hpp	145
include/barry/counters/phylo.hpp	151
include/barry/models/geese.hpp	160
include/barry/models/geese/flock-bones.hpp	160
include/barry/models/geese/flock-meet.hpp	161
include/barry/models/geese/geese-bones.hpp	161
include/barry/models/geese/geese-meat-constructors.hpp	163
include/barry/models/geese/geese-meat-likelihood.hpp	164
include/barry/models/geese/geese-meat-likelihood_exhaust.hpp	165
include/barry/models/geese/geese-meat-predict.hpp	166
include/barry/models/geese/geese-meat-simulate.hpp	167
include/barry/models/geese/geese-meat.hpp	168
include/barry/models/geese/geese-node-bones.hpp	169

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 empty 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](#) >
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

<i>counters</i>	A pointer to a <code>NetCounters</code> object (<code>Counters<Network, NetCounterData></code>).
-----------------	--

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_nodecov()

```
void counter_nodecov (
    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_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

<i>counters</i>	A pointer to a <code>PhyloCounters</code> object (<code>Counters<PhyloArray, PhyloCounterData></code>).
-----------------	---

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 1081 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 711 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 149 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 191 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 265 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 770 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 512 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 428 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 875 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 960 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 559 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 109 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 382 of file phylo.hpp.

5.4.2.14 counter_subfun()

```
void counter_subfun (
    PhyloCounters * counters,
    uint nfunA,
    uint nfunB,
    bool duplication = true ) [inline]
```

Total count of Sub-functionalization events.

It requires to specify data = {funA, funB}

Definition at line 625 of file phylo.hpp.

Chapter 6

Namespace Documentation

6.1 `barry` Namespace Reference

`barry`: Your go-to motif accountant

Namespaces

- [counters](#)

Tree class and Treeliterator class.

6.1.1 Detailed Description

`barry`: Your go-to motif accountant

6.2 `barry::counters` Namespace Reference

Tree class and Treeliterator class.

Namespaces

- [network](#)
- [phylo](#)

6.2.1 Detailed Description

Tree class and Treeliterator 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 32 of file typedefs.hpp.

6.5.2.2 NONE

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

Definition at line 33 of file typedefs.hpp.

6.5.2.3 ONE

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

Definition at line 34 of file typedefs.hpp.

6.5.2.4 TWO

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

Definition at line 35 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 UNKNOWN = -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 50 of file typedefs.hpp.

6.6.2.2 AS_ZERO

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

Definition at line 49 of file typedefs.hpp.

6.6.2.3 BOTH

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

Definition at line 43 of file typedefs.hpp.

6.6.2.4 NONE

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

Definition at line 44 of file typedefs.hpp.

6.6.2.5 ONE

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

Definition at line 45 of file typedefs.hpp.

6.6.2.6 TWO

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

Definition at line 46 of file typedefs.hpp.

6.6.2.7 UNKNOWN

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

Definition at line 48 of file typedefs.hpp.

Chapter 7

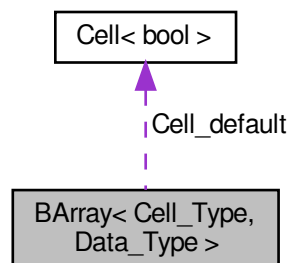
Class Documentation

7.1 BArray< Cell_Type, Data_Type > Class Template Reference

Baseline class for binary arrays.

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

Collaboration diagram for BArray< Cell_Type, Data_Type >:



Public Member Functions

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

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

Get the edgelist.

- void `transpose` ()
- void `clear` (bool hard=true)
- void `resize` (`uint` N_, `uint` M_)
- void `reserve` ()
- void `print` () const

Constructors

Parameters

N_	Number of rows
M_	Number of columns
source	An unsigned vector ranging from 0 to N_
target	An unsigned int vector ranging from 0 to M_
target	When <code>true</code> tries to add repeated observations.

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

Queries

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

Parameters

i,j	Coordinates
check_bounds	If <code>false</code> 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-wise insertion/deletion

Parameters

i,j	Row,column
check_bounds	When <i>true</i> 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 <i>swap_cells</i> , 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::UNKNOWN](#))
- void [toggle_lock](#) ([uint](#) i, [uint](#) j, bool check_bounds=true)

Column/row wise interchange

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

Arithmetic operators

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

Public Attributes

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

Static Public Attributes

- static [Cell](#)< Cell_Type > [Cell_default](#)

Friends

- class [BArrayCell< Cell_Type, Data_Type >](#)
- class [BArrayCell_const< Cell_Type, Data_Type >](#)

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_maps`, i.e. `std::vector< std::unordered_map<unsigned int, Cell_Type> >`.

Template Parameters

<i>Cell_Type</i>	Type of cell (any type).
<i>Data_Type</i>	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 58 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 61 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 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
```

7.1.3.4 get_col()

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

7.1.3.5 get_col_vec() [1/2]

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

7.1.3.6 get_col_vec() [2/2]

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

7.1.3.7 get_entries()

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

Get the edgelist.

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

Returns

`Entries<Cell_Type>`

7.1.3.8 get_row()

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

7.1.3.9 get_row_vec() [1/2]

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

7.1.3.10 `get_row_vec()` [2/2]

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

7.1.3.11 `insert_cell()` [1/3]

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

7.1.3.12 `insert_cell()` [2/3]

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

7.1.3.13 `insert_cell()` [3/3]

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

7.1.3.14 `is_empty()`

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

7.1.3.15 ncol()

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

7.1.3.16 nnozero()

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

7.1.3.17 nrow()

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

7.1.3.18 operator()() [1/2]

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

7.1.3.19 operator()() [2/2]

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

7.1.3.20 operator*=()

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

7.1.3.21 operator+=() [1/3]

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

7.1.3.22 operator+=() [2/3]

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

7.1.3.23 operator+=() [3/3]

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

7.1.3.24 operator-=() [1/3]

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

7.1.3.25 operator-=() [2/3]

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

7.1.3.26 operator-=() [3/3]

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

7.1.3.27 operator/=()

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

7.1.3.28 operator=() [1/2]

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

Move assignment.

7.1.3.29 operator=() [2/2]

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

Assignment constructor.

7.1.3.30 operator==()

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

7.1.3.31 out_of_range()

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

7.1.3.32 print()

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

7.1.3.33 reserve()

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

7.1.3.34 resize()

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

7.1.3.35 rm_cell()

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

7.1.3.36 row()

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

7.1.3.37 set_data()

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


7.1.3.38 swap_cells()

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

7.1.3.39 swap_cols()

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

7.1.3.40 swap_rows()

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

7.1.3.41 toggle_cell()

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

7.1.3.42 toggle_lock()

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

7.1.3.43 transpose()

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

7.1.3.44 zero_col()

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

7.1.3.45 zero_row()

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

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 Cell_default

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

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

7.1.5.2 data

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

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

7.1.5.3 delete_data

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

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

7.1.5.4 el_ij

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

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

7.1.5.5 el_ji

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

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

7.1.5.6 M

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

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

7.1.5.7 N

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

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

7.1.5.8 NCells

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

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

7.1.5.9 visited

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

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

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

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

- `include/barry/barray-bones.hpp`

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 Cell< Cell_Type > Class Template Reference

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

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

Public Member Functions

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

Public Attributes

- Cell_Type [value](#)
- bool [visited](#)

7.4.1 Detailed Description

```
template<class Cell_Type>
class Cell< Cell_Type >
```

[Entries](#) in [BArray](#). For now, it only has two members:

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

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

7.4.2 Constructor & Destructor Documentation

7.4.2.1 [Cell\(\)](#) [1/7]

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

7.4.2.2 Cell() [2/7]

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

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

7.4.2.3 ~Cell()

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

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

7.4.2.4 Cell() [3/7]

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

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

7.4.2.5 Cell() [4/7]

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

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

7.4.2.6 Cell() [5/7]

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

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

7.4.2.7 Cell() [6/7]

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

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

7.4.2.8 Cell() [7/7]

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

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

7.4.3 Member Function Documentation

7.4.3.1 add() [1/4]

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

7.4.3.2 add() [2/4]

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

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

7.4.3.3 add() [3/4]

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

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

7.4.3.4 add() [4/4]

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

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

7.4.3.5 operator Cell_Type()

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

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

7.4.3.6 operator=() [1/2]

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

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

7.4.3.7 operator=() [2/2]

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

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

7.4.4 Member Data Documentation

7.4.4.1 value

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

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

7.4.4.2 visited

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

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

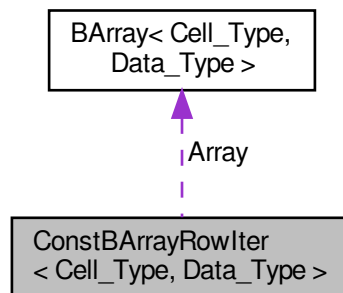
The documentation for this class was generated from the following files:

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

7.5 ConstBArrayRowIter< Cell_Type, Data_Type > Class Template Reference

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

Collaboration diagram for ConstBArrayRowIter< Cell_Type, Data_Type >:



Public Member Functions

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

Public Attributes

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

7.5.1 Detailed Description

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

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

7.5.2 Constructor & Destructor Documentation

7.5.2.1 ConstBArrayRowIter()

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

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

7.5.2.2 ~ConstBArrayRowIter()

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

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

7.5.3 Member Data Documentation

7.5.3.1 Array

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

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

7.5.3.2 current_col

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

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

7.5.3.3 current_row

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

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

7.5.3.4 iter

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

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

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

- include/barry/barray-iterator.hpp

7.6 Counter< Array_Type, Data_Type > Class Template Reference

A counter function based on change statistics.

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

Public Member Functions

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

Creator passing a counter and an initializer

Parameters

count_fun↔ _	The main counter function.
init_fun_ _	The initializer function can also be used to check if the BArray as the needed variables (see BArray::data).
data_ _	Data to be used with the counter.
delete_↔ data_ _	When <i>true</i> , 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\_)

## 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](#) = ""

### 7.6.1 Detailed Description

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

A counter function based on change statistics.

This class is used by `CountStats` and [StatsCounter](#) as a way to count statistics using change statistics.

Definition at line 38 of file counters-bones.hpp.

### 7.6.2 Constructor & Destructor Documentation

#### 7.6.2.1 Counter() [1/3]

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

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

#### 7.6.2.2 Counter() [2/3]

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

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

### 7.6.2.3 Counter() [3/3]

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

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

### 7.6.2.4 ~Counter()

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

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

## 7.6.3 Member Function Documentation

### 7.6.3.1 count()

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

Definition at line 124 of file counters-meat.hpp.

### 7.6.3.2 init()

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

Definition at line 136 of file counters-meat.hpp.

### 7.6.3.3 operator=()

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

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

## 7.6.4 Member Data Documentation

### 7.6.4.1 count\_fun

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

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

### 7.6.4.2 data

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

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

### 7.6.4.3 delete\_data

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

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

### 7.6.4.4 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.

### 7.6.4.5 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.

#### 7.6.4.6 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

## 7.7 Counters< Array\_Type, Data\_Type > Class Template Reference

Vector of counters.

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

### Public Member Functions

- [Counters](#) ()
- [~Counters](#) ()
- [Counters](#) (const [Counters](#)< Array\_Type, Data\_Type > &counter\_)
- [Counters](#)< Array\_Type, Data\_Type > [operator=](#) (const [Counters](#)< Array\_Type, Data\_Type > &counter\_)
- [Counter](#)< Array\_Type, Data\_Type > & [operator\[\]](#) (uint idx)  
*Returns a pointer to a particular counter.*
- 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](#) ()

### 7.7.1 Detailed Description

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

Vector of counters.

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

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

## 7.7.2 Constructor & Destructor Documentation

### 7.7.2.1 Counters() [1/2]

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

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

### 7.7.2.2 ~Counters()

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

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

### 7.7.2.3 Counters() [2/2]

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

Definition at line 67 of file counters-meat.hpp.

## 7.7.3 Member Function Documentation

### 7.7.3.1 add\_counter() [1/3]

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

Definition at line 157 of file counters-meat.hpp.

**7.7.3.2 add\_counter() [2/3]**

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

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

**7.7.3.3 add\_counter() [3/3]**

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

Definition at line 180 of file counters-meat.hpp.

**7.7.3.4 clear()**

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

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

**7.7.3.5 operator=()**

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

Definition at line 94 of file counters-meat.hpp.

**7.7.3.6 operator[]()**

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

Returns a pointer to a particular counter.

## Parameters

|            |                   |
|------------|-------------------|
| <i>idx</i> | Id of the counter |
|------------|-------------------|

## Returns

Counter<Array\_Type,Data\_Type>\*

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

## 7.7.3.7 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 129 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

## 7.8 Entries&lt; Cell\_Type &gt; 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](#)

## 7.8.1 Detailed Description

```
template<typename Cell_Type>
class Entries< Cell_Type >
```

A wrapper class to store `source`, `target`, `val` from a [BArray](#) object.

### Template Parameters

|                  |          |
|------------------|----------|
| <i>Cell_Type</i> | Any type |
|------------------|----------|

Definition at line 71 of file typedefs.hpp.

## 7.8.2 Constructor & Destructor Documentation

### 7.8.2.1 Entries() [1/2]

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

Definition at line 77 of file typedefs.hpp.

### 7.8.2.2 Entries() [2/2]

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

Definition at line 78 of file typedefs.hpp.

### 7.8.2.3 ~Entries()

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

Definition at line 85 of file typedefs.hpp.

## 7.8.3 Member Function Documentation

### 7.8.3.1 resize()

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

Definition at line 87 of file typedefs.hpp.



## 7.8.4 Member Data Documentation

### 7.8.4.1 source

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

Definition at line 73 of file typedefs.hpp.

### 7.8.4.2 target

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

Definition at line 74 of file typedefs.hpp.

### 7.8.4.3 val

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

Definition at line 75 of file typedefs.hpp.

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

- [include/barry/typedefs.hpp](#)

## 7.9 Flock Class Reference

A [Flock](#) is a group of [Geese](#).

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

## Public Member Functions

- [Flock](#) ()
- [~Flock](#) ()
- unsigned int [add\\_data](#) (std::vector< std::vector< unsigned int > > &annotations, std::vector< unsigned int > &geneid, std::vector< int > &parent, std::vector< bool > &duplication)  
*Add a tree to the flock.*
- void [set\\_seed](#) (const unsigned int &s)  
*Set the seed of the model.*
- void [init](#) ()
- [phylocounters::PhyloCounters](#) \* [counters\\_ptr](#) ()
- 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 [nfuncs](#) () 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

## Public Attributes

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

## 7.9.1 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.

## 7.9.2 Constructor & Destructor Documentation

### 7.9.2.1 Flock()

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

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

### 7.9.2.2 ~Flock()

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

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

## 7.9.3 Member Function Documentation

### 7.9.3.1 add\_data()

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

Add a tree to the flock.

#### Parameters

|                    |                                   |
|--------------------|-----------------------------------|
| <i>annotations</i> | see <a href="#">Geese::Geese.</a> |
| <i>geneid</i>      | see <a href="#">Geese.</a>        |
| <i>parent</i>      | see <a href="#">Geese.</a>        |
| <i>duplication</i> | see <a href="#">Geese.</a>        |

#### Returns

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

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

### 7.9.3.2 counters\_ptr()

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

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

### 7.9.3.3 init()

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

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

#### 7.9.3.4 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

|                             |                                                                                                           |
|-----------------------------|-----------------------------------------------------------------------------------------------------------|
| <i>par</i>                  | Vector of model parameters.                                                                               |
| <i>as_log</i>               | When <code>true</code> it will return the value as log.                                                   |
| <i>use_reduced_sequence</i> | When <code>true</code> (default) will compute the likelihood using the reduced sequence, which is faster. |

##### Returns

double

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

#### 7.9.3.5 nfuncs()

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

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

#### 7.9.3.6 nleafs()

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

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

#### 7.9.3.7 nnodes()

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

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

### 7.9.3.8 nterms()

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

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

### 7.9.3.9 ntrees()

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

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

### 7.9.3.10 operator()()

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

Access the i-th geese element.

#### Parameters

|                     |                                  |
|---------------------|----------------------------------|
| <i>i</i>            | Element to access                |
| <i>check_bounds</i> | When true, it will check bounds. |

#### Returns

Geese\*

Definition at line 151 of file flock-meet.hpp.

### 7.9.3.11 set\_seed()

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

Set the seed of the model.

#### Parameters

|          |                                                         |
|----------|---------------------------------------------------------|
| <i>s</i> | Passed to the <code>engine.seed()</code> member object. |
|----------|---------------------------------------------------------|

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

## 7.9.4 Member Data Documentation

### 7.9.4.1 dat

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

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

### 7.9.4.2 initialized

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

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

### 7.9.4.3 nfunctions

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

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

### 7.9.4.4 engine

```
std::mt19937 Flock::engine
```

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

### 7.9.4.5 support

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

Definition at line 23 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](#)

## 7.10 FreqTable< T > Class Template Reference

Database of statistics.

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

### Public Member Functions

- [FreqTable](#) ()
- [~FreqTable](#) ()
- void [add](#) (const std::vector< T > &x)
- [Counts\\_type as\\_vector](#) () const
- [MapVec\\_type](#)< T, uint > [get\\_data](#) () const
- const [MapVec\\_type](#)< T, uint > \* [get\\_data\\_ptr](#) () const
- void [clear](#) ()
- void [reserve](#) (unsigned int n)
- void [print](#) () const
- size\_t [size](#) () const noexcept

### 7.10.1 Detailed Description

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

Database of statistics.

This is mostly used in [Support](#).

Definition at line 16 of file statsdb.hpp.

### 7.10.2 Constructor & Destructor Documentation

#### 7.10.2.1 FreqTable()

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

Definition at line 28 of file statsdb.hpp.

#### 7.10.2.2 ~FreqTable()

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

Definition at line 29 of file statsdb.hpp.

### 7.10.3 Member Function Documentation

#### 7.10.3.1 add()

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

Definition at line 47 of file statsdb.hpp.

#### 7.10.3.2 as\_vector()

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

Definition at line 61 of file statsdb.hpp.

#### 7.10.3.3 clear()

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

Definition at line 83 of file statsdb.hpp.

#### 7.10.3.4 get\_data()

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

Definition at line 73 of file statsdb.hpp.

#### 7.10.3.5 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.



### 7.10.3.6 print()

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

Definition at line 102 of file statsdb.hpp.

### 7.10.3.7 reserve()

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

Definition at line 89 of file statsdb.hpp.

### 7.10.3.8 size()

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

Definition at line 113 of file statsdb.hpp.

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

- include/barry/statsdb.hpp

## 7.11 Geese Class Reference

Annotated Phyllo [Model](#).

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

### Public Member Functions

- [~Geese](#) ()
- void [init](#) ()
- void [inherit\\_support](#) (const [Geese](#) &model\_, bool delete\_support\_=false)
- void [calc\\_sequence](#) ([Node](#) \*n=nullptr)
- void [calc\\_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)

#### Construct a new Geese object

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

**Parameters**

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

- [Geese](#) ()
- [Geese](#) (std::vector< std::vector< unsigned int > > &annotations, std::vector< unsigned int > &geneid, std::vector< int > &parent, std::vector< bool > &duplication)
- [Geese](#) (const [Geese](#) &model\_, bool copy\_data=true)
- [Geese](#) ([Geese](#) &&x) noexcept
- [Geese](#) & operator= (const [Geese](#) &model\_)=delete
- [Geese](#) & operator= ([Geese](#) &&model\_) noexcept=delete

**Information about the model**

- unsigned int [nfuncs](#) () const noexcept
- unsigned int [nnodes](#) () const noexcept
- unsigned int [nleafs](#) () const noexcept
- unsigned int [nterms](#) () const

**Geese prediction**

*Calculate the conditional probability*

**Parameters**

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

When `res_prob` is specified, the function will attach the member vector probabilities from the [Nodes](#) objects. This contains the probability that the *i*th 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)

**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_counters = false`
- `bool delete_support = false`

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

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

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

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

### 7.11.1 Detailed Description

Annotated *Phylo Model*.

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

Definition at line 70 of file *geese-bones.hpp*.

### 7.11.2 Constructor & Destructor Documentation

#### 7.11.2.1 `Geese()` [1/4]

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

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

#### 7.11.2.2 `Geese()` [2/4]

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

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

### 7.11.2.3 Geese() [3/4]

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

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

### 7.11.2.4 Geese() [4/4]

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

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

### 7.11.2.5 ~Geese()

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

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

## 7.11.3 Member Function Documentation

### 7.11.3.1 calc\_reduced\_sequence()

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

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

### 7.11.3.2 calc\_sequence()

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

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

### 7.11.3.3 get\_probabilities()

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

Definition at line 277 of file geese-meat.hpp.

### 7.11.3.4 inherit\_support()

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

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

### 7.11.3.5 init()

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

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

### 7.11.3.6 init\_node()

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

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

### 7.11.3.7 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.

#### 7.11.3.8 likelihood\_exhaust()

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

Definition at line 7 of file geese-meat-likelihood\_exhaust.hpp.

#### 7.11.3.9 nfuncs()

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

Definition at line 293 of file geese-meat.hpp.

#### 7.11.3.10 nleafs()

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

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

#### 7.11.3.11 nnodes()

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

Definition at line 297 of file geese-meat.hpp.

#### 7.11.3.12 nterms()

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

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

#### 7.11.3.13 observed\_counts()

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

Definition at line 318 of file geese-meat.hpp.

#### 7.11.3.14 operator=() [1/2]

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

#### 7.11.3.15 operator=() [2/2]

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

#### 7.11.3.16 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 165 of file geese-meat-predict.hpp.

#### 7.11.3.17 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.

#### 7.11.3.18 print\_observed\_counts()

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

Definition at line 366 of file geese-meat.hpp.

#### 7.11.3.19 `set_seed()`

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

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

#### 7.11.3.20 `simulate()`

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

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

#### 7.11.3.21 `update_annotations()`

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

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

### 7.11.4 Member Data Documentation

#### 7.11.4.1 `counters`

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

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

#### 7.11.4.2 `delete_counters`

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

Definition at line 102 of file geese-bones.hpp.



#### 7.11.4.3 delete\_engine

```
bool Geese::delete_engine = false
```

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

#### 7.11.4.4 delete\_support

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

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

#### 7.11.4.5 initialized

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

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

#### 7.11.4.6 map\_to\_nodes

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

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

#### 7.11.4.7 nfunctions

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

Definition at line 91 of file geese-bones.hpp.

#### 7.11.4.8 nodes

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

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

#### 7.11.4.9 reduced\_sequence

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

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

#### 7.11.4.10 rengine

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

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

#### 7.11.4.11 sequence

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

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

#### 7.11.4.12 states

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

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

#### 7.11.4.13 support

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

Definition at line 86 of file geese-bones.hpp.

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

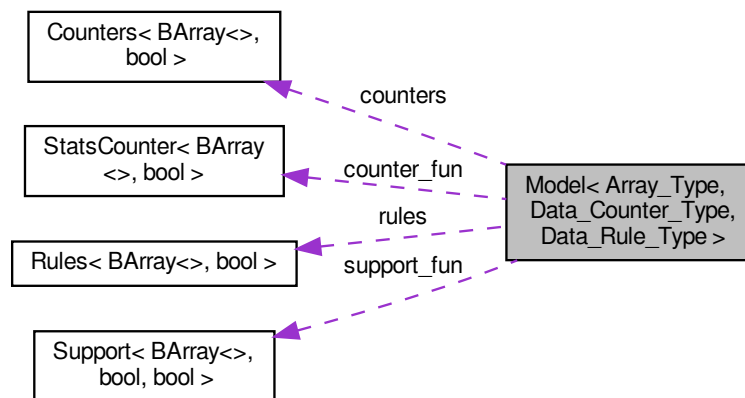
- [include/barry/models/geese/geese-bones.hpp](#)
- [include/barry/models/geese/geese-meat-constructors.hpp](#)
- [include/barry/models/geese/geese-meat-likelihood.hpp](#)
- [include/barry/models/geese/geese-meat-likelihood\\_exhaust.hpp](#)
- [include/barry/models/geese/geese-meat-predict.hpp](#)
- [include/barry/models/geese/geese-meat-simulate.hpp](#)
- [include/barry/models/geese/geese-meat.hpp](#)

## 7.12 Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type > Class Template Reference

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

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

Collaboration diagram for Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >:



### Public Member Functions

- [Model](#) ()
- [Model](#) (uint size\_)
- [Model](#) (const [Model](#)< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type > &Model\_)
- [Model](#)< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type > & [operator=](#) (const [Model](#)< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type > &Model\_)
- [~Model](#) ()
- void [store\\_psets](#) () 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)

### Wrappers for the `Counters` 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 `Rules` member.

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

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

#### Likelihood functions.

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

##### Parameters

|                     |                                                                   |
|---------------------|-------------------------------------------------------------------|
| <code>params</code> | Vector of parameters                                              |
| <code>as_log</code> | When <code>true</code> , 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

|                     |                                                                                |
|---------------------|--------------------------------------------------------------------------------|
| <code>i</code>      | Index relative to the array in the model.                                      |
| <code>params</code> | A new vector of model parameters to compute the normalizing constant.          |
| <code>as_log</code> | When <code>true</code> 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

## Public Attributes

- `std::vector< Counts\_type > stats`
- `std::vector< uint > n_arrays_per_stats`
- `MapVec\_type< double, uint > keys2support`  
*Map of types of arrays to support sets.*
- `std::vector< std::vector< double > > params\_last`  
*Vector of the previously used parameters.*
- `std::vector< double > normalizing\_constants`
- `std::vector< bool > first\_calc\_done`
- `std::function< std::vector< double > const Array_Type &> keygen = nullptr`  
*Function to extract features of the array to be hash.*

### Container space for the powerset (and its sufficient stats)

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

- `bool with\_pset = false`
- `std::vector< std::vector< Array_Type > > pset\_arrays`
- `std::vector< std::vector< std::vector< double > > > pset\_stats`
- `std::vector< std::vector< double > > pset\_probs`

### Information about the arrays used in the model

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

- `std::vector< std::vector< double > > target\_stats`
- `std::vector< uint > array\_frequency`
- `std::vector< uint > arrays2support`

### Functions to compute statistics

Arguments are recycled to save memory and computation.

- `Counters< Array_Type, Data_Counter_Type > counters`
- `Rules< Array_Type, Data_Rule_Type > rules`
- `Support< Array_Type, Data_Counter_Type, Data_Rule_Type > support\_fun`
- `StatsCounter< Array_Type, Data_Counter_Type > counter\_fun`

## Random number generation

Random number generation

- `std::mt19937 * engine = nullptr`
- `bool delete\_engine = false`
- `void set\_engine (std::mt19937 *engine_, bool delete_=false)`
- `void set\_seed (unsigned int s)`

### 7.12.1 Detailed Description

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

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

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

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

|                          |                                         |
|--------------------------|-----------------------------------------|
| <i>Array_Type</i>        | Class of <a href="#">BArray</a> object. |
| <i>Data_Counter_Type</i> | Any type.                               |
| <i>Data_Rule_Type</i>    | Any type.                               |

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

### 7.12.2 Constructor & Destructor Documentation

#### 7.12.2.1 Model() [1/3]

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

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

#### 7.12.2.2 Model() [2/3]

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

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

### 7.12.2.3 Model() [3/3]

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

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

### 7.12.2.4 ~Model()

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

Definition at line 197 of file model-bones.hpp.

## 7.12.3 Member Function Documentation

### 7.12.3.1 add\_array()

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

Adds an array to the support of not already included.

#### Parameters

|                  |                                                                                                                                                              |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Array_</i>    | array to be added                                                                                                                                            |
| <i>force_new</i> | If false, it will use <code>keygen</code> to obtain a double vector and create a hash of it. If the hash has been computed earlier, the support is recycled. |

#### Returns

The number of the array.

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

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

**7.12.3.2 add\_counter() [1/3]**

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

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

**7.12.3.3 add\_counter() [2/3]**

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

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

**7.12.3.4 add\_counter() [3/3]**

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

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

**7.12.3.5 add\_rule() [1/3]**

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

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

**7.12.3.6 add\_rule() [2/3]**

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

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



**7.12.3.7 add\_rule() [3/3]**

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

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

**7.12.3.8 get\_norm\_const()**

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

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

**7.12.3.9 get\_pset()**

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

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

**7.12.3.10 get\_stats()**

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

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

**7.12.3.11 likelihood()** [1/3]

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

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

**7.12.3.12 likelihood()** [2/3]

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

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

**7.12.3.13 likelihood()** [3/3]

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

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

**7.12.3.14 likelihood\_total()**

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

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

**7.12.3.15 nterms()**

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

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

**7.12.3.16 operator=()**

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

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

**7.12.3.17 print\_stats()**

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

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

**7.12.3.18 sample() [1/2]**

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

**7.12.3.19 sample() [2/2]**

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

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

**7.12.3.20 set\_counters()**

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

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

**7.12.3.21 set\_keygen()**

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

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

**7.12.3.22 set\_engine()**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::set_engine (
 std::mt19937 * engine_,
 bool delete_ = false) [inline]
```

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

**7.12.3.23 set\_rules()**

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

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

**7.12.3.24 set\_seed()**

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

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

### 7.12.3.25 size()

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

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

### 7.12.3.26 size\_unique()

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

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

### 7.12.3.27 store\_psets()

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

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

## 7.12.4 Member Data Documentation

### 7.12.4.1 array\_frequency

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

Definition at line 161 of file model-bones.hpp.

### 7.12.4.2 arrays2support

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

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

#### 7.12.4.3 counter\_fun

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool>
StatsCounter<Array_Type, Data_Counter_Type> Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::counter_fun
```

Definition at line 179 of file model-bones.hpp.

#### 7.12.4.4 counters

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

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

#### 7.12.4.5 delete\_rengine

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

Definition at line 112 of file model-bones.hpp.

#### 7.12.4.6 first\_calc\_done

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

Definition at line 185 of file model-bones.hpp.

#### 7.12.4.7 keygen

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

Function to extract features of the array to be hash.

Definition at line 189 of file model-bones.hpp.

#### 7.12.4.8 keys2support

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

Map of types of arrays to support sets.

This is of the same length as the vector stats.

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

#### 7.12.4.9 n\_arrays\_per\_stats

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

Definition at line 138 of file model-bones.hpp.

#### 7.12.4.10 normalizing\_constants

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

Definition at line 184 of file model-bones.hpp.

#### 7.12.4.11 params\_last

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

Vector of the previously used parameters.

Definition at line 183 of file model-bones.hpp.

#### 7.12.4.12 pset\_arrays

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool>
std::vector< std::vector< Array_Type > > Model< Array_Type, Data_Counter_Type, Data_Rule_Type >::pset_arrays
```

Definition at line 147 of file model-bones.hpp.

#### 7.12.4.13 pset\_probs

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

Definition at line 149 of file model-bones.hpp.

#### 7.12.4.14 pset\_stats

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

Definition at line 148 of file model-bones.hpp.

#### 7.12.4.15 rengine

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

Definition at line 111 of file model-bones.hpp.

#### 7.12.4.16 rules

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

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



#### 7.12.4.17 stats

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

Definition at line 137 of file model-bones.hpp.

#### 7.12.4.18 support\_fun

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

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

#### 7.12.4.19 target\_stats

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

Definition at line 160 of file model-bones.hpp.

#### 7.12.4.20 with\_pset

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

Definition at line 146 of file model-bones.hpp.

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

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

## 7.13 NetCounterData Class Reference

Data class used to store arbitrary uint or double vectors.

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

## Public Member Functions

- [NetCounterData](#) ()
- [NetCounterData](#) (const std::vector< [uint](#) > indices\_, const std::vector< double > numbers\_)
- [~NetCounterData](#) ()

## Public Attributes

- std::vector< [uint](#) > [indices](#)
- std::vector< double > [numbers](#)

### 7.13.1 Detailed Description

Data class used to store arbitrary uint or double vectors.

Definition at line 61 of file network.hpp.

### 7.13.2 Constructor & Destructor Documentation

#### 7.13.2.1 NetCounterData() [1/2]

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

Definition at line 67 of file network.hpp.

#### 7.13.2.2 NetCounterData() [2/2]

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

Definition at line 68 of file network.hpp.

#### 7.13.2.3 ~NetCounterData()

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

Definition at line 73 of file network.hpp.

### 7.13.3 Member Data Documentation

#### 7.13.3.1 indices

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

Definition at line 64 of file network.hpp.

#### 7.13.3.2 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](#)

## 7.14 NetworkData Class Reference

Data class for Networks.

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

### Public Member Functions

- [NetworkData](#) ()
- [NetworkData](#) (std::vector< double > vertex\_attr\_, bool directed\_=true)  
*Constructor using a single attribute.*
- [NetworkData](#) (std::vector< std::vector< double > > vertex\_attr\_, bool directed\_=true)  
*Constructor using multiple attributes.*
- [~NetworkData](#) ()

### Public Attributes

- bool [directed](#) = true
- std::vector< std::vector< double > > [vertex\\_attr](#)

### 7.14.1 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`.

### 7.14.2 Constructor & Destructor Documentation

#### 7.14.2.1 NetworkData() [1/3]

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

Definition at line 30 of file `network.hpp`.

#### 7.14.2.2 NetworkData() [2/3]

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

Constructor using a single attribute.

##### Parameters

|                     |                                                                      |
|---------------------|----------------------------------------------------------------------|
| <i>vertex_attr_</i> | Double vector of length equal to the number of vertices in the data. |
| <i>directed_</i>    | When <code>true</code> the graph as treated as directed.             |

Definition at line 38 of file `network.hpp`.

#### 7.14.2.3 NetworkData() [3/3]

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

Constructor using multiple attributes.

## Parameters

|                           |                                                                                                                                                                  |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>vertex_↵<br/>attr_</i> | 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. |
| <i>directed_</i>          | When <code>true</code> the graph as treated as directed.                                                                                                         |

Definition at line 50 of file network.hpp.

## 7.14.2.4 ~NetworkData()

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

Definition at line 56 of file network.hpp.

## 7.14.3 Member Data Documentation

## 7.14.3.1 directed

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

Definition at line 27 of file network.hpp.

## 7.14.3.2 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](#)

## 7.15 Node Class Reference

A single node for the model.

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

Collaboration diagram for Node:



## Public Member Functions

- [~Node](#) ()
- int [get\\_parent](#) () const
- bool [is\\_leaf](#) () const 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.*

### 7.15.1 Detailed Description

A single node for the model.

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

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

### 7.15.2 Constructor & Destructor Documentation

### 7.15.2.1 Node() [1/5]

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

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

### 7.15.2.2 Node() [2/5]

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

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

### 7.15.2.3 Node() [3/5]

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

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

### 7.15.2.4 Node() [4/5]

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

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

### 7.15.2.5 Node() [5/5]

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

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

#### 7.15.2.6 ~Node()

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

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

### 7.15.3 Member Function Documentation

#### 7.15.3.1 get\_parent()

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

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

#### 7.15.3.2 is\_leaf()

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

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

### 7.15.4 Member Data Documentation

#### 7.15.4.1 annotations

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

Observed annotations (only defined for [Geese](#))

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

#### 7.15.4.2 array

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

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



### 7.15.4.3 arrays

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

Arrays given all possible states.

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

### 7.15.4.4 duplication

```
bool Node::duplication
```

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

### 7.15.4.5 id

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

Id of the node (as specified in the input)

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

### 7.15.4.6 narray

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

ID of the array in the model.

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

### 7.15.4.7 offspring

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

Offspring nodes.

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

#### 7.15.4.8 ord

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

Order in which the node was created.

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

#### 7.15.4.9 parent

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

Parent node.

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

#### 7.15.4.10 probability

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

The probability of observing each state.

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

#### 7.15.4.11 subtree\_prob

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

Induced subtree probabilities.

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

#### 7.15.4.12 visited

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

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

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

- [include/barry/models/geese/geese-node-bones.hpp](#)

## 7.16 NodeData Class Reference

Data definition for the `PhyloArray` class.

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

### Public Member Functions

- [NodeData](#) ()
- [NodeData](#) (const std::vector< double > &lengths\_, const std::vector< bool > &states\_, bool duplication\_  
\_=true)
- [~NodeData](#) ()

### Public Attributes

- std::vector< double > [lengths](#)
- std::vector< bool > [states](#)
- bool [duplication](#) = true

#### 7.16.1 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`.

#### 7.16.2 Constructor & Destructor Documentation

##### 7.16.2.1 NodeData() [1/2]

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

Definition at line 41 of file `phylo.hpp`.

##### 7.16.2.2 NodeData() [2/2]

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

Definition at line 43 of file `phylo.hpp`.

### 7.16.2.3 ~NodeData()

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

Definition at line 49 of file phylo.hpp.

## 7.16.3 Member Data Documentation

### 7.16.3.1 blengths

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

Branch length.

Definition at line 29 of file phylo.hpp.

### 7.16.3.2 duplication

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

Definition at line 39 of file phylo.hpp.

### 7.16.3.3 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:

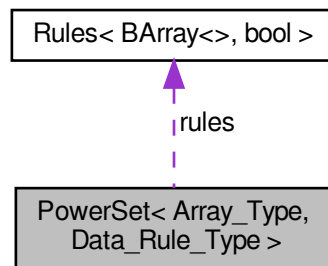
- [include/barry/counters/phylo.hpp](#)

## 7.17 PowerSet< Array\_Type, Data\_Rule\_Type > Class Template Reference

Powerset of a binary array.

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

Collaboration diagram for PowerSet< Array\_Type, Data\_Rule\_Type >:



### Public Member Functions

- void `init_support` ()
- void `calc` ()
- void `reset` (uint N\_, uint M\_)

### Construct and destroy a PowerSet object

- `PowerSet` ()
- `PowerSet` (uint N\_, uint M\_)
- `PowerSet` (const Array\_Type &array)
- `~PowerSet` ()

### Wrappers for the `Rules` 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](#)

### 7.17.1 Detailed Description

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

Powerset of a binary array.

Template Parameters

|                       |  |
|-----------------------|--|
| <i>Array_Type</i>     |  |
| <i>Data_Rule_Type</i> |  |

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

### 7.17.2 Constructor & Destructor Documentation

#### 7.17.2.1 PowerSet() [1/3]

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

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

#### 7.17.2.2 PowerSet() [2/3]

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

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

**7.17.2.3 PowerSet()** [3/3]

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

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

**7.17.2.4 ~PowerSet()**

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

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

**7.17.3 Member Function Documentation****7.17.3.1 add\_rule()** [1/3]

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

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

**7.17.3.2 add\_rule()** [2/3]

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

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

**7.17.3.3 add\_rule()** [3/3]

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

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

#### 7.17.3.4 begin()

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

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

#### 7.17.3.5 calc()

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

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

#### 7.17.3.6 end()

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

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

#### 7.17.3.7 get\_data()

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

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

#### 7.17.3.8 get\_data\_ptr()

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

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



### 7.17.3.9 init\_support()

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

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

### 7.17.3.10 operator[]()

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

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

### 7.17.3.11 reset()

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

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

### 7.17.3.12 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.

## 7.17.4 Member Data Documentation

### 7.17.4.1 coordinates\_free

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

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

#### 7.17.4.2 coordinates\_locked

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

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

#### 7.17.4.3 data

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

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

#### 7.17.4.4 EmptyArray

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

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

#### 7.17.4.5 M

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

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

#### 7.17.4.6 N

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

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

## 7.17.4.7 rules

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

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

## 7.17.4.8 rules\_deleted

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

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

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

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

## 7.18 Rule&lt; Array\_Type, Data\_Type &gt; Class Template Reference

[Rule](#) for determining if a cell should be included in a sequence.

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

## Public Member Functions

- [~Rule](#) ()
- bool [locked](#) (const Array\_Type &a, [uint](#) i, [uint](#) j)

## Construct a new Rule object

Construct a new [Rule](#) object

## Parameters

|                                |                                                                                                         |
|--------------------------------|---------------------------------------------------------------------------------------------------------|
| <code>fun_</code>              | <i>A function of type <code>Rule_fun_type</code>.</i>                                                   |
| <code>dat_</code>              | <i>Data pointer to be passed to <code>fun_</code></i>                                                   |
| <code>delete_↔<br/>dat_</code> | <i>When <code>true</code>, the <a href="#">Rule</a> destructor will delete the pointer, if defined.</i> |

- [Rule](#) ()
- [Rule](#) ([Rule\\_fun\\_type](#)< Array\_Type, Data\_Type > fun\_, Data\_Type \*dat\_=nullptr, bool delete\_dat\_=false)

## 7.18.1 Detailed Description

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

[Rule](#) for determining if a cell should be included in a sequence.

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

#### Template Parameters

|                   |                                             |
|-------------------|---------------------------------------------|
| <i>Array_Type</i> | An object of class <a href="#">BArray</a> . |
| <i>Data_Type</i>  | Any type.                                   |

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

## 7.18.2 Constructor & Destructor Documentation

### 7.18.2.1 Rule() [1/2]

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

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

### 7.18.2.2 Rule() [2/2]

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

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

### 7.18.2.3 ~Rule()

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

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

### 7.18.3 Member Function Documentation

#### 7.18.3.1 locked()

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

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

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

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

## 7.19 Rules< Array\_Type, Data\_Type > Class Template Reference

Vector of objects of class [Rule](#).

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

### Public Member Functions

- [Rules](#) ()
- [Rules](#) (const [Rules](#)< Array\_Type, Data\_Type > &rules\_)
- [Rules](#)< Array\_Type, Data\_Type > [operator=](#) (const [Rules](#)< Array\_Type, Data\_Type > &rules\_)
- [~Rules](#) ()
- [uint size](#) () const noexcept
- bool [locked](#) (const Array\_Type &a, [uint](#) i, [uint](#) j)  
*Check whether a given cell is free or locked.*
- void [clear](#) ()
- void [get\\_seq](#) (const Array\_Type &a, std::vector< std::pair< [uint](#), [uint](#) > > \*free, std::vector< std::pair< [uint](#), [uint](#) > > \*locked=nullptr)  
*Computes the sequence of free and locked cells in an [BArray](#).*

### Rule adding

#### Parameters

|      |  |
|------|--|
| rule |  |
|------|--|

- void [add\\_rule](#) ([Rule](#)< Array\_Type, Data\_Type > &rule)
- void [add\\_rule](#) ([Rule](#)< Array\_Type, Data\_Type > \*rule)

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

### 7.19.1 Detailed Description

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

Vector of objects of class `Rule`.

#### Template Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <i>Array_Type</i> | An object of class <code>BArray</code> |
| <i>Data_Type</i>  | Any type.                              |

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

### 7.19.2 Constructor & Destructor Documentation

#### 7.19.2.1 Rules() [1/2]

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

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

#### 7.19.2.2 Rules() [2/2]

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

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

#### 7.19.2.3 ~Rules()

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

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

### 7.19.3 Member Function Documentation

#### 7.19.3.1 add\_rule() [1/3]

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

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

#### 7.19.3.2 add\_rule() [2/3]

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

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

#### 7.19.3.3 add\_rule() [3/3]

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

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

#### 7.19.3.4 clear()

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

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

#### 7.19.3.5 get\_seq()

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

Computes the sequence of free and locked cells in an [BArray](#).

## Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>a</i>      | An object of class <a href="#">BArray</a> .                              |
| <i>free</i>   | Pointer to a vector of pairs (i, j) listing the free cells.              |
| <i>locked</i> | (optional) Pointer to a vector of pairs (i, j) listing the locked cells. |

## Returns

Nothing.

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

**7.19.3.6 locked()**

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

Check whether a given cell is free or locked.

## Parameters

|          |                                 |
|----------|---------------------------------|
| <i>a</i> | A <a href="#">BArray</a> object |
| <i>i</i> | row position                    |
| <i>j</i> | col position                    |

## Returns

true If the cell is locked  
false If the cell is free

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

**7.19.3.7 operator=()**

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

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



## 7.19.3.8 size()

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

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

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

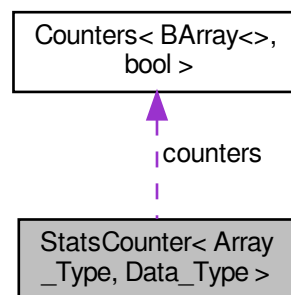
- include/barry/rules-bones.hpp
- include/barry/rules-meat.hpp

## 7.20 StatsCounter&lt; Array\_Type, Data\_Type &gt; Class Template Reference

Count stats for a single Array.

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

Collaboration diagram for StatsCounter< Array\_Type, Data\_Type >:



## Public Member Functions

- **StatsCounter** (const Array\_Type \*Array\_)  
*Creator of a StatsCounter*
- **StatsCounter** ()  
*Can be created without setting the array.*
- **~StatsCounter** ()
- void **reset\_array** (const Array\_Type \*Array\_)  
*Changes the reference array for the counting.*
- void **add\_counter** (Counter< Array\_Type, Data\_Type > \*f\_)
- void **add\_counter** (Counter< Array\_Type, Data\_Type > f\_)
- void **set\_counters** (Counters< Array\_Type, Data\_Type > \*counters\_)
- void **count\_init** (uint i, uint j)  
*Counter functions This function recurses through the entries of Array and at each step of adding a new cell it uses the functions to list the statistics.*
- void **count\_current** (uint i, uint j)
- std::vector< double > **count\_all** ()

## Public Attributes

- const Array\_Type \* [Array](#)
- Array\_Type [EmptyArray](#)
- std::vector< double > [current\\_stats](#)
- [Counters](#)< Array\_Type, Data\_Type > \* [counters](#)
- bool [counter\\_deleted](#) = false

### 7.20.1 Detailed Description

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
class StatsCounter< Array_Type, Data_Type >
```

Count stats for a single Array.

Users can a list of functions that can be used with this. The baseline set of arguments is a pointer to a binary array and a dataset to add the counts to.

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

### 7.20.2 Constructor & Destructor Documentation

#### 7.20.2.1 StatsCounter() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
StatsCounter< Array_Type, Data_Type >::StatsCounter (
 const Array_Type * Array_) [inline]
```

Creator of a [StatsCounter](#)

##### Parameters

|                         |                                               |
|-------------------------|-----------------------------------------------|
| <a href="#">Array</a> ↵ | A const pointer to a <a href="#">BArray</a> . |
| —                       |                                               |

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

#### 7.20.2.2 StatsCounter() [2/2]

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

Can be created without setting the array.

Definition at line 49 of file statscounter-bones.hpp.

### 7.20.2.3 ~StatsCounter()

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

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

## 7.20.3 Member Function Documentation

### 7.20.3.1 add\_counter() [1/2]

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

Definition at line 25 of file statscounter-meat.hpp.

### 7.20.3.2 add\_counter() [2/2]

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::add_counter (
 Counter< Array_Type, Data_Type > f_) [inline]
```

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

### 7.20.3.3 count\_all()

```
template<typename Array_Type , typename Data_Type >
std::vector< double > StatsCounter< Array_Type, Data_Type >::count_all [inline]
```

Definition at line 99 of file statscounter-meat.hpp.

### 7.20.3.4 count\_current()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::count_current (
 uint i,
 uint j) [inline]
```

Definition at line 81 of file statscounter-meat.hpp.

### 7.20.3.5 count\_init()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::count_init (
 uint i,
 uint j) [inline]
```

**Counter** functions This function recurses through the entries of `Array` and at each step of adding a new cell it uses the functions to list the statistics.

Definition at line 61 of file statscounter-meat.hpp.

### 7.20.3.6 reset\_array()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::reset_array (
 const Array_Type * Array_) [inline]
```

Changes the reference array for the counting.

#### Parameters

|                     |                                                          |
|---------------------|----------------------------------------------------------|
| <code>Array_</code> | A pointer to an array of class <code>Array_Type</code> . |
| —                   |                                                          |

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

### 7.20.3.7 set\_counters()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::set_counters (
 Counters< Array_Type, Data_Type > * counters_) [inline]
```

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

## 7.20.4 Member Data Documentation

### 7.20.4.1 Array

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
const Array_Type* StatsCounter< Array_Type, Data_Type >::Array
```

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

#### 7.20.4.2 counter\_deleted

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
bool StatsCounter< Array_Type, Data_Type >::counter_deleted = false
```

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

#### 7.20.4.3 counters

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Counters<Array_Type,Data_Type>* StatsCounter< Array_Type, Data_Type >::counters
```

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

#### 7.20.4.4 current\_stats

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::vector< double > StatsCounter< Array_Type, Data_Type >::current_stats
```

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

#### 7.20.4.5 EmptyArray

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Array_Type StatsCounter< Array_Type, Data_Type >::EmptyArray
```

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

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

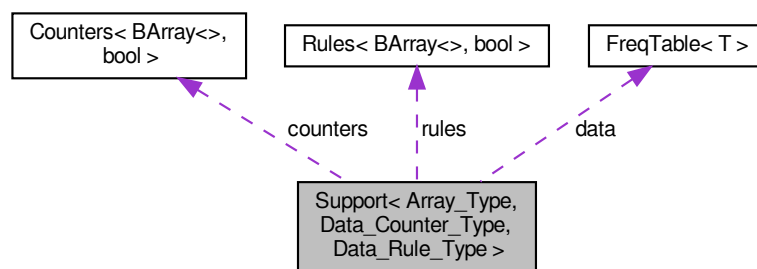
- [include/barry/statscounter-bones.hpp](#)
- [include/barry/statscounter-meat.hpp](#)

## 7.21 Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type > Class Template Reference

Compute the support of sufficient statistics.

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

Collaboration diagram for Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >:



### Public Member Functions

- [Support](#) (const Array\_Type &Array\_)  
*Constructor passing a reference Array.*
- [Support](#) (uint N\_, uint M\_)  
*Constructor specifying the dimensions of the array (empty).*
- [Support](#) ()
- [~Support](#) ()
- void [init\\_support](#) (std::vector< Array\_Type > \*array\_bank=nullptr, std::vector< std::vector< double > > \*stats\_bank=nullptr)
- void [calc](#) (std::vector< Array\_Type > \*array\_bank=nullptr, std::vector< std::vector< double > > \*stats\_↵ bank=nullptr, unsigned int max\_num\_elements\_=0u)  
*Computes the entire support.*
- [Counts\\_type](#) [get\\_counts](#) () const
- const [MapVec\\_type](#) \* [get\\_counts\\_ptr](#) () const
- void [print](#) () const

### Resets the support calculator

*If needed, the counters of a support object can be reused.*

#### Parameters

|         |                                                    |
|---------|----------------------------------------------------|
| Array_↵ | New array over which the support will be computed. |
| —       |                                                    |

- void [reset\\_array](#) ()
- void [reset\\_array](#) (const Array\_Type &Array\_)

## Manage counters

### Parameters

|           |                                   |
|-----------|-----------------------------------|
| f_        | A counter to be added.            |
| counters↔ | A vector of counters to be added. |
| —         |                                   |

- void [add\\_counter](#) (Counter< Array\_Type, Data\_Counter\_Type > \*f\_)
- void [add\\_counter](#) (Counter< Array\_Type, Data\_Counter\_Type > f\_)
- void [set\\_counters](#) (Counters< Array\_Type, Data\_Counter\_Type > \*counters\_)

## Manage rules

### Parameters

|           |                                |
|-----------|--------------------------------|
| f_        | A rule to be added.            |
| counters↔ | A vector of rules to be added. |
| —         |                                |

- void [add\\_rule](#) (Rule< Array\_Type, Data\_Rule\_Type > \*f\_)
- void [add\\_rule](#) (Rule< Array\_Type, Data\_Rule\_Type > f\_)
- void [set\\_rules](#) (Rules< Array\_Type, Data\_Rule\_Type > \*rules\_)

## Public Attributes

- Array\_Type [EmptyArray](#)  
Reference array to generate the support.
- [FreqTable](#) data
- Counters< Array\_Type, Data\_Counter\_Type > \* [counters](#)
- Rules< Array\_Type, Data\_Rule\_Type > \* [rules](#)
- [uint](#) N
- [uint](#) M
- bool [counter\\_deleted](#) = false
- bool [rules\\_deleted](#) = false
- [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](#)

### 7.21.1 Detailed Description

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

Compute the support of sufficient statistics.

Given an array and a set of counters, this object iterates throughout the support set of the Array while at the same time computing the support of the sufficient statistics.

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

## 7.21.2 Constructor & Destructor Documentation

### 7.21.2.1 Support() [1/3]

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

Constructor passing a reference Array.

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

### 7.21.2.2 Support() [2/3]

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

Constructor specifying the dimensions of the array (empty).

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

### 7.21.2.3 Support() [3/3]

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

Definition at line 70 of file support-bones.hpp.

### 7.21.2.4 ~Support()

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

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



### 7.21.3 Member Function Documentation

#### 7.21.3.1 add\_counter() [1/2]

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

Definition at line 178 of file support-meat.hpp.

#### 7.21.3.2 add\_counter() [2/2]

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

Definition at line 188 of file support-meat.hpp.

#### 7.21.3.3 add\_rule() [1/2]

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

Definition at line 215 of file support-meat.hpp.

#### 7.21.3.4 add\_rule() [2/2]

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

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

#### 7.21.3.5 calc()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::calc (
 std::vector< Array_Type > * array_bank = nullptr,
 std::vector< std::vector< double > > * stats_bank = nullptr,
 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

|                   |                                                                                                  |
|-------------------|--------------------------------------------------------------------------------------------------|
| <i>array_bank</i> | If specified, the counter will add to the vector each possible state of the array, as it counts. |
| <i>stats_bank</i> | If specified, the counter will add to the vector each possible set of statistics, as it counts.  |

Definition at line 152 of file support-meat.hpp.

### 7.21.3.6 get\_counts()

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

Definition at line 252 of file support-meat.hpp.

### 7.21.3.7 get\_counts\_ptr()

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

Definition at line 259 of file support-meat.hpp.

### 7.21.3.8 init\_support()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::init_support (
 std::vector< Array_Type > * array_bank = nullptr,
 std::vector< std::vector< double > > * stats_bank = nullptr) [inline]
```

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

### 7.21.3.9 print()

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

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

**7.21.3.10 reset\_array() [1/2]**

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

Definition at line 67 of file support-meat.hpp.

**7.21.3.11 reset\_array() [2/2]**

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

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

**7.21.3.12 set\_counters()**

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

Definition at line 198 of file support-meat.hpp.

**7.21.3.13 set\_rules()**

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

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

**7.21.4 Member Data Documentation****7.21.4.1 change\_stats**

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

Definition at line 52 of file support-bones.hpp.

#### 7.21.4.2 coordinates\_free

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

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

#### 7.21.4.3 coordinates\_locked

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

Definition at line 51 of file support-bones.hpp.

#### 7.21.4.4 counter\_deleted

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::counter_deleted = false
```

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

#### 7.21.4.5 counters

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool>
Counters<Array_Type,Data_Counter_Type>* Support< Array_Type, Data_Counter_Type, Data_Rule_↵
Type >::counters
```

Definition at line 40 of file support-bones.hpp.

#### 7.21.4.6 current\_stats

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

Definition at line 49 of file support-bones.hpp.

#### 7.21.4.7 data

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool>
FreqTable Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::data
```

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

#### 7.21.4.8 EmptyArray

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool>
Array_Type Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::EmptyArray
```

Reference array to generate the support.

Definition at line 38 of file support-bones.hpp.

#### 7.21.4.9 M

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

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

#### 7.21.4.10 max\_num\_elements

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::max_num_elements = BARRY_MAX_NUM_ELEMENTS
```

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

#### 7.21.4.11 N

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type >::N
```

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

#### 7.21.4.12 rules

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

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

#### 7.21.4.13 rules\_deleted

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

Definition at line 45 of file support-bones.hpp.

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

- [include/barry/support-bones.hpp](#)
- [include/barry/support-meat.hpp](#)

## 7.22 vecHasher< T > Struct Template Reference

```
#include <typedefs.hpp>
```

### Public Member Functions

- `std::size_t operator() (std::vector< T > const &dat) const` noexcept

#### 7.22.1 Detailed Description

```
template<typename T>
struct vecHasher< T >
```

Definition at line 98 of file typedefs.hpp.

#### 7.22.2 Member Function Documentation

##### 7.22.2.1 operator()()

```
template<typename T >
std::size_t vecHasher< T >::operator() (
 std::vector< T > const & dat) const [inline], [noexcept]
```

Definition at line 99 of file typedefs.hpp.

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

- [include/barry/typedefs.hpp](#)

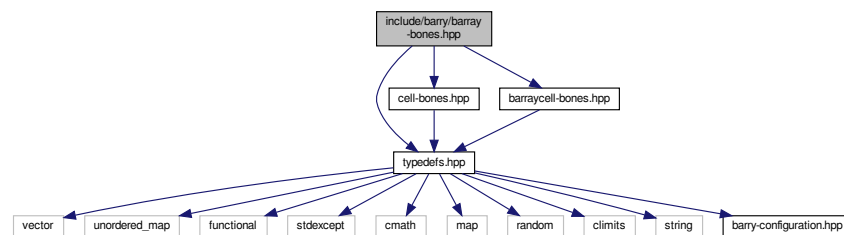
## Chapter 8

# File Documentation

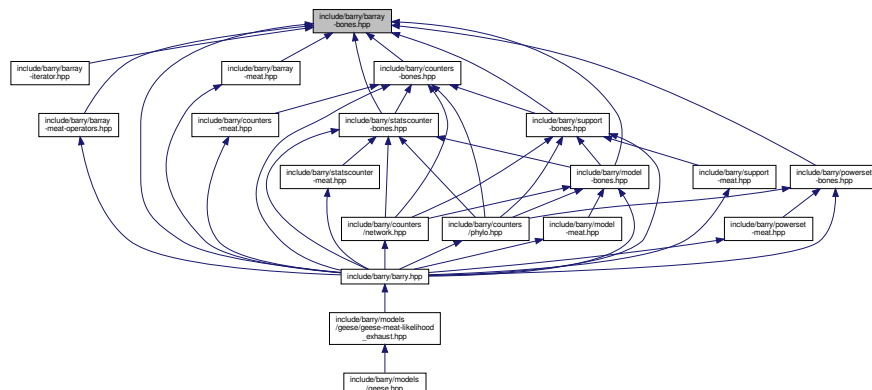
### 8.1 include/barry/barray-bones.hpp File Reference

```
#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
```

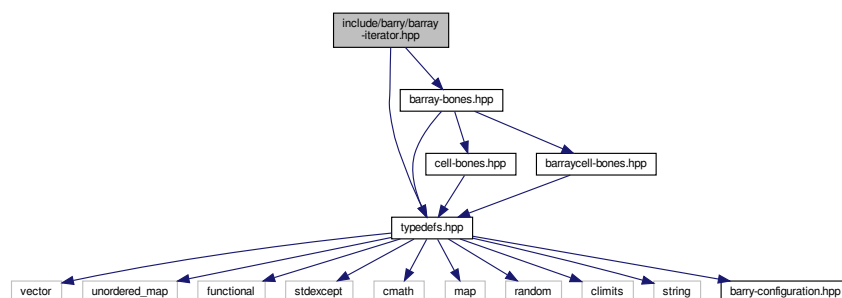
Definition at line 8 of file `barray-bones.hpp`.

## 8.2 include/barry/barray-iterator.hpp File Reference

```
#include "typedefs.hpp"
```

```
#include "barray-bones.hpp"
```

Include dependency graph for `barray-iterator.hpp`:



## Classes

- class [ConstBArrayRowIter< Cell\\_Type, Data\\_Type >](#)

## Macros

- `#define` [BARRAY\\_ITERATOR\\_HPP](#) 1



## 8.2.1 Macro Definition Documentation

### 8.2.1.1 BARRAY\_ITERATOR\_HPP

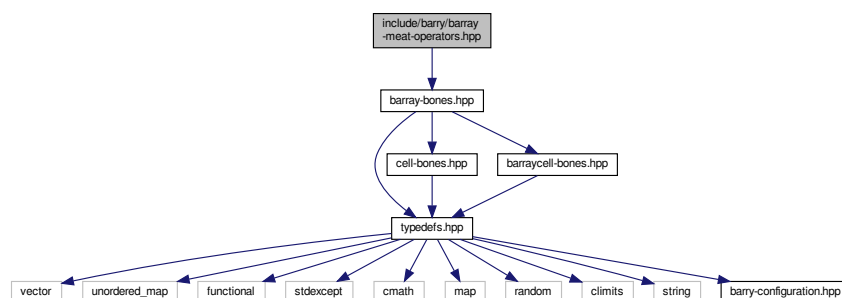
```
#define BARRAY_ITERATOR_HPP 1
```

Definition at line 7 of file barray-iterator.hpp.

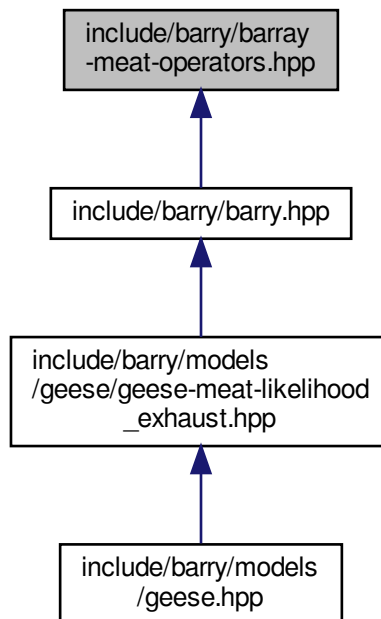
## 8.3 include/barry/barray-meat-operators.hpp File Reference

```
#include "barray-bones.hpp"
```

Include dependency graph for barray-meat-operators.hpp:



This graph shows which files directly or indirectly include this file:



## 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 Function Documentation

#### 8.3.1.1 checkdim\_()

```

template<typename Cell_Type , typename Data_Type >
void checkdim_ (
 const BArray< Cell_Type, Data_Type > & lhs,
 const BArray< Cell_Type, Data_Type > & rhs) [inline]

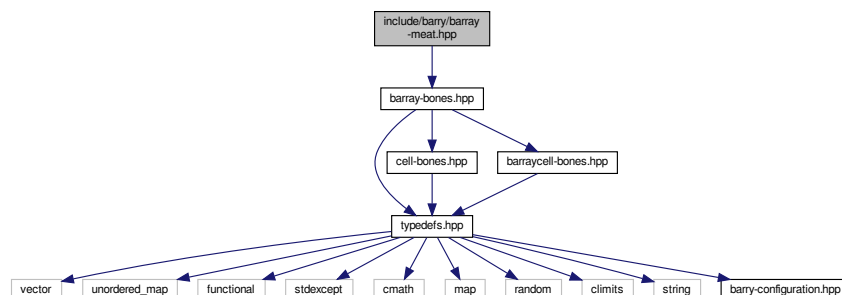
```

Definition at line 8 of file `barray-meat-operators.hpp`.

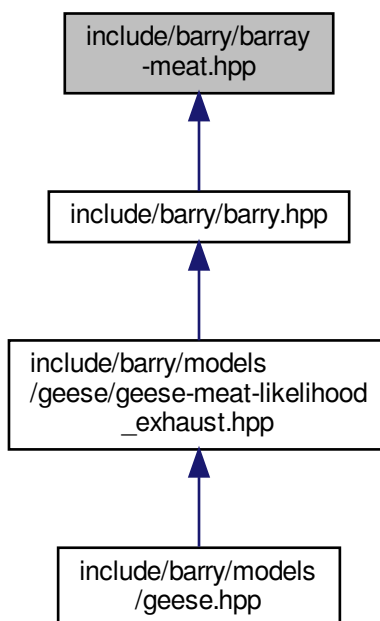
## 8.4 include/barry/barray-meat.hpp File Reference

```
#include "barray-bones.hpp"
```

Include dependency graph for barray-meat.hpp:



This graph shows which files directly or indirectly include this file:

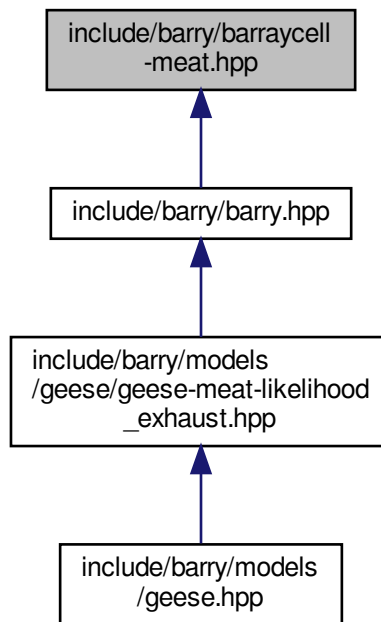


## 8.5 include/barry/barraycell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

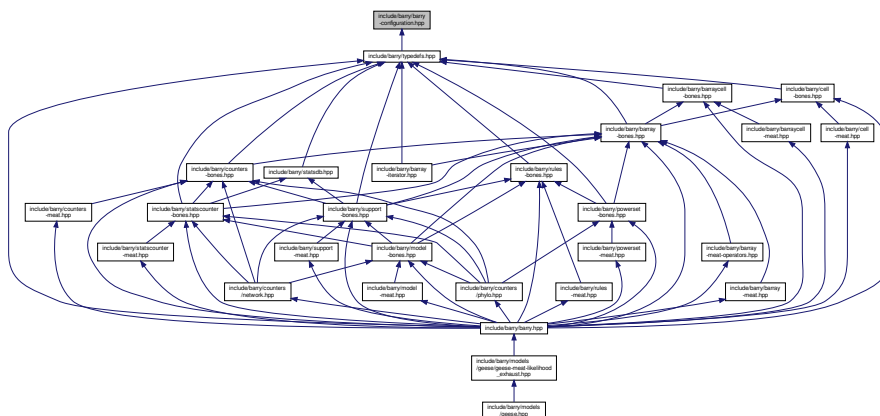


This graph shows which files directly or indirectly include this file:



## 8.7 include/barry/barry-configuration.hpp File Reference

This graph shows which files directly or indirectly include this file:



### Macros

- #define `BARRY_MAX_NUM_ELEMENTS` `static_cast< unsigned int >(UINT_MAX/2u)`

## Configuration MACROS

These are mostly related to performance. The definitions follow:

- `BARRY_USE_UNORDERED_MAP` If specified, then `barry` is compiled using `std::unordered_map`. Otherwise it will use `std::map` for the arrays.
- `BARRY_USE_SAFE_EXP` When specified, it will multiply all likelihoods in `Model` by  $(1/-100)/(1/-100)$  so that numerical overflows are avoided.
- `BARRY_CHECK_FINITE` When specified, it will introduce a macro
  - `#define BARRY_SAFE_EXP -100.0`
  - `#define BARRY_ISFINITE(a)`
  - `#define BARRY_CHECK_SUPPORT(x, maxs)`
  - `template<typename Ta, typename Tb >`  
`using Map = std::map< Ta, Tb >`

### 8.7.1 Macro Definition Documentation

#### 8.7.1.1 BARRY\_CHECK\_SUPPORT

```
#define BARRY_CHECK_SUPPORT(
 x,
 maxs)
```

Definition at line 46 of file `barry-configuration.hpp`.

#### 8.7.1.2 BARRY\_ISFINITE

```
#define BARRY_ISFINITE(
 a)
```

Definition at line 39 of file `barry-configuration.hpp`.

#### 8.7.1.3 BARRY\_MAX\_NUM\_ELEMENTS

```
#define BARRY_MAX_NUM_ELEMENTS static_cast< unsigned int >(UINT_MAX/2u)
```

Definition at line 5 of file `barry-configuration.hpp`.

#### 8.7.1.4 BARRY\_SAFE\_EXP

```
#define BARRY_SAFE_EXP -100.0
```

Definition at line 32 of file barry-configuration.hpp.

### 8.7.2 Typedef Documentation

#### 8.7.2.1 Map

```
template<typename Ta , typename Tb >
using Map = std::map<Ta,Tb>
```

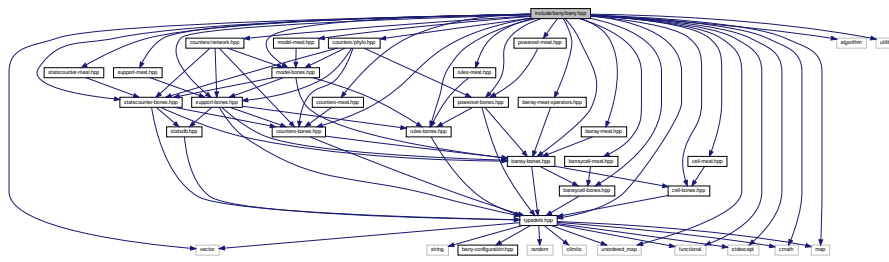
Definition at line 26 of file barry-configuration.hpp.

## 8.8 include/barry/barry.hpp File Reference

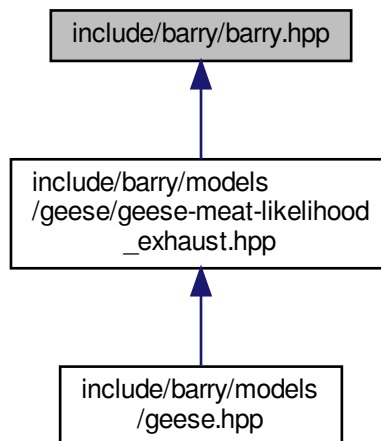
```
#include <vector>
#include <unordered_map>
#include <functional>
#include <stdexcept>
#include <cmath>
#include <map>
#include <algorithm>
#include <utility>
#include "typedefs.hpp"
#include "cell-bones.hpp"
#include "cell-meat.hpp"
#include "barray-bones.hpp"
#include "barraycell-bones.hpp"
#include "barray-meat.hpp"
#include "barraycell-meat.hpp"
#include "barray-meat-operators.hpp"
#include "counters-bones.hpp"
#include "counters-meat.hpp"
#include "statscounter-bones.hpp"
#include "statscounter-meat.hpp"
#include "support-bones.hpp"
#include "support-meat.hpp"
#include "powerset-bones.hpp"
#include "powerset-meat.hpp"
#include "model-bones.hpp"
#include "model-meat.hpp"
#include "rules-bones.hpp"
#include "rules-meat.hpp"
#include "counters/network.hpp"
```

```
#include "counters/phylo.hpp"
```

Include dependency graph for barry.hpp:



This graph shows which files directly or indirectly include this file:



## Namespaces

- [barry](#)  
*barry: Your go-to motif accountant*
- [barry::counters](#)  
*Tree class and Treeliterator class.*
- [barry::counters::network](#)
- [barry::counters::phylo](#)

## Macros

- `#define COUNTER_FUNCTION(a)`
- `#define COUNTER_LAMBDA(a)`
- `#define RULE_FUNCTION(a)`
- `#define RULE_LAMBDA(a)`



## 8.8.1 Macro Definition Documentation

### 8.8.1.1 COUNTER\_FUNCTION

```
#define COUNTER_FUNCTION(
 a)
```

**Value:**

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
inline double (a) (const Array_Type & Array, uint i, uint j, Data_Type * data) \
{
```

Definition at line 64 of file barry.hpp.

### 8.8.1.2 COUNTER\_LAMBDA

```
#define COUNTER_LAMBDA(
 a)
```

**Value:**

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
Counter_fun_type<Array_Type, Data_Type> a = \
{ [] (const Array_Type & Array, uint i, uint j, Data_Type * data)
```

Definition at line 67 of file barry.hpp.

### 8.8.1.3 RULE\_FUNCTION

```
#define RULE_FUNCTION(
 a)
```

**Value:**

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
inline bool (a) (const Array_Type & Array, uint i, uint j, Data_Type * data) \
{
```

Definition at line 71 of file barry.hpp.

### 8.8.1.4 RULE\_LAMBDA

```
#define RULE_LAMBDA(
 a)
```

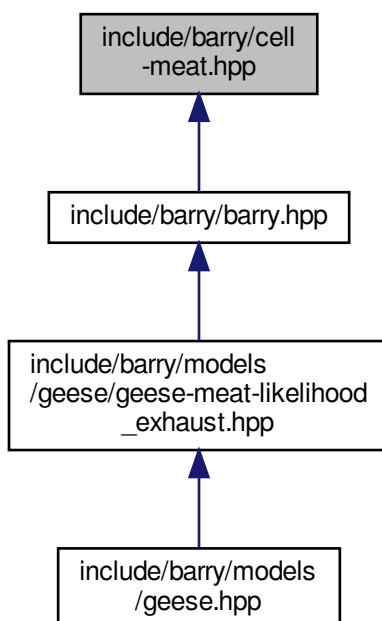
**Value:**

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
Rule_fun_type<Array_Type, Data_Type> a = \
{ [] (const Array_Type & Array, uint i, uint j, Data_Type * data)
```

Definition at line 74 of file barry.hpp.



This graph shows which files directly or indirectly include this file:



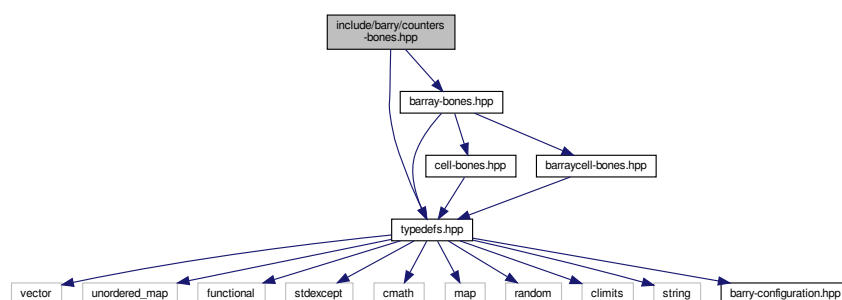
## 8.11 include/barry/col-bones.hpp File Reference

## 8.12 include/barry/counters-bones.hpp File Reference

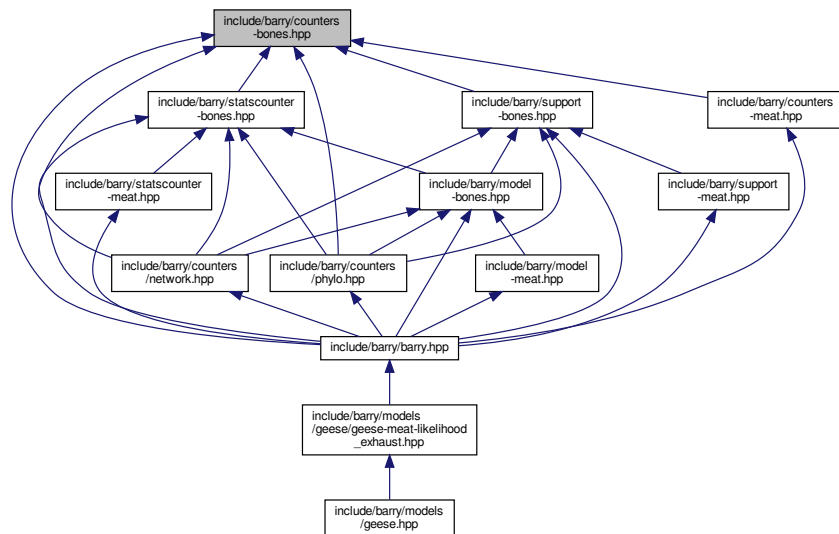
```
#include "typedefs.hpp"
```

```
#include "barray-bones.hpp"
```

Include dependency graph for counters-bones.hpp:



This graph shows which files directly or indirectly include this file:



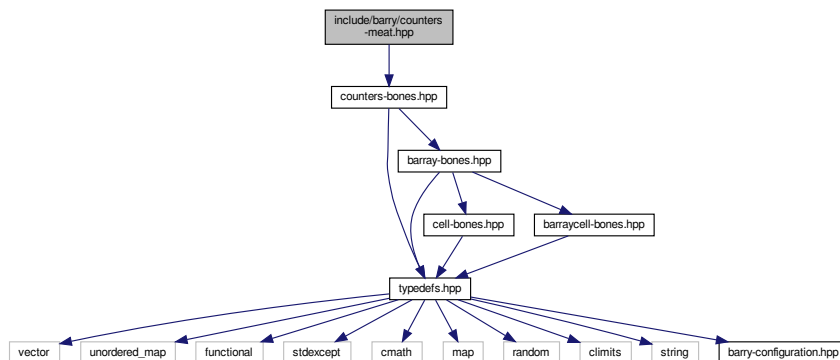
## Classes

- class [Counter< Array\\_Type, Data\\_Type >](#)  
*A counter function based on change statistics.*
- class [Counters< Array\\_Type, Data\\_Type >](#)  
*Vector of counters.*

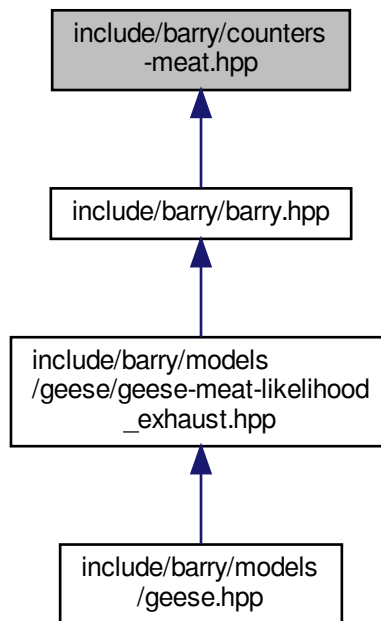
## 8.13 include/barry/counters-meat.hpp File Reference

```
#include "counters-bones.hpp"
```

Include dependency graph for counters-meat.hpp:



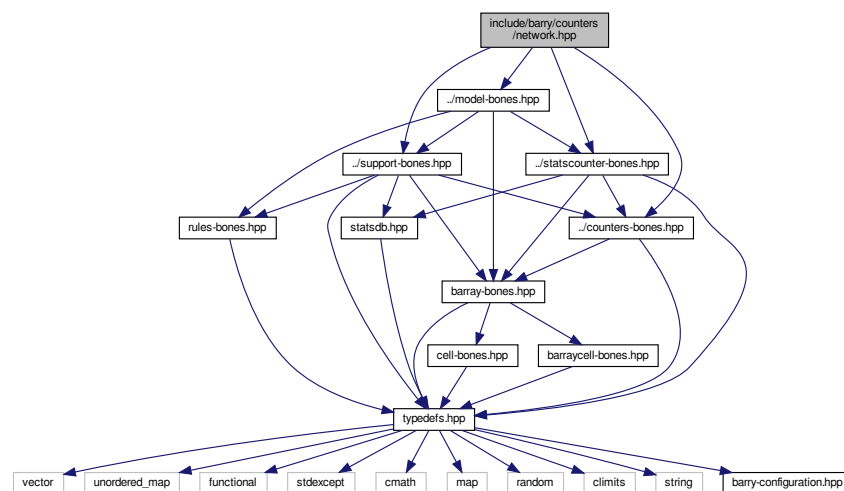
This graph shows which files directly or indirectly include this file:



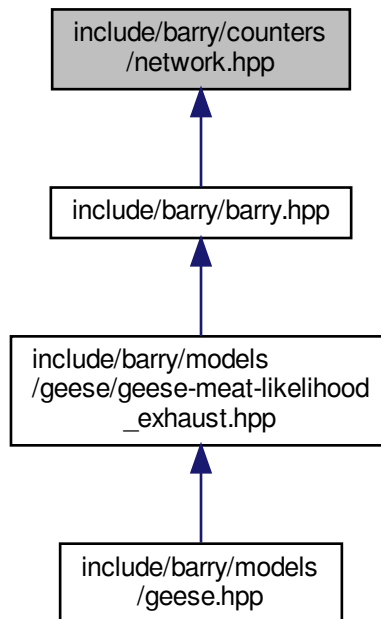
## 8.14 include/barry/counters/network.hpp File Reference

```
#include "../counters-bones.hpp"
#include "../support-bones.hpp"
#include "../statscounter-bones.hpp"
#include "../model-bones.hpp"
```

Include dependency graph for network.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [NetworkData](#)  
*Data class for Networks.*
- class [NetCounterData](#)  
*Data class used to store arbitrary uint or double vectors.*

## Macros

- #define [NET\\_C\\_DATA\\_IDX](#)(i) (data->indices[i])
- #define [NET\\_C\\_DATA\\_NUM](#)(i) (data->numbers[i])

### Macros for defining counters

- #define [NETWORK\\_COUNTER](#)(a)
- #define [NETWORK\\_COUNTER\\_LAMBDA](#)(a)

### Macros for defining rules

- #define [NETWORK\\_RULE](#)(a)
- #define [NETWORK\\_RULE\\_LAMBDA](#)(a)

## Typedefs

Convenient typedefs for network objects.

- typedef [BArray](#)< double, [NetworkData](#) > [Network](#)
- typedef [Counter](#)< [Network](#), [NetCounterData](#) > [NetCounter](#)
- typedef [Counters](#)< [Network](#), [NetCounterData](#) > [NetCounters](#)
- typedef [Support](#)< [Network](#), [NetCounterData](#) > [NetSupport](#)
- typedef [StatsCounter](#)< [Network](#), [NetCounterData](#) > [NetStatsCounter](#)
- typedef [Model](#)< [Network](#), [NetCounterData](#) > [NetModel](#)
- typedef [Rule](#)< [Network](#), bool > [NetRule](#)
- typedef [Rules](#)< [Network](#), bool > [NetRules](#)

## Functions

- 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\\_nodcov](#) ([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 <a href="#">NetRules</a> object ( <a href="#">Rules</a> < <a href="#">Network</a> , bool>). |
|-------|------------------------------------------------------------------------------------------------------------|

- void [rules\\_zerodiag](#) ([NetRules](#) \*rules)  
*Number of edges.*

## 8.14.1 Macro Definition Documentation

### 8.14.1.1 NET\_C\_DATA\_IDX

```
#define NET_C_DATA_IDX(
 i) (data->indices[i])
```

Definition at line 79 of file network.hpp.

### 8.14.1.2 NET\_C\_DATA\_NUM

```
#define NET_C_DATA_NUM(
 i) (data->numbers[i])
```

Definition at line 80 of file network.hpp.

### 8.14.1.3 NETWORK\_COUNTER

```
#define NETWORK_COUNTER(
 a)
```

**Value:**

```
inline double (a) \
(const Network & Array, uint i, uint j, NetCounterData * data)
```

Function for definition of a network counter function

Definition at line 101 of file network.hpp.

### 8.14.1.4 NETWORK\_COUNTER\_LAMBDA

```
#define NETWORK_COUNTER_LAMBDA(
 a)
```

**Value:**

```
Counter_fun_type<Network, NetCounterData> a = \
[] (const Network & Array, uint i, uint j, NetCounterData * data)
```

Lambda function for definition of a network counter function

Definition at line 104 of file network.hpp.



### 8.14.1.5 NETWORK\_RULE

```
#define NETWORK_RULE(
 a)
```

**Value:**

```
inline bool (a) \
(const Network & Array, uint i, uint j, bool * data)
```

Function for definition of a network counter function

Definition at line 113 of file network.hpp.

### 8.14.1.6 NETWORK\_RULE\_LAMBDA

```
#define NETWORK_RULE_LAMBDA(
 a)
```

**Value:**

```
Rule_fun_type<Network, bool> a = \
[] (const Network & Array, uint i, uint j, bool * data)
```

Lambda function for definition of a network counter function

Definition at line 116 of file network.hpp.

## 8.14.2 Typedef Documentation

### 8.14.2.1 NetCounter

```
typedef Counter<Network, NetCounterData > NetCounter
```

Definition at line 88 of file network.hpp.

### 8.14.2.2 NetCounters

```
typedef Counters< Network, NetCounterData> NetCounters
```

Definition at line 89 of file network.hpp.

#### 8.14.2.3 NetModel

```
typedef Model<Network, NetCounterData> NetModel
```

Definition at line 92 of file network.hpp.

#### 8.14.2.4 NetRule

```
typedef Rule<Network, bool> NetRule
```

Definition at line 93 of file network.hpp.

#### 8.14.2.5 NetRules

```
typedef Rules<Network, bool> NetRules
```

Definition at line 94 of file network.hpp.

#### 8.14.2.6 NetStatsCounter

```
typedef StatsCounter<Network, NetCounterData> NetStatsCounter
```

Definition at line 91 of file network.hpp.

#### 8.14.2.7 NetSupport

```
typedef Support<Network, NetCounterData > NetSupport
```

Definition at line 90 of file network.hpp.

#### 8.14.2.8 Network

```
typedef BArray<double, NetworkData> Network
```

Definition at line 87 of file network.hpp.

### 8.14.3 Function Documentation

#### 8.14.3.1 rules\_zerodiag()

```
void rules_zerodiag (
 NetRules * rules) [inline]
```

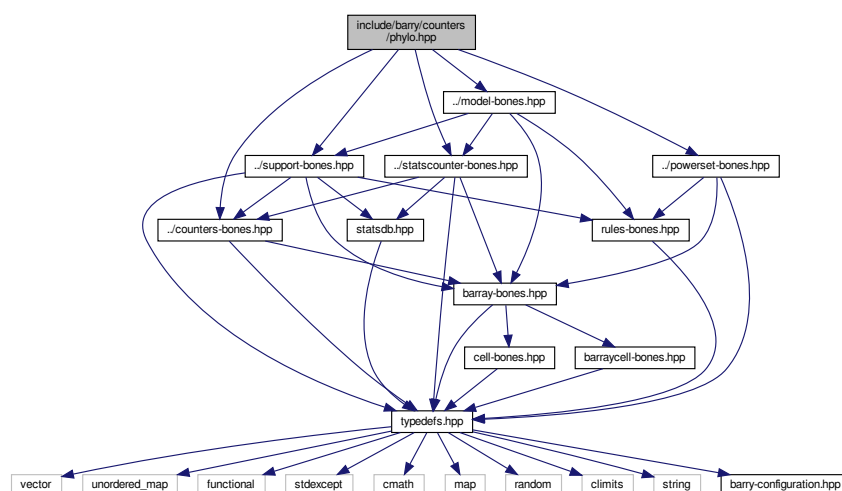
Number of edges.

Definition at line 742 of file network.hpp.

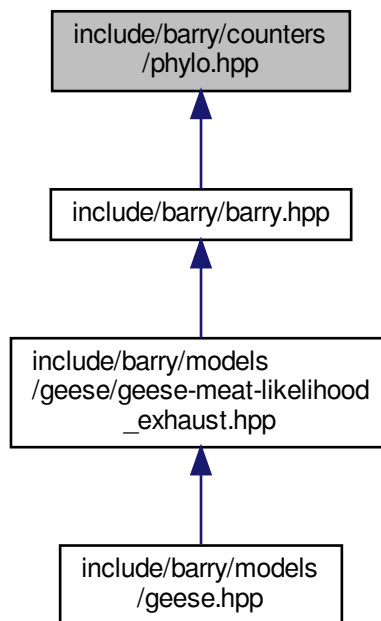
## 8.15 include/barry/counters/phylo.hpp File Reference

```
#include "../counters-bones.hpp"
#include "../support-bones.hpp"
#include "../statscounter-bones.hpp"
#include "../model-bones.hpp"
#include "../powerset-bones.hpp"
```

Include dependency graph for phylo.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [NodeData](#)

*Data definition for the `PhyloArray` class.*

## Macros

- `#define PHYLO_COUNTER(a)`  
*Extension of a simple counter.*
- `#define PHYLO_COUNTER_LAMBDA(a)`
- `#define PHYLO_CHECK_MISSING()`

## Typedefs

**Convenient typedefs for Node objects.**

- `typedef BArray< uint, NodeData > PhyloArray`
- `typedef Counter< PhyloArray, PhyloCounterData > PhyloCounter`
- `typedef Counters< PhyloArray, PhyloCounterData > PhyloCounters`
- `typedef Rule< PhyloArray, PhyloRuleData > PhyloRule`
- `typedef Rules< PhyloArray, PhyloRuleData > PhyloRules`
- `typedef Support< PhyloArray, PhyloCounterData, PhyloRuleData > PhyloSupport`
- `typedef StatsCounter< PhyloArray, PhyloCounterData > PhyloStatsCounter`
- `typedef Model< PhyloArray, PhyloCounterData, PhyloRuleData > PhyloModel`
- `typedef PowerSet< PhyloArray, PhyloRuleData > PhyloPowerSet`

## Functions

- 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_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.*
  
- #define `PHYLO_C_DATA_IDX(i)` (data.operator[])(i)
- typedef std::vector< uint > `PhyloCounterData`
- typedef std::vector< std::pair< uint, uint > > `PhyloRuleData`

### 8.15.1 Macro Definition Documentation

#### 8.15.1.1 PHYLO\_C\_DATA\_IDX

```
#define PHYLO_C_DATA_IDX(
 i) (data.operator[])(i)
```

Definition at line 56 of file phylo.hpp.

### 8.15.1.2 PHYLO\_CHECK\_MISSING

```
#define PHYLO_CHECK_MISSING()
```

#### Value:

```
if (Array.data == nullptr) \
 throw std::logic_error("The array data is nullptr."); \
if (data == nullptr) \
 throw std::logic_error("The counter data is nullptr.")
```

Definition at line 91 of file phylo.hpp.

### 8.15.1.3 PHYLO\_COUNTER

```
#define PHYLO_COUNTER(
 a)
```

#### Value:

```
inline double (a) (const PhyloArray & Array, uint i, \
 uint j, PhyloCounterData * data)
```

Extension of a simple counter.

It allows specifying extra arguments, in particular, the corresponding sets of rows to which this statistic may be relevant. This could be important in the case of, for example, counting correlation type statistics between function 1 and 2, and between function 1 and 3.

Definition at line 85 of file phylo.hpp.

### 8.15.1.4 PHYLO\_COUNTER\_LAMBDA

```
#define PHYLO_COUNTER_LAMBDA(
 a)
```

#### Value:

```
Counter_fun_type<PhyloArray, PhyloCounterData> a = \
 [] (const PhyloArray & Array, uint i, uint j, PhyloCounterData * data)
```

Definition at line 88 of file phylo.hpp.

## 8.15.2 Typedef Documentation

### 8.15.2.1 PhyloArray

```
typedef BArray<uint, NodeData> PhyloArray
```

Definition at line 63 of file phylo.hpp.

### 8.15.2.2 PhyloCounter

```
typedef Counter<PhyloArray, PhyloCounterData > PhyloCounter
```

Definition at line 64 of file phylo.hpp.

### 8.15.2.3 PhyloCounterData

```
typedef std::vector< uint > PhyloCounterData
```

Definition at line 53 of file phylo.hpp.

### 8.15.2.4 PhyloCounters

```
typedef Counters< PhyloArray, PhyloCounterData> PhyloCounters
```

Definition at line 65 of file phylo.hpp.

### 8.15.2.5 PhyloModel

```
typedef Model<PhyloArray, PhyloCounterData, PhyloRuleData> PhyloModel
```

Definition at line 70 of file phylo.hpp.

### 8.15.2.6 PhyloPowerSet

```
typedef PowerSet<PhyloArray, PhyloRuleData> PhyloPowerSet
```

Definition at line 71 of file phylo.hpp.

### 8.15.2.7 PhyloRule

```
typedef Rule<PhyloArray,PhyloRuleData> PhyloRule
```

Definition at line 66 of file phylo.hpp.

#### 8.15.2.8 PhyloRuleData

```
typedef std::vector< std::pair< uint, uint > > PhyloRuleData
```

Definition at line 54 of file phylo.hpp.

#### 8.15.2.9 PhyloRules

```
typedef Rules<PhyloArray, PhyloRuleData> PhyloRules
```

Definition at line 67 of file phylo.hpp.

#### 8.15.2.10 PhyloStatsCounter

```
typedef StatsCounter<PhyloArray, PhyloCounterData> PhyloStatsCounter
```

Definition at line 69 of file phylo.hpp.

#### 8.15.2.11 PhyloSupport

```
typedef Support<PhyloArray, PhyloCounterData, PhyloRuleData> PhyloSupport
```

Definition at line 68 of file phylo.hpp.

### 8.15.3 Function Documentation

#### 8.15.3.1 get\_last\_name()

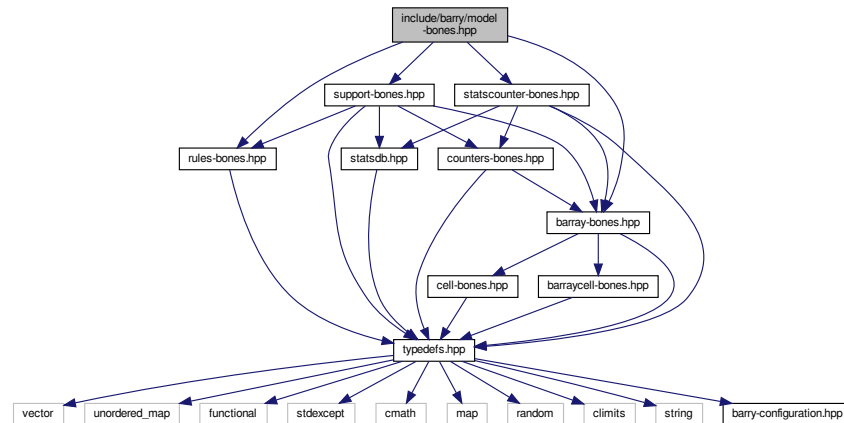
```
std::string get_last_name (
 bool d) [inline]
```

Definition at line 96 of file phylo.hpp.

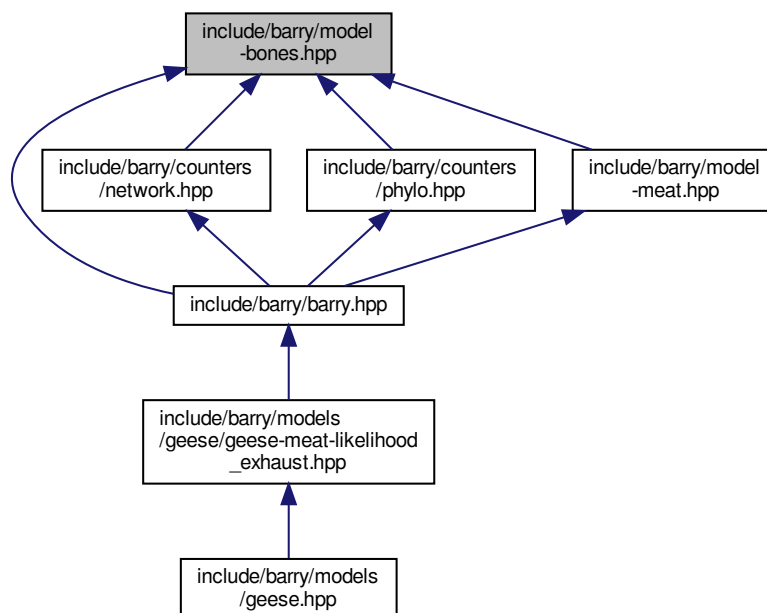


## 8.16 include/barry/model-bones.hpp File Reference

```
#include "barray-bones.hpp"
#include "support-bones.hpp"
#include "statscounter-bones.hpp"
#include "rules-bones.hpp"
Include dependency graph for model-bones.hpp:
```



This graph shows which files directly or indirectly include this file:



### Classes

- class [Model](#)< [Array\\_Type](#), [Data\\_Counter\\_Type](#), [Data\\_Rule\\_Type](#) >

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

## Functions

- double [update\\_normalizing\\_constant](#) (const std::vector< double > &params, const [Counts\\_type](#) &support)
- double [likelihood\\_](#) (const std::vector< double > &target\_stats, const std::vector< double > &params, const double normalizing\_constant, bool log\_=false)
- template<typename Array\_Type >  
std::vector< double > [keygen\\_default](#) (const Array\_Type &Array\_)  
*Array Hasher class (used for computing support)*

### 8.16.1 Function Documentation

#### 8.16.1.1 keygen\_default()

```
template<typename Array_Type >
std::vector< double > keygen_default (
 const Array_Type & Array_) [inline]
```

Array Hasher class (used for computing support)

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

#### 8.16.1.2 likelihood\_()

```
double likelihood_ (
 const std::vector< double > & target_stats,
 const std::vector< double > & params,
 const double normalizing_constant,
 bool log_ = false) [inline]
```

Definition at line 40 of file model-bones.hpp.

#### 8.16.1.3 update\_normalizing\_constant()

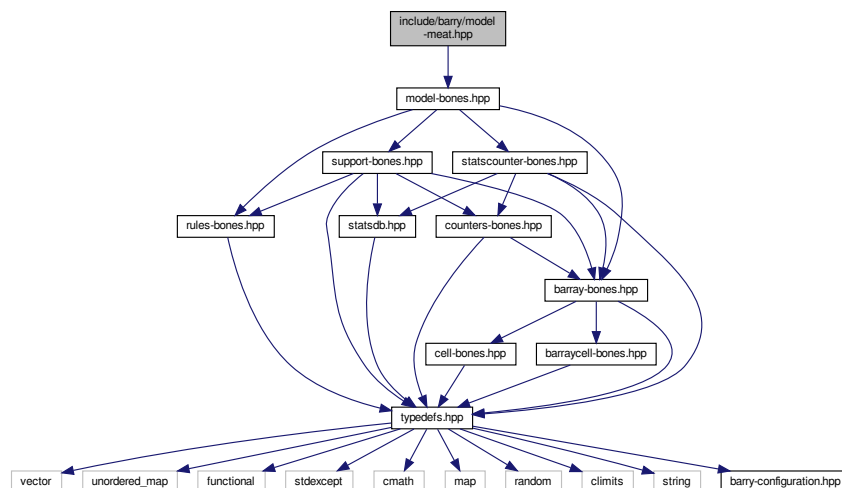
```
double update_normalizing_constant (
 const std::vector< double > & params,
 const Counts_type & support) [inline]
```

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

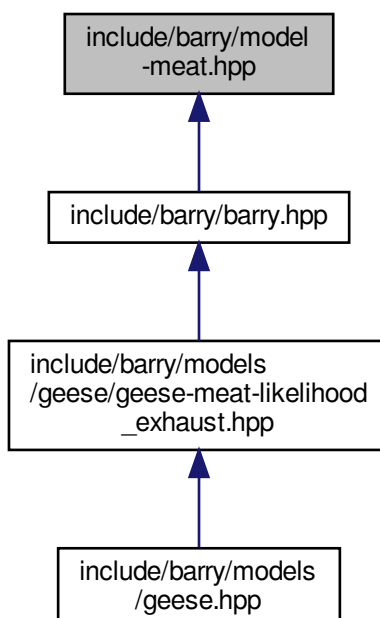
## 8.17 include/barry/model-meat.hpp File Reference

```
#include "model-bones.hpp"
```

Include dependency graph for model-meat.hpp:



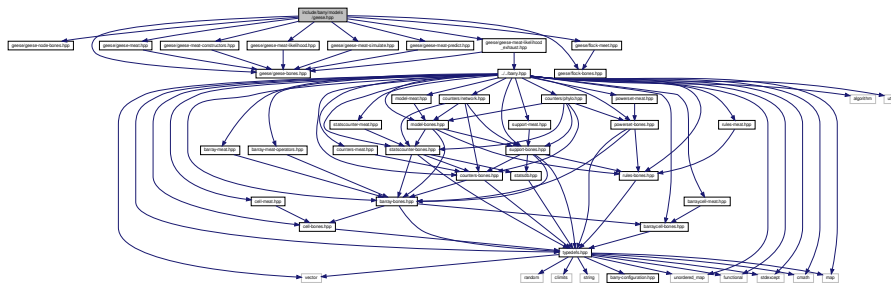
This graph shows which files directly or indirectly include this file:



## 8.18 include/barry/models/geese.hpp File Reference

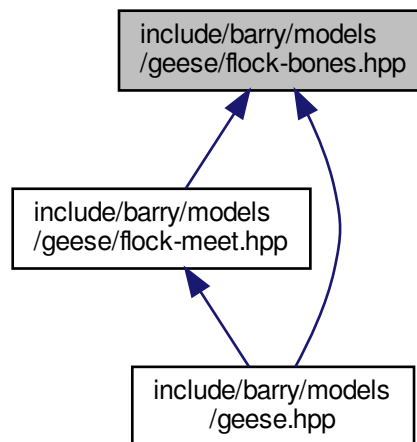
```
#include "geese/geese-node-bones.hpp"
#include "geese/geese-bones.hpp"
#include "geese/geese-meat.hpp"
#include "geese/geese-meat-constructors.hpp"
#include "geese/geese-meat-likelihood.hpp"
#include "geese/geese-meat-likelihood_exhaust.hpp"
#include "geese/geese-meat-simulate.hpp"
#include "geese/geese-meat-predict.hpp"
#include "geese/flock-bones.hpp"
#include "geese/flock-meet.hpp"
```

Include dependency graph for geese.hpp:



## 8.19 include/barry/models/geese/flock-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



### Classes

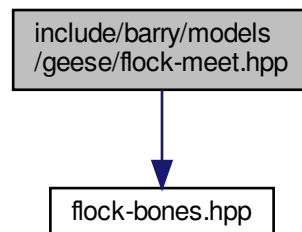
- class [Flock](#)

A [Flock](#) is a group of [Geese](#).

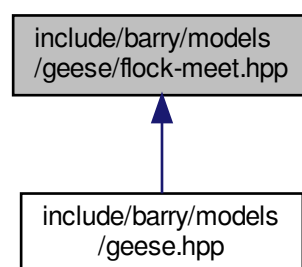
## 8.20 include/barry/models/geese/flock-meet.hpp File Reference

```
#include "flock-bones.hpp"
```

Include dependency graph for flock-meet.hpp:

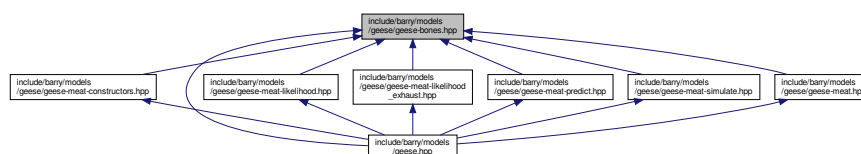


This graph shows which files directly or indirectly include this file:



## 8.21 include/barry/models/geese/geese-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



## Classes

- class [Geese](#)  
*Annotated Phylo [Model](#).*

## Macros

- `#define` [INITIALIZED\(\)](#)

## Functions

- `template<typename Ta , typename Tb >`  
`std::vector< Ta > vector\_caster (const std::vector< Tb > &x)`
- [RULE\\_FUNCTION](#) (rule\_empty\_free)
- `std::vector< double > keygen\_full (const phylocounters::PhyloArray &array)`
- `bool vec\_diff (const std::vector< unsigned int > &s, const std::vector< unsigned int > &a)`

### 8.21.1 Macro Definition Documentation

#### 8.21.1.1 INITIALIZED

```
#define INITIALIZED()
```

**Value:**

```
if (!this->initialized) \
 throw std::logic_error("The model has not been initialized yet.");
```

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

### 8.21.2 Function Documentation

#### 8.21.2.1 keygen\_full()

```
std::vector< double > keygen_full (
 const phylocounters::PhyloArray & array) [inline]
```

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

### 8.21.2.2 RULE\_FUNCTION()

```
RULE_FUNCTION (
 rule_empty_free)
```

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

### 8.21.2.3 vec\_diff()

```
bool vec_diff (
 const std::vector< unsigned int > & s,
 const std::vector< unsigned int > & a) [inline]
```

Definition at line 51 of file geese-bones.hpp.

### 8.21.2.4 vector\_caster()

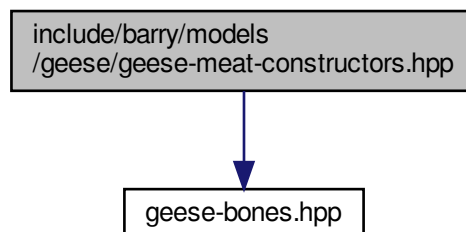
```
template<typename Ta , typename Tb >
std::vector< Ta > vector_caster (
 const std::vector< Tb > & x) [inline]
```

Definition at line 10 of file geese-bones.hpp.

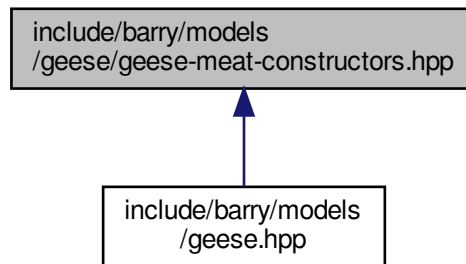
## 8.22 include/barry/models/geese/geese-meat-constructors.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat-constructors.hpp:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define GEESE_MEAT_CONSTRUCTORS_HPP 1`

### 8.22.1 Macro Definition Documentation

#### 8.22.1.1 GEESE\_MEAT\_CONSTRUCTORS\_HPP

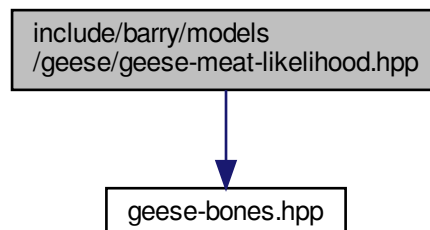
```
#define GEESE_MEAT_CONSTRUCTORS_HPP 1
```

Definition at line 4 of file `geese-meat-constructors.hpp`.

## 8.23 include/barry/models/geese/geese-meat-likelihood.hpp File Reference

```
#include "geese-bones.hpp"
```

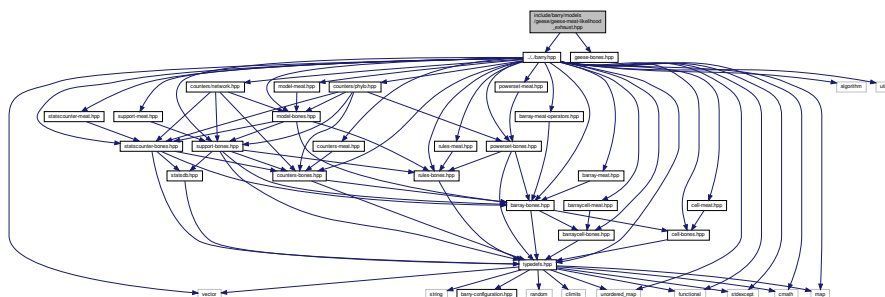
Include dependency graph for `geese-meat-likelihood.hpp`:



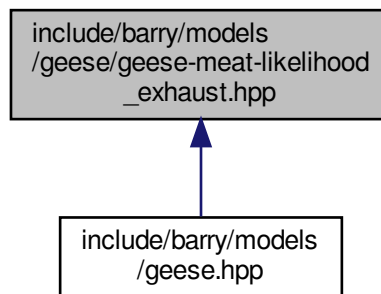


```
graph BT; A["include/barry/models/geese/geese-meat-likelihood.hpp"] --> B["include/barry/models/geese.hpp"]
```

```
#include "../..barry.hpp"
#include "geese-bones.hpp"
Include dependency graph for geese-meat-likelihood exhaust.hpp:
```



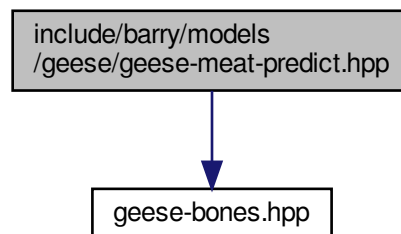
This graph shows which files directly or indirectly include this file:



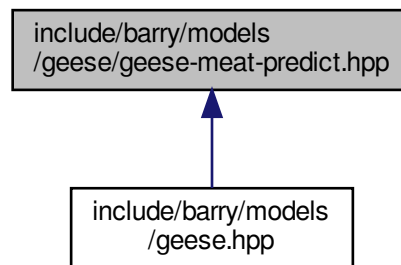
## 8.25 include/barry/models/geese/geese-meat-predict.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat-predict.hpp:



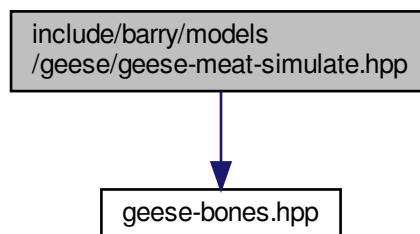
This graph shows which files directly or indirectly include this file:



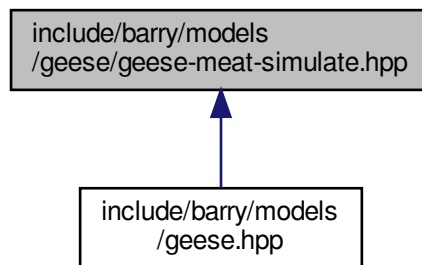
## 8.26 include/barry/models/geese/geese-meat-simulate.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat-simulate.hpp:



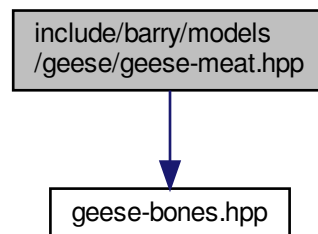
This graph shows which files directly or indirectly include this file:



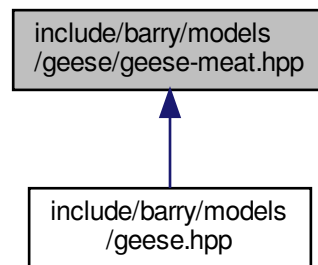
## 8.27 include/barry/models/geese/geese-meat.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat.hpp:

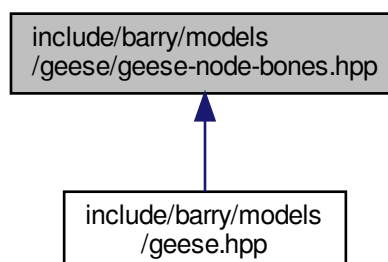


This graph shows which files directly or indirectly include this file:



## 8.28 include/barry/models/geese/geese-node-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



### Classes

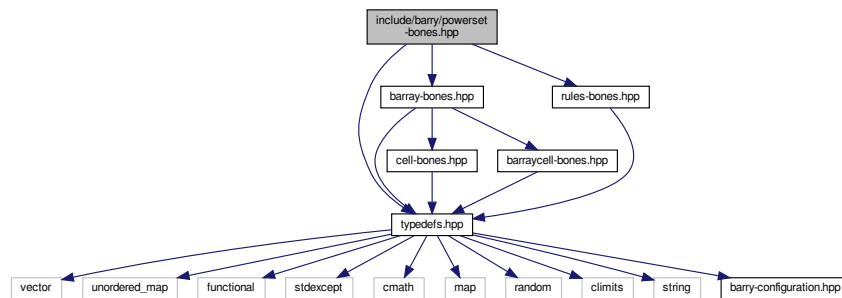
- class [Node](#)  
*A single node for the model.*

## 8.29 include/barry/powerset-bones.hpp File Reference

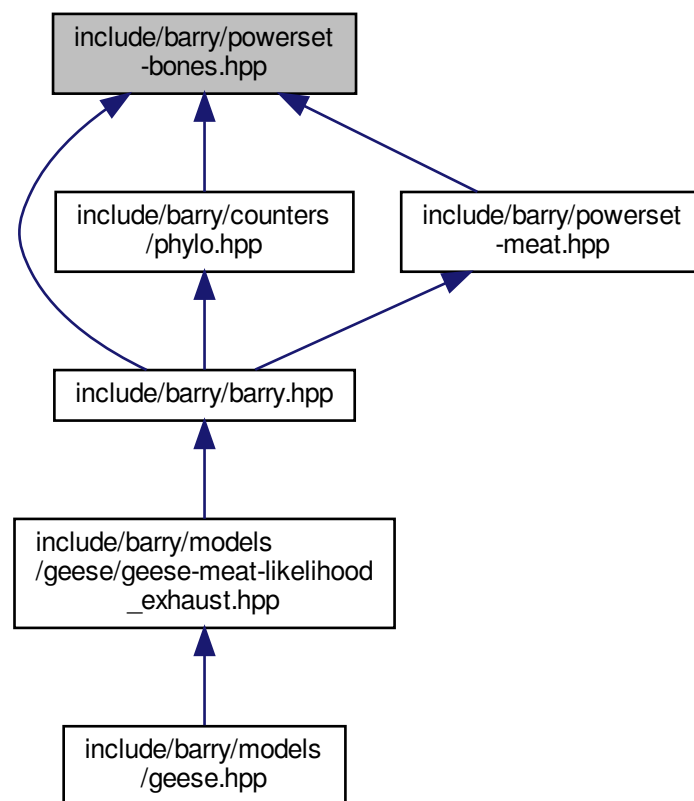
```
#include "typedefs.hpp"
#include "barray-bones.hpp"
```

```
#include "rules-bones.hpp"
```

Include dependency graph for powerset-bones.hpp:



This graph shows which files directly or indirectly include this file:



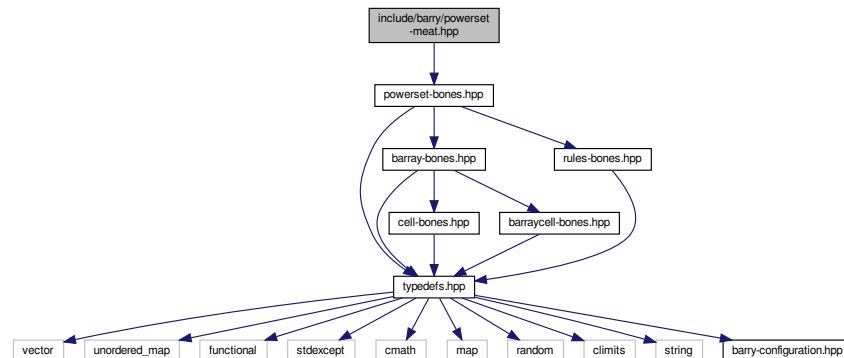
## Classes

- class [PowerSet< Array\\_Type, Data\\_Rule\\_Type >](#)  
*Powerset of a binary array.*

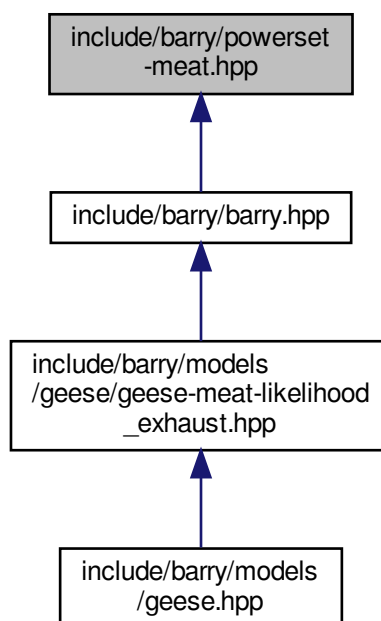
## 8.30 include/barry/powerset-meat.hpp File Reference

```
#include "powerset-bones.hpp"
```

Include dependency graph for powerset-meat.hpp:



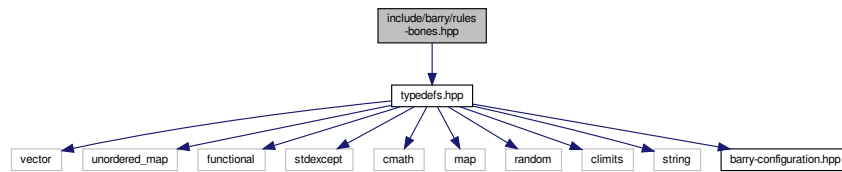
This graph shows which files directly or indirectly include this file:



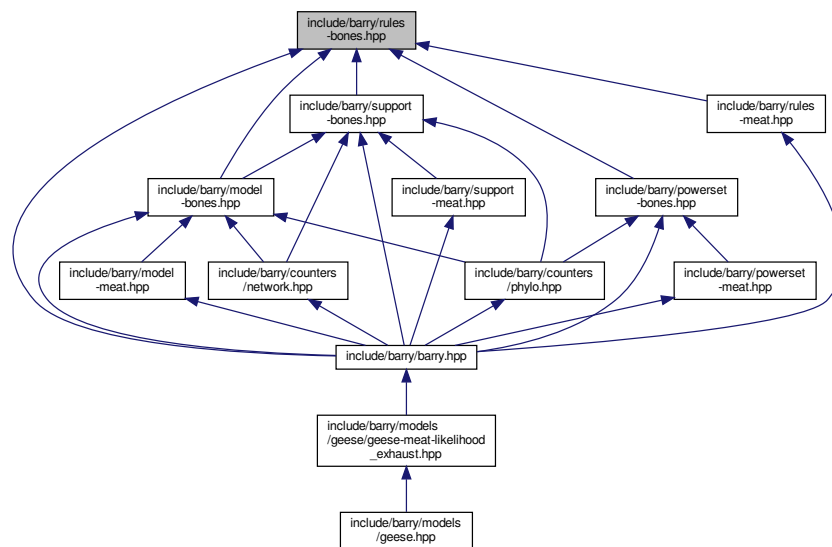
## 8.31 include/barry/rules-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for rules-bones.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [Rule< Array\\_Type, Data\\_Type >](#)  
*Rule* for determining if a cell should be included in a sequence.
- class [Rules< Array\\_Type, Data\\_Type >](#)  
*Vector* of objects of class *Rule*.

## Functions

- `template<typename Array_Type , typename Data_Type >`  
`bool rule_fun_default (const Array_Type *array, uint i, uint j, Data_Type *dat)`

### 8.31.1 Function Documentation



## 8.31.1.1 rule\_fun\_default()

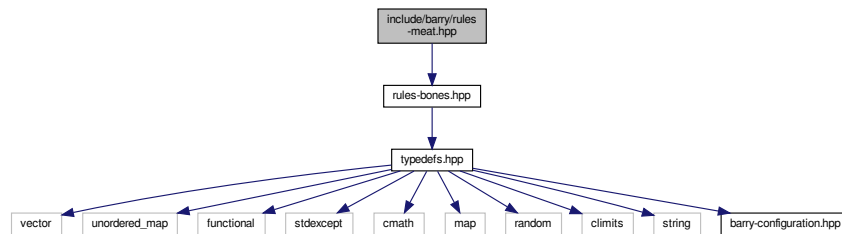
```
template<typename Array_Type , typename Data_Type >
bool rule_fun_default (
 const Array_Type * array,
 uint i,
 uint j,
 Data_Type * dat)
```

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

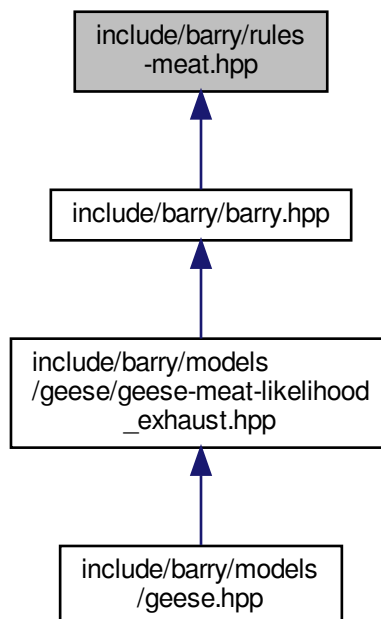
## 8.32 include/barry/rules-meat.hpp File Reference

```
#include "rules-bones.hpp"
```

Include dependency graph for rules-meat.hpp:



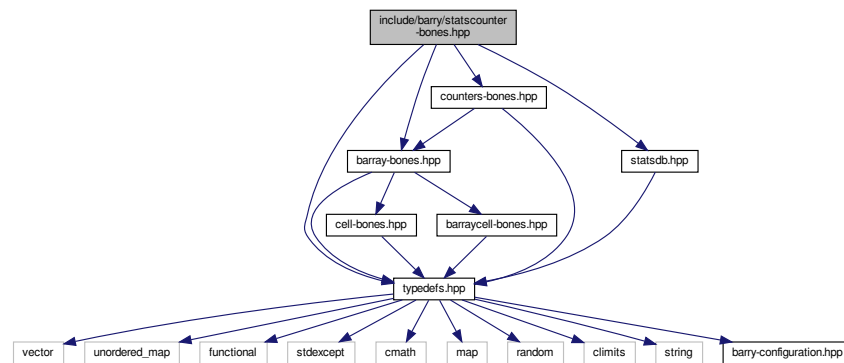
This graph shows which files directly or indirectly include this file:



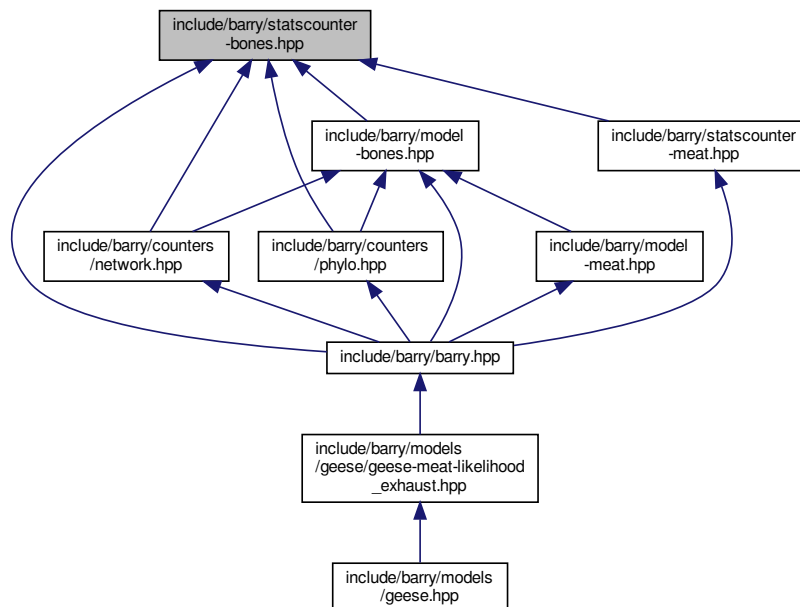
### 8.33 include/barry/statscounter-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "barray-bones.hpp"
#include "statsdb.hpp"
#include "counters-bones.hpp"
```

Include dependency graph for statscounter-bones.hpp:



This graph shows which files directly or indirectly include this file:



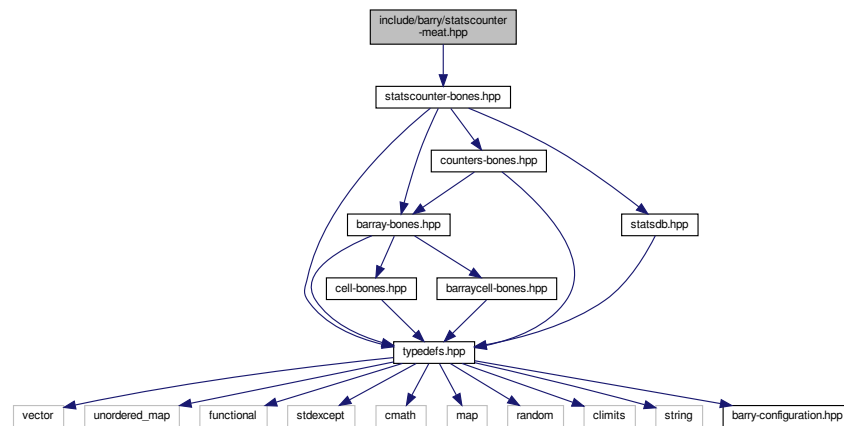
### Classes

- class `StatsCounter< Array_Type, Data_Type >`  
Count stats for a single Array.

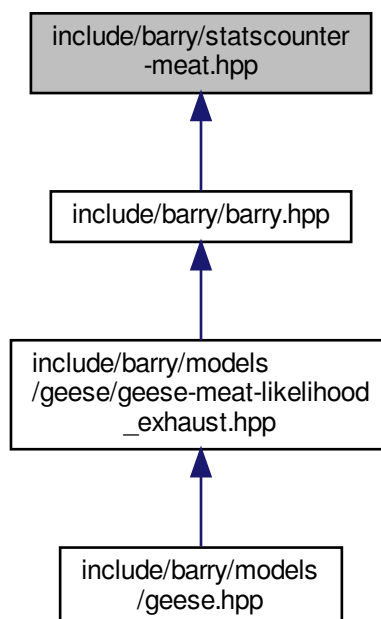
## 8.34 include/barry/statscounter-meat.hpp File Reference

```
#include "statscounter-bones.hpp"
```

Include dependency graph for statscounter-meat.hpp:



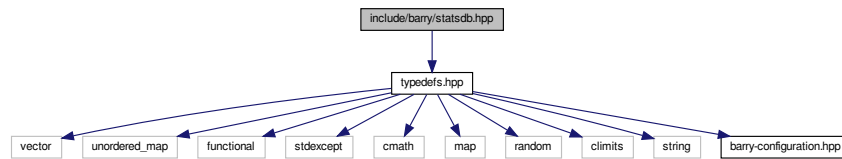
This graph shows which files directly or indirectly include this file:



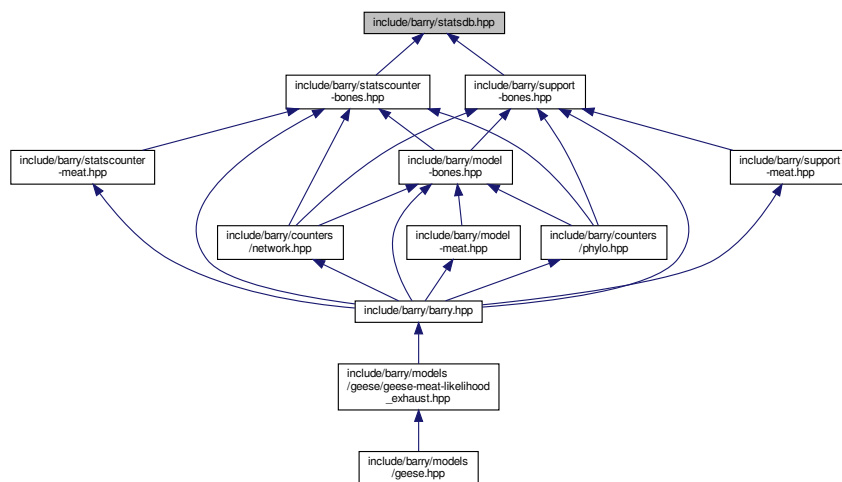
## 8.35 include/barry/statsdb.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for statsdb.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [FreqTable< T >](#)  
*Database of statistics.*

## 8.36 include/barry/support-bones.hpp File Reference

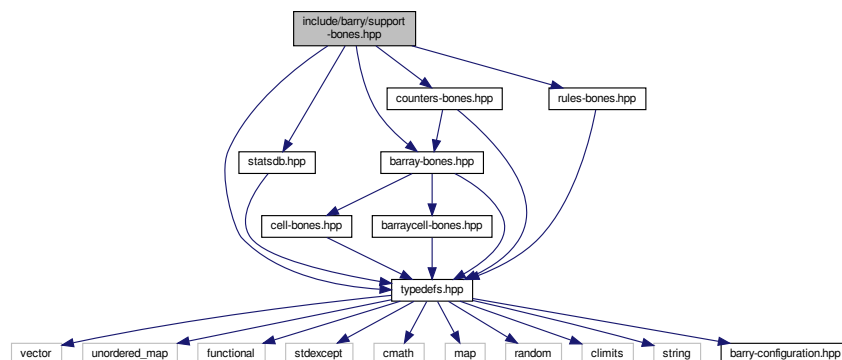
```

#include "typedefs.hpp"
#include "barry-bones.hpp"
#include "statsdb.hpp"
#include "counters-bones.hpp"

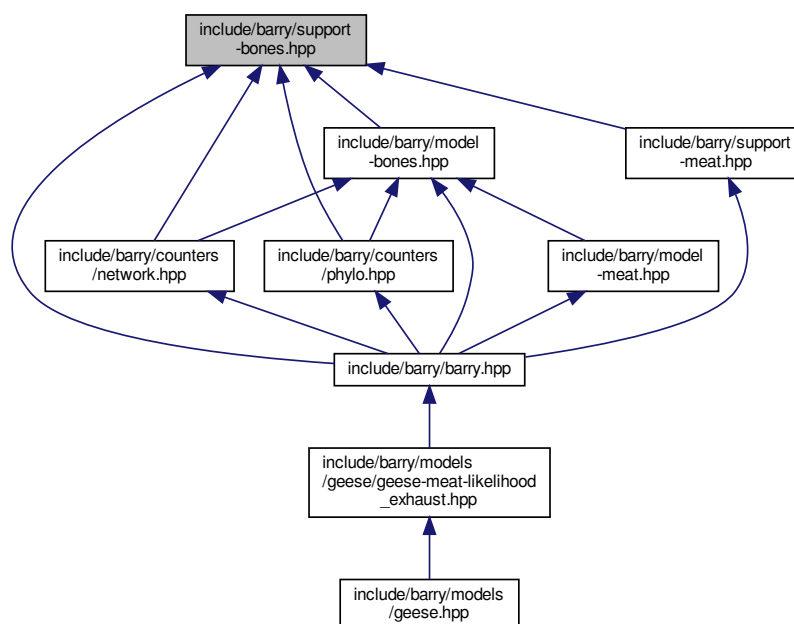
```

```
#include "rules-bones.hpp"
```

Include dependency graph for support-bones.hpp:



This graph shows which files directly or indirectly include this file:



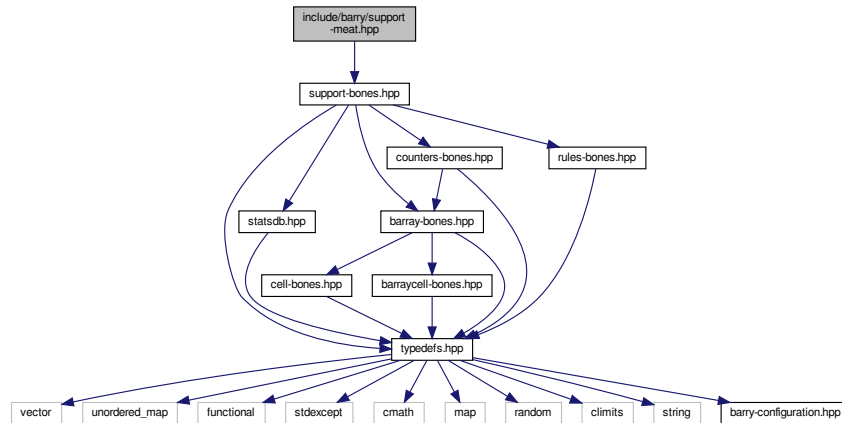
## Classes

- class [Support< Array\\_Type, Data\\_Counter\\_Type, Data\\_Rule\\_Type >](#)  
Compute the support of sufficient statistics.

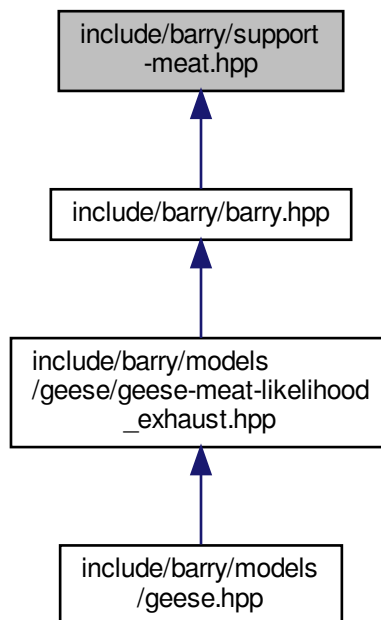
## 8.37 include/barry/support-meat.hpp File Reference

```
#include "support-bones.hpp"
```

Include dependency graph for support-meat.hpp:



This graph shows which files directly or indirectly include this file:



### Macros

- `#define BARRY_SUPPORT_MEAT_HPP 1`



## Namespaces

- [CHECK](#)

*Integer constants used to specify which cell should be check.*

- [EXISTS](#)

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

## Macros

- `#define ROW(a) this->el_ij[a]`
- `#define COL(a) this->el_ji[a]`
- `#define A\_ROW(a) Array.el_ij[a]`
- `#define A\_COL(a) Array.el_ji[a]`

## Typedefs

- `typedef unsigned int uint`
- `typedef std::vector< std::pair< std::vector< double >, uint > > Counts\_type`
- `template<typename Cell_Type >  
using Row\_type = Map< uint, Cell< Cell_Type > >`
- `template<typename Cell_Type >  
using Col\_type = Map< uint, Cell< Cell_Type > * >`
- `template<typename Ta = double, typename Tb = uint>  
using MapVec\_type = std::unordered_map< std::vector< Ta >, Tb, vecHasher< Ta > >`
- `template<typename Array_Type , typename Data_Type >  
using Counter\_fun\_type = std::function< double(const Array_Type &, uint, uint, Data_Type *)>`  
*[Counter](#) and rule functions.*
- `template<typename Array_Type , typename Data_Type >  
using Rule\_fun\_type = std::function< bool(const Array_Type &, uint, uint, Data_Type *)>`

## Functions

- `template<typename T >  
T vec\_inner\_prod (const std::vector< T > &a, const std::vector< T > &b)`
- `template<typename T >  
bool vec\_equal (const std::vector< T > &a, const std::vector< T > &b)`  
*Compares if -a- and -b- are equal.*
- `template<typename T >  
bool vec\_equal\_approx (const std::vector< T > &a, const std::vector< T > &b, double eps=1e-10)`



## Variables

- const int [CHECK::BOTH](#) = -1
- const int [CHECK::NONE](#) = 0
- const int [CHECK::ONE](#) = 1
- const int [CHECK::TWO](#) = 2
- const int [EXISTS::BOTH](#) = -1
- const int [EXISTS::NONE](#) = 0
- const int [EXISTS::ONE](#) = 1
- const int [EXISTS::TWO](#) = 1
- const int [EXISTS::UNKNOWN](#) = -1
- const int [EXISTS::AS\\_ZERO](#) = 0
- const int [EXISTS::AS\\_ONE](#) = 1

## 8.38.1 Macro Definition Documentation

### 8.38.1.1 A\_COL

```
#define A_COL(
 a) Array.el_ji[a]
```

Definition at line 124 of file typedefs.hpp.

### 8.38.1.2 A\_ROW

```
#define A_ROW(
 a) Array.el_ij[a]
```

Definition at line 123 of file typedefs.hpp.

### 8.38.1.3 COL

```
#define COL(
 a) this->el_ji[a]
```

Definition at line 24 of file typedefs.hpp.

### 8.38.1.4 ROW

```
#define ROW(
 a) this->el_ij[a]
```

Definition at line 23 of file typedefs.hpp.

## 8.38.2 Typedef Documentation

### 8.38.2.1 Col\_type

```
template<typename Cell_Type >
using Col_type = Map< uint, Cell<Cell_Type>* >
```

Definition at line 63 of file typedefs.hpp.

### 8.38.2.2 Counter\_fun\_type

```
template<typename Array_Type , typename Data_Type >
using Counter_fun_type = std::function<double(const Array_Type &, uint, uint, Data_Type *)>
```

[Counter](#) and rule functions.

#### Parameters

|                   |                                                                                   |
|-------------------|-----------------------------------------------------------------------------------|
| <i>Array_Type</i> | a <a href="#">BArray</a>                                                          |
| <i>unit,uint</i>  | Focal cell                                                                        |
| <i>Data_Type</i>  | 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 137 of file typedefs.hpp.

### 8.38.2.3 Counts\_type

```
typedef std::vector< std::pair< std::vector<double>, uint > > Counts_type
```

Definition at line 56 of file typedefs.hpp.

### 8.38.2.4 MapVec\_type

```
template<typename Ta = double, typename Tb = uint>
using MapVec_type = std::unordered_map< std::vector< Ta >, Tb, vecHasher<Ta> >
```

Definition at line 117 of file typedefs.hpp.

### 8.38.2.5 Row\_type

```
template<typename Cell_Type >
using Row_type = Map< uint, Cell<Cell_Type> >
```

Definition at line 60 of file typedefs.hpp.

### 8.38.2.6 Rule\_fun\_type

```
template<typename Array_Type , typename Data_Type >
using Rule_fun_type = std::function<bool(const Array_Type &, uint, uint, Data_Type *)>
```

Definition at line 140 of file typedefs.hpp.

### 8.38.2.7 uint

```
typedef unsigned int uint
```

Definition at line 20 of file typedefs.hpp.

## 8.38.3 Function Documentation

### 8.38.3.1 vec\_equal()

```
template<typename T >
bool vec_equal (
 const std::vector< T > & a,
 const std::vector< T > & b) [inline]
```

Compares if -a- and -b- are equal.

#### Parameters

|            |                                |
|------------|--------------------------------|
| <i>a,b</i> | Two vectors of the same length |
|------------|--------------------------------|

#### Returns

`true` if all elements are equal.

Definition at line 151 of file typedefs.hpp.

### 8.38.3.2 `vec_equal_approx()`

```
template<typename T >
bool vec_equal_approx (
 const std::vector< T > & a,
 const std::vector< T > & b,
 double eps = 1e-10) [inline]
```

Definition at line 169 of file typedefs.hpp.

### 8.38.3.3 `vec_inner_prod()`

```
template<typename T >
T vec_inner_prod (
 const std::vector< T > & a,
 const std::vector< T > & b) [inline]
```

Definition at line 189 of file typedefs.hpp.

## 8.39 README.md File Reference

# Index

- ~BArray
  - BArray< Cell\_Type, Data\_Type >, [29](#)
- ~BArrayCell
  - BArrayCell< Cell\_Type, Data\_Type >, [41](#)
- ~BArrayCell\_const
  - BArrayCell\_const< Cell\_Type, Data\_Type >, [44](#)
- ~Cell
  - Cell< Cell\_Type >, [47](#)
- ~ConstBArrayRowIter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [51](#)
- ~Counter
  - Counter< Array\_Type, Data\_Type >, [54](#)
- ~Counters
  - Counters< Array\_Type, Data\_Type >, [57](#)
- ~Entries
  - Entries< Cell\_Type >, [60](#)
- ~Flock
  - Flock, [62](#)
- ~FreqTable
  - FreqTable< T >, [67](#)
- ~Geese
  - Geese, [72](#)
- ~Model
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [83](#)
- ~NetCounterData
  - NetCounterData, [94](#)
- ~NetworkData
  - NetworkData, [97](#)
- ~Node
  - Node, [99](#)
- ~NodeData
  - NodeData, [103](#)
- ~PowerSet
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [107](#)
- ~Rule
  - Rule< Array\_Type, Data\_Type >, [112](#)
- ~Rules
  - Rules< Array\_Type, Data\_Type >, [114](#)
- ~StatsCounter
  - StatsCounter< Array\_Type, Data\_Type >, [118](#)
- ~Support
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [124](#)
- A\_COL
  - typedefs.hpp, [181](#)
- A\_ROW
  - typedefs.hpp, [181](#)
- add
  - Cell< Cell\_Type >, [48](#)
  - FreqTable< T >, [68](#)
- add\_array
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [83](#)
- add\_counter
  - Counters< Array\_Type, Data\_Type >, [57](#), [58](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [83](#), [84](#)
  - StatsCounter< Array\_Type, Data\_Type >, [119](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [125](#)
- add\_data
  - Flock, [63](#)
- add\_rule
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [84](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [107](#)
  - Rules< Array\_Type, Data\_Type >, [115](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [125](#)
- annotations
  - Node, [100](#)
- Array
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [51](#)
  - StatsCounter< Array\_Type, Data\_Type >, [120](#)
- array
  - Node, [100](#)
- array\_frequency
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [89](#)
- arrays
  - Node, [100](#)
- arrays2support
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [89](#)
- AS\_ONE
  - EXISTS, [23](#)
- as\_vector
  - FreqTable< T >, [68](#)
- AS\_ZERO
  - EXISTS, [23](#)
- BArray
  - BArray< Cell\_Type, Data\_Type >, [28](#), [29](#)
- BArray< Cell\_Type, Data\_Type >, [25](#)
  - ~BArray, [29](#)
  - BArray, [28](#), [29](#)
  - BArrayCell< Cell\_Type, Data\_Type >, [38](#)
  - BArrayCell\_const< Cell\_Type, Data\_Type >, [38](#)

- Cell\_default, 38
- clear, 30
- col, 30
- data, 39
- delete\_data, 39
- el\_ij, 39
- el\_ji, 39
- get\_cell, 30
- get\_col, 30
- get\_col\_vec, 30, 31
- get\_entries, 31
- get\_row, 31
- get\_row\_vec, 31
- insert\_cell, 32
- is\_empty, 32
- M, 39
- N, 39
- NCells, 40
- ncol, 32
- nnozero, 33
- nrow, 33
- operator\*=, 33
- operator(), 33
- operator+=, 33, 34
- operator-=, 34
- operator/=: 34
- operator=, 35
- operator==, 35
- out\_of\_range, 35
- print, 35
- reserve, 35
- resize, 36
- rm\_cell, 36
- row, 36
- set\_data, 36
- swap\_cells, 36
- swap\_cols, 37
- swap\_rows, 37
- toggle\_cell, 37
- toggle\_lock, 37
- transpose, 37
- visited, 40
- zero\_col, 38
- zero\_row, 38
- barray-bones.hpp
  - BARRAY\_BONES\_HPP, 132
- barray-iterator.hpp
  - BARRAY\_ITERATOR\_HPP, 133
- barray-meat-operators.hpp
  - checkdim\_, 134
- BARRAY\_BONES\_HPP
  - barray-bones.hpp, 132
- BARRAY\_ITERATOR\_HPP
  - barray-iterator.hpp, 133
- BArrayCell
  - BArrayCell< Cell\_Type, Data\_Type >, 41
- BArrayCell< Cell\_Type, Data\_Type >, 40
  - ~BArrayCell, 41
- BArray< Cell\_Type, Data\_Type >, 38
- BArrayCell, 41
  - operator Cell\_Type, 41
  - operator\*=, 41
  - operator+=, 42
  - operator-=, 42
  - operator/=: 42
  - operator=, 42
  - operator==, 42
- BArrayCell\_const
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 43
- BArrayCell\_const< Cell\_Type, Data\_Type >, 43
  - ~BArrayCell\_const, 44
  - BArray< Cell\_Type, Data\_Type >, 38
  - BArrayCell\_const, 43
  - operator Cell\_Type, 44
  - operator!=, 44
  - operator<, 44
  - operator<=: 44
  - operator>, 45
  - operator>=: 45
  - operator==, 45
- barry, 21
- barry-configuration.hpp
  - BARRY\_CHECK\_SUPPORT, 138
  - BARRY\_ISFINITE, 138
  - BARRY\_MAX\_NUM\_ELEMENTS, 138
  - BARRY\_SAFE\_EXP, 138
  - Map, 139
- barry.hpp
  - COUNTER\_FUNCTION, 141
  - COUNTER\_LAMBDA, 141
  - RULE\_FUNCTION, 141
  - RULE\_LAMBDA, 141
- barry::counters, 21
- barry::counters::network, 22
- barry::counters::phylo, 22
- BARRY\_CHECK\_SUPPORT
  - barry-configuration.hpp, 138
- BARRY\_ISFINITE
  - barry-configuration.hpp, 138
- BARRY\_MAX\_NUM\_ELEMENTS
  - barry-configuration.hpp, 138
- BARRY\_SAFE\_EXP
  - barry-configuration.hpp, 138
- BARRY\_SUPPORT\_MEAT\_HPP
  - support-meat.hpp, 179
- begin
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 107
- blengths
  - NodeData, 104
- BOTH
  - CHECK, 22
  - EXISTS, 23
- calc
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 108
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, 125

- calc\_reduced\_sequence
  - Geese, [72](#)
- calc\_sequence
  - Geese, [72](#)
- Cell
  - Cell< Cell\_Type >, [46–48](#)
- Cell< Cell\_Type >, [45](#)
  - ~Cell, [47](#)
  - add, [48](#)
  - Cell, [46–48](#)
  - operator Cell\_Type, [49](#)
  - operator=, [49](#)
  - value, [49](#)
  - visited, [49](#)
- Cell\_default
  - BArray< Cell\_Type, Data\_Type >, [38](#)
- change\_stats
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [127](#)
- CHECK, [22](#)
  - BOTH, [22](#)
  - NONE, [22](#)
  - ONE, [22](#)
  - TWO, [22](#)
- checkdim\_
  - barray-meat-operators.hpp, [134](#)
- clear
  - BArray< Cell\_Type, Data\_Type >, [30](#)
  - Counters< Array\_Type, Data\_Type >, [58](#)
  - FreqTable< T >, [68](#)
  - Rules< Array\_Type, Data\_Type >, [115](#)
- COL
  - typedefs.hpp, [181](#)
- col
  - BArray< Cell\_Type, Data\_Type >, [30](#)
- Col\_type
  - typedefs.hpp, [182](#)
- ConstBArrayRowIter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [51](#)
- ConstBArrayRowIter< Cell\_Type, Data\_Type >, [50](#)
  - ~ConstBArrayRowIter, [51](#)
  - Array, [51](#)
  - ConstBArrayRowIter, [51](#)
  - current\_col, [51](#)
  - current\_row, [51](#)
  - iter, [52](#)
- coordinates\_free
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [109](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [127](#)
- coordinates\_locked
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [109](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [128](#)
- count
  - Counter< Array\_Type, Data\_Type >, [54](#)
- count\_all
  - StatsCounter< Array\_Type, Data\_Type >, [119](#)
- count\_current
  - StatsCounter< Array\_Type, Data\_Type >, [119](#)
- count\_fun
  - Counter< Array\_Type, Data\_Type >, [55](#)
- count\_init
  - StatsCounter< Array\_Type, Data\_Type >, [119](#)
- Counter
  - Counter< Array\_Type, Data\_Type >, [53](#)
- Counter< Array\_Type, Data\_Type >, [52](#)
  - ~Counter, [54](#)
  - count, [54](#)
  - count\_fun, [55](#)
  - Counter, [53](#)
  - data, [55](#)
  - delete\_data, [55](#)
  - desc, [55](#)
  - init, [54](#)
  - init\_fun, [55](#)
  - name, [55](#)
  - operator=, [54](#)
- counter\_absdiff
  - Network counters, [11](#)
- counter\_co\_opt
  - Phylo counters, [16](#)
- counter\_cogain
  - Phylo counters, [16](#)
- counter\_ctriads
  - Network counters, [11](#)
- counter\_degree
  - Network counters, [11](#)
- counter\_deleted
  - StatsCounter< Array\_Type, Data\_Type >, [120](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [128](#)
- counter\_density
  - Network counters, [11](#)
- counter\_diff
  - Network counters, [11](#)
- counter\_edges
  - Network counters, [12](#)
- counter\_fun
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [89](#)
- Counter\_fun\_type
  - typedefs.hpp, [182](#)
- COUNTER\_FUNCTION
  - barry.hpp, [141](#)
- counter\_gains
  - Phylo counters, [16](#)
- counter\_gains\_k\_offspring
  - Phylo counters, [17](#)
- counter\_genes\_changing
  - Phylo counters, [17](#)
- counter\_iddegree
  - Network counters, [12](#)
- counter\_iddegree15
  - Network counters, [12](#)
- counter\_isolates

- Network counters, [12](#)
- counter\_istar2
  - Network counters, [12](#)
- COUNTER\_LAMBDA
  - barry.hpp, [141](#)
- counter\_longest
  - Phylo counters, [17](#)
- counter\_loss
  - Phylo counters, [17](#)
- counter\_maxfun
  - Phylo counters, [18](#)
- counter\_mutual
  - Network counters, [13](#)
- counter\_neofun
  - Phylo counters, [18](#)
- counter\_neofun\_a2b
  - Phylo counters, [18](#)
- counter\_nodecov
  - Network counters, [13](#)
- counter\_nodeicov
  - Network counters, [13](#)
- counter\_nodematch
  - Network counters, [13](#)
- counter\_nodeecov
  - Network counters, [13](#)
- counter\_odegree
  - Network counters, [14](#)
- counter\_odegree15
  - Network counters, [14](#)
- counter\_ostar2
  - Network counters, [14](#)
- counter\_overall\_changes
  - Phylo counters, [18](#)
- counter\_overall\_gains
  - Phylo counters, [19](#)
- counter\_overall\_loss
  - Phylo counters, [19](#)
- counter\_subfun
  - Phylo counters, [19](#)
- counter\_triads
  - Network counters, [14](#)
- Counters
  - Counters< Array\_Type, Data\_Type >, [57](#)
- counters
  - Geese, [76](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [90](#)
  - StatsCounter< Array\_Type, Data\_Type >, [121](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [128](#)
- Counters< Array\_Type, Data\_Type >, [56](#)
  - ~Counters, [57](#)
  - add\_counter, [57](#), [58](#)
  - clear, [58](#)
  - Counters, [57](#)
  - operator=, [58](#)
  - operator[], [58](#)
  - size, [59](#)
- counters\_ptr
  - Flock, [63](#)
- Counting, [9](#)
- Counts\_type
  - typedefs.hpp, [182](#)
- current\_col
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [51](#)
- current\_row
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [51](#)
- current\_stats
  - StatsCounter< Array\_Type, Data\_Type >, [121](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [128](#)
- dat
  - Flock, [66](#)
- data
  - BArray< Cell\_Type, Data\_Type >, [39](#)
  - Counter< Array\_Type, Data\_Type >, [55](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [110](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [128](#)
- delete\_counters
  - Geese, [76](#)
- delete\_data
  - BArray< Cell\_Type, Data\_Type >, [39](#)
  - Counter< Array\_Type, Data\_Type >, [55](#)
- delete\_rengine
  - Geese, [76](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [90](#)
- delete\_support
  - Geese, [77](#)
- desc
  - Counter< Array\_Type, Data\_Type >, [55](#)
- directed
  - NetworkData, [97](#)
- duplication
  - Node, [101](#)
  - NodeData, [104](#)
- el\_ij
  - BArray< Cell\_Type, Data\_Type >, [39](#)
- el\_ji
  - BArray< Cell\_Type, Data\_Type >, [39](#)
- EmptyArray
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [110](#)
  - StatsCounter< Array\_Type, Data\_Type >, [121](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [129](#)
- end
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [108](#)
- Entries
  - Entries< Cell\_Type >, [60](#)
- Entries< Cell\_Type >, [59](#)
  - ~Entries, [60](#)
  - Entries, [60](#)
  - resize, [60](#)
  - source, [61](#)



- target, 61
- val, 61
- EXISTS, 23
  - AS\_ONE, 23
  - AS\_ZERO, 23
  - BOTH, 23
  - NONE, 24
  - ONE, 24
  - TWO, 24
  - UNKNOWN, 24
- first\_calc\_done
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, 90
- Flock, 61
  - ~Flock, 62
  - add\_data, 63
  - counters\_ptr, 63
  - dat, 66
  - Flock, 62
  - init, 63
  - initialized, 66
  - likelihood\_joint, 63
  - nfunctions, 66
  - nfuns, 64
  - nleafs, 64
  - nnodes, 64
  - nterms, 64
  - ntrees, 65
  - operator(), 65
  - rengine, 66
  - set\_seed, 65
  - support, 66
- FreqTable
  - FreqTable< T >, 67
- FreqTable< T >, 67
  - ~FreqTable, 67
  - add, 68
  - as\_vector, 68
  - clear, 68
  - FreqTable, 67
  - get\_data, 68
  - get\_data\_ptr, 68
  - print, 68
  - reserve, 69
  - size, 69
- Geese, 69
  - ~Geese, 72
  - calc\_reduced\_sequence, 72
  - calc\_sequence, 72
  - counters, 76
  - delete\_counters, 76
  - delete\_rengine, 76
  - delete\_support, 77
  - Geese, 71, 72
  - get\_probabilities, 72
  - inherit\_support, 73
  - init, 73
  - init\_node, 73
  - initialized, 77
  - likelihood, 73
  - likelihood\_exhaust, 73
  - map\_to\_nodes, 77
  - nfunctions, 77
  - nfuns, 74
  - nleafs, 74
  - nnodes, 74
  - nodes, 77
  - nterms, 74
  - observed\_counts, 74
  - operator=, 74, 75
  - predict, 75
  - predict\_backend, 75
  - print\_observed\_counts, 75
  - reduced\_sequence, 77
  - rengine, 78
  - sequence, 78
  - set\_seed, 75
  - simulate, 76
  - states, 78
  - support, 78
  - update\_annotations, 76
- geese-bones.hpp
  - INITIALIZED, 162
  - keygen\_full, 162
  - RULE\_FUNCTION, 162
  - vec\_diff, 163
  - vector\_caster, 163
- geese-meat-constructors.hpp
  - GEESE\_MEAT\_CONSTRUCTORS\_HPP, 164
- GEESE\_MEAT\_CONSTRUCTORS\_HPP
  - geese-meat-constructors.hpp, 164
- get\_cell
  - BArray< Cell\_Type, Data\_Type >, 30
- get\_col
  - BArray< Cell\_Type, Data\_Type >, 30
- get\_col\_vec
  - BArray< Cell\_Type, Data\_Type >, 30, 31
- get\_counts
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, 126
- get\_counts\_ptr
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, 126
- get\_data
  - FreqTable< T >, 68
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 108
- get\_data\_ptr
  - FreqTable< T >, 68
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 108
- get\_entries
  - BArray< Cell\_Type, Data\_Type >, 31
- get\_last\_name
  - phylo.hpp, 156
- get\_norm\_const

- Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type >, 85
- get\_parent
  - Node, 100
- get\_probabilities
  - Geese, 72
- get\_pset
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type >, 85
- get\_row
  - BArray< Cell\_Type, Data\_Type >, 31
- get\_row\_vec
  - BArray< Cell\_Type, Data\_Type >, 31
- get\_seq
  - Rules< Array\_Type, Data\_Type >, 115
- get\_stats
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type >, 85
- id
  - Node, 101
- include/barry/barray-bones.hpp, 131
- include/barry/barray-iterator.hpp, 132
- include/barry/barray-meat-operators.hpp, 133
- include/barry/barray-meat.hpp, 135
- include/barry/barraycell-bones.hpp, 135
- include/barry/barraycell-meat.hpp, 136
- include/barry/barry-configuration.hpp, 137
- include/barry/barry.hpp, 139
- include/barry/cell-bones.hpp, 142
- include/barry/cell-meat.hpp, 142
- include/barry/col-bones.hpp, 143
- include/barry/counters-bones.hpp, 143
- include/barry/counters-meat.hpp, 144
- include/barry/counters/network.hpp, 145
- include/barry/counters/phylo.hpp, 151
- include/barry/model-bones.hpp, 157
- include/barry/model-meat.hpp, 159
- include/barry/models/geese.hpp, 160
- include/barry/models/geese/flock-bones.hpp, 160
- include/barry/models/geese/flock-meet.hpp, 161
- include/barry/models/geese/geese-bones.hpp, 161
- include/barry/models/geese/geese-meat-constructors.hpp,  
163
- include/barry/models/geese/geese-meat-likelihood.hpp,  
164
- include/barry/models/geese/geese-meat-likelihood\_exhaust.hpp,  
165
- include/barry/models/geese/geese-meat-predict.hpp,  
166
- include/barry/models/geese/geese-meat-simulate.hpp,  
167
- include/barry/models/geese/geese-meat.hpp, 168
- include/barry/models/geese/geese-node-bones.hpp,  
169
- include/barry/powerset-bones.hpp, 169
- include/barry/powerset-meat.hpp, 171
- include/barry/rules-bones.hpp, 171
- include/barry/rules-meat.hpp, 173
- include/barry/statscounter-bones.hpp, 174
- include/barry/statscounter-meat.hpp, 175
- include/barry/statsdb.hpp, 175
- include/barry/support-bones.hpp, 176
- include/barry/support-meat.hpp, 178
- include/barry/typedefs.hpp, 179
- indices
  - NetCounterData, 95
- inherit\_support
  - Geese, 73
- init
  - Counter< Array\_Type, Data\_Type >, 54
  - Flock, 63
  - Geese, 73
- init\_fun
  - Counter< Array\_Type, Data\_Type >, 55
- init\_node
  - Geese, 73
- init\_support
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 108
  - Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type >, 126
- INITIALIZED
  - geese-bones.hpp, 162
- initialized
  - Flock, 66
  - Geese, 77
- insert\_cell
  - BArray< Cell\_Type, Data\_Type >, 32
- is\_empty
  - BArray< Cell\_Type, Data\_Type >, 32
- is\_leaf
  - Node, 100
- iter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, 52
- keygen
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type >, 90
- keygen\_default
  - model-bones.hpp, 158
- keygen\_full
  - geese-bones.hpp, 162
- keys2support
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type >, 90
- likelihood
  - Geese, 73
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type >, 85, 86
- likelihood\_
  - model-bones.hpp, 158
- likelihood\_exhaust
  - Geese, 73
- likelihood\_joint
  - Flock, 63
- likelihood\_total

- Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, 86
- locked
  - Rule< Array\_Type, Data\_Type >, 113
  - Rules< Array\_Type, Data\_Type >, 116
- M
  - BArray< Cell\_Type, Data\_Type >, 39
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 110
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, 129
- Map
  - barry-configuration.hpp, 139
- map\_to\_nodes
  - Geese, 77
- MapVec\_type
  - typedefs.hpp, 182
- max\_num\_elements
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, 129
- Model
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, 82
- Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type>
  - >, 79
  - ~Model, 83
  - add\_array, 83
  - add\_counter, 83, 84
  - add\_rule, 84
  - array\_frequency, 89
  - arrays2support, 89
  - counter\_fun, 89
  - counters, 90
  - delete\_engine, 90
  - first\_calc\_done, 90
  - get\_norm\_const, 85
  - get\_pset, 85
  - get\_stats, 85
  - keygen, 90
  - keys2support, 90
  - likelihood, 85, 86
  - likelihood\_total, 86
  - Model, 82
  - n\_arrays\_per\_stats, 91
  - normalizing\_constants, 91
  - nterms, 86
  - operator=, 87
  - params\_last, 91
  - print\_stats, 87
  - pset\_arrays, 91
  - pset\_probs, 92
  - pset\_stats, 92
  - engine, 92
  - rules, 92
  - sample, 87
  - set\_counters, 87
  - set\_keygen, 88
  - set\_engine, 88
  - set\_rules, 88
  - set\_seed, 88
  - size, 88
  - size\_unique, 89
  - stats, 92
  - store\_psets, 89
  - support\_fun, 93
  - target\_stats, 93
  - with\_pset, 93
- model-bones.hpp
  - keygen\_default, 158
  - likelihood\_, 158
  - update\_normalizing\_constant, 158
- N
  - BArray< Cell\_Type, Data\_Type >, 39
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 110
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, 129
- n\_arrays\_per\_stats
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, 91
- name
  - Counter< Array\_Type, Data\_Type >, 55
- narray
  - Node, 101
- NCells
  - BArray< Cell\_Type, Data\_Type >, 40
- ncol
  - BArray< Cell\_Type, Data\_Type >, 32
- NET\_C\_DATA\_IDX
  - network.hpp, 148
- NET\_C\_DATA\_NUM
  - network.hpp, 148
- NetCounter
  - network.hpp, 149
- NetCounterData, 93
  - ~NetCounterData, 94
  - indices, 95
  - NetCounterData, 94
  - numbers, 95
- NetCounters
  - network.hpp, 149
- NetModel
  - network.hpp, 149
- NetRule
  - network.hpp, 150
- NetRules
  - network.hpp, 150
- NetStatsCounter
  - network.hpp, 150
- NetSupport
  - network.hpp, 150
- Network
  - network.hpp, 150
- Network counters, 10
  - counter\_absdiff, 11
  - counter\_ctriads, 11
  - counter\_degree, 11
  - counter\_density, 11

- counter\_diff, 11
- counter\_edges, 12
- counter\_iddegree, 12
- counter\_iddegree15, 12
- counter\_isolates, 12
- counter\_istar2, 12
- counter\_mutual, 13
- counter\_nodecov, 13
- counter\_nodeicov, 13
- counter\_nodematch, 13
- counter\_nodeocov, 13
- counter\_odegree, 14
- counter\_odegree15, 14
- counter\_ostar2, 14
- counter\_ttriads, 14
- NETWORK\_COUNTER, 14
- network.hpp
  - NET\_C\_DATA\_IDX, 148
  - NET\_C\_DATA\_NUM, 148
  - NetCounter, 149
  - NetCounters, 149
  - NetModel, 149
  - NetRule, 150
  - NetRules, 150
  - NetStatsCounter, 150
  - NetSupport, 150
  - Network, 150
  - NETWORK\_COUNTER, 148
  - NETWORK\_COUNTER\_LAMBDA, 148
  - NETWORK\_RULE, 148
  - NETWORK\_RULE\_LAMBDA, 149
  - rules\_zerodiag, 151
- NETWORK\_COUNTER
  - Network counters, 14
  - network.hpp, 148
- NETWORK\_COUNTER\_LAMBDA
  - network.hpp, 148
- NETWORK\_RULE
  - network.hpp, 148
- NETWORK\_RULE\_LAMBDA
  - network.hpp, 149
- NetworkData, 95
  - ~NetworkData, 97
  - directed, 97
  - NetworkData, 96
  - vertex\_attr, 97
- nfunctions
  - Flock, 66
  - Geese, 77
- nfuncs
  - Flock, 64
  - Geese, 74
- nleafs
  - Flock, 64
  - Geese, 74
- nnodes
  - Flock, 64
  - Geese, 74
- nnozero
  - BArray< Cell\_Type, Data\_Type >, 33
- Node, 97
  - ~Node, 99
  - annotations, 100
  - array, 100
  - arrays, 100
  - duplication, 101
  - get\_parent, 100
  - id, 101
  - is\_leaf, 100
  - narray, 101
  - Node, 98, 99
  - offspring, 101
  - ord, 101
  - parent, 102
  - probability, 102
  - subtree\_prob, 102
  - visited, 102
- NodeData, 103
  - ~NodeData, 103
  - blengths, 104
  - duplication, 104
  - NodeData, 103
  - states, 104
- nodes
  - Geese, 77
- NONE
  - CHECK, 22
  - EXISTS, 24
- normalizing\_constants
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, 91
- nrow
  - BArray< Cell\_Type, Data\_Type >, 33
- nterms
  - Flock, 64
  - Geese, 74
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, 86
- ntrees
  - Flock, 65
- numbers
  - NetCounterData, 95
- observed\_counts
  - Geese, 74
- offspring
  - Node, 101
- ONE
  - CHECK, 22
  - EXISTS, 24
- operator Cell\_Type
  - BArrayCell< Cell\_Type, Data\_Type >, 41
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 44
  - Cell< Cell\_Type >, 49
- operator!=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 44
- operator<

- BArrayCell\_const< Cell\_Type, Data\_Type >, 44
- operator<=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 44
- operator>
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 45
- operator>=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 45
- operator\*=
  - BArray< Cell\_Type, Data\_Type >, 33
  - BArrayCell< Cell\_Type, Data\_Type >, 41
- operator()
  - BArray< Cell\_Type, Data\_Type >, 33
  - Flock, 65
  - vecHasher< T >, 130
- operator+=
  - BArray< Cell\_Type, Data\_Type >, 33, 34
  - BArrayCell< Cell\_Type, Data\_Type >, 42
- operator-=
  - BArray< Cell\_Type, Data\_Type >, 34
  - BArrayCell< Cell\_Type, Data\_Type >, 42
- operator/=
  - BArray< Cell\_Type, Data\_Type >, 34
  - BArrayCell< Cell\_Type, Data\_Type >, 42
- operator=
  - BArray< Cell\_Type, Data\_Type >, 35
  - BArrayCell< Cell\_Type, Data\_Type >, 42
  - Cell< Cell\_Type >, 49
  - Counter< Array\_Type, Data\_Type >, 54
  - Counters< Array\_Type, Data\_Type >, 58
  - Geese, 74, 75
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, 87
  - Rules< Array\_Type, Data\_Type >, 116
- operator==
  - BArray< Cell\_Type, Data\_Type >, 35
  - BArrayCell< Cell\_Type, Data\_Type >, 42
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 45
- operator[]
  - Counters< Array\_Type, Data\_Type >, 58
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 109
- ord
  - Node, 101
- out\_of\_range
  - BArray< Cell\_Type, Data\_Type >, 35
- params\_last
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, 91
- parent
  - Node, 102
- Phylo counters, 15
  - counter\_co\_opt, 16
  - counter\_cogain, 16
  - counter\_gains, 16
  - counter\_gains\_k\_offspring, 17
  - counter\_genes\_changing, 17
  - counter\_longest, 17
  - counter\_loss, 17
  - counter\_maxfun, 18
  - counter\_neofun, 18
  - counter\_neofun\_a2b, 18
  - counter\_overall\_changes, 18
  - counter\_overall\_gains, 19
  - counter\_overall\_loss, 19
  - counter\_subfun, 19
- phylo.hpp
  - get\_last\_name, 156
  - PHYLO\_C\_DATA\_IDX, 153
  - PHYLO\_CHECK\_MISSING, 153
  - PHYLO\_COUNTER, 154
  - PHYLO\_COUNTER\_LAMBDA, 154
  - PhyloArray, 154
  - PhyloCounter, 154
  - PhyloCounterData, 155
  - PhyloCounters, 155
  - PhyloModel, 155
  - PhyloPowerSet, 155
  - PhyloRule, 155
  - PhyloRuleData, 155
  - PhyloRules, 156
  - PhyloStatsCounter, 156
  - PhyloSupport, 156
- PHYLO\_C\_DATA\_IDX
  - phylo.hpp, 153
- PHYLO\_CHECK\_MISSING
  - phylo.hpp, 153
- PHYLO\_COUNTER
  - phylo.hpp, 154
- PHYLO\_COUNTER\_LAMBDA
  - phylo.hpp, 154
- PhyloArray
  - phylo.hpp, 154
- PhyloCounter
  - phylo.hpp, 154
- PhyloCounterData
  - phylo.hpp, 155
- PhyloCounters
  - phylo.hpp, 155
- PhyloModel
  - phylo.hpp, 155
- PhyloPowerSet
  - phylo.hpp, 155
- PhyloRule
  - phylo.hpp, 155
- PhyloRuleData
  - phylo.hpp, 155
- PhyloRules
  - phylo.hpp, 156
- PhyloStatsCounter
  - phylo.hpp, 156
- PhyloSupport
  - phylo.hpp, 156
- PowerSet
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 106
- PowerSet< Array\_Type, Data\_Rule\_Type >, 105
  - ~PowerSet, 107
  - add\_rule, 107

- begin, [107](#)
- calc, [108](#)
- coordinates\_free, [109](#)
- coordinates\_locked, [109](#)
- data, [110](#)
- EmptyArray, [110](#)
- end, [108](#)
- get\_data, [108](#)
- get\_data\_ptr, [108](#)
- init\_support, [108](#)
- M, [110](#)
- N, [110](#)
- operator[], [109](#)
- PowerSet, [106](#)
- reset, [109](#)
- rules, [110](#)
- rules\_deleted, [111](#)
- size, [109](#)
- predict
  - Geese, [75](#)
- predict\_backend
  - Geese, [75](#)
- print
  - BArray< Cell\_Type, Data\_Type >, [35](#)
  - FreqTable< T >, [68](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [126](#)
- print\_observed\_counts
  - Geese, [75](#)
- print\_stats
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [87](#)
- probability
  - Node, [102](#)
- pset\_arrays
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [91](#)
- pset\_probs
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [92](#)
- pset\_stats
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [92](#)
- README.md, [184](#)
- reduced\_sequence
  - Geese, [77](#)
- rengine
  - Flock, [66](#)
  - Geese, [78](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [92](#)
- reserve
  - BArray< Cell\_Type, Data\_Type >, [35](#)
  - FreqTable< T >, [69](#)
- reset
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [109](#)
- reset\_array
  - StatsCounter< Array\_Type, Data\_Type >, [120](#)
- Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [126](#), [127](#)
- resize
  - BArray< Cell\_Type, Data\_Type >, [36](#)
  - Entries< Cell\_Type >, [60](#)
- rm\_cell
  - BArray< Cell\_Type, Data\_Type >, [36](#)
- ROW
  - typedefs.hpp, [181](#)
- row
  - BArray< Cell\_Type, Data\_Type >, [36](#)
- Row\_type
  - typedefs.hpp, [182](#)
- Rule
  - Rule< Array\_Type, Data\_Type >, [112](#)
- Rule< Array\_Type, Data\_Type >, [111](#)
- ~Rule, [112](#)
- locked, [113](#)
- Rule, [112](#)
- rule\_fun\_default
  - rules-bones.hpp, [172](#)
- Rule\_fun\_type
  - typedefs.hpp, [183](#)
- RULE\_FUNCTION
  - barry.hpp, [141](#)
  - geese-bones.hpp, [162](#)
- RULE\_LAMBDA
  - barry.hpp, [141](#)
- Rules
  - Rules< Array\_Type, Data\_Type >, [114](#)
- rules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [92](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [110](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [129](#)
  - Rules< Array\_Type, Data\_Type >, [113](#)
  - ~Rules, [114](#)
  - add\_rule, [115](#)
  - clear, [115](#)
  - get\_seq, [115](#)
  - locked, [116](#)
  - operator=, [116](#)
  - Rules, [114](#)
  - size, [116](#)
- rules-bones.hpp
  - rule\_fun\_default, [172](#)
- rules\_deleted
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [111](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [130](#)
- rules\_zerodiag
  - network.hpp, [151](#)
- sample
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [87](#)
- sequence
  - Geese, [78](#)

- set\_counters
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [87](#)
  - StatsCounter< Array\_Type, Data\_Type >, [120](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [127](#)
- set\_data
  - BArray< Cell\_Type, Data\_Type >, [36](#)
- set\_keygen
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [88](#)
- set\_rengine
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [88](#)
- set\_rules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [88](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [127](#)
- set\_seed
  - Flock, [65](#)
  - Geese, [75](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [88](#)
- simulate
  - Geese, [76](#)
- size
  - Counters< Array\_Type, Data\_Type >, [59](#)
  - FreqTable< T >, [69](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [88](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [109](#)
  - Rules< Array\_Type, Data\_Type >, [116](#)
- size\_unique
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [89](#)
- source
  - Entries< Cell\_Type >, [61](#)
- states
  - Geese, [78](#)
  - NodeData, [104](#)
- Statistical Models, [9](#)
- stats
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [92](#)
- StatsCounter
  - StatsCounter< Array\_Type, Data\_Type >, [118](#)
- StatsCounter< Array\_Type, Data\_Type >, [117](#)
- ~StatsCounter, [118](#)
- add\_counter, [119](#)
- Array, [120](#)
- count\_all, [119](#)
- count\_current, [119](#)
- count\_init, [119](#)
- counter\_deleted, [120](#)
- counters, [121](#)
- current\_stats, [121](#)
- EmptyArray, [121](#)
- reset\_array, [120](#)
- set\_counters, [120](#)
- StatsCounter, [118](#)
- store\_psets
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [89](#)
- subtree\_prob
  - Node, [102](#)
- Support
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [124](#)
- support
  - Flock, [66](#)
  - Geese, [78](#)
- Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [122](#)
- ~Support, [124](#)
- add\_counter, [125](#)
- add\_rule, [125](#)
- calc, [125](#)
- change\_stats, [127](#)
- coordinates\_free, [127](#)
- coordinates\_locked, [128](#)
- counter\_deleted, [128](#)
- counters, [128](#)
- current\_stats, [128](#)
- data, [128](#)
- EmptyArray, [129](#)
- get\_counts, [126](#)
- get\_counts\_ptr, [126](#)
- init\_support, [126](#)
- M, [129](#)
- max\_num\_elements, [129](#)
- N, [129](#)
- print, [126](#)
- reset\_array, [126](#), [127](#)
- rules, [129](#)
- rules\_deleted, [130](#)
- set\_counters, [127](#)
- set\_rules, [127](#)
- Support, [124](#)
- support-meat.hpp
  - BARRY\_SUPPORT\_MEAT\_HPP, [179](#)
- support\_fun
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [93](#)
- swap\_cells
  - BArray< Cell\_Type, Data\_Type >, [36](#)
- swap\_cols
  - BArray< Cell\_Type, Data\_Type >, [37](#)
- swap\_rows
  - BArray< Cell\_Type, Data\_Type >, [37](#)
- target
  - Entries< Cell\_Type >, [61](#)
- target\_stats
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type >, [93](#)
- toggle\_cell

- BArray< Cell\_Type, Data\_Type >, [37](#)
- toggle\_lock
  - BArray< Cell\_Type, Data\_Type >, [37](#)
- transpose
  - BArray< Cell\_Type, Data\_Type >, [37](#)
- TWO
  - CHECK, [22](#)
  - EXISTS, [24](#)
- typedefs.hpp
  - A\_COL, [181](#)
  - A\_ROW, [181](#)
  - COL, [181](#)
  - Col\_type, [182](#)
  - Counter\_fun\_type, [182](#)
  - Counts\_type, [182](#)
  - MapVec\_type, [182](#)
  - ROW, [181](#)
  - Row\_type, [182](#)
  - Rule\_fun\_type, [183](#)
  - uint, [183](#)
  - vec\_equal, [183](#)
  - vec\_equal\_approx, [183](#)
  - vec\_inner\_prod, [184](#)
- uint
  - typedefs.hpp, [183](#)
- UNKNOWN
  - EXISTS, [24](#)
- update\_annotations
  - Geese, [76](#)
- update\_normalizing\_constant
  - model-bones.hpp, [158](#)
- val
  - Entries< Cell\_Type >, [61](#)
- value
  - Cell< Cell\_Type >, [49](#)
- vec\_diff
  - geese-bones.hpp, [163](#)
- vec\_equal
  - typedefs.hpp, [183](#)
- vec\_equal\_approx
  - typedefs.hpp, [183](#)
- vec\_inner\_prod
  - typedefs.hpp, [184](#)
- vecHasher< T >, [130](#)
  - operator(), [130](#)
- vector\_caster
  - geese-bones.hpp, [163](#)
- vertex\_attr
  - NetworkData, [97](#)
- visited
  - BArray< Cell\_Type, Data\_Type >, [40](#)
  - Cell< Cell\_Type >, [49](#)
  - Node, [102](#)
- with\_pset
  - Model<     Array\_Type,     Data\_Counter\_Type,  
          Data\_Rule\_Type >, [93](#)
- zero\_col
  - BArray< Cell\_Type, Data\_Type >, [38](#)
- zero\_row
  - BArray< Cell\_Type, Data\_Type >, [38](#)