

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
5.5 Phylo rules	20
5.5.1 Detailed Description	20
5.5.2 Function Documentation	20
5.5.2.1 rule_dyn_limit_changes()	20
6 Namespace Documentation	23
6.1 barry Namespace Reference	23
6.1.1 Detailed Description	23
6.2 barry::counters Namespace Reference	23
6.2.1 Detailed Description	23
6.3 barry::counters::network Namespace Reference	24
6.4 barry::counters::phylo Namespace Reference	24
6.5 CHECK Namespace Reference	24
6.5.1 Detailed Description	24
6.5.2 Variable Documentation	24
6.5.2.1 BOTH	24
6.5.2.2 NONE	24
6.5.2.3 ONE	24
6.5.2.4 TWO	25
6.6 EXISTS Namespace Reference	25
6.6.1 Detailed Description	25
6.6.2 Variable Documentation	25
6.6.2.1 AS_ONE	25
6.6.2.2 AS_ZERO	25
6.6.2.3 BOTH	26
6.6.2.4 NONE	26
6.6.2.5 ONE	26
6.6.2.6 TWO	26
6.6.2.7 UNKNOWN	26
7 Class Documentation	27

7.1 BArray< Cell_Type, Data_Type > Class Template Reference	27
7.1.1 Detailed Description	29
7.1.2 Constructor & Destructor Documentation	30
7.1.2.1 BArray() [1/6]	30
7.1.2.2 BArray() [2/6]	30
7.1.2.3 BArray() [3/6]	30
7.1.2.4 BArray() [4/6]	31
7.1.2.5 BArray() [5/6]	31
7.1.2.6 BArray() [6/6]	31
7.1.2.7 ~BArray()	31
7.1.3 Member Function Documentation	31
7.1.3.1 clear()	31
7.1.3.2 col()	32
7.1.3.3 D() [1/2]	32
7.1.3.4 D() [2/2]	32
7.1.3.5 default_val()	32
7.1.3.6 get_cell()	32
7.1.3.7 get_col()	32
7.1.3.8 get_col_vec() [1/2]	33
7.1.3.9 get_col_vec() [2/2]	33
7.1.3.10 get_entries()	33
7.1.3.11 get_row()	33
7.1.3.12 get_row_vec() [1/2]	33
7.1.3.13 get_row_vec() [2/2]	34
7.1.3.14 insert_cell() [1/3]	34
7.1.3.15 insert_cell() [2/3]	34
7.1.3.16 insert_cell() [3/3]	34
7.1.3.17 is_empty()	34
7.1.3.18 ncol()	35
7.1.3.19 nnozero()	35
7.1.3.20 nrow()	35
7.1.3.21 operator()() [1/2]	35
7.1.3.22 operator()() [2/2]	35
7.1.3.23 operator*=()	35
7.1.3.24 operator+=() [1/3]	36
7.1.3.25 operator+=() [2/3]	36
7.1.3.26 operator+=() [3/3]	36
7.1.3.27 operator-=() [1/3]	36
7.1.3.28 operator-=() [2/3]	36
7.1.3.29 operator-=() [3/3]	36
7.1.3.30 operator/=()	37
7.1.3.31 operator=() [1/2]	37

7.1.3.32 operator=() [2/2]	37
7.1.3.33 operator==()	37
7.1.3.34 out_of_range()	37
7.1.3.35 print()	37
7.1.3.36 reserve()	38
7.1.3.37 resize()	38
7.1.3.38 rm_cell()	38
7.1.3.39 row()	38
7.1.3.40 set_data()	38
7.1.3.41 swap_cells()	39
7.1.3.42 swap_cols()	39
7.1.3.43 swap_rows()	39
7.1.3.44 toggle_cell()	39
7.1.3.45 toggle_lock()	40
7.1.3.46 transpose()	40
7.1.3.47 zero_col()	40
7.1.3.48 zero_row()	40
7.1.4 Friends And Related Function Documentation	40
7.1.4.1 BArrayCell< Cell_Type, Data_Type >	40
7.1.4.2 BArrayCell_const< Cell_Type, Data_Type >	41
7.1.5 Member Data Documentation	41
7.1.5.1 visited	41
7.2 BArrayCell< Cell_Type, Data_Type > Class Template Reference	41
7.2.1 Detailed Description	41
7.2.2 Constructor & Destructor Documentation	42
7.2.2.1 BArrayCell()	42
7.2.2.2 ~BArrayCell()	42
7.2.3 Member Function Documentation	42
7.2.3.1 operator Cell_Type()	42
7.2.3.2 operator*=()	42
7.2.3.3 operator+=()	43
7.2.3.4 operator-=()	43
7.2.3.5 operator/=()	43
7.2.3.6 operator=()	43
7.2.3.7 operator==()	43
7.3 BArrayCell_const< Cell_Type, Data_Type > Class Template Reference	44
7.3.1 Detailed Description	44
7.3.2 Constructor & Destructor Documentation	44
7.3.2.1 BArrayCell_const()	44
7.3.2.2 ~BArrayCell_const()	44
7.3.3 Member Function Documentation	45
7.3.3.1 operator Cell_Type()	45

7.3.3.2 operator!=(())	45
7.3.3.3 operator<()	45
7.3.3.4 operator<=()	45
7.3.3.5 operator==(())	45
7.3.3.6 operator>()	46
7.3.3.7 operator>=()	46
7.4 BArrayCol< Cell_Type, Data_Type > Class Template Reference	46
7.4.1 Detailed Description	46
7.4.2 Constructor & Destructor Documentation	47
7.4.2.1 BArrayCol()	47
7.4.2.2 ~BArrayCol()	47
7.4.3 Member Function Documentation	47
7.4.3.1 begin()	47
7.4.3.2 end()	47
7.4.3.3 operator Cell_Type()	48
7.4.3.4 operator*=(())	48
7.4.3.5 operator+=(())	48
7.4.3.6 operator-=(())	48
7.4.3.7 operator/=(())	48
7.4.3.8 operator=()	49
7.4.3.9 operator==(())	49
7.5 BArrayCol_const< Cell_Type, Data_Type > Class Template Reference	49
7.5.1 Detailed Description	49
7.5.2 Constructor & Destructor Documentation	50
7.5.2.1 ~BArrayCol_const()	50
7.5.3 Member Function Documentation	50
7.5.3.1 BArrayCol()	50
7.5.3.2 operator!=(())	50
7.5.3.3 operator<()	50
7.5.3.4 operator<=()	51
7.5.3.5 operator==(())	51
7.5.3.6 operator>()	51
7.5.3.7 operator>=()	51
7.6 BArrayDense< Cell_Type, Data_Type > Class Template Reference	51
7.6.1 Detailed Description	52
7.6.2 Constructor & Destructor Documentation	52
7.6.2.1 BArrayDense() [1/2]	52
7.6.2.2 BArrayDense() [2/2]	52
7.6.2.3 ~BArrayDense()	53
7.6.3 Member Function Documentation	53
7.6.3.1 elements_ptr()	53
7.6.3.2 elements_raw()	53

7.6.3.3 fill()	53
7.6.3.4 ncol()	53
7.6.3.5 nrow()	54
7.6.3.6 operator>()	54
7.6.3.7 operator[]()	54
7.6.3.8 print()	54
7.7 Cell< Cell_Type > Class Template Reference	54
7.7.1 Detailed Description	55
7.7.2 Constructor & Destructor Documentation	55
7.7.2.1 Cell() [1/7]	55
7.7.2.2 Cell() [2/7]	56
7.7.2.3 ~Cell()	56
7.7.2.4 Cell() [3/7]	56
7.7.2.5 Cell() [4/7]	56
7.7.2.6 Cell() [5/7]	56
7.7.2.7 Cell() [6/7]	57
7.7.2.8 Cell() [7/7]	57
7.7.3 Member Function Documentation	57
7.7.3.1 add() [1/4]	57
7.7.3.2 add() [2/4]	57
7.7.3.3 add() [3/4]	57
7.7.3.4 add() [4/4]	58
7.7.3.5 operator Cell_Type()	58
7.7.3.6 operator=() [1/2]	58
7.7.3.7 operator=() [2/2]	58
7.7.4 Member Data Documentation	58
7.7.4.1 value	58
7.7.4.2 visited	59
7.8 ConstBArrayRowIter< Cell_Type, Data_Type > Class Template Reference	59
7.8.1 Detailed Description	60
7.8.2 Constructor & Destructor Documentation	60
7.8.2.1 ConstBArrayRowIter()	60
7.8.2.2 ~ConstBArrayRowIter()	60
7.8.3 Member Data Documentation	60
7.8.3.1 Array	60
7.8.3.2 current_col	60
7.8.3.3 current_row	61
7.8.3.4 iter	61
7.9 Counter< Array_Type, Data_Type > Class Template Reference	61
7.9.1 Detailed Description	62
7.9.2 Constructor & Destructor Documentation	62
7.9.2.1 Counter() [1/4]	62

7.9.2.2 Counter() [2/4]	63
7.9.2.3 Counter() [3/4]	63
7.9.2.4 Counter() [4/4]	63
7.9.2.5 ~Counter()	63
7.9.3 Member Function Documentation	63
7.9.3.1 count()	64
7.9.3.2 init()	64
7.9.3.3 operator=() [1/2]	64
7.9.3.4 operator=() [2/2]	64
7.9.4 Member Data Documentation	64
7.9.4.1 count_fun	65
7.9.4.2 data	65
7.9.4.3 delete_data	65
7.9.4.4 desc	65
7.9.4.5 init_fun	65
7.9.4.6 name	66
7.10 Counters< Array_Type, Data_Type > Class Template Reference	66
7.10.1 Detailed Description	66
7.10.2 Constructor & Destructor Documentation	67
7.10.2.1 Counters() [1/3]	67
7.10.2.2 ~Counters()	67
7.10.2.3 Counters() [2/3]	67
7.10.2.4 Counters() [3/3]	67
7.10.3 Member Function Documentation	68
7.10.3.1 add_counter() [1/3]	68
7.10.3.2 add_counter() [2/3]	68
7.10.3.3 add_counter() [3/3]	68
7.10.3.4 clear()	69
7.10.3.5 operator=() [1/2]	69
7.10.3.6 operator=() [2/2]	69
7.10.3.7 operator[]()	70
7.10.3.8 size()	70
7.11 Entries< Cell_Type > Class Template Reference	70
7.11.1 Detailed Description	71
7.11.2 Constructor & Destructor Documentation	71
7.11.2.1 Entries() [1/2]	71
7.11.2.2 Entries() [2/2]	71
7.11.2.3 ~Entries()	71
7.11.3 Member Function Documentation	72
7.11.3.1 resize()	72
7.11.4 Member Data Documentation	72
7.11.4.1 source	72

7.11.4.2 target	72
7.11.4.3 val	72
7.12 Flock Class Reference	73
7.12.1 Detailed Description	73
7.12.2 Constructor & Destructor Documentation	74
7.12.2.1 Flock()	74
7.12.2.2 ~Flock()	74
7.12.3 Member Function Documentation	74
7.12.3.1 add_data()	74
7.12.3.2 get_counters()	75
7.12.3.3 init()	75
7.12.3.4 likelihood_joint()	75
7.12.3.5 nfuncs()	75
7.12.3.6 nleaves()	76
7.12.3.7 nnodes()	76
7.12.3.8 nterms()	76
7.12.3.9 ntrees()	76
7.12.3.10 operator()()	76
7.12.3.11 set_seed()	77
7.12.4 Member Data Documentation	77
7.12.4.1 dat	77
7.12.4.2 initialized	77
7.12.4.3 nfunctions	77
7.12.4.4 rengine	78
7.12.4.5 support	78
7.13 FreqTable< T > Class Template Reference	78
7.13.1 Detailed Description	78
7.13.2 Constructor & Destructor Documentation	79
7.13.2.1 FreqTable()	79
7.13.2.2 ~FreqTable()	79
7.13.3 Member Function Documentation	79
7.13.3.1 add()	79
7.13.3.2 as_vector()	79
7.13.3.3 clear()	80
7.13.3.4 get_data()	80
7.13.3.5 get_data_ptr()	80
7.13.3.6 print()	80
7.13.3.7 reserve()	80
7.13.3.8 size()	81
7.14 Geese Class Reference	81
7.14.1 Detailed Description	83
7.14.2 Constructor & Destructor Documentation	83

7.14.2.1 Geese() [1/4]	83
7.14.2.2 Geese() [2/4]	83
7.14.2.3 Geese() [3/4]	83
7.14.2.4 Geese() [4/4]	83
7.14.2.5 ~Geese()	84
7.14.3 Member Function Documentation	84
7.14.3.1 calc_reduced_sequence()	84
7.14.3.2 calc_sequence()	84
7.14.3.3 get_counters()	84
7.14.3.4 get_probabilities()	84
7.14.3.5 get_engine()	84
7.14.3.6 get_states()	85
7.14.3.7 get_support()	85
7.14.3.8 inherit_support()	85
7.14.3.9 init()	85
7.14.3.10 init_node()	85
7.14.3.11 likelihood()	85
7.14.3.12 likelihood_exhaust()	86
7.14.3.13 nfuncs()	86
7.14.3.14 nleaves()	86
7.14.3.15 nnodes()	86
7.14.3.16 nterms()	86
7.14.3.17 observed_counts()	86
7.14.3.18 operator=() [1/2]	87
7.14.3.19 operator=() [2/2]	87
7.14.3.20 predict()	87
7.14.3.21 predict_backend()	87
7.14.3.22 print_observed_counts()	87
7.14.3.23 set_seed()	88
7.14.3.24 simulate()	88
7.14.3.25 update_annotations()	88
7.14.4 Member Data Documentation	88
7.14.4.1 delete_counters	88
7.14.4.2 delete_engine	88
7.14.4.3 delete_support	89
7.14.4.4 initialized	89
7.14.4.5 map_to_nodes	89
7.14.4.6 nfunctions	89
7.14.4.7 nodes	89
7.14.4.8 reduced_sequence	89
7.14.4.9 sequence	90

7.15 Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference	90
7.15.1 Detailed Description	92
7.15.2 Constructor & Destructor Documentation	92
7.15.2.1 Model() [1/3]	92
7.15.2.2 Model() [2/3]	93
7.15.2.3 Model() [3/3]	93
7.15.2.4 ~Model()	93
7.15.3 Member Function Documentation	93
7.15.3.1 add_array()	93
7.15.3.2 add_counter() [1/3]	94
7.15.3.3 add_counter() [2/3]	94
7.15.3.4 add_counter() [3/3]	94
7.15.3.5 add_rule() [1/3]	94
7.15.3.6 add_rule() [2/3]	95
7.15.3.7 add_rule() [3/3]	95
7.15.3.8 add_rule_dyn() [1/3]	95
7.15.3.9 add_rule_dyn() [2/3]	95
7.15.3.10 add_rule_dyn() [3/3]	96
7.15.3.11 get_counters()	96
7.15.3.12 get_norm_const()	96
7.15.3.13 get_pset()	96
7.15.3.14 get_rengine()	97
7.15.3.15 get_rules()	97
7.15.3.16 get_rules_dyn()	97
7.15.3.17 get_stats()	97
7.15.3.18 likelihood() [1/3]	98
7.15.3.19 likelihood() [2/3]	98
7.15.3.20 likelihood() [3/3]	98
7.15.3.21 likelihood_total()	98
7.15.3.22 nterms()	99
7.15.3.23 operator=()	99
7.15.3.24 print_stats()	99
7.15.3.25 sample() [1/2]	99
7.15.3.26 sample() [2/2]	100
7.15.3.27 set_counters()	100
7.15.3.28 set_keygen()	100
7.15.3.29 set_rengine()	100
7.15.3.30 set_rules()	101
7.15.3.31 set_rules_dyn()	101
7.15.3.32 set_seed()	101
7.15.3.33 size()	101

7.15.3.34 size_unique()	101
7.15.3.35 store_psets()	102
7.16 NetCounterData Class Reference	102
7.16.1 Detailed Description	102
7.16.2 Constructor & Destructor Documentation	102
7.16.2.1 NetCounterData() [1/2]	102
7.16.2.2 NetCounterData() [2/2]	103
7.16.2.3 ~NetCounterData()	103
7.16.3 Member Data Documentation	103
7.16.3.1 indices	103
7.16.3.2 numbers	103
7.17 NetworkData Class Reference	103
7.17.1 Detailed Description	104
7.17.2 Constructor & Destructor Documentation	104
7.17.2.1 NetworkData() [1/3]	104
7.17.2.2 NetworkData() [2/3]	104
7.17.2.3 NetworkData() [3/3]	105
7.17.2.4 ~NetworkData()	105
7.17.3 Member Data Documentation	105
7.17.3.1 directed	105
7.17.3.2 vertex_attr	106
7.18 Node Class Reference	106
7.18.1 Detailed Description	107
7.18.2 Constructor & Destructor Documentation	107
7.18.2.1 Node() [1/5]	107
7.18.2.2 Node() [2/5]	108
7.18.2.3 Node() [3/5]	108
7.18.2.4 Node() [4/5]	108
7.18.2.5 Node() [5/5]	108
7.18.2.6 ~Node()	108
7.18.3 Member Function Documentation	108
7.18.3.1 get_parent()	109
7.18.3.2 is_leaf()	109
7.18.4 Member Data Documentation	109
7.18.4.1 annotations	109
7.18.4.2 array	109
7.18.4.3 arrays	109
7.18.4.4 duplication	110
7.18.4.5 id	110
7.18.4.6 narray	110
7.18.4.7 offspring	110
7.18.4.8 ord	110

7.18.4.9 parent	111
7.18.4.10 probability	111
7.18.4.11 subtree_prob	111
7.18.4.12 visited	111
7.19 NodeData Class Reference	111
7.19.1 Detailed Description	112
7.19.2 Constructor & Destructor Documentation	112
7.19.2.1 NodeData() [1/2]	112
7.19.2.2 NodeData() [2/2]	112
7.19.2.3 ~NodeData()	113
7.19.3 Member Data Documentation	113
7.19.3.1 blengths	113
7.19.3.2 duplication	113
7.19.3.3 states	113
7.20 PhyloRuleDynData Class Reference	113
7.20.1 Detailed Description	114
7.20.2 Constructor & Destructor Documentation	114
7.20.2.1 PhyloRuleDynData()	114
7.20.2.2 ~PhyloRuleDynData()	114
7.20.3 Member Data Documentation	114
7.20.3.1 counts	115
7.20.3.2 duplication	115
7.20.3.3 lb	115
7.20.3.4 pos	115
7.20.3.5 ub	115
7.21 PowerSet< Array_Type, Data_Rule_Type > Class Template Reference	116
7.21.1 Detailed Description	117
7.21.2 Constructor & Destructor Documentation	117
7.21.2.1 PowerSet() [1/3]	117
7.21.2.2 PowerSet() [2/3]	117
7.21.2.3 PowerSet() [3/3]	118
7.21.2.4 ~PowerSet()	118
7.21.3 Member Function Documentation	118
7.21.3.1 add_rule() [1/3]	118
7.21.3.2 add_rule() [2/3]	118
7.21.3.3 add_rule() [3/3]	118
7.21.3.4 begin()	119
7.21.3.5 calc()	119
7.21.3.6 end()	119
7.21.3.7 get_data()	119
7.21.3.8 get_data_ptr()	119
7.21.3.9 init_support()	120

7.21.3.10 operator[]()	120
7.21.3.11 reset()	120
7.21.3.12 size()	120
7.21.4 Member Data Documentation	120
7.21.4.1 coordinates_free	120
7.21.4.2 coordinates_locked	121
7.21.4.3 data	121
7.21.4.4 EmptyArray	121
7.21.4.5 M	121
7.21.4.6 N	121
7.21.4.7 rules	122
7.21.4.8 rules_deleted	122
7.22 Rule< Array_Type, Data_Type > Class Template Reference	122
7.22.1 Detailed Description	123
7.22.2 Constructor & Destructor Documentation	123
7.22.2.1 Rule() [1/2]	123
7.22.2.2 Rule() [2/2]	123
7.22.2.3 ~Rule()	123
7.22.3 Member Function Documentation	124
7.22.3.1 D()	124
7.22.3.2 operator()()	124
7.23 Rules< Array_Type, Data_Type > Class Template Reference	124
7.23.1 Detailed Description	125
7.23.2 Constructor & Destructor Documentation	125
7.23.2.1 Rules() [1/2]	125
7.23.2.2 Rules() [2/2]	125
7.23.2.3 ~Rules()	126
7.23.3 Member Function Documentation	126
7.23.3.1 add_rule() [1/3]	126
7.23.3.2 add_rule() [2/3]	126
7.23.3.3 add_rule() [3/3]	126
7.23.3.4 clear()	126
7.23.3.5 get_seq()	126
7.23.3.6 operator()()	127
7.23.3.7 operator=()	127
7.23.3.8 size()	128
7.24 StatsCounter< Array_Type, Data_Type > Class Template Reference	128
7.24.1 Detailed Description	128
7.24.2 Constructor & Destructor Documentation	129
7.24.2.1 StatsCounter() [1/2]	129
7.24.2.2 StatsCounter() [2/2]	129
7.24.2.3 ~StatsCounter()	129

7.24.3 Member Function Documentation	129
7.24.3.1 add_counter() [1/2]	129
7.24.3.2 add_counter() [2/2]	130
7.24.3.3 count_all()	130
7.24.3.4 count_current()	130
7.24.3.5 count_init()	130
7.24.3.6 get_counters()	130
7.24.3.7 reset_array()	130
7.24.3.8 set_counters()	131
7.25 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference	131
7.25.1 Detailed Description	134
7.25.2 Constructor & Destructor Documentation	134
7.25.2.1 Support() [1/3]	134
7.25.2.2 Support() [2/3]	134
7.25.2.3 Support() [3/3]	135
7.25.2.4 ~Support()	135
7.25.3 Member Function Documentation	135
7.25.3.1 add_counter() [1/2]	135
7.25.3.2 add_counter() [2/2]	135
7.25.3.3 add_rule() [1/2]	136
7.25.3.4 add_rule() [2/2]	136
7.25.3.5 add_rule_dyn() [1/2]	136
7.25.3.6 add_rule_dyn() [2/2]	136
7.25.3.7 calc()	136
7.25.3.8 get_counters()	137
7.25.3.9 get_counts()	137
7.25.3.10 get_counts_ptr()	137
7.25.3.11 get_current_stats()	137
7.25.3.12 get_data()	138
7.25.3.13 get_rules()	138
7.25.3.14 get_rules_dyn()	138
7.25.3.15 init_support()	138
7.25.3.16 print()	139
7.25.3.17 reset_array() [1/2]	139
7.25.3.18 reset_array() [2/2]	139
7.25.3.19 set_counters()	139
7.25.3.20 set_rules()	139
7.25.3.21 set_rules_dyn()	140
7.25.4 Member Data Documentation	140
7.25.4.1 change_stats	140
7.25.4.2 coordinates_free	140

7.25.4.3	coordinates_locked	140
7.25.4.4	current_stats	141
7.25.4.5	delete_counters	141
7.25.4.6	delete_rules	141
7.25.4.7	delete_rules_dyn	141
7.25.4.8	M	141
7.25.4.9	max_num_elements	142
7.25.4.10	N	142
7.26	vecHasher< T > Struct Template Reference	142
7.26.1	Detailed Description	142
7.26.2	Member Function Documentation	142
7.26.2.1	operator()()	142
8	File Documentation	143
8.1	include/barry/barray-bones.hpp File Reference	143
8.1.1	Macro Definition Documentation	144
8.1.1.1	BARRAY_BONES_HPP	144
8.2	include/barry/barray-iterator.hpp File Reference	144
8.2.1	Macro Definition Documentation	145
8.2.1.1	BARRAY_ITERATOR_HPP	145
8.3	include/barry/barray-meat-operators.hpp File Reference	146
8.3.1	Macro Definition Documentation	147
8.3.1.1	COL	147
8.3.1.2	ROW	147
8.3.2	Function Documentation	147
8.3.2.1	checkdim_()	147
8.4	include/barry/barray-meat.hpp File Reference	148
8.4.1	Macro Definition Documentation	149
8.4.1.1	COL	149
8.4.1.2	ROW	149
8.5	include/barry/barraycell-bones.hpp File Reference	149
8.6	include/barry/barraycell-meat.hpp File Reference	150
8.7	include/barry/barraycol-bones.hpp File Reference	151
8.8	include/barry/barraycol-meat.hpp File Reference	152
8.8.1	Macro Definition Documentation	153
8.8.1.1	BARRY_BARRAYCOL_MEAT_HPP	153
8.9	include/barry/barraydense-bones.hpp File Reference	153
8.10	include/barry/barry-configuration.hpp File Reference	154
8.10.1	Macro Definition Documentation	154
8.10.1.1	BARRY_CHECK_SUPPORT	154
8.10.1.2	BARRY_ISFINITE	155
8.10.1.3	BARRY_MAX_NUM_ELEMENTS	155

8.10.1.4 BARRY_SAFE_EXP	155
8.10.2 Typedef Documentation	155
8.10.2.1 Map	155
8.11 include/barry/barry.hpp File Reference	156
8.11.1 Macro Definition Documentation	157
8.11.1.1 BARRY_HPP	157
8.11.1.2 COUNTER_FUNCTION	157
8.11.1.3 COUNTER_LAMBDA	157
8.11.1.4 RULE_FUNCTION	158
8.11.1.5 RULE_LAMBDA	158
8.12 include/barry/cell-bones.hpp File Reference	158
8.13 include/barry/cell-meat.hpp File Reference	159
8.14 include/barry/col-bones.hpp File Reference	160
8.15 include/barry/counters-bones.hpp File Reference	160
8.16 include/barry/counters-meat.hpp File Reference	161
8.17 include/barry/counters/network.hpp File Reference	162
8.17.1 Macro Definition Documentation	165
8.17.1.1 NET_C_DATA_IDX	165
8.17.1.2 NET_C_DATA_NUM	165
8.17.1.3 NETWORK_COUNTER	165
8.17.1.4 NETWORK_COUNTER_LAMBDA	166
8.17.1.5 NETWORK_RULE	166
8.17.1.6 NETWORK_RULE_LAMBDA	166
8.17.2 Typedef Documentation	166
8.17.2.1 NetCounter	166
8.17.2.2 NetCounters	167
8.17.2.3 NetModel	167
8.17.2.4 NetRule	167
8.17.2.5 NetRules	167
8.17.2.6 NetStatsCounter	167
8.17.2.7 NetSupport	167
8.17.2.8 Network	168
8.17.3 Function Documentation	168
8.17.3.1 rules_zerodiag()	168
8.18 include/barry/counters/phylo.hpp File Reference	168
8.18.1 Macro Definition Documentation	170
8.18.1.1 PHYLO_CHECK_MISSING	170
8.18.1.2 PHYLO_COUNTER_LAMBDA	171
8.18.1.3 PHYLO_RULE_DYN_LAMBDA	171
8.18.2 Typedef Documentation	171
8.18.2.1 PhyloArray	171
8.18.2.2 PhyloCounter	171

8.18.2.3 PhyloCounterData	172
8.18.2.4 PhyloCounters	172
8.18.2.5 PhyloModel	172
8.18.2.6 PhyloPowerSet	172
8.18.2.7 PhyloRule	172
8.18.2.8 PhyloRuleData	172
8.18.2.9 PhyloRuleDyn	173
8.18.2.10 PhyloRules	173
8.18.2.11 PhyloRulesDyn	173
8.18.2.12 PhyloStatsCounter	173
8.18.2.13 PhyloSupport	173
8.18.3 Function Documentation	173
8.18.3.1 get_last_name()	173
8.19 include/barry/model-bones.hpp File Reference	174
8.19.1 Function Documentation	175
8.19.1.1 keygen_default()	175
8.19.1.2 likelihood_()	175
8.19.1.3 update_normalizing_constant()	175
8.20 include/barry/model-meat.hpp File Reference	176
8.21 include/barry/models/geese.hpp File Reference	176
8.22 include/barry/models/geese/flock-bones.hpp File Reference	177
8.23 include/barry/models/geese/flock-meet.hpp File Reference	178
8.24 include/barry/models/geese/geese-bones.hpp File Reference	178
8.24.1 Macro Definition Documentation	179
8.24.1.1 INITIALIZED	179
8.24.2 Function Documentation	179
8.24.2.1 keygen_full()	179
8.24.2.2 RULE_FUNCTION()	179
8.24.2.3 vec_diff()	180
8.24.2.4 vector_caster()	180
8.25 include/barry/models/geese/geese-meat-constructors.hpp File Reference	180
8.26 include/barry/models/geese/geese-meat-likelihood.hpp File Reference	181
8.27 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference	182
8.28 include/barry/models/geese/geese-meat-predict.hpp File Reference	182
8.29 include/barry/models/geese/geese-meat-simulate.hpp File Reference	183
8.30 include/barry/models/geese/geese-meat.hpp File Reference	183
8.31 include/barry/models/geese/geese-node-bones.hpp File Reference	184
8.32 include/barry/powerset-bones.hpp File Reference	184
8.33 include/barry/powerset-meat.hpp File Reference	186
8.34 include/barry/rules-bones.hpp File Reference	187
8.34.1 Function Documentation	188
8.34.1.1 rule_fun_default()	188

8.35 include/barry/rules-meat.hpp File Reference	189
8.36 include/barry/statscounter-bones.hpp File Reference	189
8.37 include/barry/statscounter-meat.hpp File Reference	191
8.38 include/barry/statsdb.hpp File Reference	192
8.39 include/barry/support-bones.hpp File Reference	192
8.40 include/barry/support-meat.hpp File Reference	194
8.40.1 Macro Definition Documentation	195
8.40.1.1 BARRY_SUPPORT_MEAT_HPP	195
8.41 include/barry/typedefs.hpp File Reference	195
8.41.1 Typedef Documentation	197
8.41.1.1 Col_type	197
8.41.1.2 Counter_fun_type	197
8.41.1.3 Counts_type	197
8.41.1.4 MapVec_type	197
8.41.1.5 Row_type	198
8.41.1.6 Rule_fun_type	198
8.41.1.7 uint	198
8.41.2 Function Documentation	198
8.41.2.1 vec_equal()	198
8.41.2.2 vec_equal_approx()	199
8.41.2.3 vec_inner_prod()	199
8.42 README.md File Reference	199

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
Phylo rules	20

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	27
BArrayCell< Cell_Type, Data_Type >	41
BArrayCell_const< Cell_Type, Data_Type >	44
BArrayCol< Cell_Type, Data_Type >	46
BArrayCol_const< Cell_Type, Data_Type >	49
BArrayDense< Cell_Type, Data_Type >	
Dense bi-dimensional array	51
Cell< Cell_Type >	
Entries in BArray . For now, it only has two members:	54
ConstBArrayRowIter< Cell_Type, Data_Type >	59
Counter< Array_Type, Data_Type >	
A counter function based on change statistics	61
Counters< Array_Type, Data_Type >	
Vector of counters	66
Entries< Cell_Type >	
A wrapper class to store <code>source</code> , <code>target</code> , <code>val</code> from a BArray object	70
Flock	
A Flock is a group of Geese	73
FreqTable< T >	
Database of statistics	78
Geese	
Annotated Phylo Model	81
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >	
General framework for discrete exponential models. This class allows generating discrete exponential models in the form of a linear exponential model:	90
NetCounterData	
Data class used to store arbitrary uint or double vectors	102
NetworkData	
Data class for Networks	103
Node	
A single node for the model	106
NodeData	
Data definition for the PhyloArray class	111
PhyloRuleDynData	113

PowerSet< Array_Type, Data_Rule_Type >	
Powerset of a binary array	116
Rule< Array_Type, Data_Type >	
Rule for determining if a cell should be included in a sequence	122
Rules< Array_Type, Data_Type >	
Vector of objects of class Rule	124
StatsCounter< Array_Type, Data_Type >	
Count stats for a single Array	128
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >	
Compute the support of sufficient statistics	131
vecHasher< T >	142

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

include/barry/barray-bones.hpp	143
include/barry/barray-iterator.hpp	144
include/barry/barray-meat-operators.hpp	146
include/barry/barray-meat.hpp	148
include/barry/barraycell-bones.hpp	149
include/barry/barraycell-meat.hpp	150
include/barry/barraycol-bones.hpp	151
include/barry/barraycol-meat.hpp	152
include/barry/barraydense-bones.hpp	153
include/barry/barray-configuration.hpp	154
include/barry/barray.hpp	156
include/barry/cell-bones.hpp	158
include/barry/cell-meat.hpp	159
include/barry/col-bones.hpp	160
include/barry/counters-bones.hpp	160
include/barry/counters-meat.hpp	161
include/barry/model-bones.hpp	174
include/barry/model-meat.hpp	176
include/barry/powerset-bones.hpp	184
include/barry/powerset-meat.hpp	186
include/barry/rules-bones.hpp	187
include/barry/rules-meat.hpp	189
include/barry/statscounter-bones.hpp	189
include/barry/statscounter-meat.hpp	191
include/barry/statsdb.hpp	192
include/barry/support-bones.hpp	192
include/barry/support-meat.hpp	194
include/barry/typedefs.hpp	195
include/barry/counters/network.hpp	162
include/barry/counters/phylo.hpp	168
include/barry/models/geese.hpp	176
include/barry/models/geese/flock-bones.hpp	177
include/barry/models/geese/flock-meet.hpp	178
include/barry/models/geese/geese-bones.hpp	178
include/barry/models/geese/geese-meat-constructors.hpp	180

include/barry/models/geese/geese-meat-likelihood.hpp	181
include/barry/models/geese/geese-meat-likelihood_exhaust.hpp	182
include/barry/models/geese/geese-meat-predict.hpp	182
include/barry/models/geese/geese-meat-simulate.hpp	183
include/barry/models/geese/geese-meat.hpp	183
include/barry/models/geese/geese-node-bones.hpp	184

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](#), [Data_Rule_Dyn_Type](#) >
General framework for discrete exponential models. This class allows generating discrete exponential models in the form of a linear exponential model:
- class [Flock](#)
A [Flock](#) is a group of [Geese](#).
- class [Geese](#)
Annotated Phylo [Model](#).

5.2.1 Detailed Description

Statistical models available in `barry`.

5.3 Network counters

[Counters](#) for network models.

Functions

- void [counter_edges](#) ([NetCounters](#) *counters)
Number of edges.
- void [counter_isolates](#) ([NetCounters](#) *counters)
Number of isolated vertices.
- void [counter_mutual](#) ([NetCounters](#) *counters)
Number of mutual ties.
- void [counter_istar2](#) ([NetCounters](#) *counters)
- void [counter_ostar2](#) ([NetCounters](#) *counters)
- void [counter_ttriads](#) ([NetCounters](#) *counters)
- void [counter_ctriads](#) ([NetCounters](#) *counters)
- void [counter_density](#) ([NetCounters](#) *counters)
- void [counter_idegree15](#) ([NetCounters](#) *counters)
- void [counter_odegree15](#) ([NetCounters](#) *counters)
- void [counter_absdiff](#) ([NetCounters](#) *counters, [uint](#) attr_id, double alpha=1.0)
Sum of absolute attribute difference between ego and alter.
- void [counter_diff](#) ([NetCounters](#) *counters, [uint](#) attr_id, double alpha=1.0, double tail_head=true)
Sum of attribute difference between ego and alter to pow(alpha)
- [NETWORK_COUNTER](#) (init_single_attr)
- void [counter_nodeicov](#) ([NetCounters](#) *counters, [uint](#) attr_id)
- void [counter_nodeocov](#) ([NetCounters](#) *counters, [uint](#) attr_id)
- void [counter_nodecov](#) ([NetCounters](#) *counters, [uint](#) attr_id)
- void [counter_nodematch](#) ([NetCounters](#) *counters, [uint](#) attr_id)
- void [counter_idegree](#) ([NetCounters](#) *counters, std::vector< [uint](#) > d)
Counts number of vertices with a given in-degree.
- void [counter_odegree](#) ([NetCounters](#) *counters, std::vector< [uint](#) > d)
Counts number of vertices with a given out-degree.
- void [counter_degree](#) ([NetCounters](#) *counters, std::vector< [uint](#) > d)
Counts number of vertices with a given out-degree.

5.3.1 Detailed Description

[Counters](#) for network models.

Parameters

<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 1084 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 714 of file phylo.hpp.

5.4.2.3 counter_gains()

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

Functional gains for a specific function (nfun).

Definition at line 152 of file phylo.hpp.

5.4.2.4 counter_gains_k_offspring()

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

k genes gain function nfun

Definition at line 194 of file phylo.hpp.

5.4.2.5 counter_genes_changing()

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

Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)

Definition at line 268 of file phylo.hpp.

5.4.2.6 counter_longest()

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

Longest branch mutates (either by gain or by loss)

Definition at line 773 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 515 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 431 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 878 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 963 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 562 of file phylo.hpp.

5.4.2.12 counter_overall_gains()

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

Overall functional gains.

Total number of gains (irrespective of the function).

Definition at line 112 of file phylo.hpp.

5.4.2.13 counter_overall_loss()

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

Overall functional loss.

Definition at line 385 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 628 of file phylo.hpp.

5.5 Phylo rules

[Rules](#) for phylogenetic modeling.

Classes

- class [PhyloRuleDynData](#)

Functions

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

5.5.1 Detailed Description

[Rules](#) for phylogenetic modeling.

Parameters

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

5.5.2 Function Documentation

5.5.2.1 [rule_dyn_limit_changes\(\)](#)

```
void rule_dyn_limit_changes (
    PhyloSupport * support,
    uint pos,
    uint lb,
    uint ub,
    bool duplication = true ) [inline]
```

Overall functional gains.

Parameters

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

Returns

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

Definition at line 1217 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 20 of file typedefs.hpp.

6.5.2.2 NONE

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

Definition at line 21 of file typedefs.hpp.

6.5.2.3 ONE

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

Definition at line 22 of file typedefs.hpp.

6.5.2.4 TWO

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

Definition at line 23 of file typedefs.hpp.

6.6 EXISTS Namespace Reference

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

Variables

- const int BOTH = -1
- const int NONE = 0
- const int ONE = 1
- const int TWO = 1
- const int 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 38 of file typedefs.hpp.

6.6.2.2 AS_ZERO

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

Definition at line 37 of file typedefs.hpp.

6.6.2.3 BOTH

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

Definition at line 31 of file typedefs.hpp.

6.6.2.4 NONE

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

Definition at line 32 of file typedefs.hpp.

6.6.2.5 ONE

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

Definition at line 33 of file typedefs.hpp.

6.6.2.6 TWO

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

Definition at line 34 of file typedefs.hpp.

6.6.2.7 UNKNOWN

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

Definition at line 36 of file typedefs.hpp.

Chapter 7

Class Documentation

7.1 BArray< Cell_Type, Data_Type > Class Template Reference

Baseline class for binary arrays.

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

Public Member Functions

- bool `operator==` (const `BArray`< `Cell_Type`, `Data_Type` > &`Array_`)
- `~BArray` ()
- void `out_of_range` (uint i, uint j) const
- `Cell_Type` `get_cell` (uint i, uint j, bool check_bounds=true) const
- const `Row_type`< `Cell_Type` > * `get_row` (uint i, bool check_bounds=true) const
- const `Col_type`< `Cell_Type` > * `get_col` (uint i, bool check_bounds=true) const
- std::vector< `Cell_Type` > `get_col_vec` (uint i, bool check_bounds=true) const
- std::vector< `Cell_Type` > `get_row_vec` (uint i, bool check_bounds=true) const
- void `get_col_vec` (std::vector< `Cell_Type` > *x, uint i, bool check_bounds=true) const
- void `get_row_vec` (std::vector< `Cell_Type` > *x, uint i, bool check_bounds=true) const
- const `Row_type`< `Cell_Type` > & `row` (uint i, bool check_bounds=true) const
- const `Col_type`< `Cell_Type` > & `col` (uint i, bool check_bounds=true) const
- `Entries`< `Cell_Type` > `get_entries` () const
 - Get the edgelist.*
- void `transpose` ()
- void `clear` (bool hard=true)
- void `resize` (uint N_, uint M_)
- void `reserve` ()
- void `print` () const

Constructors

Parameters

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

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

- void **set_data** (Data_Type *data_, bool delete_data_=false)

Set the data object.

- Data_Type * **D** ()
- const Data_Type * **D** () const

Queries

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

Parameters

i,j	<i>Coordinates</i>
check_bounds	<i>If false avoids checking bounds.</i>

- bool **is_empty** (uint i, uint j, bool check_bounds=true) const
- uint **nrow** () const noexcept
- uint **ncol** () const noexcept
- uint **nnozero** () const noexcept
- **Cell**< Cell_Type > **default_val** () const

Cell-wise insertion/deletion

Parameters

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

- **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

- `bool visited` = `false`

Friends

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

7.1.1 Detailed Description

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

Baseline class for binary arrays.

`BArray` class objects are arbitrary arrays in which non-empty cells hold data of type `Cell_Type`. The non-empty cells are stored by row and indexed using `unordered_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 60 of file barray-bones.hpp.

7.1.2.2 BArray() [2/6]

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

Empty array.

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

7.1.2.3 BArray() [3/6]

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

Edgelist with data.

7.1.2.4 BArray() [4/6]

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

Edgelist with no data (simpler)

7.1.2.5 BArray() [5/6]

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

Copy constructor.

7.1.2.6 BArray() [6/6]

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

Move operator.

7.1.2.7 ~BArray()

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

7.1.3 Member Function Documentation

7.1.3.1 clear()

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

7.1.3.2 col()

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

7.1.3.3 D() [1/2]

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

7.1.3.4 D() [2/2]

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

7.1.3.5 default_val()

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

7.1.3.6 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
```

7.1.3.7 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
```

7.1.3.8 get_col_vec() [1/2]

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

7.1.3.9 get_col_vec() [2/2]

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

7.1.3.10 get_entries()

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

Get the edgelist.

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

Returns

`Entries<Cell_Type>`

7.1.3.11 get_row()

```
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
```

7.1.3.12 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
```

7.1.3.13 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
```

7.1.3.14 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 )
```

7.1.3.15 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 )
```

7.1.3.16 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 )
```

7.1.3.17 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
```

7.1.3.18 ncol()

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

7.1.3.19 nnozero()

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

7.1.3.20 nrow()

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

7.1.3.21 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 )
```

7.1.3.22 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
```

7.1.3.23 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 )
```

7.1.3.24 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-=() [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 )
```

7.1.3.28 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 )
```

7.1.3.29 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 )
```


7.1.3.30 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 )
```

7.1.3.31 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.

7.1.3.32 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.

7.1.3.33 operator==()

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

7.1.3.34 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
```

7.1.3.35 print()

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

7.1.3.36 reserve()

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

7.1.3.37 resize()

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

7.1.3.38 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 )
```

7.1.3.39 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
```

7.1.3.40 set_data()

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

Set the data object.

Parameters

<i>data_</i>	
<i>delete_↔</i>	
<i>data_</i>	

7.1.3.41 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 )
```

7.1.3.42 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 )
```

7.1.3.43 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 )
```

7.1.3.44 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 )
```

7.1.3.45 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 )
```

7.1.3.46 transpose()

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

7.1.3.47 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 )
```

7.1.3.48 zero_row()

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

7.1.4 Friends And Related Function Documentation

7.1.4.1 BArrayCell< Cell_Type, Data_Type >

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

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

7.1.4.2 BArrayCell_const< Cell_Type, Data_Type >

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

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

7.1.5 Member Data Documentation

7.1.5.1 visited

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

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

Definition at line 45 of file barray-bones.hpp.

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

- [include/barry/barray-bones.hpp](#)

7.2 BArrayCell< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

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

7.2.1 Detailed Description

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

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

7.2.2 Constructor & Destructor Documentation

7.2.2.1 BArrayCell()

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

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

7.2.2.2 ~BArrayCell()

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

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

7.2.3 Member Function Documentation

7.2.3.1 operator Cell_Type()

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

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

7.2.3.2 operator*=()

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

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

7.2.3.3 operator+=()

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

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

7.2.3.4 operator-=()

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

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

7.2.3.5 operator/=()

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

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

7.2.3.6 operator=()

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

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

7.2.3.7 operator==()

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

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

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

- [include/barry/barraycell-bones.hpp](#)
- [include/barry/barraycell-meat.hpp](#)

7.3 BArrayCell_const< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

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

7.3.1 Detailed Description

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

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

7.3.2 Constructor & Destructor Documentation

7.3.2.1 BArrayCell_const()

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

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

7.3.2.2 ~BArrayCell_const()

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

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

7.3.3 Member Function Documentation

7.3.3.1 operator Cell_Type()

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

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

7.3.3.2 operator"!="()

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

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

7.3.3.3 operator<()

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

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

7.3.3.4 operator<=()

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

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

7.3.3.5 operator==(())

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

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

7.3.3.6 operator>()

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

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

7.3.3.7 operator>=()

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

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

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

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

7.4 BArrayCol< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

- [BArrayCol](#) ([BArray](#)< Cell_Type, Data_Type > *Array_, [uint](#) i_, bool check_bounds=true)
- [~BArrayCol](#) ()
- 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
- [Col_type](#)< Cell_Type >::iterator [begin](#) () noexcept
- [Col_type](#)< Cell_Type >::iterator [end](#) () noexcept

7.4.1 Detailed Description

```
template<typename Cell_Type = bool, typename Data_Type = bool>
class BArrayCol< Cell_Type, Data_Type >
```

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

7.4.2 Constructor & Destructor Documentation

7.4.2.1 BArrayCol()

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

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

7.4.2.2 ~BArrayCol()

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

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

7.4.3 Member Function Documentation

7.4.3.1 begin()

```
template<typename Cell_Type , typename Data_Type >
Col_type< Cell_Type >::iterator BArrayCol< Cell_Type, Data_Type >::begin [inline], [noexcept]
```

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

7.4.3.2 end()

```
template<typename Cell_Type , typename Data_Type >
Col_type< Cell_Type >::iterator BArrayCol< Cell_Type, Data_Type >::end [inline], [noexcept]
```

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

7.4.3.3 operator Cell_Type()

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

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

7.4.3.4 operator*=()

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

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

7.4.3.5 operator+=()

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

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

7.4.3.6 operator-=()

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

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

7.4.3.7 operator/=()

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

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

7.4.3.8 operator=()

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

Definition at line 7 of file `barraycol-meat.hpp`.

7.4.3.9 operator==()

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

Definition at line 63 of file `barraycol-meat.hpp`.

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

- `include/barry/barraycol-bones.hpp`
- `include/barry/barraycol-meat.hpp`

7.5 BArrayCol_const< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

- `BArrayCol` (const `BArray`< Cell_Type, Data_Type > *Array_, `uint` i_, bool check_bounds=true)
- `~BArrayCol_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

7.5.1 Detailed Description

```
template<typename Cell_Type = bool, typename Data_Type = bool>
class BArrayCol_const< Cell_Type, Data_Type >
```

Definition at line 42 of file `barraycol-bones.hpp`.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 ~BArrayCol_const()

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

Definition at line 60 of file `barraycol-bones.hpp`.

7.5.3 Member Function Documentation

7.5.3.1 BArrayCol()

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

Definition at line 50 of file `barraycol-bones.hpp`.

7.5.3.2 operator"!="()

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

Definition at line 92 of file `barraycol-meat.hpp`.

7.5.3.3 operator<()

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

Definition at line 97 of file `barraycol-meat.hpp`.

7.5.3.4 operator<=()

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

Definition at line 107 of file `barraycol-meat.hpp`.

7.5.3.5 operator==(

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

Definition at line 87 of file `barraycol-meat.hpp`.

7.5.3.6 operator>()

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

Definition at line 102 of file `barraycol-meat.hpp`.

7.5.3.7 operator>=()

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

Definition at line 112 of file `barraycol-meat.hpp`.

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

- `include/barry/barraycol-bones.hpp`
- `include/barry/barraycol-meat.hpp`

7.6 BArrayDense< Cell_Type, Data_Type > Class Template Reference

Dense bi-dimensional array.

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

Public Member Functions

- [BArrayDense](#) ()
- [BArrayDense](#) ([uint](#) N_, [uint](#) M_, [std::vector](#)< [Cell_Type](#) > [elements_](#)={})
- [~BArrayDense](#) ()
- void [fill](#) (const [Cell_Type](#) &d)
- const [std::vector](#)< [Cell_Type](#) > & [elements_raw](#) () const noexcept
- const [std::vector](#)< [Cell_Type](#) > * [elements_ptr](#) () const noexcept
- [uint](#) [nrow](#) () const noexcept
- [uint](#) [ncol](#) () const noexcept
- [Cell_Type](#) [operator](#)() ([uint](#) i, [uint](#) j, bool [check_bounds](#)=true) const
- [Cell_Type](#) [operator](#)[] ([uint](#) i) const
- void [print](#) () const

7.6.1 Detailed Description

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

Dense bi-dimensional array.

[elements](#) is stored in a [std::vector](#), in col-major order.

Template Parameters

<i>Cell_Type</i>	
------------------	--

Definition at line 13 of file [barraydense-bones.hpp](#).

7.6.2 Constructor & Destructor Documentation

7.6.2.1 BArrayDense() [1/2]

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

Definition at line 43 of file [barraydense-bones.hpp](#).

7.6.2.2 BArrayDense() [2/2]

```
template<typename Cell_Type , typename Data_Type >
BArrayDense< Cell_Type, Data_Type >::BArrayDense (
    uint N_,
    uint M_,
    std::vector< Cell\_Type > elements\_ = {} ) [inline]
```

Definition at line 55 of file [barraydense-bones.hpp](#).

7.6.2.3 ~BArrayDense()

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

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

7.6.3 Member Function Documentation

7.6.3.1 elements_ptr()

```
template<typename Cell_Type , typename Data_Type >
const std::vector< Cell_Type > * BArrayDense< Cell_Type, Data_Type >::elements_ptr [inline],
[noexcept]
```

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

7.6.3.2 elements_raw()

```
template<typename Cell_Type , typename Data_Type >
const std::vector< Cell_Type > & BArrayDense< Cell_Type, Data_Type >::elements_raw [inline],
[noexcept]
```

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

7.6.3.3 fill()

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::fill (
    const Cell_Type & d ) [inline]
```

Definition at line 70 of file barraydense-bones.hpp.

7.6.3.4 ncol()

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

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

7.6.3.5 nrow()

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

Definition at line 91 of file `barraydense-bones.hpp`.

7.6.3.6 operator()()

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

Definition at line 103 of file `barraydense-bones.hpp`.

7.6.3.7 operator[]()

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

Definition at line 122 of file `barraydense-bones.hpp`.

7.6.3.8 print()

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

Definition at line 128 of file `barraydense-bones.hpp`.

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

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

7.7 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](#)

7.7.1 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.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 Cell() [1/7]

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

7.7.2.2 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.

7.7.2.3 ~Cell()

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

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

7.7.2.4 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.

7.7.2.5 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.

7.7.2.6 Cell() [5/7]

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

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

7.7.2.7 Cell() [6/7]

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

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

7.7.2.8 Cell() [7/7]

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

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

7.7.3 Member Function Documentation

7.7.3.1 add() [1/4]

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

7.7.3.2 add() [2/4]

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

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

7.7.3.3 add() [3/4]

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

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

7.7.3.4 add() [4/4]

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

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

7.7.3.5 operator Cell_Type()

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

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

7.7.3.6 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.

7.7.3.7 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.

7.7.4 Member Data Documentation

7.7.4.1 value

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

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

7.7.4.2 visited

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

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

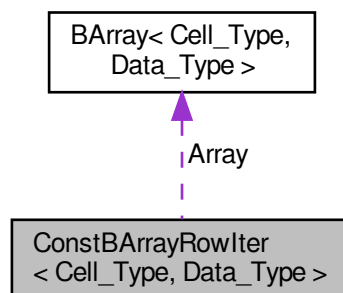
The documentation for this class was generated from the following files:

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

7.8 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`

7.8.1 Detailed Description

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

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

7.8.2 Constructor & Destructor Documentation

7.8.2.1 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.

7.8.2.2 ~ConstBArrayRowIter()

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

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

7.8.3 Member Data Documentation

7.8.3.1 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.

7.8.3.2 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.

7.8.3.3 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.

7.8.3.4 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

7.9 Counter< Array_Type, Data_Type > Class Template Reference

A counter function based on change statistics.

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

Public Member Functions

- [~Counter](#) ()
- double [count](#) (Array_Type &Array, [uint](#) i, [uint](#) j)
- double [init](#) (Array_Type &Array, [uint](#) i, [uint](#) j)

Creator passing a counter and an initializer

Parameters

count_fun_↔ _	The main counter function.
init_fun_	The initializer function can also be used to check if the BArray as the needed variables (see BArray::data).
data_	Data to be used with the counter.
delete_↔ data_	When <code>true</code> , the destructor will delete the pointer in the main data.

- [Counter](#) ()
- [Counter](#) ([Counter_fun_type](#)< Array_Type, Data_Type > count_fun_, [Counter_fun_type](#)< Array_Type, Data_Type > init_fun_=nullptr, Data_Type *data_=nullptr, bool delete_data_=false, std::string name_↔="", std::string desc_="")

- `Counter` (const `Counter`< Array_Type, Data_Type > &counter_)
Copy constructor.
- `Counter` (`Counter`< Array_Type, Data_Type > &&counter_) noexcept
Move constructor.
- `Counter`< Array_Type, Data_Type > `operator=` (const `Counter`< Array_Type, Data_Type > &counter_)
Copy assignment.
- `Counter`< Array_Type, Data_Type > & `operator=` (`Counter`< Array_Type, Data_Type > &&counter_)
Move assignment.

Public Attributes

- `Counter_fun_type`< Array_Type, Data_Type > `count_fun`
- `Counter_fun_type`< Array_Type, Data_Type > `init_fun`
- Data_Type * `data` = nullptr
- bool `delete_data` = false
- std::string `name` = ""
- std::string `desc` = ""

7.9.1 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.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 Counter() [1/4]

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

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

7.9.2.2 Counter() [2/4]

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

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

7.9.2.3 Counter() [3/4]

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

Copy constructor.

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

7.9.2.4 Counter() [4/4]

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

Move constructor.

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

7.9.2.5 ~Counter()

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

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

7.9.3 Member Function Documentation

7.9.3.1 count()

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

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

7.9.3.2 init()

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

Definition at line 126 of file counters-meat.hpp.

7.9.3.3 operator=() [1/2]

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

Copy assignment.

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

7.9.3.4 operator=() [2/2]

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

Move assignment.

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

7.9.4 Member Data Documentation

7.9.4.1 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.

7.9.4.2 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.

7.9.4.3 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.

7.9.4.4 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.

7.9.4.5 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.

7.9.4.6 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

7.10 Counters< Array_Type, Data_Type > Class Template Reference

Vector of counters.

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

Public Member Functions

- [Counters](#) ()
- [~Counters](#) ()
- [Counters](#) (const [Counters](#)< Array_Type, Data_Type > &counter_)
Copy constructor.
- [Counters](#) ([Counters](#)< Array_Type, Data_Type > &&counters_) noexcept
Move constructor.
- [Counters](#)< Array_Type, Data_Type > [operator=](#) (const [Counters](#)< Array_Type, Data_Type > &counter_)
Copy assignment constructor.
- [Counters](#)< Array_Type, Data_Type > & [operator=](#) ([Counters](#)< Array_Type, Data_Type > &&counter_) noexcept
Move assignment constructor.
- [Counter](#)< Array_Type, Data_Type > & [operator\[\]](#) (uint idx)
Returns a pointer to a particular counter.
- std::size_t [size](#) () const noexcept
Number of counters in the set.
- void [add_counter](#) ([Counter](#)< Array_Type, Data_Type > &counter)
- void [add_counter](#) ([Counter](#)< Array_Type, Data_Type > *counter)
- void [add_counter](#) ([Counter_fun_type](#)< Array_Type, Data_Type > count_fun_, [Counter_fun_type](#)< Array_Type, Data_Type > init_fun_=nullptr, Data_Type *data_=nullptr, bool delete_data_=false, std::string name_="", std::string desc_="")
- void [clear](#) ()

7.10.1 Detailed Description

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

Vector of counters.

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

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

7.10.2 Constructor & Destructor Documentation

7.10.2.1 Counters() [1/3]

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

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

7.10.2.2 ~Counters()

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

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

7.10.2.3 Counters() [2/3]

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

Copy constructor.

Parameters

<i>counter_↔</i>	
—	

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

7.10.2.4 Counters() [3/3]

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

Move constructor.

Parameters

<i>counters</i> ↔	
—	

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

7.10.3 Member Function Documentation

7.10.3.1 add_counter() [1/3]

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

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

7.10.3.2 add_counter() [2/3]

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

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

7.10.3.3 add_counter() [3/3]

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

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

7.10.3.4 clear()

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

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

7.10.3.5 operator=() [1/2]

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

Copy assignment constructor.

Parameters

<i>counter_↔</i>	
—	

Returns

Counters<Array_Type,Data_Type>

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

7.10.3.6 operator=() [2/2]

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

Move assignment constructor.

Parameters

<i>counter_↔</i>	
—	

Returns

Counters<Array_Type,Data_Type>&

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

7.10.3.7 operator[]()

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

Returns a pointer to a particular counter.

Parameters

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

Returns

Counter<Array_Type,Data_Type>*

Definition at line 153 of file counters-meat.hpp.

7.10.3.8 size()

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

Number of counters in the set.

Returns

uint

Definition at line 159 of file counters-bones.hpp.

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

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

7.11 Entries< Cell_Type > Class Template Reference

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

```
#include <typedefs.hpp>
```

Public Member Functions

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

Public Attributes

- `std::vector< uint >` [source](#)
- `std::vector< uint >` [target](#)
- `std::vector< Cell_Type >` [val](#)

7.11.1 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 59 of file `typedefs.hpp`.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 Entries() [1/2]

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

Definition at line 65 of file `typedefs.hpp`.

7.11.2.2 Entries() [2/2]

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

Definition at line 66 of file `typedefs.hpp`.

7.11.2.3 ~Entries()

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

Definition at line 73 of file `typedefs.hpp`.

7.11.3 Member Function Documentation

7.11.3.1 `resize()`

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

Definition at line 75 of file typedefs.hpp.

7.11.4 Member Data Documentation

7.11.4.1 `source`

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

Definition at line 61 of file typedefs.hpp.

7.11.4.2 `target`

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

Definition at line 62 of file typedefs.hpp.

7.11.4.3 `val`

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

Definition at line 63 of file typedefs.hpp.

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

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

7.12 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](#) * [get_counters](#) ()
- 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 [engine](#)
- [phylocounters::PhyloModel](#) [support](#) = [phylocounters::PhyloModel](#)()

7.12.1 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.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 Flock()

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

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

7.12.2.2 ~Flock()

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

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

7.12.3 Member Function Documentation

7.12.3.1 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 Geese::Geese .
<i>geneid</i>	see Geese .
<i>parent</i>	see Geese .
<i>duplication</i>	see Geese .

Returns

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

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

7.12.3.2 get_counters()

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

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

7.12.3.3 init()

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

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

7.12.3.4 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.

7.12.3.5 nfuncs()

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

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

7.12.3.6 nleafs()

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

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

7.12.3.7 nnodes()

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

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

7.12.3.8 nterms()

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

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

7.12.3.9 ntrees()

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

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

7.12.3.10 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.

7.12.3.11 set_seed()

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

Set the seed of the model.

Parameters

s	Passed to the <code>rengine.seed()</code> member object.
---	--

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

7.12.4 Member Data Documentation

7.12.4.1 dat

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

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

7.12.4.2 initialized

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

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

7.12.4.3 nfunctions

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

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

7.12.4.4 engine

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

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

7.12.4.5 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

7.13 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

7.13.1 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.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 FreqTable()

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

Definition at line 28 of file statsdb.hpp.

7.13.2.2 ~FreqTable()

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

Definition at line 29 of file statsdb.hpp.

7.13.3 Member Function Documentation

7.13.3.1 add()

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

Definition at line 47 of file statsdb.hpp.

7.13.3.2 as_vector()

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

Definition at line 61 of file statsdb.hpp.

7.13.3.3 clear()

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

Definition at line 83 of file statsdb.hpp.

7.13.3.4 get_data()

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

Definition at line 73 of file statsdb.hpp.

7.13.3.5 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.

7.13.3.6 print()

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

Definition at line 102 of file statsdb.hpp.

7.13.3.7 reserve()

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

Definition at line 89 of file statsdb.hpp.

7.13.3.8 size()

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

Definition at line 126 of file statsdb.hpp.

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

- include/barry/statsdb.hpp

7.14 Geese Class Reference

Annotated Phylo [Model](#).

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

Public Member Functions

- [~Geese](#) ()
- void [init](#) ()
- void [inherit_support](#) (const [Geese](#) &model_, bool delete_support_=false)
- void [calc_sequence](#) ([Node](#) *n=nullptr)
- void [calc_reduced_sequence](#) ()
- double [likelihood](#) (const std::vector< double > &par, bool as_log=false, bool use_reduced_sequence=true)
- double [likelihood_exhaust](#) (const std::vector< double > &par)
- std::vector< double > [get_probabilities](#) () const
- void [set_seed](#) (const unsigned int &s)
- std::vector< std::vector< unsigned int > > [simulate](#) (const std::vector< double > &par)
- std::vector< std::vector< double > > [observed_counts](#) ()
- void [print_observed_counts](#) ()
- void [init_node](#) ([Node](#) &n)
- void [update_annotations](#) (unsigned int nodeid, std::vector< unsigned int > newann)
- std::mt19937 * [get_rengine](#) ()
- [phylocounters::PhyloCounters](#) * [get_counters](#) ()
- [phylocounters::PhyloModel](#) * [get_support](#) ()
- std::vector< std::vector< bool > > [get_states](#) ()

Construct a new Geese object

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

Parameters

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

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

Information about the model

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

When *res_prob* is specified, the function will attach the member vector probabilities from the [Nodes](#) objects. This contains the probability that the *ith* node has either of the possible states.

Returns

std::vector< double > Returns the posterior probability

- std::vector< std::vector< double > > [predict](#) (const std::vector< double > &par, std::vector< std::vector< double > > *res_prob=nullptr, bool leave_one_out=false, bool only_annotated=false, bool use_reduced_sequence=true)
- std::vector< std::vector< double > > [predict_backend](#) (const std::vector< double > &par, bool use_reduced_sequence, const std::vector< uint > &preorder)

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_engine](#) = false
- bool [delete_counters](#) = false
- bool [delete_support](#) = false

7.14.1 Detailed Description

Annotated Phylo [Model](#).

A list of available terms for this model can be found in the [Phylo counters](#) section.

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

7.14.2 Constructor & Destructor Documentation

7.14.2.1 Geese() [1/4]

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

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

7.14.2.2 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.

7.14.2.3 Geese() [3/4]

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

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

7.14.2.4 Geese() [4/4]

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

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

7.14.2.5 ~Geese()

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

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

7.14.3 Member Function Documentation

7.14.3.1 calc_reduced_sequence()

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

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

7.14.3.2 calc_sequence()

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

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

7.14.3.3 get_counters()

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

Definition at line 432 of file geese-meat.hpp.

7.14.3.4 get_probabilities()

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

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

7.14.3.5 get_rengine()

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

Definition at line 428 of file geese-meat.hpp.

7.14.3.6 get_states()

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

Definition at line 440 of file geese-meat.hpp.

7.14.3.7 get_support()

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

Definition at line 436 of file geese-meat.hpp.

7.14.3.8 inherit_support()

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

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

7.14.3.9 init()

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

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

7.14.3.10 init_node()

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

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

7.14.3.11 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.

7.14.3.12 likelihood_exhaust()

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

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

7.14.3.13 nfuncs()

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

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

7.14.3.14 nleafs()

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

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

7.14.3.15 nnodes()

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

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

7.14.3.16 nterms()

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

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

7.14.3.17 observed_counts()

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

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

7.14.3.18 operator=() [1/2]

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

7.14.3.19 operator=() [2/2]

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

7.14.3.20 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 166 of file geese-meat-predict.hpp.

7.14.3.21 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.

7.14.3.22 print_observed_counts()

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

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

7.14.3.23 `set_seed()`

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

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

7.14.3.24 `simulate()`

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

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

7.14.3.25 `update_annotations()`

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

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

7.14.4 Member Data Documentation

7.14.4.1 `delete_counters`

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

Definition at line 107 of file geese-bones.hpp.

7.14.4.2 `delete_engine`

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

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

7.14.4.3 delete_support

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

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

7.14.4.4 initialized

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

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

7.14.4.5 map_to_nodes

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

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

7.14.4.6 nfunctions

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

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

7.14.4.7 nodes

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

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

7.14.4.8 reduced_sequence

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

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

7.14.4.9 sequence

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

Definition at line 101 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

7.15 Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference

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

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

Public Member Functions

- void [set_engine](#) (std::mt19937 *engine_, bool delete_=false)
- void [set_seed](#) (unsigned int s)
- [Model](#) ()
- [Model](#) (uint size_)
- [Model](#) (const [Model](#)< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > &Model_)
- [Model](#)< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > & [operator=](#) (const [Model](#)< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > &Model_)
- [~Model](#) ()
- void [store_psets](#) () noexcept
- void [set_keygen](#) (std::function< std::vector< double > (const Array_Type &)> keygen_)
- [uint](#) [add_array](#) (const Array_Type &Array_, bool force_new=false)
Adds an array to the support of not already included.
- void [print_stats](#) (uint i) const
- Array_Type [sample](#) (const Array_Type &Array_, const std::vector< double > ¶ms={})
- Array_Type [sample](#) (const [uint](#) &i, const std::vector< double > ¶ms)
- const std::mt19937 * [get_engine](#) () const
- [Counters](#)< Array_Type, Data_Counter_Type > * [get_counters](#) ()
- [Rules](#)< Array_Type, Data_Rule_Type > & [get_rules](#) ()
- [Rules](#)< Array_Type, Data_Rule_Dyn_Type > & [get_rules_dyn](#) ()

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

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

- void [add_counter](#) ([Counter](#)< Array_Type, Data_Counter_Type > &counter)
- void [add_counter](#) ([Counter](#)< Array_Type, Data_Counter_Type > *counter)
- void [add_counter](#) ([Counter_fun_type](#)< Array_Type, Data_Counter_Type > count_fun_, [Counter_fun_type](#)< Array_Type, Data_Counter_Type > init_fun_=nullptr, Data_Counter_Type *data_=nullptr, bool delete_data_=false)
- void [set_counters](#) ([Counters](#)< Array_Type, Data_Counter_Type > *counters_)

Wrappers for the `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_)
- void [add_rule_dyn](#) ([Rule](#)< Array_Type, Data_Rule_Dyn_Type > &rule)
- void [add_rule_dyn](#) ([Rule](#)< Array_Type, Data_Rule_Dyn_Type > *rule)
- void [add_rule_dyn](#) ([Rule_fun_type](#)< Array_Type, Data_Rule_Dyn_Type > count_fun_, Data_Rule_Dyn_Type *data_=nullptr, bool delete_data_=false)
- void [set_rules_dyn](#) ([Rules](#)< Array_Type, Data_Rule_Dyn_Type > *rules_)

Likelihood functions.

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

Parameters

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

7.15.1 Detailed Description

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

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

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

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

<code>Array_Type</code>	Class of <code>BArray</code> object.
<code>Data_Counter_Type</code>	Any type.
<code>Data_Rule_Type</code>	Any type.

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

7.15.2 Constructor & Destructor Documentation**7.15.2.1 Model() [1/3]**

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

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

7.15.2.2 Model() [2/3]

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

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

7.15.2.3 Model() [3/3]

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

Definition at line 56 of file model-meat.hpp.

7.15.2.4 ~Model()

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

Definition at line 203 of file model-bones.hpp.

7.15.3 Member Function Documentation

7.15.3.1 add_array()

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

Adds an array to the support of not already included.

Parameters

<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 302 of file model-meat.hpp.

7.15.3.2 add_counter() [1/3]

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

Definition at line 145 of file model-meat.hpp.

7.15.3.3 add_counter() [2/3]

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

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

7.15.3.4 add_counter() [3/3]

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

Definition at line 164 of file model-meat.hpp.

7.15.3.5 add_rule() [1/3]

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

Definition at line 203 of file model-meat.hpp.

7.15.3.6 add_rule() [2/3]

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

Definition at line 212 of file model-meat.hpp.

7.15.3.7 add_rule() [3/3]

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

Definition at line 222 of file model-meat.hpp.

7.15.3.8 add_rule_dyn() [1/3]

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

Definition at line 252 of file model-meat.hpp.

7.15.3.9 add_rule_dyn() [2/3]

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

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

7.15.3.10 add_rule_dyn() [3/3]

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

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

7.15.3.11 get_counters()

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

Definition at line 667 of file model-meat.hpp.

7.15.3.12 get_norm_const()

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

Definition at line 533 of file model-meat.hpp.

7.15.3.13 get_pset()

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

Definition at line 565 of file model-meat.hpp.

7.15.3.14 get_rengine()

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

Definition at line 662 of file model-meat.hpp.

7.15.3.15 get_rules()

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

Definition at line 672 of file model-meat.hpp.

7.15.3.16 get_rules_dyn()

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

Definition at line 677 of file model-meat.hpp.

7.15.3.17 get_stats()

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

Definition at line 578 of file model-meat.hpp.

7.15.3.18 likelihood() [1/3]

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

Definition at line 419 of file model-meat.hpp.

7.15.3.19 likelihood() [2/3]

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

Definition at line 459 of file model-meat.hpp.

7.15.3.20 likelihood() [3/3]

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

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

7.15.3.21 likelihood_total()

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

Definition at line 493 of file model-meat.hpp.

7.15.3.22 nterms()

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

Definition at line 619 of file model-meat.hpp.

7.15.3.23 operator=()

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

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

7.15.3.24 print_stats()

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

Definition at line 590 of file model-meat.hpp.

7.15.3.25 sample() [1/2]

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

7.15.3.26 sample() [2/2]

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

Definition at line 626 of file model-meat.hpp.

7.15.3.27 set_counters()

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

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

7.15.3.28 set_keygen()

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

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

7.15.3.29 set_engine()

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

Definition at line 175 of file model-bones.hpp.

7.15.3.30 set_rules()

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

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

7.15.3.31 set_rules_dyn()

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

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

7.15.3.32 set_seed()

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

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

7.15.3.33 size()

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

Definition at line 609 of file model-meat.hpp.

7.15.3.34 size_unique()

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

Definition at line 614 of file model-meat.hpp.

7.15.3.35 store_psets()

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

Definition at line 129 of file model-meat.hpp.

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

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

7.16 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](#)

7.16.1 Detailed Description

Data class used to store arbitrary uint or double vectors.

Definition at line 61 of file network.hpp.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 NetCounterData() [1/2]

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

Definition at line 67 of file network.hpp.

7.16.2.2 NetCounterData() [2/2]

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

Definition at line 68 of file network.hpp.

7.16.2.3 ~NetCounterData()

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

Definition at line 73 of file network.hpp.

7.16.3 Member Data Documentation

7.16.3.1 indices

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

Definition at line 64 of file network.hpp.

7.16.3.2 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](#)

7.17 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](#)

7.17.1 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.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 NetworkData() [1/3]

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

Definition at line 30 of file network.hpp.

7.17.2.2 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.

7.17.2.3 NetworkData() [3/3]

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

Constructor using multiple attributes.

Parameters

<i>vertex_↔ 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.

7.17.2.4 ~NetworkData()

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

Definition at line 56 of file network.hpp.

7.17.3 Member Data Documentation

7.17.3.1 directed

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

Definition at line 27 of file network.hpp.

7.17.3.2 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](#)

7.18 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.

7.18.1 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`.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 `Node()` [1/5]

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

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

7.18.2.2 Node() [2/5]

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

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

7.18.2.3 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.

7.18.2.4 Node() [4/5]

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

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

7.18.2.5 Node() [5/5]

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

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

7.18.2.6 ~Node()

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

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

7.18.3 Member Function Documentation

7.18.3.1 get_parent()

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

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

7.18.3.2 is_leaf()

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

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

7.18.4 Member Data Documentation

7.18.4.1 annotations

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

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

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

7.18.4.2 array

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

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

7.18.4.3 arrays

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

Arrays given all possible states.

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

7.18.4.4 duplication

```
bool Node::duplication
```

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

7.18.4.5 id

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

Id of the node (as specified in the input)

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

7.18.4.6 narray

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

ID of the array in the model.

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

7.18.4.7 offspring

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

Offspring nodes.

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

7.18.4.8 ord

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

Order in which the node was created.

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

7.18.4.9 parent

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

Parent node.

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

7.18.4.10 probability

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

The probability of observing each state.

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

7.18.4.11 subtree_prob

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

Induced subtree probabilities.

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

7.18.4.12 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](#)

7.19 NodeData Class Reference

Data definition for the `PhyloArray` class.

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

Public Member Functions

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

Public Attributes

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

7.19.1 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`.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 NodeData() [1/2]

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

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

7.19.2.2 NodeData() [2/2]

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

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

7.19.2.3 ~NodeData()

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

Definition at line 49 of file phylo.hpp.

7.19.3 Member Data Documentation

7.19.3.1 blengths

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

Branch length.

Definition at line 29 of file phylo.hpp.

7.19.3.2 duplication

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

Definition at line 39 of file phylo.hpp.

7.19.3.3 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](#)

7.20 PhyloRuleDynData Class Reference

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

Public Member Functions

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

Public Attributes

- const std::vector< double > * [counts](#)
- [uint](#) [pos](#)
- [uint](#) [lb](#)
- [uint](#) [ub](#)
- bool [duplication](#)

7.20.1 Detailed Description

Definition at line 1189 of file phylo.hpp.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 [PhyloRuleDynData\(\)](#)

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

Definition at line 1196 of file phylo.hpp.

7.20.2.2 [~PhyloRuleDynData\(\)](#)

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

Definition at line 1205 of file phylo.hpp.

7.20.3 Member Data Documentation

7.20.3.1 counts

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

Definition at line 1191 of file phylo.hpp.

7.20.3.2 duplication

```
bool PhyloRuleDynData::duplication
```

Definition at line 1195 of file phylo.hpp.

7.20.3.3 lb

```
uint PhyloRuleDynData::lb
```

Definition at line 1193 of file phylo.hpp.

7.20.3.4 pos

```
uint PhyloRuleDynData::pos
```

Definition at line 1192 of file phylo.hpp.

7.20.3.5 ub

```
uint PhyloRuleDynData::ub
```

Definition at line 1194 of file phylo.hpp.

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

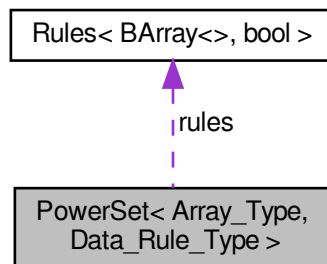
- include/barry/counters/[phylo.hpp](#)

7.21 PowerSet< Array_Type, Data_Rule_Type > Class Template Reference

Powerset of a binary array.

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

Collaboration diagram for PowerSet< Array_Type, Data_Rule_Type >:



Public Member Functions

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

Construct and destroy a PowerSet object

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

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

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

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

Getter functions

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

Public Attributes

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

7.21.1 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.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 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.

7.21.2.2 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.

7.21.2.3 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.

7.21.2.4 ~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.

7.21.3 Member Function Documentation**7.21.3.1 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.

7.21.3.2 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.

7.21.3.3 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.

7.21.3.4 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.

7.21.3.5 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.

7.21.3.6 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.

7.21.3.7 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.

7.21.3.8 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.

7.21.3.9 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.

7.21.3.10 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.

7.21.3.11 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.

7.21.3.12 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.

7.21.4 Member Data Documentation

7.21.4.1 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.

7.21.4.2 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.

7.21.4.3 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.

7.21.4.4 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.

7.21.4.5 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.

7.21.4.6 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.

7.21.4.7 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.

7.21.4.8 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

7.22 Rule< Array_Type, Data_Type > Class Template Reference

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

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

Public Member Functions

- [~Rule](#) ()
- Data_Type * [D](#) ()
Read/Write access to the data.
- bool [operator\(\)](#) (const Array_Type &a, [uint](#) i, [uint](#) j)

Construct a new Rule object

Construct a new [Rule](#) object

Parameters

fun_	A function of type <i>Rule_fun_type</i> .
dat_	Data pointer to be passed to <i>fun_</i>
delete_↔ dat_	When <i>true</i> , the Rule destructor will delete the pointer, if defined.

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

7.22.1 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 BArray .
<i>Data_Type</i>	Any type.

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

7.22.2 Constructor & Destructor Documentation

7.22.2.1 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.

7.22.2.2 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.

7.22.2.3 ~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.

7.22.3 Member Function Documentation

7.22.3.1 D()

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

Read/Write access to the data.

7.22.3.2 operator()()

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

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

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

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

7.23 Rules< Array_Type, Data_Type > Class Template Reference

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

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

Public Member Functions

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

Rule adding

Parameters

rule	
------	--

- void [add_rule](#) ([Rule](#)< Array_Type, Data_Type > &rule)
- void [add_rule](#) ([Rule](#)< Array_Type, Data_Type > *rule)
- void [add_rule](#) ([Rule_fun_type](#)< Array_Type, Data_Type > rule_, Data_Type *data_=nullptr, bool delete←_data_=false)

7.23.1 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 BArray
<i>Data_Type</i>	Any type.

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

7.23.2 Constructor & Destructor Documentation**7.23.2.1 Rules() [1/2]**

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

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

7.23.2.2 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.

7.23.2.3 ~Rules()

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

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

7.23.3 Member Function Documentation

7.23.3.1 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.

7.23.3.2 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.

7.23.3.3 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.

7.23.3.4 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.

7.23.3.5 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 <code>BArray</code> .
<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.

7.23.3.6 operator()

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

Check whether a given cell is free or locked.

Parameters

<i>a</i>	A <code>BArray</code> 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.

7.23.3.7 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.

7.23.3.8 size()

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

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

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

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

7.24 StatsCounter< Array_Type, Data_Type > Class Template Reference

Count stats for a single Array.

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

Public Member Functions

- [StatsCounter](#) (const Array_Type *Array_)
Creator of a [StatsCounter](#)
- [StatsCounter](#) ()
Can be created without setting the array.
- [~StatsCounter](#) ()
- void [reset_array](#) (const Array_Type *Array_)
Changes the reference array for the counting.
- void [add_counter](#) ([Counter](#)< Array_Type, Data_Type > *f_)
- void [add_counter](#) ([Counter](#)< Array_Type, Data_Type > f_)
- void [set_counters](#) ([Counters](#)< Array_Type, Data_Type > *counters_)
- void [count_init](#) (uint i, uint j)
*[Counter](#) functions This function recurses through the entries of *Array* and at each step of adding a new cell it uses the functions to list the statistics.*
- void [count_current](#) (uint i, uint j)
- std::vector< double > [count_all](#) ()
- [Counters](#)< Array_Type, Data_Type > * [get_counters](#) ()

7.24.1 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.

7.24.2 Constructor & Destructor Documentation

7.24.2.1 StatsCounter() [1/2]

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

Creator of a [StatsCounter](#)

Parameters

Array ↔	A const pointer to a BArray .
—	

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

7.24.2.2 StatsCounter() [2/2]

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

Can be created without setting the array.

Definition at line 51 of file statscounter-bones.hpp.

7.24.2.3 ~StatsCounter()

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

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

7.24.3 Member Function Documentation

7.24.3.1 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.

7.24.3.2 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.

7.24.3.3 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.

7.24.3.4 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.

7.24.3.5 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.

7.24.3.6 get_counters()

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

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

7.24.3.7 reset_array()

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

Changes the reference array for the counting.

Parameters

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

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

7.24.3.8 set_counters()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::set_counters (
    Counters< Array_Type, Data_Type > * counters_ ) [inline]
```

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

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

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

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

Compute the support of sufficient statistics.

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

Public Member Functions

- [Support](#) (const `Array_Type` &`Array_`)
Constructor passing a reference Array.
- [Support](#) (uint `N_`, uint `M_`)
Constructor specifying the dimensions of the array (empty).
- [Support](#) ()
- [~Support](#) ()
- void [init_support](#) (std::vector< `Array_Type` > *`array_bank`=nullptr, std::vector< std::vector< double > > *`stats_bank`=nullptr)
- void [calc](#) (std::vector< `Array_Type` > *`array_bank`=nullptr, std::vector< std::vector< double > > *`stats_↵`, bank=nullptr, unsigned int `max_num_elements_`=0u)
Computes the entire support.
- [Counts_type](#) [get_counts](#) () const
- const [MapVec_type](#) * [get_counts_ptr](#) () const
- const std::vector< double > & [get_current_stats](#) () const
List current statistics.
- void [print](#) () const
- const [FreqTable](#) & [get_data](#) () const
- [Counters](#)< `Array_Type`, `Data_Counter_Type` > * [get_counters](#) ()

Vector of counter functions.

- [Rules](#)< Array_Type, Data_Rule_Type > * [get_rules](#) ()

Vector of static rules (cells to iterate).

- [Rules](#)< Array_Type, Data_Rule_Dyn_Type > * [get_rules_dyn](#) ()

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

Resets the support calculator

If needed, the counters of a support object can be reused.

Parameters

Array↔ —	New array over which the support will be computed.
-------------	--

- void [reset_array](#) ()
- void [reset_array](#) (const Array_Type &Array_)

Manage counters

Parameters

f_ —	A counter to be added.
counters↔ —	A vector of counters to be added.

- void [add_counter](#) (Counter< Array_Type, Data_Counter_Type > *f_)
- void [add_counter](#) (Counter< Array_Type, Data_Counter_Type > f_)
- void [set_counters](#) (Counters< Array_Type, Data_Counter_Type > *counters_)

Manage rules

Parameters

f_ —	A rule to be added.
counters↔ —	A vector of rules to be added.

- void [add_rule](#) (Rule< Array_Type, Data_Rule_Type > *f_)
- void [add_rule](#) (Rule< Array_Type, Data_Rule_Type > f_)
- void [set_rules](#) (Rules< Array_Type, Data_Rule_Type > *rules_)
- void [add_rule_dyn](#) (Rule< Array_Type, Data_Rule_Dyn_Type > *f_)
- void [add_rule_dyn](#) (Rule< Array_Type, Data_Rule_Dyn_Type > f_)
- void [set_rules_dyn](#) (Rules< Array_Type, Data_Rule_Dyn_Type > *rules_)

Public Attributes

- [uint](#) N
- [uint](#) M
- bool [delete_counters](#) = true
- bool [delete_rules](#) = true
- bool [delete_rules_dyn](#) = true
- [uint](#) max_num_elements = BARRY_MAX_NUM_ELEMENTS
- std::vector< double > [current_stats](#)
- std::vector< std::pair< [uint](#), [uint](#) > > [coordinates_free](#)
- std::vector< std::pair< [uint](#), [uint](#) > > [coordinates_locked](#)
- std::vector< std::vector< double > > [change_stats](#)

7.25.1 Detailed Description

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

Compute the support of sufficient statistics.

Given an array and a set of counters, this object iterates throughout the support set of the Array while at the same time computing the support of the sufficient statistics.

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

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

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

7.25.2 Constructor & Destructor Documentation

7.25.2.1 Support() [1/3]

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

Constructor passing a reference Array.

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

7.25.2.2 Support() [2/3]

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

Constructor specifying the dimensions of the array (empty).

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

7.25.2.3 Support() [3/3]

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

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

7.25.2.4 ~Support()

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

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

7.25.3 Member Function Documentation

7.25.3.1 add_counter() [1/2]

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

Definition at line 204 of file support-meat.hpp.

7.25.3.2 add_counter() [2/2]

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

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

7.25.3.3 add_rule() [1/2]

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

Definition at line 241 of file support-meat.hpp.

7.25.3.4 add_rule() [2/2]

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

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

7.25.3.5 add_rule_dyn() [1/2]

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

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

7.25.3.6 add_rule_dyn() [2/2]

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

Definition at line 286 of file support-meat.hpp.

7.25.3.7 calc()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::calc (
    std::vector< Array_Type > * array_bank = nullptr,
    std::vector< std::vector< double > > * stats_bank = nullptr,
    unsigned int max_num_elements_ = 0u ) [inline]
```

Computes the entire support.

Not to be used by the user. Sets the starting point in the array (column-major).

Parameters

<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 178 of file support-meat.hpp.

7.25.3.8 get_counters()

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

Vector of couter functions.

Definition at line 349 of file support-meat.hpp.

7.25.3.9 get_counts()

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

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

7.25.3.10 get_counts_ptr()

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

Definition at line 320 of file support-meat.hpp.

7.25.3.11 get_current_stats()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
const std::vector< double > & Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_↵
Rule_Dyn_Type >::get_current_stats [inline]
```

List current statistics.

Definition at line 327 of file support-meat.hpp.

7.25.3.12 get_data()

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

Definition at line 344 of file support-meat.hpp.

7.25.3.13 get_rules()

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

Vector of static rules (cells to iterate).

Definition at line 354 of file support-meat.hpp.

7.25.3.14 get_rules_dyn()

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

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

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

7.25.3.15 init_support()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::init_↵
support (
    std::vector< Array_Type > * array_bank = nullptr,
    std::vector< std::vector< double > > * stats_bank = nullptr ) [inline]
```

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

7.25.3.16 print()

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

Definition at line 332 of file support-meat.hpp.

7.25.3.17 reset_array() [1/2]

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

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

7.25.3.18 reset_array() [2/2]

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

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

7.25.3.19 set_counters()

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

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

7.25.3.20 set_rules()

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

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

7.25.3.21 set_rules_dyn()

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

Definition at line 296 of file support-meat.hpp.

7.25.4 Member Data Documentation

7.25.4.1 change_stats

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

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

7.25.4.2 coordinates_free

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

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

7.25.4.3 coordinates_locked

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

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

7.25.4.4 current_stats

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

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

7.25.4.5 delete_counters

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

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

7.25.4.6 delete_rules

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

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

7.25.4.7 delete_rules_dyn

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

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

7.25.4.8 M

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

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

7.25.4.9 max_num_elements

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::max_num_elements = BARRY_MAX_NUM_ELEMENTS
```

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

7.25.4.10 N

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::N
```

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

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

- [include/barry/support-bones.hpp](#)
- [include/barry/support-meat.hpp](#)

7.26 vecHasher< T > Struct Template Reference

```
#include <typedefs.hpp>
```

Public Member Functions

- `std::size_t operator() (std::vector< T > const &dat) const` noexcept

7.26.1 Detailed Description

```
template<typename T>
struct vecHasher< T >
```

Definition at line 86 of file typedefs.hpp.

7.26.2 Member Function Documentation

7.26.2.1 operator()()

```
template<typename T >
std::size_t vecHasher< T >::operator() (
    std::vector< T > const & dat ) const [inline], [noexcept]
```

Definition at line 87 of file typedefs.hpp.

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

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

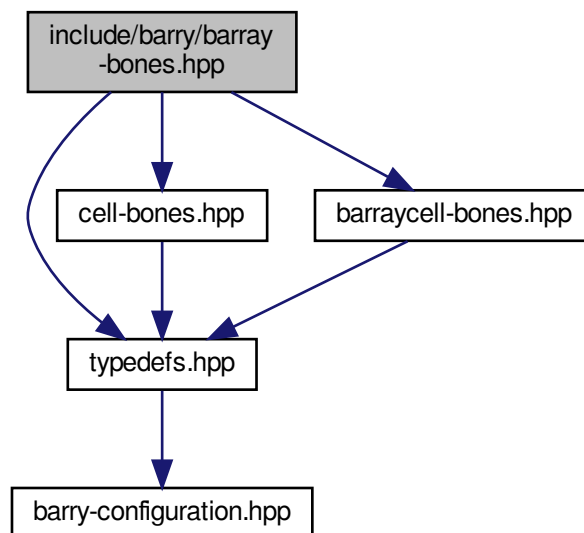
Chapter 8

File Documentation

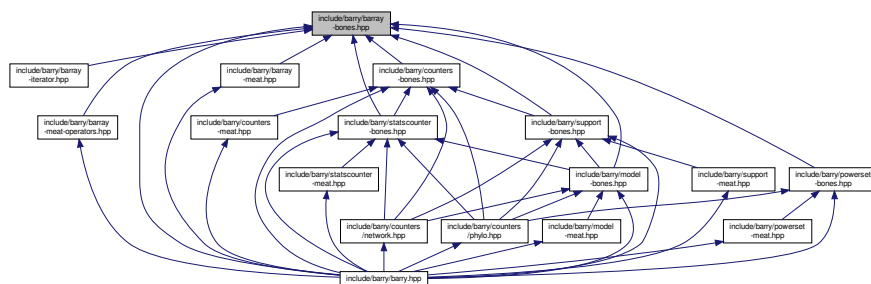
8.1 include/barry/barray-bones.hpp File Reference

```
#include "typedefs.hpp"  
#include "cell-bones.hpp"  
#include "barraycell-bones.hpp"
```

Include dependency graph for barray-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class `BArray< Cell_Type, Data_Type >`
Baseline class for binary arrays.

Macros

- `#define BARRAY_BONES_HPP 1`

8.1.1 Macro Definition Documentation

8.1.1.1 BARRAY_BONES_HPP

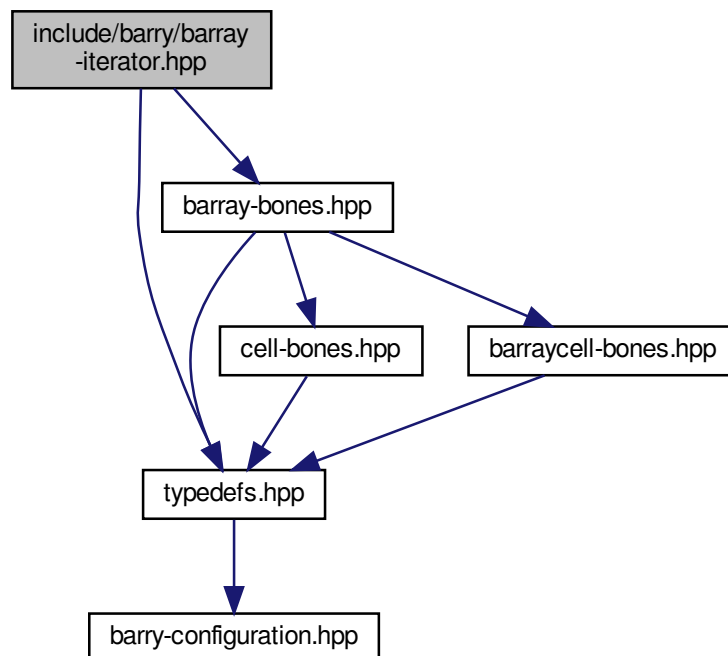
```
#define BARRAY_BONES_HPP 1
```

Definition at line 8 of file `barray-bones.hpp`.

8.2 include/barry/barray-iterator.hpp File Reference

```
#include "typedefs.hpp"
#include "barray-bones.hpp"
```

Include dependency graph for barray-iterator.hpp:



Classes

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

Macros

- `#define` [BARRAY_ITERATOR_HPP](#) 1

8.2.1 Macro Definition Documentation

8.2.1.1 BARRAY_ITERATOR_HPP

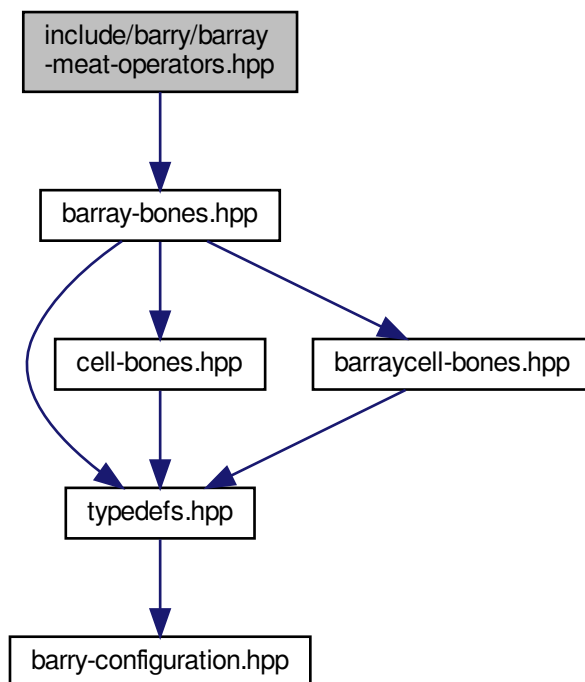
```
#define BARRAY_ITERATOR_HPP 1
```

Definition at line 7 of file `barray-iterator.hpp`.

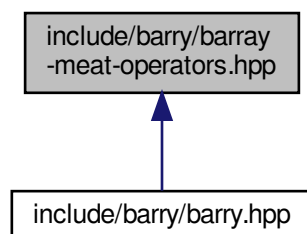
8.3 include/barry/barray-meat-operators.hpp File Reference

```
#include "barray-bones.hpp"
```

Include dependency graph for barray-meat-operators.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- `#define ROW(a) this->el_ij[a]`
- `#define COL(a) this->el_ji[a]`

Functions

- `template<typename Cell_Type , typename Data_Type >`
`void checkdim_ (const BArray< Cell_Type, Data_Type > &lhs, const BArray< Cell_Type, Data_Type > &rhs)`

8.3.1 Macro Definition Documentation

8.3.1.1 COL

```
#define COL(  
    a ) this->el_ji[a]
```

Definition at line 8 of file barray-meat-operators.hpp.

8.3.1.2 ROW

```
#define ROW(  
    a ) this->el_ij[a]
```

Definition at line 7 of file barray-meat-operators.hpp.

8.3.2 Function Documentation

8.3.2.1 checkdim_()

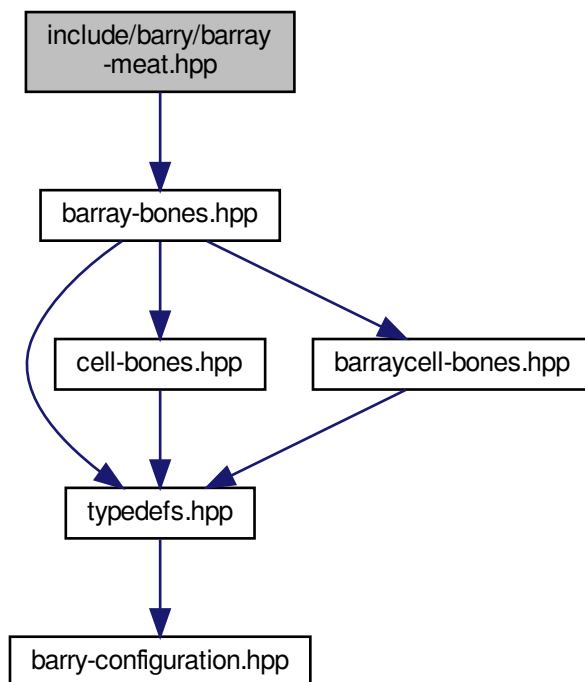
```
template<typename Cell_Type , typename Data_Type >  
void checkdim_ (  
    const BArray< Cell_Type, Data_Type > & lhs,  
    const BArray< Cell_Type, Data_Type > & rhs ) [inline]
```

Definition at line 11 of file barray-meat-operators.hpp.

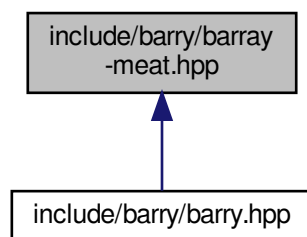
8.4 include/barry/barray-meat.hpp File Reference

```
#include "barray-bones.hpp"
```

Include dependency graph for barray-meat.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- `#define ROW(a) this->el_ij[a]`
- `#define COL(a) this->el_ji[a]`

8.4.1 Macro Definition Documentation

8.4.1.1 COL

```
#define COL(  
    a ) this->el_ji[a]
```

Definition at line 8 of file barray-meat.hpp.

8.4.1.2 ROW

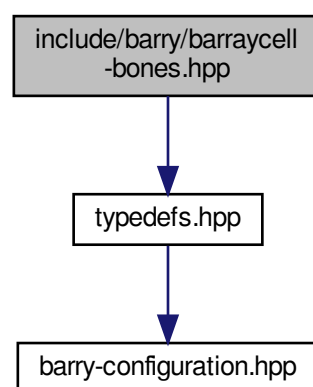
```
#define ROW(  
    a ) this->el_ij[a]
```

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

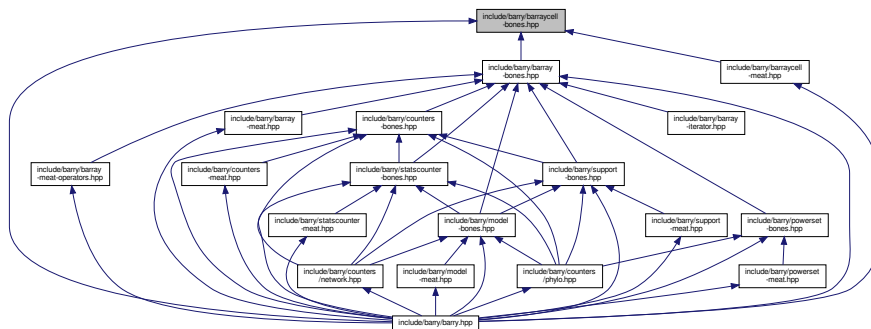
8.5 include/barry/barraycell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barraycell-bones.hpp:



This graph shows which files directly or indirectly include this file:



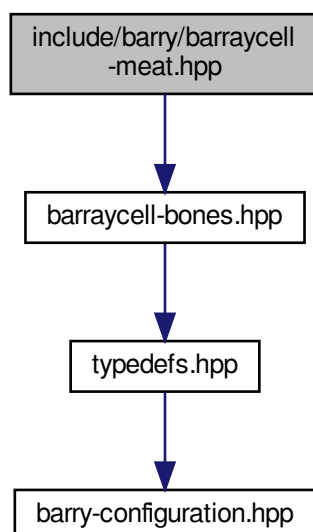
Classes

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

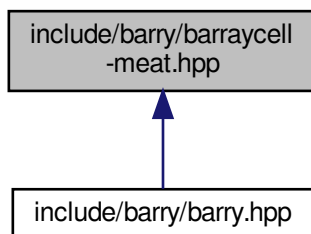
8.6 include/barry/barraycell-meat.hpp File Reference

```
#include "barraycell-bones.hpp"
```

Include dependency graph for barraycell-meat.hpp:



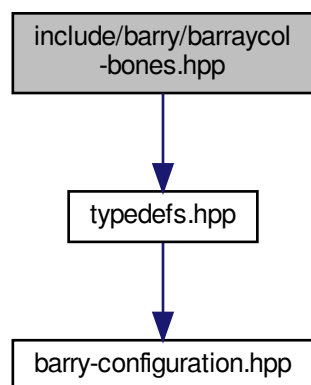
This graph shows which files directly or indirectly include this file:



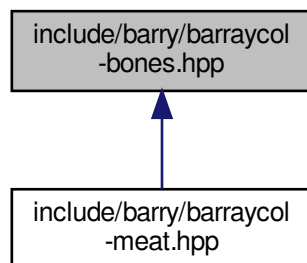
8.7 include/barry/barraycol-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barraycol-bones.hpp:



This graph shows which files directly or indirectly include this file:



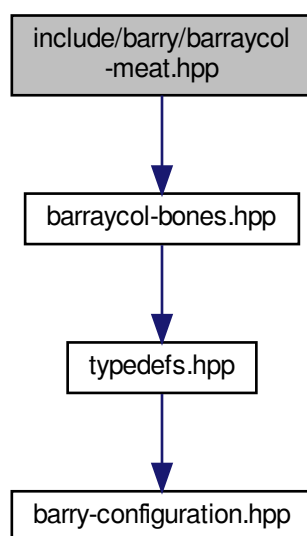
Classes

- class [BArrayCol< Cell_Type, Data_Type >](#)
- class [BArrayCol_const< Cell_Type, Data_Type >](#)

8.8 include/barry/barraycol-meat.hpp File Reference

```
#include "barraycol-bones.hpp"
```

Include dependency graph for `barraycol-meat.hpp`:



Macros

- `#define BARRY_BARRAYCOL_MEAT_HPP 1`

8.8.1 Macro Definition Documentation

8.8.1.1 BARRY_BARRAYCOL_MEAT_HPP

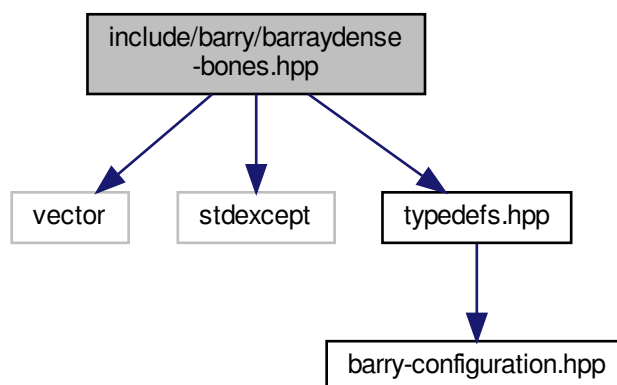
```
#define BARRY_BARRAYCOL_MEAT_HPP 1
```

Definition at line 4 of file barraycol-meat.hpp.

8.9 include/barry/barraydense-bones.hpp File Reference

```
#include <vector>
#include <stdexcept>
#include "typedefs.hpp"
```

Include dependency graph for barraydense-bones.hpp:



Classes

- class `BArrayDense< Cell_Type, Data_Type >`
Dense bi-dimensional array.

8.10.1.2 BARRY_ISFINITE

```
#define BARRY_ISFINITE(  
    a )
```

Definition at line 39 of file barry-configuration.hpp.

8.10.1.3 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.

8.10.1.4 BARRY_SAFE_EXP

```
#define BARRY_SAFE_EXP -100.0
```

Definition at line 32 of file barry-configuration.hpp.

8.10.2 Typedef Documentation

8.10.2.1 Map

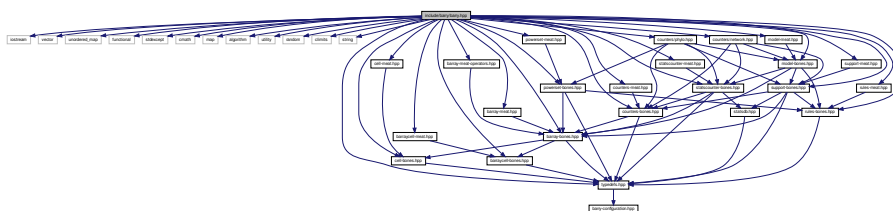
```
template<typename Ta , typename Tb >  
using Map = std::map<Ta,Tb>
```

Definition at line 26 of file barry-configuration.hpp.

8.11 include/barry/barry.hpp File Reference

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <functional>
#include <stdexcept>
#include <cmath>
#include <map>
#include <algorithm>
#include <utility>
#include <random>
#include <climits>
#include <string>
#include "typedefs.hpp"
#include "cell-bones.hpp"
#include "cell-meat.hpp"
#include "barray-bones.hpp"
#include "barraycell-bones.hpp"
#include "barray-meat.hpp"
#include "barraycell-meat.hpp"
#include "barray-meat-operators.hpp"
#include "counters-bones.hpp"
#include "counters-meat.hpp"
#include "statscounter-bones.hpp"
#include "statscounter-meat.hpp"
#include "support-bones.hpp"
#include "support-meat.hpp"
#include "powerset-bones.hpp"
#include "powerset-meat.hpp"
#include "model-bones.hpp"
#include "model-meat.hpp"
#include "rules-bones.hpp"
#include "rules-meat.hpp"
#include "counters/network.hpp"
#include "counters/phylo.hpp"
```

Include dependency graph for barry.hpp:



Namespaces

- [barry](#)
barry: Your go-to motif accountant
- [barry::counters](#)
Tree class and Treeliterator class.
- [barry::counters::network](#)
- [barry::counters::phylo](#)

Macros

- #define [BARRY_HPP](#)
- #define [COUNTER_FUNCTION](#)(a)
- #define [COUNTER_LAMBDA](#)(a)
- #define [RULE_FUNCTION](#)(a)
- #define [RULE_LAMBDA](#)(a)

8.11.1 Macro Definition Documentation

8.11.1.1 BARRY_HPP

```
#define BARRY_HPP
```

Definition at line 19 of file barry.hpp.

8.11.1.2 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 70 of file barry.hpp.

8.11.1.3 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 73 of file barry.hpp.

8.11.1.4 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 77 of file barry.hpp.

8.11.1.5 RULE_LAMBDA

```
#define RULE_LAMBDA(  
    a )
```

Value:

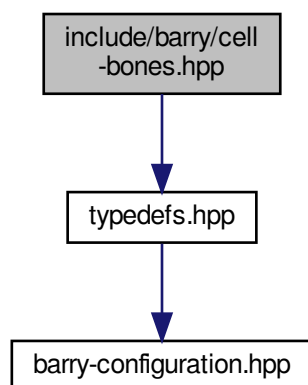
```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \  
Rule_fun_type<Array_Type, Data_Type> a = \  
[] (const Array_Type & Array, uint i, uint j, Data_Type * data)
```

Definition at line 80 of file barry.hpp.

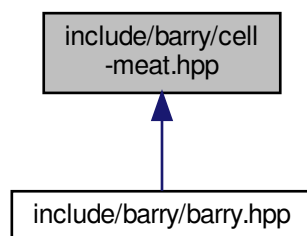
8.12 include/barry/cell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for cell-bones.hpp:



This graph shows which files directly or indirectly include this file:



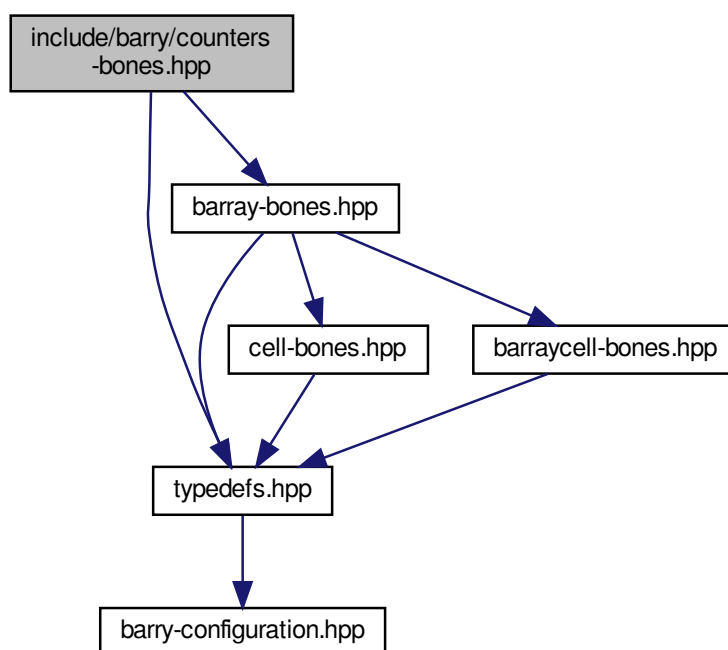
8.14 include/barry/col-bones.hpp File Reference

8.15 include/barry/counters-bones.hpp File Reference

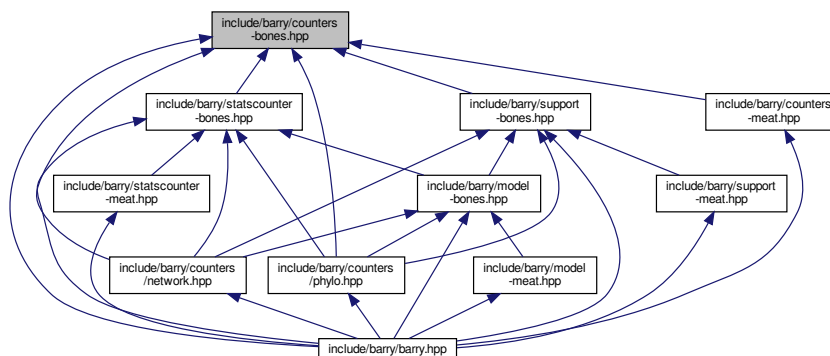
```
#include "typedefs.hpp"
```

```
#include "barray-bones.hpp"
```

Include dependency graph for counters-bones.hpp:



This graph shows which files directly or indirectly include this file:



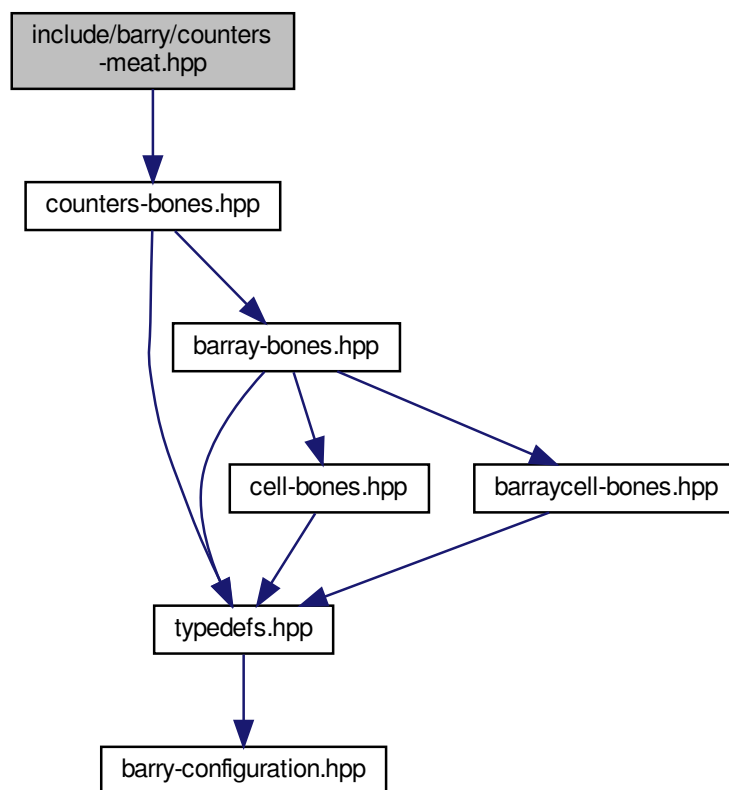
Classes

- class `Counter< Array_Type, Data_Type >`
A counter function based on change statistics.
- class `Counters< Array_Type, Data_Type >`
Vector of counters.

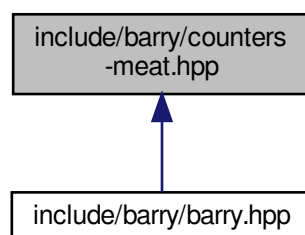
8.16 include/barry/counters-meat.hpp File Reference

```
#include "counters-bones.hpp"
```

Include dependency graph for counters-meat.hpp:



This graph shows which files directly or indirectly include this file:



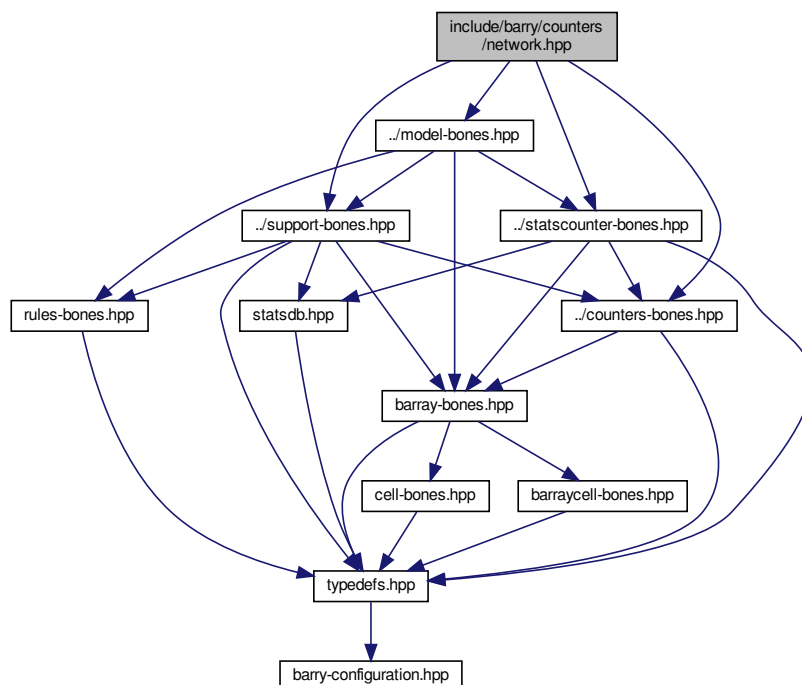
8.17 include/barry/counters/network.hpp File Reference

```

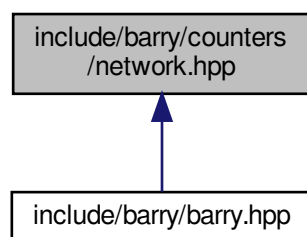
#include "../counters-bones.hpp"
#include "../support-bones.hpp"

```

```
#include "../statscounter-bones.hpp"
#include "../model-bones.hpp"
Include dependency graph for network.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [NetworkData](#)
Data class for Networks.
- class [NetCounterData](#)
Data class used to store arbitrary uint or double vectors.

Macros

- #define `NET_C_DATA_IDX(i)` (`data->indices[i]`)
- #define `NET_C_DATA_NUM(i)` (`data->numbers[i]`)

Macros for defining counters

- #define `NETWORK_COUNTER(a)`
- #define `NETWORK_COUNTER_LAMBDA(a)`

Macros for defining rules

- #define `NETWORK_RULE(a)`
- #define `NETWORK_RULE_LAMBDA(a)`

Typedefs

Convenient typedefs for network objects.

- typedef `BArray< double, NetworkData > Network`
- typedef `Counter< Network, NetCounterData > NetCounter`
- typedef `Counters< Network, NetCounterData > NetCounters`
- typedef `Support< Network, NetCounterData > NetSupport`
- typedef `StatsCounter< Network, NetCounterData > NetStatsCounter`
- typedef `Model< Network, NetCounterData > NetModel`
- typedef `Rule< Network, bool > NetRule`
- typedef `Rules< Network, bool > NetRules`

Functions

- void `counter_edges (NetCounters *counters)`
Number of edges.
- void `counter_isolates (NetCounters *counters)`
Number of isolated vertices.
- void `counter_mutual (NetCounters *counters)`
Number of mutual ties.
- void `counter_istar2 (NetCounters *counters)`
- void `counter_ostar2 (NetCounters *counters)`
- void `counter_ttriads (NetCounters *counters)`
- void `counter_ctriads (NetCounters *counters)`
- void `counter_density (NetCounters *counters)`
- void `counter_idegree15 (NetCounters *counters)`
- void `counter_odegree15 (NetCounters *counters)`
- void `counter_absdiff (NetCounters *counters, uint attr_id, double alpha=1.0)`
Sum of absolute attribute difference between ego and alter.
- void `counter_diff (NetCounters *counters, uint attr_id, double alpha=1.0, double tail_head=true)`
Sum of attribute difference between ego and alter to pow(alpha)
- `NETWORK_COUNTER` (`init_single_attr`)
- void `counter_nodeicov (NetCounters *counters, uint attr_id)`
- void `counter_nodeocov (NetCounters *counters, uint attr_id)`

- void `counter_nodecov` (`NetCounters` *counters, `uint` attr_id)
- void `counter_nodematch` (`NetCounters` *counters, `uint` attr_id)
- void `counter_iddegree` (`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 <code>NetRules</code> object (<code>Rules<Network, bool></code>).
-------	--

- void `rules_zerodiag` (`NetRules` *rules)
Number of edges.

8.17.1 Macro Definition Documentation

8.17.1.1 NET_C_DATA_IDX

```
#define NET_C_DATA_IDX(  
    i ) (data->indices[i])
```

Definition at line 79 of file network.hpp.

8.17.1.2 NET_C_DATA_NUM

```
#define NET_C_DATA_NUM(  
    i ) (data->numbers[i])
```

Definition at line 80 of file network.hpp.

8.17.1.3 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.

8.17.1.4 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.

8.17.1.5 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.

8.17.1.6 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.

8.17.2 Typedef Documentation

8.17.2.1 NetCounter

```
typedef Counter<Network, NetCounterData > NetCounter
```

Definition at line 88 of file network.hpp.

8.17.2.2 NetCounters

```
typedef Counters< Network, NetCounterData> NetCounters
```

Definition at line 89 of file network.hpp.

8.17.2.3 NetModel

```
typedef Model<Network, NetCounterData> NetModel
```

Definition at line 92 of file network.hpp.

8.17.2.4 NetRule

```
typedef Rule<Network, bool> NetRule
```

Definition at line 93 of file network.hpp.

8.17.2.5 NetRules

```
typedef Rules<Network, bool> NetRules
```

Definition at line 94 of file network.hpp.

8.17.2.6 NetStatsCounter

```
typedef StatsCounter<Network, NetCounterData> NetStatsCounter
```

Definition at line 91 of file network.hpp.

8.17.2.7 NetSupport

```
typedef Support<Network, NetCounterData > NetSupport
```

Definition at line 90 of file network.hpp.

8.17.2.8 Network

```
typedef BArray<double, NetworkData> Network
```

Definition at line 87 of file network.hpp.

8.17.3 Function Documentation

8.17.3.1 rules_zerodiag()

```
void rules_zerodiag (
    NetRules * rules ) [inline]
```

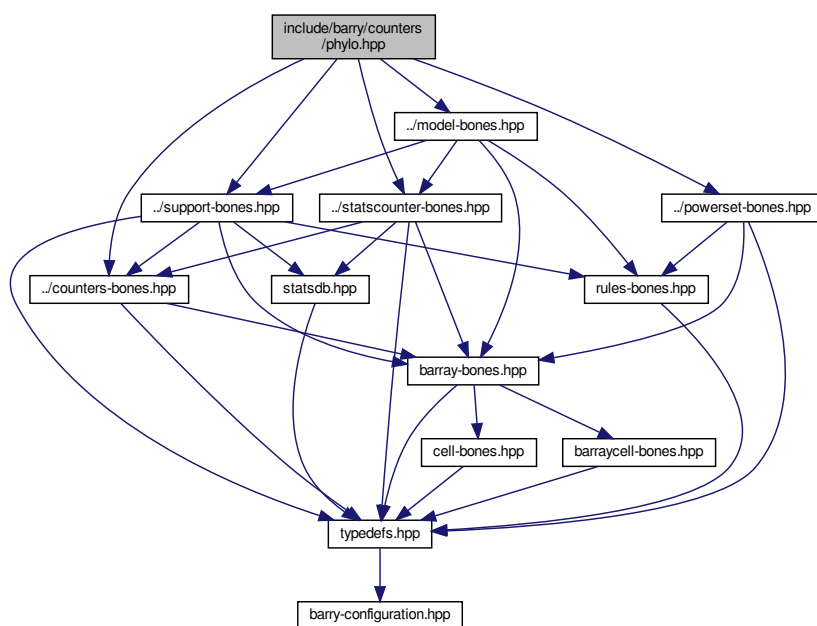
Number of edges.

Definition at line 742 of file network.hpp.

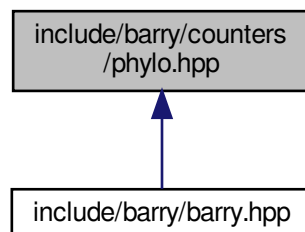
8.18 include/barry/counters/phylo.hpp File Reference

```
#include "../counters-bones.hpp"
#include "../support-bones.hpp"
#include "../statscounter-bones.hpp"
#include "../model-bones.hpp"
#include "../powerset-bones.hpp"
```

Include dependency graph for phylo.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [NodeData](#)
Data definition for the `PhyloArray` class.
- class [PhyloRuleDynData](#)

Macros

- `#define` [PHYLO_COUNTER_LAMBDA\(a\)](#)
Extension of a simple counter.
- `#define` [PHYLO_RULE_DYN_LAMBDA\(a\)](#)
- `#define` [PHYLO_CHECK_MISSING\(\)](#)

Typedefs

- `typedef` `std::vector< uint >` [PhyloCounterData](#)
- `typedef` `std::vector< std::pair< uint, uint > >` [PhyloRuleData](#)

Convenient typedefs for Node objects.

- `typedef` [BArray< \[uint\]\(#\), \[NodeData\]\(#\) >](#) [PhyloArray](#)
- `typedef` [Counter< \[PhyloArray\]\(#\), \[PhyloCounterData\]\(#\) >](#) [PhyloCounter](#)
- `typedef` [Counters< \[PhyloArray\]\(#\), \[PhyloCounterData\]\(#\) >](#) [PhyloCounters](#)
- `typedef` [Rule< \[PhyloArray\]\(#\), \[PhyloRuleData\]\(#\) >](#) [PhyloRule](#)
- `typedef` [Rules< \[PhyloArray\]\(#\), \[PhyloRuleData\]\(#\) >](#) [PhyloRules](#)
- `typedef` [Rule< \[PhyloArray\]\(#\), \[PhyloRuleDynData\]\(#\) >](#) [PhyloRuleDyn](#)
- `typedef` [Rules< \[PhyloArray\]\(#\), \[PhyloRuleDynData\]\(#\) >](#) [PhyloRulesDyn](#)
- `typedef` [Support< \[PhyloArray\]\(#\), \[PhyloCounterData\]\(#\), \[PhyloRuleData\]\(#\), \[PhyloRuleDynData\]\(#\) >](#) [PhyloSupport](#)
- `typedef` [StatsCounter< \[PhyloArray\]\(#\), \[PhyloCounterData\]\(#\) >](#) [PhyloStatsCounter](#)
- `typedef` [Model< \[PhyloArray\]\(#\), \[PhyloCounterData\]\(#\), \[PhyloRuleData\]\(#\), \[PhyloRuleDynData\]\(#\) >](#) [PhyloModel](#)
- `typedef` [PowerSet< \[PhyloArray\]\(#\), \[PhyloRuleData\]\(#\) >](#) [PhyloPowerSet](#)

Functions

- `std::string get_last_name` (bool d)
- void `counter_overall_gains` (PhyloCounters *counters, bool duplication=true)
Overall functional gains.
- void `counter_gains` (PhyloCounters *counters, std::vector< uint > nfun, bool duplication=true)
Functional gains for a specific function (nfun).
- void `counter_gains_k_offspring` (PhyloCounters *counters, std::vector< uint > nfun, uint k=1u, bool duplication=true)
k genes gain function nfun
- void `counter_genes_changing` (PhyloCounters *counters, bool duplication=true)
Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)
- void `counter_overall_loss` (PhyloCounters *counters, bool duplication=true)
Overall functional loss.
- void `counter_maxfuns` (PhyloCounters *counters, uint lb, uint ub, bool duplication=true)
Cap the number of functions per gene.
- void `counter_loss` (PhyloCounters *counters, std::vector< uint > nfun, bool duplication=true)
Total count of losses for an specific function.
- void `counter_overall_changes` (PhyloCounters *counters, bool duplication=true)
Total number of changes. Use this statistic to account for "preservation".
- void `counter_subfun` (PhyloCounters *counters, uint nfunA, uint nfunB, bool duplication=true)
Total count of Sub-functionalization events.
- void `counter_cogain` (PhyloCounters *counters, uint nfunA, uint nfunB, bool duplication=true)
Co-evolution (joint gain or loss)
- void `counter_longest` (PhyloCounters *counters)
Longest branch mutates (either by gain or by loss)
- void `counter_neofun` (PhyloCounters *counters, uint nfunA, uint nfunB, bool duplication=true)
Total number of neofunctionalization events.
- void `counter_neofun_a2b` (PhyloCounters *counters, uint nfunA, uint nfunB, bool duplication=true)
Total number of neofunctionalization events.
- void `counter_co_opt` (PhyloCounters *counters, uint nfunA, uint nfunB, bool duplication=true)
Function co-opting.
- void `rule_dyn_limit_changes` (PhyloSupport *support, uint pos, uint lb, uint ub, bool duplication=true)
Overall functional gains.

8.18.1 Macro Definition Documentation

8.18.1.1 PHYLO_CHECK_MISSING

```
#define PHYLO_CHECK_MISSING( )
```

Value:

```
if (Array.D() == nullptr) \
throw std::logic_error("The array data is nullptr."); \
if (data == nullptr) \
throw std::logic_error("The counter/rule data is nullptr.")
```

Definition at line 94 of file phylo.hpp.

8.18.1.2 PHYLO_COUNTER_LAMBDA

```
#define PHYLO_COUNTER_LAMBDA(  
    a )
```

Value:

```
Counter_fun_type<PhyloArray, PhyloCounterData> a = \  
[] (const PhyloArray & Array, uint i, uint j, PhyloCounterData * data)
```

Extension of a simple counter.

It allows specifying extra arguments, in particular, the corresponding sets of rows to which this statistic may be relevant. This could be important in the case of, for example, counting correlation type statistics between function 1 and 2, and between function 1 and 3.

Definition at line 88 of file phylo.hpp.

8.18.1.3 PHYLO_RULE_DYN_LAMBDA

```
#define PHYLO_RULE_DYN_LAMBDA(  
    a )
```

Value:

```
Rule_fun_type<PhyloArray, PhyloRuleDynData> a = \  
[] (const PhyloArray & Array, uint i, uint j, PhyloRuleDynData * data)
```

Definition at line 91 of file phylo.hpp.

8.18.2 Typedef Documentation

8.18.2.1 PhyloArray

```
typedef BArray<uint, NodeData> PhyloArray
```

Definition at line 61 of file phylo.hpp.

8.18.2.2 PhyloCounter

```
typedef Counter<PhyloArray, PhyloCounterData > PhyloCounter
```

Definition at line 62 of file phylo.hpp.

8.18.2.3 PhyloCounterData

```
typedef std::vector< uint > PhyloCounterData
```

Definition at line 53 of file phylo.hpp.

8.18.2.4 PhyloCounters

```
typedef Counters< PhyloArray, PhyloCounterData> PhyloCounters
```

Definition at line 63 of file phylo.hpp.

8.18.2.5 PhyloModel

```
typedef Model<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel
```

Definition at line 73 of file phylo.hpp.

8.18.2.6 PhyloPowerSet

```
typedef PowerSet<PhyloArray, PhyloRuleData> PhyloPowerSet
```

Definition at line 74 of file phylo.hpp.

8.18.2.7 PhyloRule

```
typedef Rule<PhyloArray,PhyloRuleData> PhyloRule
```

Definition at line 65 of file phylo.hpp.

8.18.2.8 PhyloRuleData

```
typedef std::vector< std::pair< uint, uint > > PhyloRuleData
```

Definition at line 54 of file phylo.hpp.

8.18.2.9 PhyloRuleDyn

```
typedef Rule<PhyloArray,PhyloRuleDynData> PhyloRuleDyn
```

Definition at line 68 of file phylo.hpp.

8.18.2.10 PhyloRules

```
typedef Rules<PhyloArray,PhyloRuleData> PhyloRules
```

Definition at line 66 of file phylo.hpp.

8.18.2.11 PhyloRulesDyn

```
typedef Rules<PhyloArray,PhyloRuleDynData> PhyloRulesDyn
```

Definition at line 69 of file phylo.hpp.

8.18.2.12 PhyloStatsCounter

```
typedef StatsCounter<PhyloArray, PhyloCounterData> PhyloStatsCounter
```

Definition at line 72 of file phylo.hpp.

8.18.2.13 PhyloSupport

```
typedef Support<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport
```

Definition at line 71 of file phylo.hpp.

8.18.3 Function Documentation

8.18.3.1 get_last_name()

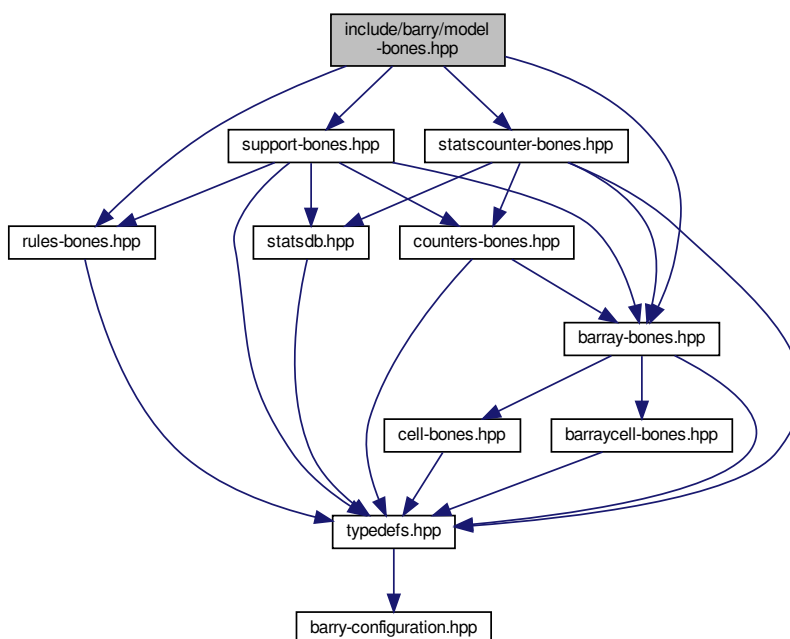
```
std::string get_last_name (  
    bool d ) [inline]
```

Definition at line 99 of file phylo.hpp.

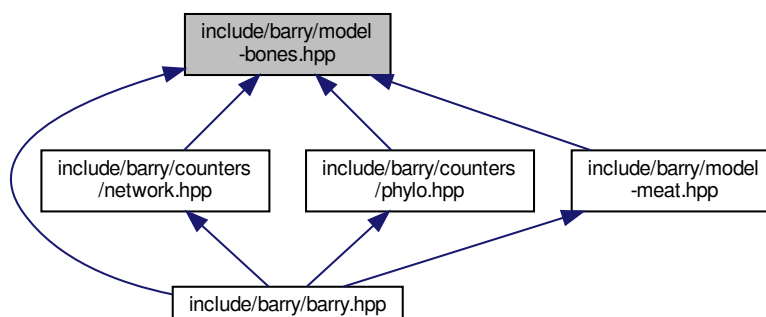
8.19 include/barry/model-bones.hpp File Reference

```
#include "barray-bones.hpp"
#include "support-bones.hpp"
#include "statscounter-bones.hpp"
#include "rules-bones.hpp"
```

Include dependency graph for model-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >](#)

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

Functions

- double [update_normalizing_constant](#) (const std::vector< double > ¶ms, const [Counts_type](#) &support)
- double [likelihood_](#) (const std::vector< double > &target_stats, const std::vector< double > ¶ms, 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)

8.19.1 Function Documentation

8.19.1.1 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.

8.19.1.2 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.

8.19.1.3 update_normalizing_constant()

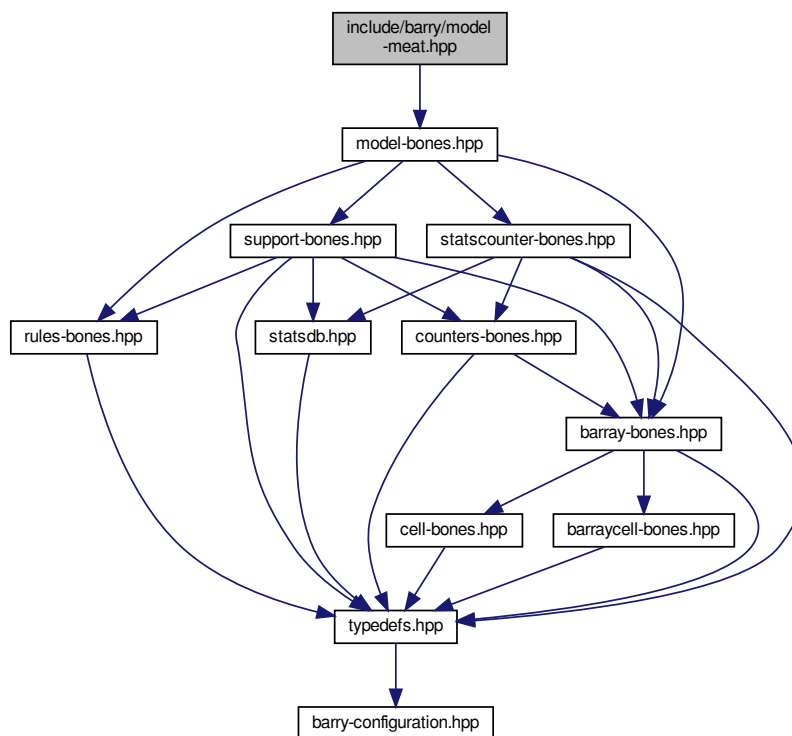
```
double update_normalizing_constant (  
    const std::vector< double > & params,  
    const Counts\_type & support ) [inline]
```

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

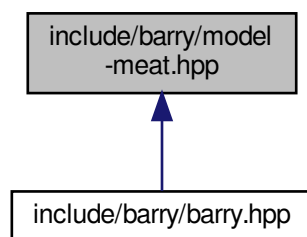
8.20 include/barry/model-meat.hpp File Reference

```
#include "model-bones.hpp"
```

Include dependency graph for model-meat.hpp:



This graph shows which files directly or indirectly include this file:

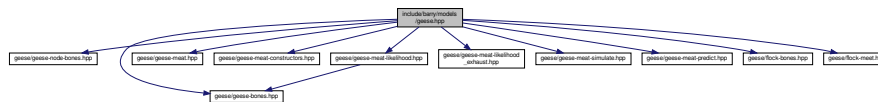


8.21 include/barry/models/geese.hpp File Reference

```
#include "geese/geese-node-bones.hpp"
```

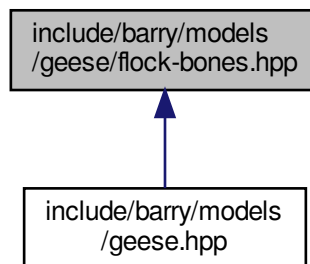
```
#include "geese/geese-bones.hpp"
```

```
#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:
```



8.22 include/barry/models/geese/flock-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:

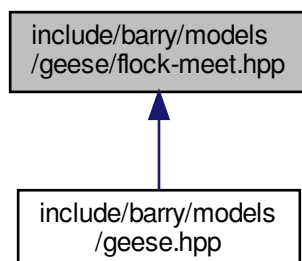


Classes

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

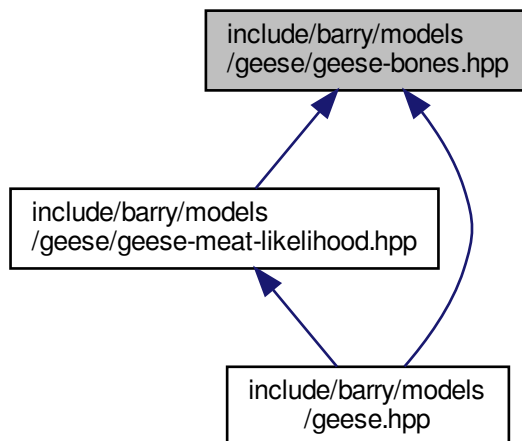
8.23 include/barry/models/geese/flock-meet.hpp File Reference

This graph shows which files directly or indirectly include this file:



8.24 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)`

8.24.1 Macro Definition Documentation

8.24.1.1 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.

8.24.2 Function Documentation

8.24.2.1 keygen_full()

```
std::vector< double > keygen_full (
    const phylocounters::PhyloArray & array ) [inline]
```

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

8.24.2.2 RULE_FUNCTION()

```
RULE_FUNCTION (
    rule_empty_free )
```

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

8.24.2.3 `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.

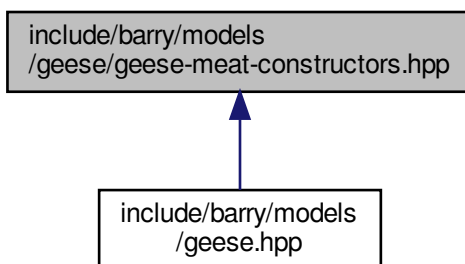
8.24.2.4 `vector_caster()`

```
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.

8.25 `include/barry/models/geese/geese-meat-constructors.hpp` File Reference

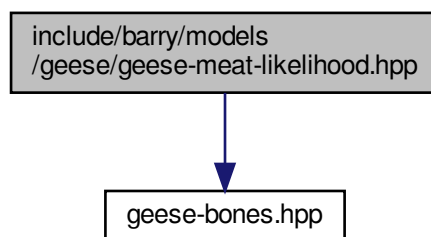
This graph shows which files directly or indirectly include this file:



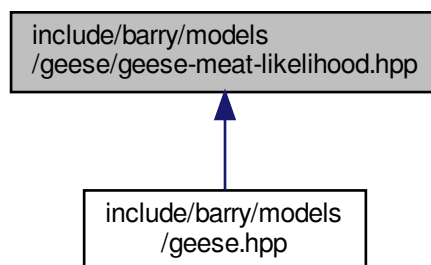
8.26 include/barry/models/geese/geese-meat-likelihood.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat-likelihood.hpp:

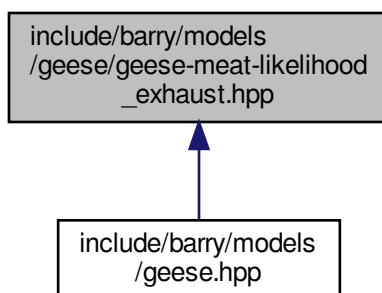


This graph shows which files directly or indirectly include this file:



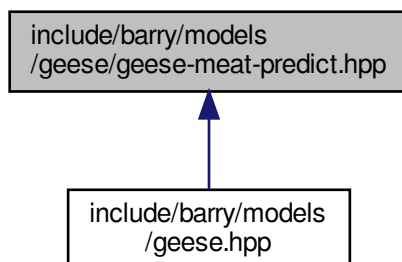
8.27 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference

This graph shows which files directly or indirectly include this file:



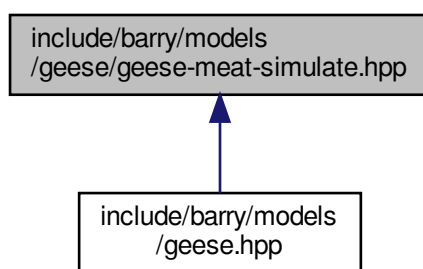
8.28 include/barry/models/geese/geese-meat-predict.hpp File Reference

This graph shows which files directly or indirectly include this file:



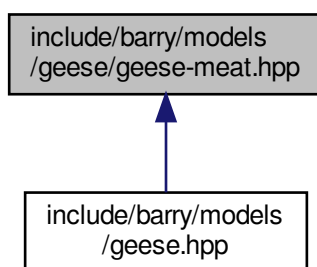
8.29 include/barry/models/geese/geese-meat-simulate.hpp File Reference

This graph shows which files directly or indirectly include this file:



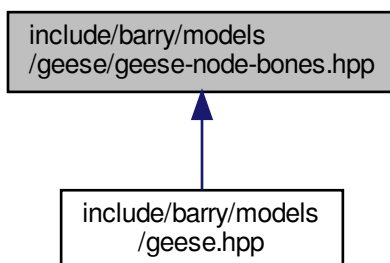
8.30 include/barry/models/geese/geese-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



8.31 include/barry/models/geese/geese-node-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

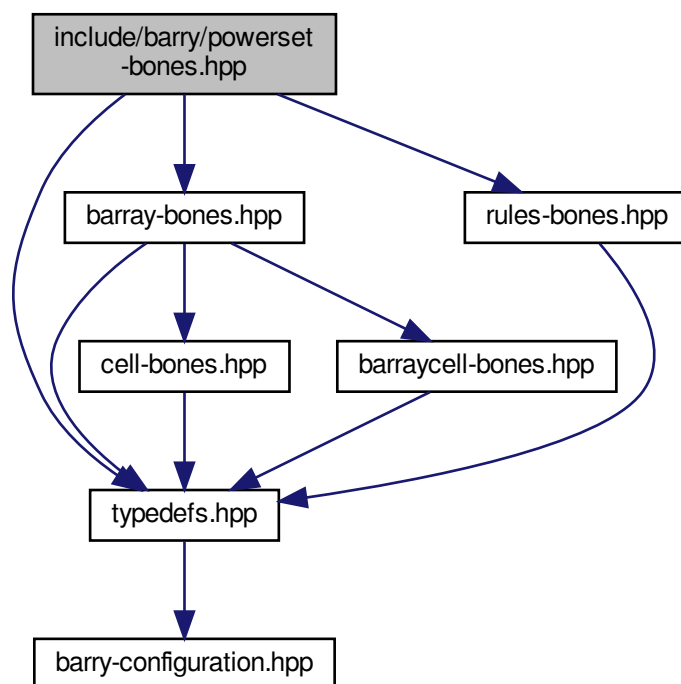
- class [Node](#)

A single node for the model.

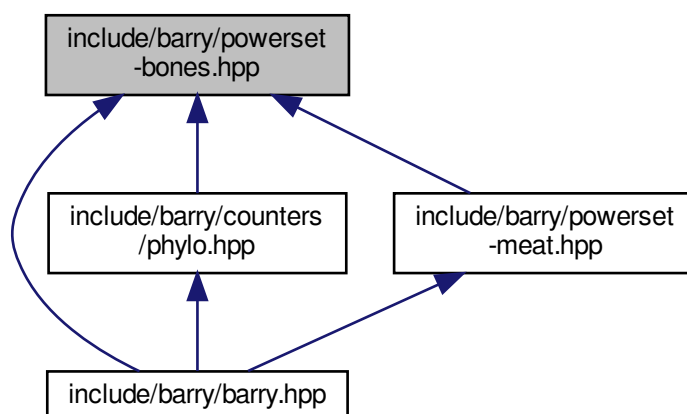
8.32 include/barry/powerset-bones.hpp File Reference

```
#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

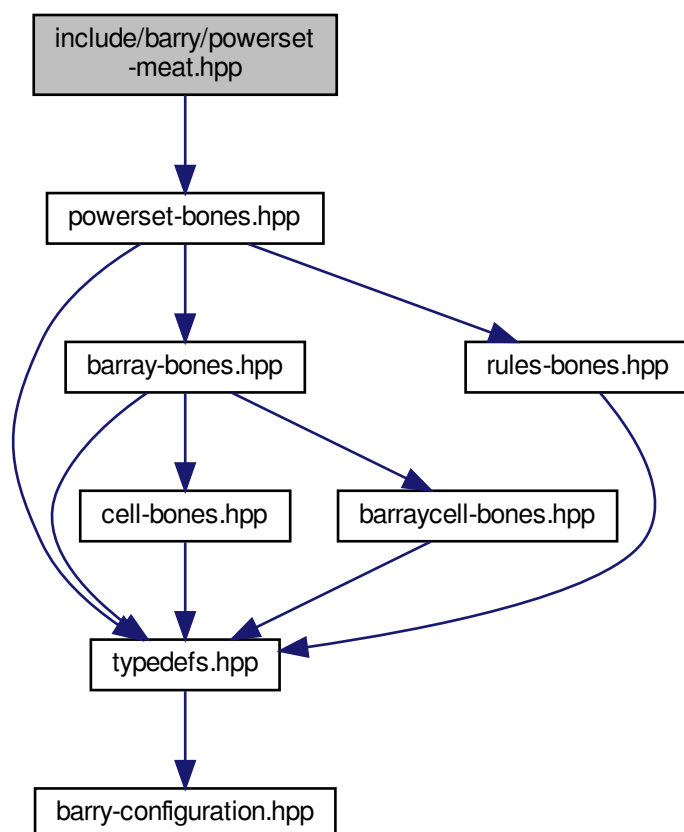
- class [PowerSet< Array_Type, Data_Rule_Type >](#)

Powerset of a binary array.

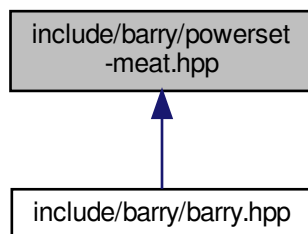
8.33 include/barry/powerset-meat.hpp File Reference

```
#include "powerset-bones.hpp"
```

Include dependency graph for powerset-meat.hpp:



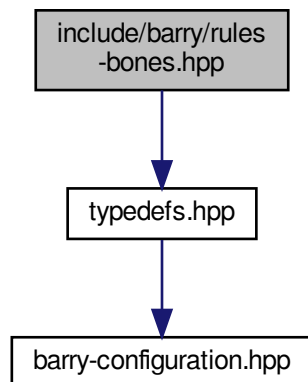
This graph shows which files directly or indirectly include this file:



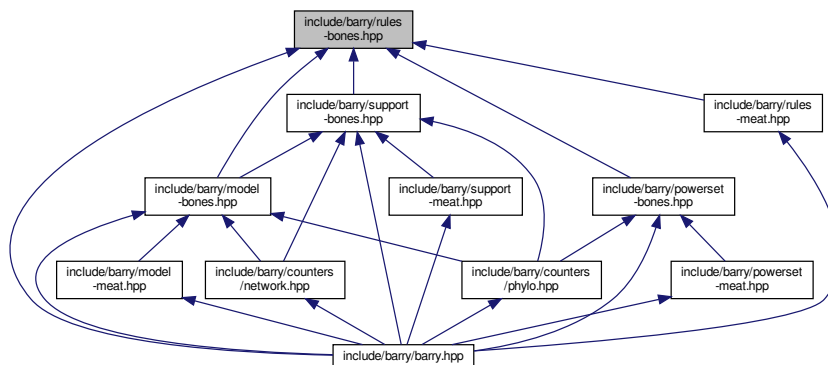
8.34 include/barry/rules-bones.hpp File Reference

```
#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)`

8.34.1 Function Documentation

8.34.1.1 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 )

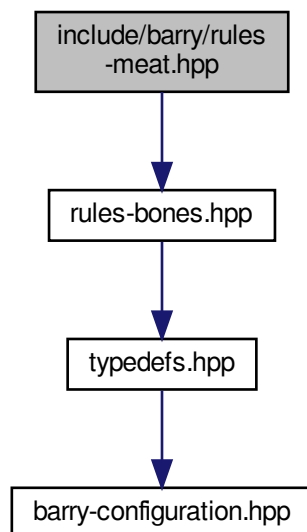
```

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

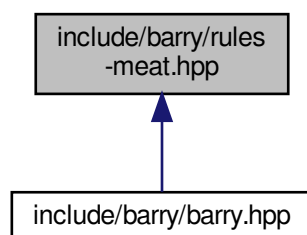
8.35 include/barry/rules-meat.hpp File Reference

```
#include "rules-bones.hpp"
```

Include dependency graph for rules-meat.hpp:



This graph shows which files directly or indirectly include this file:



8.36 include/barry/statscounter-bones.hpp File Reference

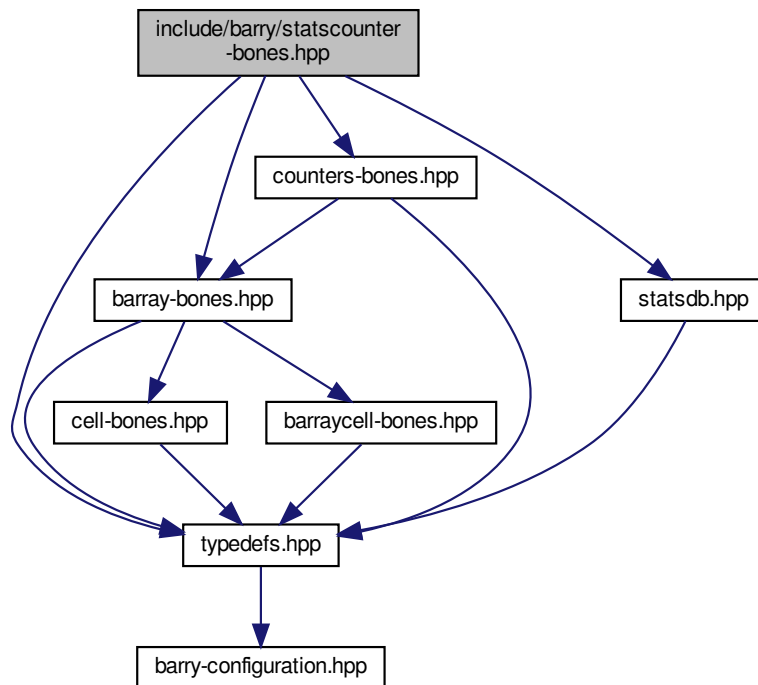
```
#include "typedefs.hpp"
```

```
#include "barray-bones.hpp"
```

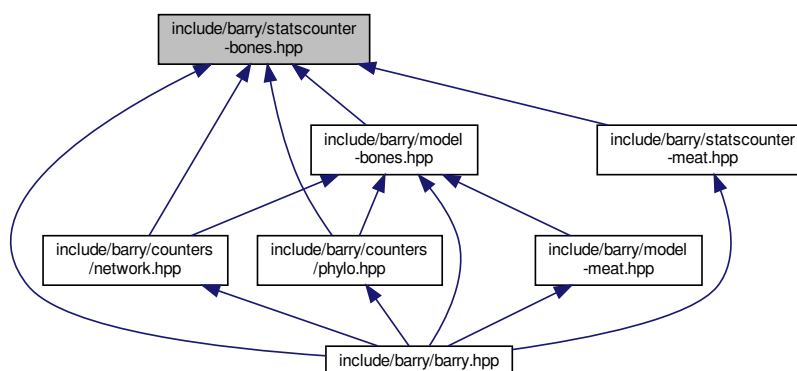
```
#include "statsdb.hpp"
```

```
#include "counters-bones.hpp"
```

Include dependency graph for statscounter-bones.hpp:



This graph shows which files directly or indirectly include this file:



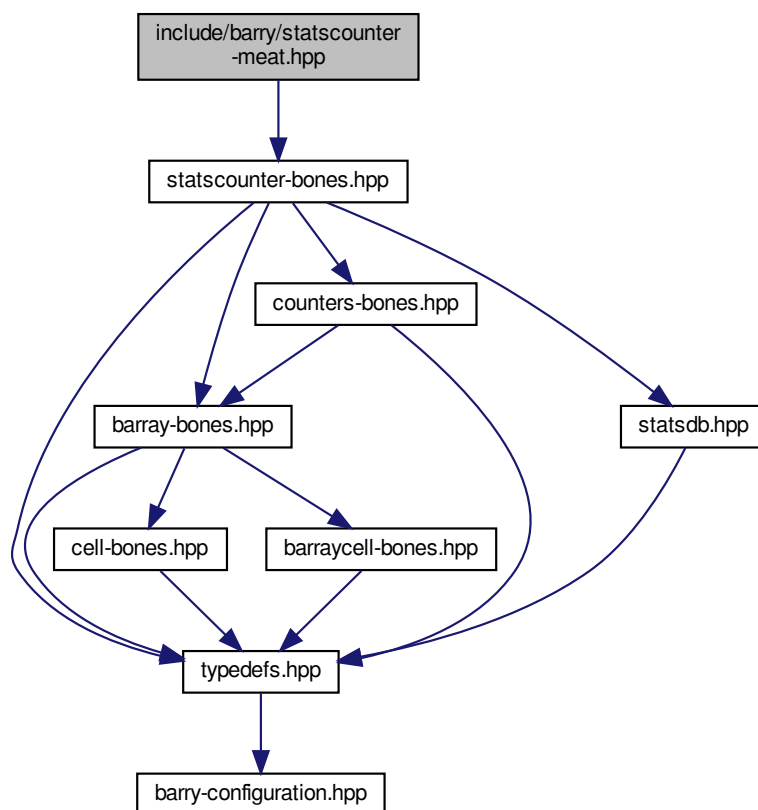
Classes

- class `StatsCounter< Array_Type, Data_Type >`
Count stats for a single Array.

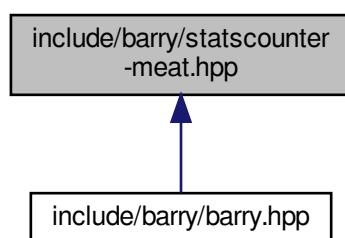
8.37 include/barry/statscounter-meat.hpp File Reference

```
#include "statscounter-bones.hpp"
```

Include dependency graph for statscounter-meat.hpp:



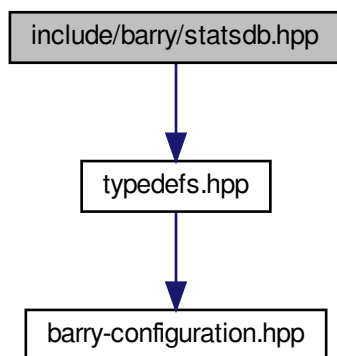
This graph shows which files directly or indirectly include this file:



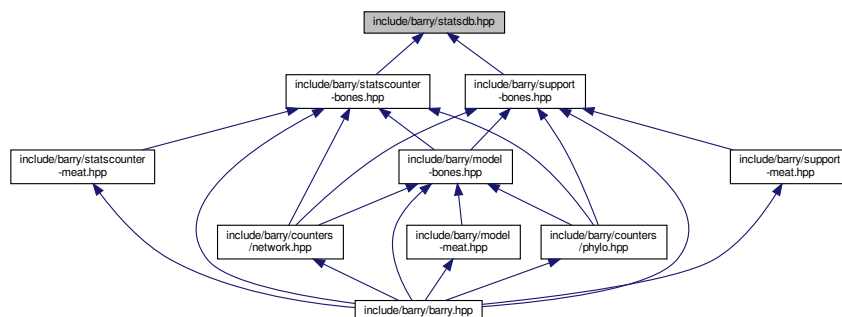
8.38 include/barry/statsdb.hpp File Reference

```
#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.

8.39 include/barry/support-bones.hpp File Reference

```
#include "typedefs.hpp"
```

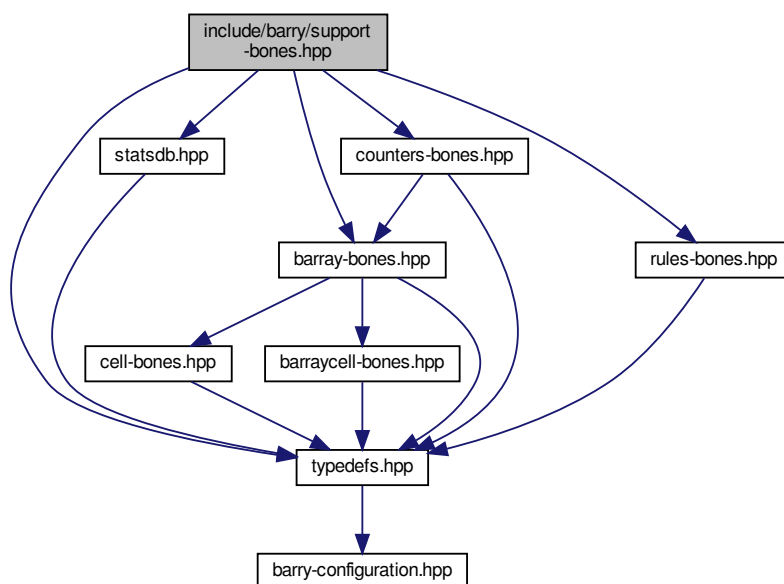
```
#include "barray-bones.hpp"
```

```
#include "statsdb.hpp"
```

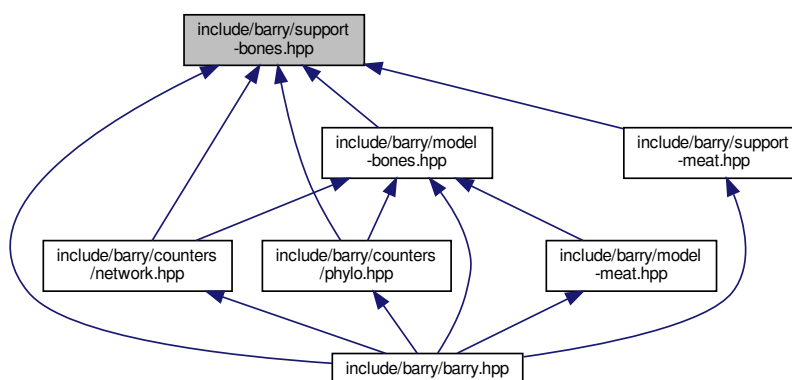
```
#include "counters-bones.hpp"
```

```
#include "rules-bones.hpp"
```

Include dependency graph for support-bones.hpp:



This graph shows which files directly or indirectly include this file:



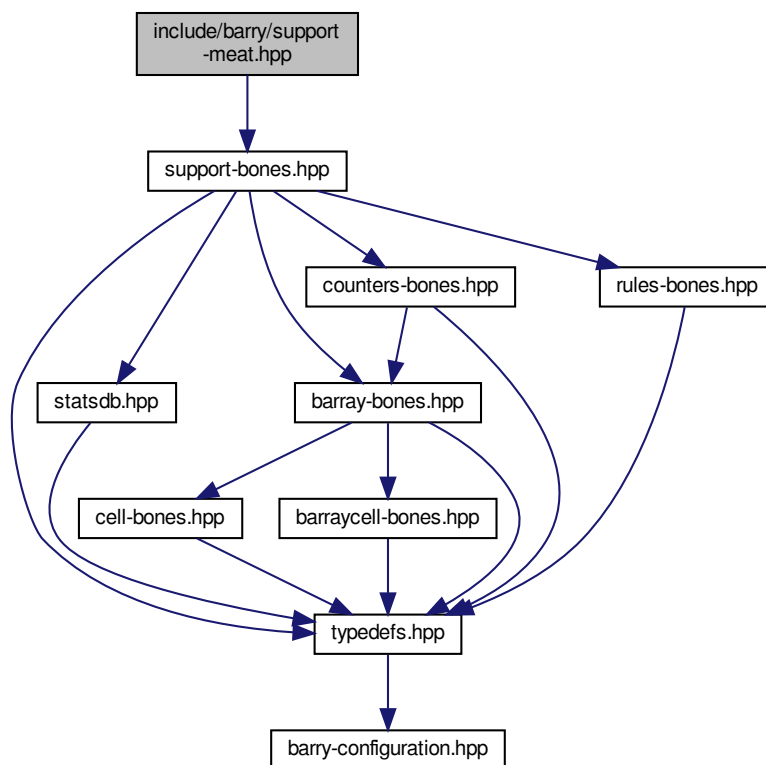
Classes

- class `Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >`
Compute the support of sufficient statistics.

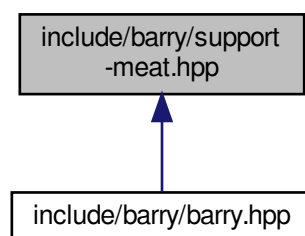
8.40 include/barry/support-meat.hpp File Reference

```
#include "support-bones.hpp"
```

Include dependency graph for support-meat.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- `#define BARRY_SUPPORT_MEAT_HPP` 1

8.40.1 Macro Definition Documentation

8.40.1.1 BARRY_SUPPORT_MEAT_HPP

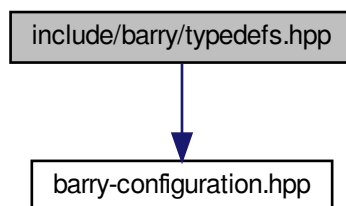
```
#define BARRY_SUPPORT_MEAT_HPP 1
```

Definition at line 4 of file support-meat.hpp.

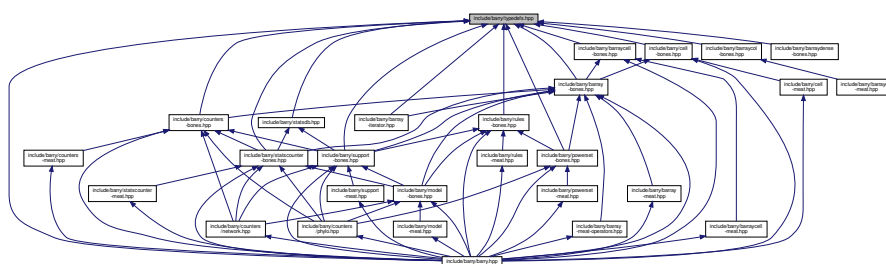
8.41 include/barry/typedefs.hpp File Reference

```
#include "barry-configuration.hpp"
```

Include dependency graph for typedefs.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Entries< Cell_Type >](#)
A wrapper class to store source, target, val from a [BArray](#) object.
- struct [vecHasher< T >](#)

Namespaces

- [CHECK](#)
Integer constants used to specify which cell should be check.
- [EXISTS](#)
Integer constants used to specify which cell should be check to exist or not.

Typedefs

- typedef unsigned int [uint](#)
- typedef std::vector< std::pair< std::vector< double >, [uint](#) > > [Counts_type](#)
- template<typename Cell_Type >
using [Row_type](#) = Map< [uint](#), Cell< Cell_Type > >
- template<typename Cell_Type >
using [Col_type](#) = Map< [uint](#), Cell< Cell_Type > * >
- template<typename Ta = double, typename Tb = uint>
using [MapVec_type](#) = std::unordered_map< std::vector< Ta >, Tb, [vecHasher](#)< Ta > >
- template<typename Array_Type , typename Data_Type >
using [Counter_fun_type](#) = std::function< double(const Array_Type &, [uint](#), [uint](#), Data_Type *)>
[Counter](#) and rule functions.
- template<typename Array_Type , typename Data_Type >
using [Rule_fun_type](#) = std::function< bool(const Array_Type &, [uint](#), [uint](#), Data_Type *)>

Functions

- template<typename T >
T [vec_inner_prod](#) (const std::vector< T > &a, const std::vector< T > &b)
- template<typename T >
bool [vec_equal](#) (const std::vector< T > &a, const std::vector< T > &b)
Compares if -a- and -b- are equal.
- template<typename T >
bool [vec_equal_approx](#) (const std::vector< T > &a, const std::vector< T > &b, double eps=1e-10)

Variables

- const int [CHECK::BOTH](#) = -1
- const int [CHECK::NONE](#) = 0
- const int [CHECK::ONE](#) = 1
- const int [CHECK::TWO](#) = 2
- const int [EXISTS::BOTH](#) = -1
- const int [EXISTS::NONE](#) = 0
- const int [EXISTS::ONE](#) = 1
- const int [EXISTS::TWO](#) = 1
- const int [EXISTS::UNKNOWN](#) = -1
- const int [EXISTS::AS_ZERO](#) = 0
- const int [EXISTS::AS_ONE](#) = 1

8.41.1 Typedef Documentation

8.41.1.1 Col_type

```
template<typename Cell_Type >
using Col_type = Map< uint, Cell<Cell_Type>* >
```

Definition at line 51 of file typedefs.hpp.

8.41.1.2 Counter_fun_type

```
template<typename Array_Type , typename Data_Type >
using Counter_fun_type = std::function<double(const Array_Type &, uint, uint, Data_Type *)>
```

[Counter](#) and rule functions.

Parameters

<i>Array_Type</i>	a BArray
<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 123 of file typedefs.hpp.

8.41.1.3 Counts_type

```
typedef std::vector< std::pair< std::vector<double>, uint > > Counts_type
```

Definition at line 44 of file typedefs.hpp.

8.41.1.4 MapVec_type

```
template<typename Ta = double, typename Tb = uint>
using MapVec_type = std::unordered_map< std::vector< Ta >, Tb, vecHasher<Ta> >
```

Definition at line 105 of file typedefs.hpp.

8.41.1.5 Row_type

```
template<typename Cell_Type >
using Row_type = Map< uint, Cell<Cell_Type> >
```

Definition at line 48 of file typedefs.hpp.

8.41.1.6 Rule_fun_type

```
template<typename Array_Type , typename Data_Type >
using Rule_fun_type = std::function<bool(const Array_Type &, uint, uint, Data_Type *)>
```

Definition at line 126 of file typedefs.hpp.

8.41.1.7 uint

```
typedef unsigned int uint
```

Definition at line 10 of file typedefs.hpp.

8.41.2 Function Documentation

8.41.2.1 vec_equal()

```
template<typename T >
bool vec_equal (
    const std::vector< T > & a,
    const std::vector< T > & b ) [inline]
```

Compares if -a- and -b- are equal.

Parameters

<i>a,b</i>	Two vectors of the same length
------------	--------------------------------

Returns

`true` if all elements are equal.

Definition at line 137 of file typedefs.hpp.

8.41.2.2 `vec_equal_approx()`

```
template<typename T >
bool vec_equal_approx (
    const std::vector< T > & a,
    const std::vector< T > & b,
    double eps = 1e-10 ) [inline]
```

Definition at line 155 of file typedefs.hpp.

8.41.2.3 `vec_inner_prod()`

```
template<typename T >
T vec_inner_prod (
    const std::vector< T > & a,
    const std::vector< T > & b ) [inline]
```

Definition at line 175 of file typedefs.hpp.

8.42 README.md File Reference

Index

- ~BArray
 - BArray< Cell_Type, Data_Type >, [31](#)
- ~BArrayCell
 - BArrayCell< Cell_Type, Data_Type >, [42](#)
- ~BArrayCell_const
 - BArrayCell_const< Cell_Type, Data_Type >, [44](#)
- ~BArrayCol
 - BArrayCol< Cell_Type, Data_Type >, [47](#)
- ~BArrayCol_const
 - BArrayCol_const< Cell_Type, Data_Type >, [50](#)
- ~BArrayDense
 - BArrayDense< Cell_Type, Data_Type >, [52](#)
- ~Cell
 - Cell< Cell_Type >, [56](#)
- ~ConstBArrayRowIter
 - ConstBArrayRowIter< Cell_Type, Data_Type >, [60](#)
- ~Counter
 - Counter< Array_Type, Data_Type >, [63](#)
- ~Counters
 - Counters< Array_Type, Data_Type >, [67](#)
- ~Entries
 - Entries< Cell_Type >, [71](#)
- ~Flock
 - Flock, [74](#)
- ~FreqTable
 - FreqTable< T >, [79](#)
- ~Geese
 - Geese, [83](#)
- ~Model
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [93](#)
- ~NetCounterData
 - NetCounterData, [103](#)
- ~NetworkData
 - NetworkData, [105](#)
- ~Node
 - Node, [108](#)
- ~NodeData
 - NodeData, [112](#)
- ~PhyloRuleDynData
 - PhyloRuleDynData, [114](#)
- ~PowerSet
 - PowerSet< Array_Type, Data_Rule_Type >, [118](#)
- ~Rule
 - Rule< Array_Type, Data_Type >, [123](#)
- ~Rules
 - Rules< Array_Type, Data_Type >, [125](#)
- ~StatsCounter
 - StatsCounter< Array_Type, Data_Type >, [129](#)
- ~Support
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [135](#)
- add
 - Cell< Cell_Type >, [57](#)
 - FreqTable< T >, [79](#)
- add_array
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [93](#)
- add_counter
 - Counters< Array_Type, Data_Type >, [68](#)
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [94](#)
 - StatsCounter< Array_Type, Data_Type >, [129](#)
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [135](#)
- add_data
 - Flock, [74](#)
- add_rule
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [94](#), [95](#)
 - PowerSet< Array_Type, Data_Rule_Type >, [118](#)
 - Rules< Array_Type, Data_Type >, [126](#)
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [135](#), [136](#)
- add_rule_dyn
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [95](#)
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [136](#)
- annotations
 - Node, [109](#)
- Array
 - ConstBArrayRowIter< Cell_Type, Data_Type >, [60](#)
- array
 - Node, [109](#)
- arrays
 - Node, [109](#)
- AS_ONE
 - EXISTS, [25](#)

- as_vector
 - FreqTable< T >, 79
- AS_ZERO
 - EXISTS, 25
- BArray
 - BArray< Cell_Type, Data_Type >, 30, 31
- BArray< Cell_Type, Data_Type >, 27
 - ~BArray, 31
 - BArray, 30, 31
 - BArrayCell< Cell_Type, Data_Type >, 40
 - BArrayCell_const< Cell_Type, Data_Type >, 40
 - clear, 31
 - col, 31
 - D, 32
 - default_val, 32
 - get_cell, 32
 - get_col, 32
 - get_col_vec, 32, 33
 - get_entries, 33
 - get_row, 33
 - get_row_vec, 33
 - insert_cell, 34
 - is_empty, 34
 - ncol, 34
 - nnozero, 35
 - nrow, 35
 - operator*=: 35
 - operator(), 35
 - operator+=, 35, 36
 - operator-=, 36
 - operator/=: 36
 - operator=, 37
 - operator==, 37
 - out_of_range, 37
 - print, 37
 - reserve, 37
 - resize, 38
 - rm_cell, 38
 - row, 38
 - set_data, 38
 - swap_cells, 39
 - swap_cols, 39
 - swap_rows, 39
 - toggle_cell, 39
 - toggle_lock, 39
 - transpose, 40
 - visited, 41
 - zero_col, 40
 - zero_row, 40
- barray-bones.hpp
 - BARRAY_BONES_HPP, 144
- barray-iterator.hpp
 - BARRAY_ITERATOR_HPP, 145
- barray-meat-operators.hpp
 - checkdim_, 147
 - COL, 147
 - ROW, 147
- barray-meat.hpp
 - COL, 149
 - ROW, 149
- BARRAY_BONES_HPP
 - barray-bones.hpp, 144
- BARRAY_ITERATOR_HPP
 - barray-iterator.hpp, 145
- BArrayCell
 - BArrayCell< Cell_Type, Data_Type >, 42
- BArrayCell< Cell_Type, Data_Type >, 41
 - ~BArrayCell, 42
 - BArray< Cell_Type, Data_Type >, 40
 - BArrayCell, 42
 - operator Cell_Type, 42
 - operator*=: 42
 - operator+=, 42
 - operator-=, 43
 - operator/=: 43
 - operator=, 43
 - operator==, 43
- BArrayCell_const
 - BArrayCell_const< Cell_Type, Data_Type >, 44
- BArrayCell_const< Cell_Type, Data_Type >, 44
 - ~BArrayCell_const, 44
 - BArray< Cell_Type, Data_Type >, 40
 - BArrayCell_const, 44
 - operator Cell_Type, 45
 - operator!=, 45
 - operator<, 45
 - operator<=, 45
 - operator>, 45
 - operator>=, 46
 - operator==, 45
- BArrayCol
 - BArrayCol< Cell_Type, Data_Type >, 47
 - BArrayCol_const< Cell_Type, Data_Type >, 50
- BArrayCol< Cell_Type, Data_Type >, 46
 - ~BArrayCol, 47
 - BArrayCol, 47
 - begin, 47
 - end, 47
 - operator Cell_Type, 47
 - operator*=: 48
 - operator+=, 48
 - operator-=, 48
 - operator/=: 48
 - operator=, 48
 - operator==, 49
- barraycol-meat.hpp
 - BARRY_BARRAYCOL_MEAT_HPP, 153
- BArrayCol_const< Cell_Type, Data_Type >, 49
 - ~BArrayCol_const, 50
 - BArrayCol, 50
 - operator!=, 50
 - operator<, 50
 - operator<=, 50
 - operator>, 51
 - operator>=, 51
 - operator==, 51

- BArrayDense
 - BArrayDense< Cell_Type, Data_Type >, 52
- BArrayDense< Cell_Type, Data_Type >, 51
 - ~BArrayDense, 52
 - BArrayDense, 52
 - elements_ptr, 53
 - elements_raw, 53
 - fill, 53
 - ncol, 53
 - nrow, 53
 - operator(), 54
 - operator[], 54
 - print, 54
- barry, 23
- barry-configuration.hpp
 - BARRY_CHECK_SUPPORT, 154
 - BARRY_ISFINITE, 154
 - BARRY_MAX_NUM_ELEMENTS, 155
 - BARRY_SAFE_EXP, 155
 - Map, 155
- barry.hpp
 - BARRY_HPP, 157
 - COUNTER_FUNCTION, 157
 - COUNTER_LAMBDA, 157
 - RULE_FUNCTION, 157
 - RULE_LAMBDA, 158
- barry::counters, 23
- barry::counters::network, 24
- barry::counters::phylo, 24
- BARRY_BARRAYCOL_MEAT_HPP
 - barraycol-meat.hpp, 153
- BARRY_CHECK_SUPPORT
 - barry-configuration.hpp, 154
- BARRY_HPP
 - barry.hpp, 157
- BARRY_ISFINITE
 - barry-configuration.hpp, 154
- BARRY_MAX_NUM_ELEMENTS
 - barry-configuration.hpp, 155
- BARRY_SAFE_EXP
 - barry-configuration.hpp, 155
- BARRY_SUPPORT_MEAT_HPP
 - support-meat.hpp, 195
- begin
 - BArrayCol< Cell_Type, Data_Type >, 47
 - PowerSet< Array_Type, Data_Rule_Type >, 118
- blengths
 - NodeData, 113
- BOTH
 - CHECK, 24
 - EXISTS, 25
- calc
 - PowerSet< Array_Type, Data_Rule_Type >, 119
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 136
- calc_reduced_sequence
 - Geese, 84
- calc_sequence
 - Geese, 84
- Cell
 - Cell< Cell_Type >, 55–57
- Cell< Cell_Type >, 54
 - ~Cell, 56
 - add, 57
 - Cell, 55–57
 - operator Cell_Type, 58
 - operator=, 58
 - value, 58
 - visited, 58
- change_stats
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 140
- CHECK, 24
 - BOTH, 24
 - NONE, 24
 - ONE, 24
 - TWO, 24
- checkdim_
 - barray-meat-operators.hpp, 147
- clear
 - BArray< Cell_Type, Data_Type >, 31
 - Counters< Array_Type, Data_Type >, 68
 - FreqTable< T >, 79
 - Rules< Array_Type, Data_Type >, 126
- COL
 - barray-meat-operators.hpp, 147
 - barray-meat.hpp, 149
- col
 - BArray< Cell_Type, Data_Type >, 31
- Col_type
 - typedefs.hpp, 197
- ConstBArrayRowIter
 - ConstBArrayRowIter< Cell_Type, Data_Type >, 60
- ConstBArrayRowIter< Cell_Type, Data_Type >, 59
 - ~ConstBArrayRowIter, 60
 - Array, 60
 - ConstBArrayRowIter, 60
 - current_col, 60
 - current_row, 60
 - iter, 61
- coordinates_free
 - PowerSet< Array_Type, Data_Rule_Type >, 120
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 140
- coordinates_locked
 - PowerSet< Array_Type, Data_Rule_Type >, 120
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 140
- count
 - Counter< Array_Type, Data_Type >, 63
- count_all
 - StatsCounter< Array_Type, Data_Type >, 130

- count_current
 - StatsCounter< Array_Type, Data_Type >, 130
- count_fun
 - Counter< Array_Type, Data_Type >, 64
- count_init
 - StatsCounter< Array_Type, Data_Type >, 130
- Counter
 - Counter< Array_Type, Data_Type >, 62, 63
- Counter< Array_Type, Data_Type >, 61
 - ~Counter, 63
 - count, 63
 - count_fun, 64
 - Counter, 62, 63
 - data, 65
 - delete_data, 65
 - desc, 65
 - init, 64
 - init_fun, 65
 - name, 65
 - operator=, 64
- 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_density
 - Network counters, 11
- counter_diff
 - Network counters, 11
- counter_edges
 - Network counters, 12
- Counter_fun_type
 - typedefs.hpp, 197
- COUNTER_FUNCTION
 - barry.hpp, 157
- 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, 157
- counter_longest
 - Phylo counters, 17
- counter_loss
 - Phylo counters, 17
- counter_maxfuns
 - 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_nodeocov
 - 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_ttriads
 - Network counters, 14
- Counters
 - Counters< Array_Type, Data_Type >, 67
- Counters< Array_Type, Data_Type >, 66
 - ~Counters, 67
 - add_counter, 68
 - clear, 68
 - Counters, 67
 - operator=, 69
 - operator[], 69
 - size, 70
- Counting, 9
- counts
 - PhyloRuleDynData, 114
- Counts_type
 - typedefs.hpp, 197
- current_col
 - ConstBArrayRowIter< Cell_Type, Data_Type >, 60
- current_row
 - ConstBArrayRowIter< Cell_Type, Data_Type >, 60
- current_stats
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 140

D

- BArray< Cell_Type, Data_Type >, 32
- Rule< Array_Type, Data_Type >, 124
- dat
 - Flock, 77
- data
 - Counter< Array_Type, Data_Type >, 65
 - PowerSet< Array_Type, Data_Rule_Type >, 121
- default_val
 - BArray< Cell_Type, Data_Type >, 32
- delete_counters
 - Geese, 88
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 141
- delete_data
 - Counter< Array_Type, Data_Type >, 65
- delete_engine
 - Geese, 88
- delete_rules
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 141
- delete_rules_dyn
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 141
- delete_support
 - Geese, 88
- desc
 - Counter< Array_Type, Data_Type >, 65
- directed
 - NetworkData, 105
- duplication
 - Node, 109
 - NodeData, 113
 - PhyloRuleDynData, 115
- elements_ptr
 - BArrayDense< Cell_Type, Data_Type >, 53
- elements_raw
 - BArrayDense< Cell_Type, Data_Type >, 53
- EmptyArray
 - PowerSet< Array_Type, Data_Rule_Type >, 121
- end
 - BArrayCol< Cell_Type, Data_Type >, 47
 - PowerSet< Array_Type, Data_Rule_Type >, 119
- Entries
 - Entries< Cell_Type >, 71
- Entries< Cell_Type >, 70
 - ~Entries, 71
 - Entries, 71
 - resize, 72
 - source, 72
 - target, 72
 - val, 72
- EXISTS, 25
 - AS_ONE, 25
 - AS_ZERO, 25
 - BOTH, 25
 - NONE, 26
 - ONE, 26
 - TWO, 26
 - UNKNOWN, 26
- fill
 - BArrayDense< Cell_Type, Data_Type >, 53
- Flock, 73
 - ~Flock, 74
 - add_data, 74
 - dat, 77
 - Flock, 74
 - get_counters, 74
 - init, 75
 - initialized, 77
 - likelihood_joint, 75
 - nfunctions, 77
 - nfuncs, 75
 - nleaves, 75
 - nnodes, 76
 - nterms, 76
 - ntrees, 76
 - operator(), 76
 - engine, 77
 - set_seed, 77
 - support, 78
- FreqTable
 - FreqTable< T >, 79
- FreqTable< T >, 78
 - ~FreqTable, 79
 - add, 79
 - as_vector, 79
 - clear, 79
 - FreqTable, 79
 - get_data, 80
 - get_data_ptr, 80
 - print, 80
 - reserve, 80
 - size, 80
- Geese, 81
 - ~Geese, 83
 - calc_reduced_sequence, 84
 - calc_sequence, 84
 - delete_counters, 88
 - delete_engine, 88
 - delete_support, 88
 - Geese, 83
 - get_counters, 84
 - get_probabilities, 84
 - get_engine, 84
 - get_states, 84
 - get_support, 85
 - inherit_support, 85
 - init, 85
 - init_node, 85
 - initialized, 89
 - likelihood, 85
 - likelihood_exhaust, 85

- map_to_nodes, 89
- nfunctions, 89
- nfuns, 86
- nleaves, 86
- nnodes, 86
- nodes, 89
- nterms, 86
- observed_counts, 86
- operator=, 86, 87
- predict, 87
- predict_backend, 87
- print_observed_counts, 87
- reduced_sequence, 89
- sequence, 89
- set_seed, 87
- simulate, 88
- update_annotations, 88
- geese-bones.hpp
 - INITIALIZED, 179
 - keygen_full, 179
 - RULE_FUNCTION, 179
 - vec_diff, 179
 - vector_caster, 180
- get_cell
 - BArray< Cell_Type, Data_Type >, 32
- get_col
 - BArray< Cell_Type, Data_Type >, 32
- get_col_vec
 - BArray< Cell_Type, Data_Type >, 32, 33
- get_counters
 - Flock, 74
 - Geese, 84
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 96
 - StatsCounter< Array_Type, Data_Type >, 130
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 137
- get_counts
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 137
- get_counts_ptr
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 137
- get_current_stats
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 137
- get_data
 - FreqTable< T >, 80
 - PowerSet< Array_Type, Data_Rule_Type >, 119
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 137
- get_data_ptr
 - FreqTable< T >, 80
 - PowerSet< Array_Type, Data_Rule_Type >, 119
- get_entries
 - BArray< Cell_Type, Data_Type >, 33
- get_last_name
 - phylo.hpp, 173
- get_norm_const
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 96
- get_parent
 - Node, 108
- get_probabilities
 - Geese, 84
- get_pset
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 96
- get_rengine
 - Geese, 84
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 96
- get_row
 - BArray< Cell_Type, Data_Type >, 33
- get_row_vec
 - BArray< Cell_Type, Data_Type >, 33
- get_rules
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 97
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 138
- get_rules_dyn
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 97
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 138
- get_seq
 - Rules< Array_Type, Data_Type >, 126
- get_states
 - Geese, 84
- get_stats
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 97
- get_support
 - Geese, 85
- id
 - Node, 110
- include/barry/barray-bones.hpp, 143
- include/barry/barray-iterator.hpp, 144
- include/barry/barray-meat-operators.hpp, 146
- include/barry/barray-meat.hpp, 148
- include/barry/barraycell-bones.hpp, 149

- include/barry/barraycell-meat.hpp, 150
- include/barry/barraycol-bones.hpp, 151
- include/barry/barraycol-meat.hpp, 152
- include/barry/barraydense-bones.hpp, 153
- include/barry/barry-configuration.hpp, 154
- include/barry/barry.hpp, 156
- include/barry/cell-bones.hpp, 158
- include/barry/cell-meat.hpp, 159
- include/barry/col-bones.hpp, 160
- include/barry/counters-bones.hpp, 160
- include/barry/counters-meat.hpp, 161
- include/barry/counters/network.hpp, 162
- include/barry/counters/phylo.hpp, 168
- include/barry/model-bones.hpp, 174
- include/barry/model-meat.hpp, 176
- include/barry/models/geese.hpp, 176
- include/barry/models/geese/flock-bones.hpp, 177
- include/barry/models/geese/flock-meat.hpp, 178
- include/barry/models/geese/geese-bones.hpp, 178
- include/barry/models/geese/geese-meat-constructors.hpp, 180
- include/barry/models/geese/geese-meat-likelihood.hpp, 181
- include/barry/models/geese/geese-meat-likelihood_exhaust.hpp, 182
- include/barry/models/geese/geese-meat-predict.hpp, 182
- include/barry/models/geese/geese-meat-simulate.hpp, 183
- include/barry/models/geese/geese-meat.hpp, 183
- include/barry/models/geese/geese-node-bones.hpp, 184
- include/barry/powerset-bones.hpp, 184
- include/barry/powerset-meat.hpp, 186
- include/barry/rules-bones.hpp, 187
- include/barry/rules-meat.hpp, 189
- include/barry/statscounter-bones.hpp, 189
- include/barry/statscounter-meat.hpp, 191
- include/barry/statsdb.hpp, 192
- include/barry/support-bones.hpp, 192
- include/barry/support-meat.hpp, 194
- include/barry/typedefs.hpp, 195
- indices
 - NetCounterData, 103
- inherit_support
 - Geese, 85
- init
 - Counter< Array_Type, Data_Type >, 64
 - Flock, 75
 - Geese, 85
- init_fun
 - Counter< Array_Type, Data_Type >, 65
- init_node
 - Geese, 85
- init_support
 - PowerSet< Array_Type, Data_Rule_Type >, 119
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 138
- INITIALIZED
 - geese-bones.hpp, 179
- initialized
 - Flock, 77
 - Geese, 89
- insert_cell
 - BArray< Cell_Type, Data_Type >, 34
- is_empty
 - BArray< Cell_Type, Data_Type >, 34
- is_leaf
 - Node, 109
- iter
 - ConstBArrayRowIter< Cell_Type, Data_Type >, 61
- keygen_default
 - model-bones.hpp, 175
- keygen_full
 - geese-bones.hpp, 179
- lb
 - PhyloRuleDynData, 115
- likelihood
 - Geese, 85
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 97, 98
- likelihood_
 - model-bones.hpp, 175
- likelihood_exhaust
 - Geese, 85
- likelihood_joint
 - Flock, 75
- likelihood_total
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 98
- M
 - PowerSet< Array_Type, Data_Rule_Type >, 121
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 141
- Map
 - barry-configuration.hpp, 155
- map_to_nodes
 - Geese, 89
- MapVec_type
 - typedefs.hpp, 197
- max_num_elements
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 141
- Model
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 92, 93
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 90

- ~Model, 93
- add_array, 93
- add_counter, 94
- add_rule, 94, 95
- add_rule_dyn, 95
- get_counters, 96
- get_norm_const, 96
- get_pset, 96
- get_rengine, 96
- get_rules, 97
- get_rules_dyn, 97
- get_stats, 97
- likelihood, 97, 98
- likelihood_total, 98
- Model, 92, 93
- nterms, 98
- operator=, 99
- print_stats, 99
- sample, 99
- set_counters, 100
- set_keygen, 100
- set_rengine, 100
- set_rules, 100
- set_rules_dyn, 101
- set_seed, 101
- size, 101
- size_unique, 101
- store_psets, 101
- model-bones.hpp
 - keygen_default, 175
 - likelihood_, 175
 - update_normalizing_constant, 175
- N
 - PowerSet< Array_Type, Data_Rule_Type >, 121
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 142
- name
 - Counter< Array_Type, Data_Type >, 65
- narray
 - Node, 110
- ncol
 - BArray< Cell_Type, Data_Type >, 34
 - BArrayDense< Cell_Type, Data_Type >, 53
- NET_C_DATA_IDX
 - network.hpp, 165
- NET_C_DATA_NUM
 - network.hpp, 165
- NetCounter
 - network.hpp, 166
- NetCounterData, 102
 - ~NetCounterData, 103
 - indices, 103
 - NetCounterData, 102
 - numbers, 103
- NetCounters
 - network.hpp, 166
- NetModel
 - network.hpp, 167
- NetRule
 - network.hpp, 167
- NetRules
 - network.hpp, 167
- NetStatsCounter
 - network.hpp, 167
- NetSupport
 - network.hpp, 167
- Network
 - network.hpp, 167
- 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, 165
 - NET_C_DATA_NUM, 165
 - NetCounter, 166
 - NetCounters, 166
 - NetModel, 167
 - NetRule, 167
 - NetRules, 167
 - NetStatsCounter, 167
 - NetSupport, 167
 - Network, 167
 - NETWORK_COUNTER, 165
 - NETWORK_COUNTER_LAMBDA, 165
 - NETWORK_RULE, 166
 - NETWORK_RULE_LAMBDA, 166
 - rules_zerodiag, 168
- NETWORK_COUNTER
 - Network counters, 14
 - network.hpp, 165
- NETWORK_COUNTER_LAMBDA
 - network.hpp, 165
- NETWORK_RULE
 - network.hpp, 166
- NETWORK_RULE_LAMBDA
 - network.hpp, 166
- NetworkData, 103

- ~NetworkData, 105
- directed, 105
- NetworkData, 104, 105
- vertex_attr, 105
- nfunctions
 - Flock, 77
 - Geese, 89
- nfuncs
 - Flock, 75
 - Geese, 86
- nleafs
 - Flock, 75
 - Geese, 86
- nnodes
 - Flock, 76
 - Geese, 86
- nnozero
 - BArray< Cell_Type, Data_Type >, 35
- Node, 106
 - ~Node, 108
 - annotations, 109
 - array, 109
 - arrays, 109
 - duplication, 109
 - get_parent, 108
 - id, 110
 - is_leaf, 109
 - narray, 110
 - Node, 107, 108
 - offspring, 110
 - ord, 110
 - parent, 110
 - probability, 111
 - subtree_prob, 111
 - visited, 111
- NodeData, 111
 - ~NodeData, 112
 - blengths, 113
 - duplication, 113
 - NodeData, 112
 - states, 113
- nodes
 - Geese, 89
- NONE
 - CHECK, 24
 - EXISTS, 26
- nrow
 - BArray< Cell_Type, Data_Type >, 35
 - BArrayDense< Cell_Type, Data_Type >, 53
- nterms
 - Flock, 76
 - Geese, 86
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 98
- ntrees
 - Flock, 76
- numbers
 - NetCounterData, 103
- observed_counts
 - Geese, 86
- offspring
 - Node, 110
- ONE
 - CHECK, 24
 - EXISTS, 26
- operator Cell_Type
 - BArrayCell< Cell_Type, Data_Type >, 42
 - BArrayCell_const< Cell_Type, Data_Type >, 45
 - BArrayCol< Cell_Type, Data_Type >, 47
 - Cell< Cell_Type >, 58
- operator!=
 - BArrayCell_const< Cell_Type, Data_Type >, 45
 - BArrayCol_const< Cell_Type, Data_Type >, 50
- operator<
 - BArrayCell_const< Cell_Type, Data_Type >, 45
 - BArrayCol_const< Cell_Type, Data_Type >, 50
- operator<=
 - BArrayCell_const< Cell_Type, Data_Type >, 45
 - BArrayCol_const< Cell_Type, Data_Type >, 50
- operator>
 - BArrayCell_const< Cell_Type, Data_Type >, 45
 - BArrayCol_const< Cell_Type, Data_Type >, 51
- operator>=
 - BArrayCell_const< Cell_Type, Data_Type >, 46
 - BArrayCol_const< Cell_Type, Data_Type >, 51
- operator*=
 - BArray< Cell_Type, Data_Type >, 35
 - BArrayCell< Cell_Type, Data_Type >, 42
 - BArrayCol< Cell_Type, Data_Type >, 48
- operator()
 - BArray< Cell_Type, Data_Type >, 35
 - BArrayDense< Cell_Type, Data_Type >, 54
 - Flock, 76
 - Rule< Array_Type, Data_Type >, 124
 - Rules< Array_Type, Data_Type >, 127
 - vecHasher< T >, 142
- operator+=
 - BArray< Cell_Type, Data_Type >, 35, 36
 - BArrayCell< Cell_Type, Data_Type >, 42
 - BArrayCol< Cell_Type, Data_Type >, 48
- operator-=
 - BArray< Cell_Type, Data_Type >, 36
 - BArrayCell< Cell_Type, Data_Type >, 43
 - BArrayCol< Cell_Type, Data_Type >, 48
- operator/=
 - BArray< Cell_Type, Data_Type >, 36
 - BArrayCell< Cell_Type, Data_Type >, 43
 - BArrayCol< Cell_Type, Data_Type >, 48
- operator=
 - BArray< Cell_Type, Data_Type >, 37
 - BArrayCell< Cell_Type, Data_Type >, 43
 - BArrayCol< Cell_Type, Data_Type >, 48
 - Cell< Cell_Type >, 58
 - Counter< Array_Type, Data_Type >, 64
 - Counters< Array_Type, Data_Type >, 69

- Geese, [86](#), [87](#)
- Model< Array_Type, Data_Counter_Type,
Data_Rule_Type, Data_Rule_Dyn_Type >,
[99](#)
- Rules< Array_Type, Data_Type >, [127](#)
- operator==
 - BArray< Cell_Type, Data_Type >, [37](#)
 - BArrayCell< Cell_Type, Data_Type >, [43](#)
 - BArrayCell_const< Cell_Type, Data_Type >, [45](#)
 - BArrayCol< Cell_Type, Data_Type >, [49](#)
 - BArrayCol_const< Cell_Type, Data_Type >, [51](#)
- operator[]
 - BArrayDense< Cell_Type, Data_Type >, [54](#)
 - Counters< Array_Type, Data_Type >, [69](#)
 - PowerSet< Array_Type, Data_Rule_Type >, [120](#)
- ord
 - Node, [110](#)
- out_of_range
 - BArray< Cell_Type, Data_Type >, [37](#)
- parent
 - Node, [110](#)
- 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_maxfuns, [18](#)
 - counter_neofun, [18](#)
 - counter_neofun_a2b, [18](#)
 - counter_overall_changes, [18](#)
 - counter_overall_gains, [19](#)
 - counter_overall_loss, [19](#)
 - counter_subfun, [19](#)
- Phylo rules, [20](#)
 - rule_dyn_limit_changes, [20](#)
- phylo.hpp
 - get_last_name, [173](#)
 - PHYLO_CHECK_MISSING, [170](#)
 - PHYLO_COUNTER_LAMBDA, [170](#)
 - PHYLO_RULE_DYN_LAMBDA, [171](#)
 - PhyloArray, [171](#)
 - PhyloCounter, [171](#)
 - PhyloCounterData, [171](#)
 - PhyloCounters, [172](#)
 - PhyloModel, [172](#)
 - PhyloPowerSet, [172](#)
 - PhyloRule, [172](#)
 - PhyloRuleData, [172](#)
 - PhyloRuleDyn, [172](#)
 - PhyloRules, [173](#)
 - PhyloRulesDyn, [173](#)
 - PhyloStatsCounter, [173](#)
 - PhyloSupport, [173](#)
- PHYLO_CHECK_MISSING
 - phylo.hpp, [170](#)
- PHYLO_COUNTER_LAMBDA
 - phylo.hpp, [170](#)
- PHYLO_RULE_DYN_LAMBDA
 - phylo.hpp, [171](#)
- PhyloArray
 - phylo.hpp, [171](#)
- PhyloCounter
 - phylo.hpp, [171](#)
- PhyloCounterData
 - phylo.hpp, [171](#)
- PhyloCounters
 - phylo.hpp, [172](#)
- PhyloModel
 - phylo.hpp, [172](#)
- PhyloPowerSet
 - phylo.hpp, [172](#)
- PhyloRule
 - phylo.hpp, [172](#)
- PhyloRuleData
 - phylo.hpp, [172](#)
- PhyloRuleDyn
 - phylo.hpp, [172](#)
- PhyloRuleDynData, [113](#)
 - ~PhyloRuleDynData, [114](#)
 - counts, [114](#)
 - duplication, [115](#)
 - lb, [115](#)
 - PhyloRuleDynData, [114](#)
 - pos, [115](#)
 - ub, [115](#)
- PhyloRules
 - phylo.hpp, [173](#)
- PhyloRulesDyn
 - phylo.hpp, [173](#)
- PhyloStatsCounter
 - phylo.hpp, [173](#)
- PhyloSupport
 - phylo.hpp, [173](#)
- pos
 - PhyloRuleDynData, [115](#)
- PowerSet
 - PowerSet< Array_Type, Data_Rule_Type >, [117](#)
- PowerSet< Array_Type, Data_Rule_Type >, [116](#)
 - ~PowerSet, [118](#)
 - add_rule, [118](#)
 - begin, [118](#)
 - calc, [119](#)
 - coordinates_free, [120](#)
 - coordinates_locked, [120](#)
 - data, [121](#)
 - EmptyArray, [121](#)
 - end, [119](#)
 - get_data, [119](#)
 - get_data_ptr, [119](#)
 - init_support, [119](#)
 - M, [121](#)
 - N, [121](#)
 - operator[], [120](#)

- PowerSet, [117](#)
- reset, [120](#)
- rules, [121](#)
- rules_deleted, [122](#)
- size, [120](#)
- predict
 - Geese, [87](#)
- predict_backend
 - Geese, [87](#)
- print
 - BArray< Cell_Type, Data_Type >, [37](#)
 - BArrayDense< Cell_Type, Data_Type >, [54](#)
 - FreqTable< T >, [80](#)
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [138](#)
- print_observed_counts
 - Geese, [87](#)
- print_stats
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [99](#)
- probability
 - Node, [111](#)
- README.md, [199](#)
- reduced_sequence
 - Geese, [89](#)
- rengine
 - Flock, [77](#)
- reserve
 - BArray< Cell_Type, Data_Type >, [37](#)
 - FreqTable< T >, [80](#)
- reset
 - PowerSet< Array_Type, Data_Rule_Type >, [120](#)
- reset_array
 - StatsCounter< Array_Type, Data_Type >, [130](#)
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [139](#)
- resize
 - BArray< Cell_Type, Data_Type >, [38](#)
 - Entries< Cell_Type >, [72](#)
- rm_cell
 - BArray< Cell_Type, Data_Type >, [38](#)
- ROW
 - barray-meat-operators.hpp, [147](#)
 - barray-meat.hpp, [149](#)
- row
 - BArray< Cell_Type, Data_Type >, [38](#)
- Row_type
 - typedefs.hpp, [197](#)
- Rule
 - Rule< Array_Type, Data_Type >, [123](#)
- Rule< Array_Type, Data_Type >, [122](#)
 - ~Rule, [123](#)
 - D, [124](#)
 - operator(), [124](#)
 - Rule, [123](#)
- rule_dyn_limit_changes
 - Phylo rules, [20](#)
- rule_fun_default
 - rules-bones.hpp, [188](#)
- Rule_fun_type
 - typedefs.hpp, [198](#)
- RULE_FUNCTION
 - barry.hpp, [157](#)
 - geese-bones.hpp, [179](#)
- RULE_LAMBDA
 - barry.hpp, [158](#)
- Rules
 - Rules< Array_Type, Data_Type >, [125](#)
- rules
 - PowerSet< Array_Type, Data_Rule_Type >, [121](#)
- Rules< Array_Type, Data_Type >, [124](#)
 - ~Rules, [125](#)
 - add_rule, [126](#)
 - clear, [126](#)
 - get_seq, [126](#)
 - operator(), [127](#)
 - operator=, [127](#)
 - Rules, [125](#)
 - size, [127](#)
- rules-bones.hpp
 - rule_fun_default, [188](#)
- rules_deleted
 - PowerSet< Array_Type, Data_Rule_Type >, [122](#)
- rules_zerodiag
 - network.hpp, [168](#)
- sample
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [99](#)
- sequence
 - Geese, [89](#)
- set_counters
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [100](#)
 - StatsCounter< Array_Type, Data_Type >, [131](#)
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [139](#)
- set_data
 - BArray< Cell_Type, Data_Type >, [38](#)
- set_keygen
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [100](#)
- set_rengine
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [100](#)
- set_rules
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [100](#)

- Support< Array_Type, Data_Counter_Type,
Data_Rule_Type, Data_Rule_Dyn_Type >,
139
- set_rules_dyn
 - Model< Array_Type, Data_Counter_Type,
Data_Rule_Type, Data_Rule_Dyn_Type >,
101
 - Support< Array_Type, Data_Counter_Type,
Data_Rule_Type, Data_Rule_Dyn_Type >,
139
- set_seed
 - Flock, 77
 - Geese, 87
 - Model< Array_Type, Data_Counter_Type,
Data_Rule_Type, Data_Rule_Dyn_Type >,
101
- simulate
 - Geese, 88
- size
 - Counters< Array_Type, Data_Type >, 70
 - FreqTable< T >, 80
 - Model< Array_Type, Data_Counter_Type,
Data_Rule_Type, Data_Rule_Dyn_Type >,
101
 - PowerSet< Array_Type, Data_Rule_Type >, 120
 - Rules< Array_Type, Data_Type >, 127
- size_unique
 - Model< Array_Type, Data_Counter_Type,
Data_Rule_Type, Data_Rule_Dyn_Type >,
101
- source
 - Entries< Cell_Type >, 72
- states
 - NodeData, 113
- Statistical Models, 9
- StatsCounter
 - StatsCounter< Array_Type, Data_Type >, 129
- StatsCounter< Array_Type, Data_Type >, 128
 - ~StatsCounter, 129
 - add_counter, 129
 - count_all, 130
 - count_current, 130
 - count_init, 130
 - get_counters, 130
 - reset_array, 130
 - set_counters, 131
 - StatsCounter, 129
- store_psets
 - Model< Array_Type, Data_Counter_Type,
Data_Rule_Type, Data_Rule_Dyn_Type >,
101
- subtree_prob
 - Node, 111
- Support
 - Support< Array_Type, Data_Counter_Type,
Data_Rule_Type, Data_Rule_Dyn_Type >,
134
- support
 - Flock, 78
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >, 131
 - ~Support, 135
 - add_counter, 135
 - add_rule, 135, 136
 - add_rule_dyn, 136
 - calc, 136
 - change_stats, 140
 - coordinates_free, 140
 - coordinates_locked, 140
 - current_stats, 140
 - delete_counters, 141
 - delete_rules, 141
 - delete_rules_dyn, 141
 - get_counters, 137
 - get_counts, 137
 - get_counts_ptr, 137
 - get_current_stats, 137
 - get_data, 137
 - get_rules, 138
 - get_rules_dyn, 138
 - init_support, 138
 - M, 141
 - max_num_elements, 141
 - N, 142
 - print, 138
 - reset_array, 139
 - set_counters, 139
 - set_rules, 139
 - set_rules_dyn, 139
 - Support, 134
- support-meat.hpp
 - BARRY_SUPPORT_MEAT_HPP, 195
- swap_cells
 - BArray< Cell_Type, Data_Type >, 39
- swap_cols
 - BArray< Cell_Type, Data_Type >, 39
- swap_rows
 - BArray< Cell_Type, Data_Type >, 39
- target
 - Entries< Cell_Type >, 72
- toggle_cell
 - BArray< Cell_Type, Data_Type >, 39
- toggle_lock
 - BArray< Cell_Type, Data_Type >, 39
- transpose
 - BArray< Cell_Type, Data_Type >, 40
- TWO
 - CHECK, 24
 - EXISTS, 26
- typedefs.hpp
 - Col_type, 197
 - Counter_fun_type, 197
 - Counts_type, 197
 - MapVec_type, 197
 - Row_type, 197
 - Rule_fun_type, 198

- uint, [198](#)
- vec_equal, [198](#)
- vec_equal_approx, [198](#)
- vec_inner_prod, [199](#)
- ub
 - PhyloRuleDynData, [115](#)
- uint
 - typedefs.hpp, [198](#)
- UNKNOWN
 - EXISTS, [26](#)
- update_annotations
 - Geese, [88](#)
- update_normalizing_constant
 - model-bones.hpp, [175](#)
- val
 - Entries< Cell_Type >, [72](#)
- value
 - Cell< Cell_Type >, [58](#)
- vec_diff
 - geese-bones.hpp, [179](#)
- vec_equal
 - typedefs.hpp, [198](#)
- vec_equal_approx
 - typedefs.hpp, [198](#)
- vec_inner_prod
 - typedefs.hpp, [199](#)
- vecHasher< T >, [142](#)
 - operator(), [142](#)
- vector_caster
 - geese-bones.hpp, [180](#)
- vertex_attr
 - NetworkData, [105](#)
- visited
 - BArray< Cell_Type, Data_Type >, [41](#)
 - Cell< Cell_Type >, [58](#)
 - Node, [111](#)
- zero_col
 - BArray< Cell_Type, Data_Type >, [40](#)
- zero_row
 - BArray< Cell_Type, Data_Type >, [40](#)