

barry: Your go-to motif accountant

0.0-1

Generated by Doxygen 1.9.1



<b>1 Main Page</b>	<b>1</b>
<b>2 Module Index</b>	<b>3</b>
2.1 Modules	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Module Documentation</b>	<b>9</b>
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_prop_genes_changing()	19
5.4.2.15 counter_subfun()	20
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
<b>6 Namespace Documentation</b>	<b>23</b>
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

<b>7 Class Documentation</b>	<b>27</b>
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_vec() [1/2]	32
7.1.3.8 get_col_vec() [2/2]	33
7.1.3.9 get_entries()	33
7.1.3.10 get_row_vec() [1/2]	33
7.1.3.11 get_row_vec() [2/2]	33
7.1.3.12 insert_cell() [1/3]	33
7.1.3.13 insert_cell() [2/3]	34
7.1.3.14 insert_cell() [3/3]	34
7.1.3.15 is_empty()	34
7.1.3.16 ncol()	34
7.1.3.17 nnozero()	34
7.1.3.18 nrow()	35
7.1.3.19 operator>() [1/2]	35
7.1.3.20 operator>() [2/2]	35
7.1.3.21 operator*=( )	35
7.1.3.22 operator+=( ) [1/3]	35
7.1.3.23 operator+=( ) [2/3]	35
7.1.3.24 operator+=( ) [3/3]	36
7.1.3.25 operator-=( ) [1/3]	36
7.1.3.26 operator-=( ) [2/3]	36
7.1.3.27 operator-=( ) [3/3]	36
7.1.3.28 operator/=( )	36
7.1.3.29 operator=( ) [1/2]	36
7.1.3.30 operator=( ) [2/2]	37

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

7.3.3.3 operator<>()	45
7.3.3.4 operator<=>()	45
7.3.3.5 operator==()	45
7.3.3.6 operator>()	45
7.3.3.7 operator>=()	45
7.4 BArrayDense< Cell_Type, Data_Type > Class Template Reference	46
7.4.1 Detailed Description	48
7.4.2 Constructor & Destructor Documentation	48
7.4.2.1 BArrayDense() [1/6]	49
7.4.2.2 BArrayDense() [2/6]	49
7.4.2.3 BArrayDense() [3/6]	49
7.4.2.4 BArrayDense() [4/6]	49
7.4.2.5 BArrayDense() [5/6]	50
7.4.2.6 BArrayDense() [6/6]	50
7.4.2.7 ~BArrayDense()	50
7.4.3 Member Function Documentation	50
7.4.3.1 clear()	50
7.4.3.2 col()	51
7.4.3.3 D() [1/2]	51
7.4.3.4 D() [2/2]	51
7.4.3.5 default_val()	51
7.4.3.6 get_cell()	51
7.4.3.7 get_col_vec() [1/2]	52
7.4.3.8 get_col_vec() [2/2]	52
7.4.3.9 get_entries()	52
7.4.3.10 get_row_vec() [1/2]	52
7.4.3.11 get_row_vec() [2/2]	53
7.4.3.12 insert_cell() [1/3]	53
7.4.3.13 insert_cell() [2/3]	53
7.4.3.14 insert_cell() [3/3]	53
7.4.3.15 is_empty()	54
7.4.3.16 ncol()	54
7.4.3.17 nnozero()	54
7.4.3.18 nrow()	54
7.4.3.19 operator>() [1/2]	54
7.4.3.20 operator>() [2/2]	55
7.4.3.21 operator*=( )	55
7.4.3.22 operator+=( ) [1/3]	55
7.4.3.23 operator+=( ) [2/3]	55
7.4.3.24 operator+=( ) [3/3]	55
7.4.3.25 operator-=( ) [1/3]	56
7.4.3.26 operator-=( ) [2/3]	56

7.4.3.27 operator-=( ) [ 3 / 3 ] . . . . .	56
7.4.3.28 operator/=( ) . . . . .	56
7.4.3.29 operator=( ) [ 1 / 2 ] . . . . .	56
7.4.3.30 operator=( ) [ 2 / 2 ] . . . . .	57
7.4.3.31 operator==( ) . . . . .	57
7.4.3.32 out_of_range( ) . . . . .	57
7.4.3.33 print( ) . . . . .	57
7.4.3.34 reserve( ) . . . . .	57
7.4.3.35 resize( ) . . . . .	58
7.4.3.36 rm_cell( ) . . . . .	58
7.4.3.37 row( ) . . . . .	58
7.4.3.38 set_data( ) . . . . .	58
7.4.3.39 swap_cells( ) . . . . .	59
7.4.3.40 swap_cols( ) . . . . .	59
7.4.3.41 swap_rows( ) . . . . .	59
7.4.3.42 toggle_cell( ) . . . . .	59
7.4.3.43 toggle_lock( ) . . . . .	60
7.4.3.44 transpose( ) . . . . .	60
7.4.3.45 zero_col( ) . . . . .	60
7.4.3.46 zero_row( ) . . . . .	60
7.4.4 Friends And Related Function Documentation . . . . .	60
7.4.4.1 BArrayCell< Cell_Type, Data_Type > . . . . .	60
7.4.4.2 BArrayCell_const< Cell_Type, Data_Type > . . . . .	61
7.4.5 Member Data Documentation . . . . .	61
7.4.5.1 visited . . . . .	61
7.5 BArrayDenseCell< Cell_Type, Data_Type > Class Template Reference . . . . .	61
7.5.1 Detailed Description . . . . .	62
7.5.2 Constructor & Destructor Documentation . . . . .	62
7.5.2.1 BArrayDenseCell( ) . . . . .	62
7.5.2.2 ~BArrayDenseCell( ) . . . . .	62
7.5.3 Member Function Documentation . . . . .	62
7.5.3.1 operator Cell_Type( ) . . . . .	62
7.5.3.2 operator*=( ) . . . . .	63
7.5.3.3 operator+=( ) . . . . .	63
7.5.3.4 operator-=( ) . . . . .	63
7.5.3.5 operator/=( ) . . . . .	63
7.5.3.6 operator=( ) . . . . .	63
7.5.3.7 operator==( ) . . . . .	64
7.6 BArrayDenseCell_const< Cell_Type, Data_Type > Class Template Reference . . . . .	64
7.6.1 Detailed Description . . . . .	64
7.6.2 Constructor & Destructor Documentation . . . . .	64
7.6.2.1 BArrayDenseCell_const( ) . . . . .	65



7.6.2.2 ~BArrayDenseCell_const()	65
7.6.3 Member Function Documentation	65
7.6.3.1 operator Cell_Type()	65
7.6.3.2 operator"!=( )	65
7.6.3.3 operator<( )	65
7.6.3.4 operator<=( )	66
7.6.3.5 operator==( )	66
7.6.3.6 operator>( )	66
7.6.3.7 operator>=( )	66
7.7 BArrayVector< Cell_Type, Data_Type > Class Template Reference	66
7.7.1 Detailed Description	67
7.7.2 Constructor & Destructor Documentation	67
7.7.2.1 BArrayVector()	67
7.7.2.2 ~BArrayVector()	68
7.7.3 Member Function Documentation	68
7.7.3.1 begin()	68
7.7.3.2 end()	68
7.7.3.3 is_col()	68
7.7.3.4 is_row()	69
7.7.3.5 operator std::vector< Cell_Type >()	69
7.7.3.6 operator*=( )	69
7.7.3.7 operator+=( )	69
7.7.3.8 operator-=( )	69
7.7.3.9 operator/=( )	70
7.7.3.10 operator=( )	70
7.7.3.11 operator==( )	70
7.7.3.12 size()	70
7.8 BArrayVector_const< Cell_Type, Data_Type > Class Template Reference	70
7.8.1 Detailed Description	71
7.8.2 Constructor & Destructor Documentation	71
7.8.2.1 BArrayVector_const()	71
7.8.2.2 ~BArrayVector_const()	71
7.8.3 Member Function Documentation	72
7.8.3.1 begin()	72
7.8.3.2 end()	72
7.8.3.3 is_col()	72
7.8.3.4 is_row()	72
7.8.3.5 operator std::vector< Cell_Type >()	72
7.8.3.6 operator"!=( )	73
7.8.3.7 operator<( )	73
7.8.3.8 operator<=( )	73
7.8.3.9 operator==( )	73

7.8.3.10 operator>()	73
7.8.3.11 operator>=()	74
7.8.3.12 size()	74
7.9 Cell< Cell_Type > Class Template Reference	74
7.9.1 Detailed Description	75
7.9.2 Constructor & Destructor Documentation	75
7.9.2.1 Cell() [1/7]	75
7.9.2.2 Cell() [2/7]	75
7.9.2.3 ~Cell()	75
7.9.2.4 Cell() [3/7]	76
7.9.2.5 Cell() [4/7]	76
7.9.2.6 Cell() [5/7]	76
7.9.2.7 Cell() [6/7]	76
7.9.2.8 Cell() [7/7]	76
7.9.3 Member Function Documentation	76
7.9.3.1 add() [1/4]	77
7.9.3.2 add() [2/4]	77
7.9.3.3 add() [3/4]	77
7.9.3.4 add() [4/4]	77
7.9.3.5 operator Cell_Type()	77
7.9.3.6 operator"!=(	77
7.9.3.7 operator=(	78
7.9.3.8 operator=(	78
7.9.3.9 operator==(	78
7.9.4 Member Data Documentation	78
7.9.4.1 value	78
7.9.4.2 visited	78
7.10 ConstBArrayRowIter< Cell_Type, Data_Type > Class Template Reference	79
7.10.1 Detailed Description	79
7.10.2 Constructor & Destructor Documentation	79
7.10.2.1 ConstBArrayRowIter()	80
7.10.2.2 ~ConstBArrayRowIter()	80
7.10.3 Member Data Documentation	80
7.10.3.1 Array	80
7.10.3.2 current_col	80
7.10.3.3 current_row	80
7.10.3.4 iter	81
7.11 Counter< Array_Type, Data_Type > Class Template Reference	81
7.11.1 Detailed Description	82
7.11.2 Constructor & Destructor Documentation	82
7.11.2.1 Counter() [1/4]	82
7.11.2.2 Counter() [2/4]	82

7.11.2.3 Counter() [3/4]	83
7.11.2.4 Counter() [4/4]	83
7.11.2.5 ~Counter()	83
7.11.3 Member Function Documentation	83
7.11.3.1 count()	83
7.11.3.2 init()	84
7.11.3.3 operator=() [1/2]	84
7.11.3.4 operator=() [2/2]	84
7.11.4 Member Data Documentation	84
7.11.4.1 count_fun	84
7.11.4.2 data	85
7.11.4.3 delete_data	85
7.11.4.4 desc	85
7.11.4.5 init_fun	85
7.11.4.6 name	85
7.12 Counters< Array_Type, Data_Type > Class Template Reference	86
7.12.1 Detailed Description	86
7.12.2 Constructor & Destructor Documentation	86
7.12.2.1 Counters() [1/3]	87
7.12.2.2 ~Counters()	87
7.12.2.3 Counters() [2/3]	87
7.12.2.4 Counters() [3/3]	87
7.12.3 Member Function Documentation	88
7.12.3.1 add_counter() [1/3]	88
7.12.3.2 add_counter() [2/3]	88
7.12.3.3 add_counter() [3/3]	88
7.12.3.4 clear()	88
7.12.3.5 operator=() [1/2]	88
7.12.3.6 operator=() [2/2]	89
7.12.3.7 operator[]()	89
7.12.3.8 size()	90
7.13 Entries< Cell_Type > Class Template Reference	90
7.13.1 Detailed Description	90
7.13.2 Constructor & Destructor Documentation	91
7.13.2.1 Entries() [1/2]	91
7.13.2.2 Entries() [2/2]	91
7.13.2.3 ~Entries()	91
7.13.3 Member Function Documentation	91
7.13.3.1 resize()	91
7.13.4 Member Data Documentation	92
7.13.4.1 source	92
7.13.4.2 target	92

7.13.4.3 val	92
7.14 Flock Class Reference	92
7.14.1 Detailed Description	93
7.14.2 Constructor & Destructor Documentation	93
7.14.2.1 Flock()	94
7.14.2.2 ~Flock()	94
7.14.3 Member Function Documentation	94
7.14.3.1 add_data()	94
7.14.3.2 colnames()	94
7.14.3.3 get_counters()	95
7.14.3.4 get_support()	95
7.14.3.5 init()	95
7.14.3.6 likelihood_joint()	95
7.14.3.7 nfuncs()	96
7.14.3.8 nleafs()	96
7.14.3.9 nnodes()	96
7.14.3.10 nterms()	96
7.14.3.11 ntrees()	96
7.14.3.12 operator()()	96
7.14.3.13 parse_polytomies()	97
7.14.3.14 set_seed()	97
7.14.3.15 support_size()	97
7.14.4 Member Data Documentation	97
7.14.4.1 dat	98
7.14.4.2 initialized	98
7.14.4.3 model	98
7.14.4.4 nfunctions	98
7.14.4.5 rengine	98
7.15 FreqTable< T > Class Template Reference	98
7.15.1 Detailed Description	99
7.15.2 Constructor & Destructor Documentation	99
7.15.2.1 FreqTable()	99
7.15.2.2 ~FreqTable()	99
7.15.3 Member Function Documentation	99
7.15.3.1 add()	100
7.15.3.2 as_vector()	100
7.15.3.3 clear()	100
7.15.3.4 get_data()	100
7.15.3.5 get_data_ptr()	100
7.15.3.6 print()	101
7.15.3.7 reserve()	101
7.15.3.8 size()	101

7.16 Geese Class Reference	101
7.16.1 Detailed Description	103
7.16.2 Constructor & Destructor Documentation	103
7.16.2.1 Geese() [1/4]	104
7.16.2.2 Geese() [2/4]	104
7.16.2.3 Geese() [3/4]	104
7.16.2.4 Geese() [4/4]	104
7.16.2.5 ~Geese()	104
7.16.3 Member Function Documentation	104
7.16.3.1 calc_reduced_sequence()	105
7.16.3.2 calc_sequence()	105
7.16.3.3 colnames()	105
7.16.3.4 get_annotated_nodes()	105
7.16.3.5 get_counters()	105
7.16.3.6 get_model()	105
7.16.3.7 get_probabilities()	106
7.16.3.8 get_rengine()	106
7.16.3.9 get_states()	106
7.16.3.10 get_support()	106
7.16.3.11 inherit_support()	106
7.16.3.12 init()	107
7.16.3.13 init_node()	107
7.16.3.14 likelihood()	107
7.16.3.15 likelihood_exhaust()	107
7.16.3.16 nfuncs()	107
7.16.3.17 nleafs()	108
7.16.3.18 nnodes()	108
7.16.3.19 nterms()	108
7.16.3.20 observed_counts()	108
7.16.3.21 operator=() [1/2]	108
7.16.3.22 operator=() [2/2]	108
7.16.3.23 parse_polytomies()	109
7.16.3.24 predict()	109
7.16.3.25 predict_backend()	109
7.16.3.26 predict_exhaust()	109
7.16.3.27 predict_exhaust_backend()	109
7.16.3.28 predict_sim()	110
7.16.3.29 print_observed_counts()	110
7.16.3.30 set_seed()	110
7.16.3.31 simulate()	110
7.16.3.32 support_size()	110
7.16.3.33 update_annotations()	111

7.16.4 Member Data Documentation	111
7.16.4.1 delete_engine	111
7.16.4.2 delete_support	111
7.16.4.3 initialized	111
7.16.4.4 map_to_nodes	111
7.16.4.5 nfunctions	112
7.16.4.6 nodes	112
7.16.4.7 reduced_sequence	112
7.16.4.8 sequence	112
7.17 Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference	112
7.17.1 Detailed Description	114
7.17.2 Constructor & Destructor Documentation	115
7.17.2.1 Model() [1/3]	115
7.17.2.2 Model() [2/3]	115
7.17.2.3 Model() [3/3]	115
7.17.2.4 ~Model()	116
7.17.3 Member Function Documentation	116
7.17.3.1 add_array()	116
7.17.3.2 add_counter() [1/3]	116
7.17.3.3 add_counter() [2/3]	117
7.17.3.4 add_counter() [3/3]	117
7.17.3.5 add_rule() [1/3]	117
7.17.3.6 add_rule() [2/3]	117
7.17.3.7 add_rule() [3/3]	118
7.17.3.8 add_rule_dyn() [1/3]	118
7.17.3.9 add_rule_dyn() [2/3]	118
7.17.3.10 add_rule_dyn() [3/3]	118
7.17.3.11 colnames()	119
7.17.3.12 get_counters()	119
7.17.3.13 get_norm_const()	119
7.17.3.14 get_pset()	119
7.17.3.15 get_engine()	120
7.17.3.16 get_rules()	120
7.17.3.17 get_rules_dyn()	120
7.17.3.18 get_stats()	120
7.17.3.19 get_support()	120
7.17.3.20 likelihood() [1/3]	121
7.17.3.21 likelihood() [2/3]	121
7.17.3.22 likelihood() [3/3]	121
7.17.3.23 likelihood_total()	121
7.17.3.24 nterms()	122

7.17.3.25 operator=()	122
7.17.3.26 print_stats()	122
7.17.3.27 sample() [1/2]	122
7.17.3.28 sample() [2/2]	123
7.17.3.29 set_counters()	123
7.17.3.30 set_keygen()	123
7.17.3.31 set_rengine()	123
7.17.3.32 set_rules()	124
7.17.3.33 set_rules_dyn()	124
7.17.3.34 set_seed()	124
7.17.3.35 size()	124
7.17.3.36 size_unique()	124
7.17.3.37 store_psets()	125
7.17.3.38 support_size()	125
7.18 NetCounterData Class Reference	125
7.18.1 Detailed Description	125
7.18.2 Constructor & Destructor Documentation	126
7.18.2.1 NetCounterData() [1/2]	126
7.18.2.2 NetCounterData() [2/2]	126
7.18.2.3 ~NetCounterData()	126
7.18.3 Member Data Documentation	126
7.18.3.1 indices	126
7.18.3.2 numbers	126
7.19 NetworkData Class Reference	127
7.19.1 Detailed Description	127
7.19.2 Constructor & Destructor Documentation	127
7.19.2.1 NetworkData() [1/3]	127
7.19.2.2 NetworkData() [2/3]	127
7.19.2.3 NetworkData() [3/3]	128
7.19.2.4 ~NetworkData()	128
7.19.3 Member Data Documentation	128
7.19.3.1 directed	128
7.19.3.2 vertex_attr	129
7.20 Node Class Reference	129
7.20.1 Detailed Description	130
7.20.2 Constructor & Destructor Documentation	130
7.20.2.1 Node() [1/5]	130
7.20.2.2 Node() [2/5]	131
7.20.2.3 Node() [3/5]	131
7.20.2.4 Node() [4/5]	131
7.20.2.5 Node() [5/5]	131
7.20.2.6 ~Node()	131

7.20.3 Member Function Documentation	131
7.20.3.1 get_parent()	132
7.20.3.2 is_leaf()	132
7.20.3.3 noffspring()	132
7.20.4 Member Data Documentation	132
7.20.4.1 annotations	132
7.20.4.2 array	132
7.20.4.3 arrays	133
7.20.4.4 duplication	133
7.20.4.5 id	133
7.20.4.6 narray	133
7.20.4.7 offspring	133
7.20.4.8 ord	134
7.20.4.9 parent	134
7.20.4.10 probability	134
7.20.4.11 subtree_prob	134
7.20.4.12 visited	134
7.21 NodeData Class Reference	135
7.21.1 Detailed Description	135
7.21.2 Constructor & Destructor Documentation	135
7.21.2.1 NodeData()	135
7.21.2.2 ~NodeData()	135
7.21.3 Member Data Documentation	136
7.21.3.1 blengths	136
7.21.3.2 duplication	136
7.21.3.3 states	136
7.22 PhyloRuleDynData Class Reference	136
7.22.1 Detailed Description	137
7.22.2 Constructor & Destructor Documentation	137
7.22.2.1 PhyloRuleDynData()	137
7.22.2.2 ~PhyloRuleDynData()	137
7.22.3 Member Data Documentation	137
7.22.3.1 counts	137
7.22.3.2 duplication	138
7.22.3.3 lb	138
7.22.3.4 pos	138
7.22.3.5 ub	138
7.23 PowerSet< Array_Type, Data_Rule_Type > Class Template Reference	138
7.23.1 Detailed Description	139
7.23.2 Constructor & Destructor Documentation	140
7.23.2.1 PowerSet() [1/3]	140
7.23.2.2 PowerSet() [2/3]	140



7.23.2.3 PowerSet() [3/3]	140
7.23.2.4 ~PowerSet()	140
7.23.3 Member Function Documentation	140
7.23.3.1 add_rule() [1/3]	141
7.23.3.2 add_rule() [2/3]	141
7.23.3.3 add_rule() [3/3]	141
7.23.3.4 begin()	141
7.23.3.5 calc()	141
7.23.3.6 end()	142
7.23.3.7 get_data()	142
7.23.3.8 get_data_ptr()	142
7.23.3.9 init_support()	142
7.23.3.10 operator[]()	142
7.23.3.11 reset()	143
7.23.3.12 size()	143
7.23.4 Member Data Documentation	143
7.23.4.1 coordinates_free	143
7.23.4.2 coordinates_locked	143
7.23.4.3 data	143
7.23.4.4 EmptyArray	144
7.23.4.5 M	144
7.23.4.6 N	144
7.23.4.7 rules	144
7.23.4.8 rules_deleted	144
7.24 Rule< Array_Type, Data_Type > Class Template Reference	145
7.24.1 Detailed Description	145
7.24.2 Constructor & Destructor Documentation	145
7.24.2.1 Rule() [1/2]	146
7.24.2.2 Rule() [2/2]	146
7.24.2.3 ~Rule()	146
7.24.3 Member Function Documentation	146
7.24.3.1 D()	146
7.24.3.2 operator()()	146
7.25 Rules< Array_Type, Data_Type > Class Template Reference	147
7.25.1 Detailed Description	147
7.25.2 Constructor & Destructor Documentation	147
7.25.2.1 Rules() [1/2]	148
7.25.2.2 Rules() [2/2]	148
7.25.2.3 ~Rules()	148
7.25.3 Member Function Documentation	148
7.25.3.1 add_rule() [1/3]	148
7.25.3.2 add_rule() [2/3]	148

7.25.3.3 add_rule() [3/3]	149
7.25.3.4 clear()	149
7.25.3.5 get_seq()	149
7.25.3.6 operator()()	149
7.25.3.7 operator=()	151
7.25.3.8 size()	151
7.26 StatsCounter< Array_Type, Data_Type > Class Template Reference	151
7.26.1 Detailed Description	152
7.26.2 Constructor & Destructor Documentation	152
7.26.2.1 StatsCounter() [1/2]	152
7.26.2.2 StatsCounter() [2/2]	153
7.26.2.3 ~StatsCounter()	153
7.26.3 Member Function Documentation	153
7.26.3.1 add_counter() [1/2]	153
7.26.3.2 add_counter() [2/2]	153
7.26.3.3 count_all()	154
7.26.3.4 count_current()	154
7.26.3.5 count_init()	154
7.26.3.6 get_counters()	154
7.26.3.7 reset_array()	154
7.26.3.8 set_counters()	155
7.27 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference	155
7.27.1 Detailed Description	158
7.27.2 Constructor & Destructor Documentation	158
7.27.2.1 Support() [1/3]	158
7.27.2.2 Support() [2/3]	158
7.27.2.3 Support() [3/3]	159
7.27.2.4 ~Support()	159
7.27.3 Member Function Documentation	159
7.27.3.1 add_counter() [1/2]	159
7.27.3.2 add_counter() [2/2]	159
7.27.3.3 add_rule() [1/2]	160
7.27.3.4 add_rule() [2/2]	160
7.27.3.5 add_rule_dyn() [1/2]	160
7.27.3.6 add_rule_dyn() [2/2]	160
7.27.3.7 calc()	160
7.27.3.8 get_counters()	161
7.27.3.9 get_counts()	161
7.27.3.10 get_counts_ptr()	161
7.27.3.11 get_current_stats()	161
7.27.3.12 get_data()	162

7.27.3.13 <a href="#">get_rules()</a>	162
7.27.3.14 <a href="#">get_rules_dyn()</a>	162
7.27.3.15 <a href="#">init_support()</a>	162
7.27.3.16 <a href="#">print()</a>	163
7.27.3.17 <a href="#">reset_array()</a> [1/2]	163
7.27.3.18 <a href="#">reset_array()</a> [2/2]	163
7.27.3.19 <a href="#">set_counters()</a>	163
7.27.3.20 <a href="#">set_rules()</a>	163
7.27.3.21 <a href="#">set_rules_dyn()</a>	164
7.27.4 Member Data Documentation	164
7.27.4.1 <a href="#">change_stats</a>	164
7.27.4.2 <a href="#">coordinates_free</a>	164
7.27.4.3 <a href="#">coordinates_locked</a>	164
7.27.4.4 <a href="#">current_stats</a>	165
7.27.4.5 <a href="#">delete_counters</a>	165
7.27.4.6 <a href="#">delete_rules</a>	165
7.27.4.7 <a href="#">delete_rules_dyn</a>	165
7.27.4.8 <a href="#">M</a>	165
7.27.4.9 <a href="#">max_num_elements</a>	166
7.27.4.10 <a href="#">N</a>	166
7.28 <a href="#">vecHasher&lt; T &gt; Struct Template Reference</a>	166
7.28.1 Detailed Description	166
7.28.2 Member Function Documentation	166
7.28.2.1 <a href="#">operator()()</a>	166
<b>8 File Documentation</b>	<b>167</b>
8.1 <a href="#">include/barry/barray-bones.hpp File Reference</a>	167
8.1.1 Macro Definition Documentation	168
8.1.1.1 <a href="#">BARRAY_BONES_HPP</a>	168
8.2 <a href="#">include/barry/barray-iterator.hpp File Reference</a>	168
8.3 <a href="#">include/barry/barray-meat-operators.hpp File Reference</a>	169
8.3.1 Macro Definition Documentation	170
8.3.1.1 <a href="#">BARRY_BARRAY_MEAT_OPERATORS_HPP</a>	170
8.3.1.2 <a href="#">COL</a>	170
8.3.1.3 <a href="#">ROW</a>	170
8.3.2 Function Documentation	170
8.3.2.1 <a href="#">checkdim_()</a>	170
8.4 <a href="#">include/barry/barray-meat.hpp File Reference</a>	171
8.4.1 Macro Definition Documentation	172
8.4.1.1 <a href="#">COL</a>	172
8.4.1.2 <a href="#">ROW</a>	172
8.5 <a href="#">include/barry/barraycell-bones.hpp File Reference</a>	172

8.6 include/barry/barraycell-meat.hpp File Reference . . . . .	173
8.7 include/barry/barraydense-bones.hpp File Reference . . . . .	174
8.8 include/barry/barraydense-meet.hpp File Reference . . . . .	175
8.8.1 Macro Definition Documentation . . . . .	176
8.8.1.1 BARRY_BARRAYDENSE_MEAT_HPP . . . . .	176
8.8.1.2 COL . . . . .	177
8.8.1.3 POS . . . . .	177
8.8.1.4 ROW . . . . .	177
8.8.1.5 ZERO_CELL . . . . .	177
8.9 include/barry/barraydensecell-bones.hpp File Reference . . . . .	177
8.10 include/barry/barraydensecell-meat.hpp File Reference . . . . .	178
8.10.1 Macro Definition Documentation . . . . .	179
8.10.1.1 BARRY_BARRAYDENSECELL_MEAT_HPP . . . . .	179
8.10.1.2 POS . . . . .	179
8.11 include/barry/barrayvector-bones.hpp File Reference . . . . .	179
8.12 include/barry/barrayvector-meat.hpp File Reference . . . . .	180
8.12.1 Macro Definition Documentation . . . . .	181
8.12.1.1 BARRY_BARRAYVECTOR_MEAT_HPP . . . . .	181
8.13 include/barry/barry-configuration.hpp File Reference . . . . .	181
8.13.1 Macro Definition Documentation . . . . .	182
8.13.1.1 BARRY_CHECK_SUPPORT . . . . .	182
8.13.1.2 BARRY_ISFINITE . . . . .	182
8.13.1.3 BARRY_MAX_NUM_ELEMENTS . . . . .	182
8.13.1.4 BARRY_SAFE_EXP . . . . .	182
8.13.1.5 printf_barry . . . . .	182
8.13.2 Typedef Documentation . . . . .	182
8.13.2.1 Map . . . . .	183
8.14 include/barry/barry.hpp File Reference . . . . .	183
8.14.1 Macro Definition Documentation . . . . .	184
8.14.1.1 BARRY_HPP . . . . .	184
8.14.1.2 BARRY_VERSION . . . . .	184
8.14.1.3 COUNTER_FUNCTION . . . . .	184
8.14.1.4 COUNTER_LAMBDA . . . . .	185
8.14.1.5 RULE_FUNCTION . . . . .	185
8.14.1.6 RULE_LAMBDA . . . . .	185
8.15 include/barry/cell-bones.hpp File Reference . . . . .	185
8.16 include/barry/cell-meat.hpp File Reference . . . . .	186
8.17 include/barry/col-bones.hpp File Reference . . . . .	187
8.18 include/barry/counters-bones.hpp File Reference . . . . .	187
8.19 include/barry/counters-meat.hpp File Reference . . . . .	188
8.20 include/barry/counters/network.hpp File Reference . . . . .	189
8.20.1 Macro Definition Documentation . . . . .	192

8.20.1.1 NET_C_DATA_IDX . . . . .	192
8.20.1.2 NET_C_DATA_NUM . . . . .	192
8.20.1.3 NETWORK_COUNTER . . . . .	192
8.20.1.4 NETWORK_COUNTER_LAMBDA . . . . .	193
8.20.1.5 NETWORK_RULE . . . . .	193
8.20.1.6 NETWORK_RULE_LAMBDA . . . . .	193
8.20.2 Typedef Documentation . . . . .	193
8.20.2.1 NetCounter . . . . .	193
8.20.2.2 NetCounters . . . . .	194
8.20.2.3 NetModel . . . . .	194
8.20.2.4 NetRule . . . . .	194
8.20.2.5 NetRules . . . . .	194
8.20.2.6 NetStatsCounter . . . . .	194
8.20.2.7 NetSupport . . . . .	194
8.20.2.8 Network . . . . .	195
8.20.3 Function Documentation . . . . .	195
8.20.3.1 rules_zerodiag() . . . . .	195
8.21 include/barry/counters/phylo.hpp File Reference . . . . .	195
8.21.1 Macro Definition Documentation . . . . .	197
8.21.1.1 PHYLO_CHECK_MISSING . . . . .	197
8.21.1.2 PHYLO_COUNTER_LAMBDA . . . . .	198
8.21.1.3 PHYLO_RULE_DYN_LAMBDA . . . . .	198
8.21.2 Typedef Documentation . . . . .	198
8.21.2.1 PhyloArray . . . . .	198
8.21.2.2 PhyloCounter . . . . .	198
8.21.2.3 PhyloCounterData . . . . .	199
8.21.2.4 PhyloCounters . . . . .	199
8.21.2.5 PhyloModel . . . . .	199
8.21.2.6 PhyloPowerSet . . . . .	199
8.21.2.7 PhyloRule . . . . .	199
8.21.2.8 PhyloRuleData . . . . .	199
8.21.2.9 PhyloRuleDyn . . . . .	200
8.21.2.10 PhyloRules . . . . .	200
8.21.2.11 PhyloRulesDyn . . . . .	200
8.21.2.12 PhyloStatsCounter . . . . .	200
8.21.2.13 PhyloSupport . . . . .	200
8.21.3 Function Documentation . . . . .	200
8.21.3.1 get_last_name() . . . . .	200
8.22 include/barry/model-bones.hpp File Reference . . . . .	201
8.22.1 Function Documentation . . . . .	202
8.22.1.1 keygen_default() . . . . .	202
8.22.1.2 likelihood_() . . . . .	202

8.22.1.3 <code>update_normalizing_constant()</code> . . . . .	202
8.23 <code>include/barry/model-meat.hpp</code> File Reference . . . . .	203
8.24 <code>include/barry/models/geese.hpp</code> File Reference . . . . .	203
8.25 <code>include/barry/models/geese/flock-bones.hpp</code> File Reference . . . . .	204
8.26 <code>include/barry/models/geese/flock-meet.hpp</code> File Reference . . . . .	204
8.27 <code>include/barry/models/geese/geese-bones.hpp</code> File Reference . . . . .	205
8.27.1 Macro Definition Documentation . . . . .	205
8.27.1.1 <code>INITIALIZED</code> . . . . .	206
8.27.2 Function Documentation . . . . .	206
8.27.2.1 <code>keygen_full()</code> . . . . .	206
8.27.2.2 <code>RULE_FUNCTION()</code> . . . . .	206
8.27.2.3 <code>vec_diff()</code> . . . . .	206
8.27.2.4 <code>vector_caster()</code> . . . . .	206
8.28 <code>include/barry/models/geese/geese-meat-constructors.hpp</code> File Reference . . . . .	207
8.29 <code>include/barry/models/geese/geese-meat-likelihood.hpp</code> File Reference . . . . .	207
8.30 <code>include/barry/models/geese/geese-meat-likelihood_exhaust.hpp</code> File Reference . . . . .	208
8.31 <code>include/barry/models/geese/geese-meat-predict.hpp</code> File Reference . . . . .	209
8.32 <code>include/barry/models/geese/geese-meat-predict_exhaust.hpp</code> File Reference . . . . .	209
8.33 <code>include/barry/models/geese/geese-meat-predict_sim.hpp</code> File Reference . . . . .	210
8.34 <code>include/barry/models/geese/geese-meat-simulate.hpp</code> File Reference . . . . .	210
8.35 <code>include/barry/models/geese/geese-meat.hpp</code> File Reference . . . . .	211
8.36 <code>include/barry/models/geese/geese-node-bones.hpp</code> File Reference . . . . .	211
8.37 <code>include/barry/powerset-bones.hpp</code> File Reference . . . . .	212
8.38 <code>include/barry/powerset-meat.hpp</code> File Reference . . . . .	213
8.39 <code>include/barry/rules-bones.hpp</code> File Reference . . . . .	215
8.39.1 Function Documentation . . . . .	216
8.39.1.1 <code>rule_fun_default()</code> . . . . .	216
8.40 <code>include/barry/rules-meat.hpp</code> File Reference . . . . .	216
8.41 <code>include/barry/statscounter-bones.hpp</code> File Reference . . . . .	217
8.42 <code>include/barry/statscounter-meat.hpp</code> File Reference . . . . .	218
8.43 <code>include/barry/statsdb.hpp</code> File Reference . . . . .	219
8.44 <code>include/barry/support-bones.hpp</code> File Reference . . . . .	220
8.45 <code>include/barry/support-meat.hpp</code> File Reference . . . . .	222
8.45.1 Macro Definition Documentation . . . . .	223
8.45.1.1 <code>BARRY_SUPPORT_MEAT_HPP</code> . . . . .	223
8.46 <code>include/barry/typedefs.hpp</code> File Reference . . . . .	223
8.46.1 Typedef Documentation . . . . .	225
8.46.1.1 <code>Col_type</code> . . . . .	225
8.46.1.2 <code>Counter_fun_type</code> . . . . .	225
8.46.1.3 <code>Counts_type</code> . . . . .	225
8.46.1.4 <code>MapVec_type</code> . . . . .	225
8.46.1.5 <code>Row_type</code> . . . . .	226

---

8.46.1.6 Rule_fun_type . . . . .	226
8.46.1.7 uint . . . . .	226
8.46.2 Function Documentation . . . . .	226
8.46.2.1 vec_equal() . . . . .	226
8.46.2.2 vec_equal_approx() . . . . .	227
8.46.2.3 vec_inner_prod() . . . . .	227
8.47 README.md File Reference . . . . .	227
<b>Index</b>	<b>229</b>





# 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:

<a href="#">BArray&lt; Cell_Type, Data_Type &gt;</a>	
Baseline class for binary arrays	27
<a href="#">BArrayCell&lt; Cell_Type, Data_Type &gt;</a>	41
<a href="#">BArrayCell_const&lt; Cell_Type, Data_Type &gt;</a>	43
<a href="#">BArrayDense&lt; Cell_Type, Data_Type &gt;</a>	
Baseline class for binary arrays	46
<a href="#">BArrayDenseCell&lt; Cell_Type, Data_Type &gt;</a>	61
<a href="#">BArrayDenseCell_const&lt; Cell_Type, Data_Type &gt;</a>	64
<a href="#">BArrayVector&lt; Cell_Type, Data_Type &gt;</a>	
Row or column of a <a href="#">BArray</a>	66
<a href="#">BArrayVector_const&lt; Cell_Type, Data_Type &gt;</a>	70
<a href="#">Cell&lt; Cell_Type &gt;</a>	
Entries in <a href="#">BArray</a> . For now, it only has two members:	74
<a href="#">ConstBArrayRowIter&lt; Cell_Type, Data_Type &gt;</a>	79
<a href="#">Counter&lt; Array_Type, Data_Type &gt;</a>	
A counter function based on change statistics	81
<a href="#">Counters&lt; Array_Type, Data_Type &gt;</a>	
Vector of counters	86
<a href="#">Entries&lt; Cell_Type &gt;</a>	
A wrapper class to store <code>source</code> , <code>target</code> , <code>val</code> from a <a href="#">BArray</a> object	90
<a href="#">Flock</a>	
A <a href="#">Flock</a> is a group of <a href="#">Geese</a>	92
<a href="#">FreqTable&lt; T &gt;</a>	
Database of statistics	98
<a href="#">Geese</a>	
Annotated Phylo <a href="#">Model</a>	101
<a href="#">Model&lt; Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type &gt;</a>	
General framework for discrete exponential models. This class allows generating discrete exponential models in the form of a linear exponential model:	112
<a href="#">NetCounterData</a>	
Data class used to store arbitrary uint or double vectors	125
<a href="#">NetworkData</a>	
Data class for Networks	127
<a href="#">Node</a>	
A single node for the model	129

<a href="#">NodeData</a>	
Data definition for the <code>PhyloArray</code> class	135
<a href="#">PhyloRuleDynData</a>	136
<a href="#">PowerSet&lt; Array_Type, Data_Rule_Type &gt;</a>	
Powerset of a binary array	138
<a href="#">Rule&lt; Array_Type, Data_Type &gt;</a>	
Rule for determining if a cell should be included in a sequence	145
<a href="#">Rules&lt; Array_Type, Data_Type &gt;</a>	
Vector of objects of class <code>Rule</code>	147
<a href="#">StatsCounter&lt; Array_Type, Data_Type &gt;</a>	
Count stats for a single Array	151
<a href="#">Support&lt; Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type &gt;</a>	
Compute the support of sufficient statistics	155
<a href="#">vecHasher&lt; T &gt;</a>	166

## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

include/barry/barray-bones.hpp . . . . .	167
include/barry/barray-iterator.hpp . . . . .	168
include/barry/barray-meat-operators.hpp . . . . .	169
include/barry/barray-meat.hpp . . . . .	171
include/barry/barraycell-bones.hpp . . . . .	172
include/barry/barraycell-meat.hpp . . . . .	173
include/barry/barraydense-bones.hpp . . . . .	174
include/barry/barraydense-meat.hpp . . . . .	175
include/barry/barraydensecell-bones.hpp . . . . .	177
include/barry/barraydensecell-meat.hpp . . . . .	178
include/barry/barrayvector-bones.hpp . . . . .	179
include/barry/barrayvector-meat.hpp . . . . .	180
include/barry/barray-configuration.hpp . . . . .	181
include/barry/barray.hpp . . . . .	183
include/barry/cell-bones.hpp . . . . .	185
include/barry/cell-meat.hpp . . . . .	186
include/barry/col-bones.hpp . . . . .	187
include/barry/counters-bones.hpp . . . . .	187
include/barry/counters-meat.hpp . . . . .	188
include/barry/model-bones.hpp . . . . .	201
include/barry/model-meat.hpp . . . . .	203
include/barry/powerset-bones.hpp . . . . .	212
include/barry/powerset-meat.hpp . . . . .	213
include/barry/rules-bones.hpp . . . . .	215
include/barry/rules-meat.hpp . . . . .	216
include/barry/statscounter-bones.hpp . . . . .	217
include/barry/statscounter-meat.hpp . . . . .	218
include/barry/statsdb.hpp . . . . .	219
include/barry/support-bones.hpp . . . . .	220
include/barry/support-meat.hpp . . . . .	222
include/barry/typedefs.hpp . . . . .	223
include/barry/counters/network.hpp . . . . .	189
include/barry/counters/phylo.hpp . . . . .	195
include/barry/models/geese.hpp . . . . .	203
include/barry/models/geese/flock-bones.hpp . . . . .	204

include/barry/models/geese/flock-meet.hpp . . . . .	204
include/barry/models/geese/geese-bones.hpp . . . . .	205
include/barry/models/geese/geese-meat-constructors.hpp . . . . .	207
include/barry/models/geese/geese-meat-likelihood.hpp . . . . .	207
include/barry/models/geese/geese-meat-likelihood_exhaust.hpp . . . . .	208
include/barry/models/geese/geese-meat-predict.hpp . . . . .	209
include/barry/models/geese/geese-meat-predict_exhaust.hpp . . . . .	209
include/barry/models/geese/geese-meat-predict_sim.hpp . . . . .	210
include/barry/models/geese/geese-meat-simulate.hpp . . . . .	210
include/barry/models/geese/geese-meat.hpp . . . . .	211
include/barry/models/geese/geese-node-bones.hpp . . . . .	211



## 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\\_nodcov](#) ([NetCounters](#) \*counters, [uint](#) attr\_id)
- void [counter\\_nodematch](#) ([NetCounters](#) \*counters, [uint](#) attr\_id)
- void [counter\\_idegree](#) ([NetCounters](#) \*counters, std::vector< [uint](#) > d)  
*Counts number of vertices with a given in-degree.*
- void [counter\\_odegree](#) ([NetCounters](#) \*counters, std::vector< [uint](#) > d)  
*Counts number of vertices with a given out-degree.*
- void [counter\\_degree](#) ([NetCounters](#) \*counters, std::vector< [uint](#) > d)  
*Counts number of vertices with a given out-degree.*

### 5.3.1 Detailed Description

[Counters](#) for network models.

## Parameters

<i>counters</i>	A pointer to a <code>NetCounters</code> object ( <code>Counters&lt;Network, NetCounterData&gt;</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\_iddegree()

```
void counter_iddegree (
    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\_iddegree15()

```
void counter_iddegree15 (
    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\\_prop\\_genes\\_changing](#) ([PhyloCounters](#) \*counters, bool duplication=true)  
*Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)*
- void [counter\\_overall\\_loss](#) ([PhyloCounters](#) \*counters, bool duplication=true)  
*Overall functional loss.*
- void [counter\\_maxfuns](#) ([PhyloCounters](#) \*counters, [uint](#) lb, [uint](#) ub, bool duplication=true)  
*Cap the number of functions per gene.*
- void [counter\\_loss](#) ([PhyloCounters](#) \*counters, std::vector< [uint](#) > nfun, bool duplication=true)  
*Total count of losses for an specific function.*
- void [counter\\_overall\\_changes](#) ([PhyloCounters](#) \*counters, bool duplication=true)  
*Total number of changes. Use this statistic to account for "preservation".*
- void [counter\\_subfun](#) ([PhyloCounters](#) \*counters, [uint](#) nfunA, [uint](#) nfunB, bool duplication=true)  
*Total count of Sub-functionalization events.*
- void [counter\\_cogain](#) ([PhyloCounters](#) \*counters, [uint](#) nfunA, [uint](#) nfunB, bool duplication=true)  
*Co-evolution (joint gain or loss)*
- void [counter\\_longest](#) ([PhyloCounters](#) \*counters)  
*Longest branch mutates (either by gain or by loss)*
- void [counter\\_neofun](#) ([PhyloCounters](#) \*counters, [uint](#) nfunA, [uint](#) nfunB, bool duplication=true)  
*Total number of neofunctionalization events.*
- void [counter\\_neofun\\_a2b](#) ([PhyloCounters](#) \*counters, [uint](#) nfunA, [uint](#) nfunB, bool duplication=true)  
*Total number of neofunctionalization events.*
- void [counter\\_co\\_opt](#) ([PhyloCounters](#) \*counters, [uint](#) nfunA, [uint](#) nfunB, bool duplication=true)  
*Function co-opting.*

### 5.4.1 Detailed Description

[Counters](#) for phylogenetic modeling.

## Parameters

<i>counters</i>	A pointer to a <code>PhyloCounters</code> object ( <code>Counters&lt;PhyloArray, PhyloCounterData&gt;</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 1203 of file phylo.hpp.

### 5.4.2.2 counter\_cogain()

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

Co-evolution (joint gain or loss)

Needs to specify pairs of functions (nfunA, nfunB).

Definition at line 833 of file phylo.hpp.



### 5.4.2.3 counter\_gains()

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

Functional gains for a specific function (nfun).

Definition at line 152 of file phylo.hpp.

### 5.4.2.4 counter\_gains\_k\_offspring()

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

k genes gain function nfun

Definition at line 194 of file phylo.hpp.

### 5.4.2.5 counter\_genes\_changing()

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

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

Definition at line 268 of file phylo.hpp.

### 5.4.2.6 counter\_longest()

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

Longest branch mutates (either by gain or by loss)

Definition at line 892 of file phylo.hpp.

#### 5.4.2.7 counter\_loss()

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

Total count of losses for an specific function.

Definition at line 633 of file phylo.hpp.

#### 5.4.2.8 counter\_maxfuns()

```
void counter_maxfuns (
    PhyloCounters * counters,
    uint lb,
    uint ub,
    bool duplication = true ) [inline]
```

Cap the number of functions per gene.

Definition at line 549 of file phylo.hpp.

#### 5.4.2.9 counter\_neofun()

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

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 997 of file phylo.hpp.

#### 5.4.2.10 counter\_neofun\_a2b()

```
void counter_neofun_a2b (
    PhyloCounters * counters,
    uint nfunA,
    uint nfunB,
    bool duplication = true ) [inline]
```

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 1082 of file phylo.hpp.

#### 5.4.2.11 counter\_overall\_changes()

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

Total number of changes. Use this statistic to account for "preservation".

Definition at line 680 of file phylo.hpp.

#### 5.4.2.12 counter\_overall\_gains()

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

Overall functional gains.

Total number of gains (irrespective of the function).

Definition at line 112 of file phylo.hpp.

#### 5.4.2.13 counter\_overall\_loss()

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

Overall functional loss.

Definition at line 503 of file phylo.hpp.

#### 5.4.2.14 counter\_prop\_genes\_changing()

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

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

Definition at line 386 of file phylo.hpp.

#### 5.4.2.15 counter\_subfun()

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

Total count of Sub-functionalization events.

It requires to specify data = {funA, funB}

Definition at line 747 of file phylo.hpp.

## 5.5 Phylo rules

[Rules](#) for phylogenetic modeling.

### Classes

- class [PhyloRuleDynData](#)

### Functions

- void [rule\\_dyn\\_limit\\_changes](#) ([PhyloSupport](#) \*support, [uint](#) pos, [uint](#) lb, [uint](#) ub, bool duplication=true)  
*Overall functional gains.*

#### 5.5.1 Detailed Description

[Rules](#) for phylogenetic modeling.

##### Parameters

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

#### 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 1336 of file phylo.hpp.



## Chapter 6

# Namespace Documentation

### 6.1 `barry` Namespace Reference

`barry`: Your go-to motif accountant

#### Namespaces

- [counters](#)

*Tree class and 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
- std::vector< Cell\_Type > `get_col_vec` (uint i, bool check\_bounds=true) const
- std::vector< Cell\_Type > `get_row_vec` (uint i, bool check\_bounds=true) const
- void `get_col_vec` (std::vector< Cell\_Type > \*x, uint i, bool check\_bounds=true) const
- void `get_row_vec` (std::vector< Cell\_Type > \*x, uint i, bool check\_bounds=true) const
- const `Row_type`< Cell\_Type > & `row` (uint i, bool check\_bounds=true) const
- const `Col_type`< Cell\_Type > & `col` (uint i, bool check\_bounds=true) const
- `Entries`< Cell\_Type > `get_entries` () const

*Get the edgelist.*

- void `transpose` ()
- void `clear` (bool hard=true)
- void `resize` (uint N\_, uint M\_)
- void `reserve` ()
- void `print` (const char \*fmt=nullptr,...) const

#### Constructors

##### Parameters

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

- **BArray** ()  
*Zero-size array.*
- **BArray** (uint N\_, uint M\_)  
*Empty array.*
- **BArray** (uint N\_, uint M\_, const std::vector< uint > &source, const std::vector< uint > &target, const std::vector< Cell\_Type > &value, bool add=true)  
*Edgelist with data.*
- **BArray** (uint N\_, uint M\_, const std::vector< uint > &source, const std::vector< uint > &target, bool add=true)  
*Edgelist with no data (simpler)*
- **BArray** (const **BArray**< Cell\_Type, Data\_Type > &Array\_, bool copy\_data=false)  
*Copy constructor.*
- **BArray**< Cell\_Type, Data\_Type > & **operator=** (const **BArray**< Cell\_Type, Data\_Type > &Array\_)  
*Assignment constructor.*
- **BArray** (**BArray**< Cell\_Type, Data\_Type > &&x) noexcept  
*Move operator.*
- **BArray**< Cell\_Type, Data\_Type > & **operator=** (**BArray**< Cell\_Type, Data\_Type > &&x) noexcept  
*Move assignment.*

- 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\_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
```



**7.1.3.8 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
```

**7.1.3.9 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>`

**7.1.3.10 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
```

**7.1.3.11 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
```

**7.1.3.12 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 )
```

#### 7.1.3.13 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 )
```

#### 7.1.3.14 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 )
```

#### 7.1.3.15 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
```

#### 7.1.3.16 ncol()

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

#### 7.1.3.17 nnozero()

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

**7.1.3.18 nrow()**

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

**7.1.3.19 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 )
```

**7.1.3.20 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
```

**7.1.3.21 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 )
```

**7.1.3.22 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 )
```

**7.1.3.23 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 )
```

**7.1.3.24 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 )
```

**7.1.3.25 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 )
```

**7.1.3.26 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 )
```

**7.1.3.27 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 )
```

**7.1.3.28 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 )
```

**7.1.3.29 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.

#### 7.1.3.30 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.

#### 7.1.3.31 operator==()

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

#### 7.1.3.32 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
```

#### 7.1.3.33 print()

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

#### 7.1.3.34 reserve()

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

#### 7.1.3.35 resize()

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

**7.1.3.36 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 )
```

**7.1.3.37 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
```

**7.1.3.38 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_ ↔ data_</i>	

**7.1.3.39 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 )
```

#### 7.1.3.40 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 )
```

#### 7.1.3.41 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 )
```

#### 7.1.3.42 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 )
```

#### 7.1.3.43 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 )
```

#### 7.1.3.44 transpose()

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

#### 7.1.3.45 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 )
```

#### 7.1.3.46 zero\_row()

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

### 7.1.4 Friends And Related Function Documentation

#### 7.1.4.1 BArrayCell< Cell\_Type, Data\_Type >

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

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

#### 7.1.4.2 BArrayCell\_const< Cell\_Type, Data\_Type >

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

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

### 7.1.5 Member Data Documentation

#### 7.1.5.1 visited

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

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

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

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

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



## 7.2 BArrayCell< Cell\_Type, Data\_Type > Class Template Reference

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

### Public Member Functions

- [BArrayCell](#) ([BArray](#)< Cell\_Type, Data\_Type > \*Array\_, [uint](#) i\_, [uint](#) j\_, bool check\_bounds=true)
- [~BArrayCell](#) ()
- void [operator=](#) (const Cell\_Type &val)
- void [operator+=](#) (const Cell\_Type &val)
- void [operator-=](#) (const Cell\_Type &val)
- void [operator\\*=](#) (const Cell\_Type &val)
- void [operator/=](#) (const Cell\_Type &val)
- [operator Cell\\_Type](#) () const
- bool [operator==](#) (const Cell\_Type &val) const

### 7.2.1 Detailed Description

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

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

### 7.2.2 Constructor & Destructor Documentation

#### 7.2.2.1 BArrayCell()

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

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

#### 7.2.2.2 ~BArrayCell()

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

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

## 7.2.3 Member Function Documentation

### 7.2.3.1 operator Cell\_Type()

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

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

### 7.2.3.2 operator\*=( )

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

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

### 7.2.3.3 operator+=( )

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

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

### 7.2.3.4 operator-=( )

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

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

### 7.2.3.5 operator/=( )

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

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

#### 7.2.3.6 operator=()

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

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

#### 7.2.3.7 operator==()

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

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

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

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

## 7.3 BArrayCell\_const< Cell\_Type, Data\_Type > Class Template Reference

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

### Public Member Functions

- [BArrayCell\\_const](#) (const [BArray](#)< Cell\_Type, Data\_Type > \*Array\_, [uint](#) i\_, [uint](#) j\_, bool check\_bounds=true)
- [~BArrayCell\\_const](#) ()
- [operator Cell\\_Type](#) () const
- bool [operator==](#) (const Cell\_Type &val) const
- bool [operator!=](#) (const Cell\_Type &val) const
- bool [operator<](#) (const Cell\_Type &val) const
- bool [operator>](#) (const Cell\_Type &val) const
- bool [operator<=](#) (const Cell\_Type &val) const
- bool [operator>=](#) (const Cell\_Type &val) const

### 7.3.1 Detailed Description

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

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

## 7.3.2 Constructor & Destructor Documentation

### 7.3.2.1 BArrayCell\_const()

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

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

### 7.3.2.2 ~BArrayCell\_const()

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

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

## 7.3.3 Member Function Documentation

### 7.3.3.1 operator Cell\_Type()

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

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

### 7.3.3.2 operator"!=(

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

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

### 7.3.3.3 operator<()

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

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

### 7.3.3.4 operator<=()

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

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

### 7.3.3.5 operator==()

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

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

### 7.3.3.6 operator>()

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

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

### 7.3.3.7 operator>=()

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

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

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

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

## 7.4 BArrayDense< Cell\_Type, Data\_Type > Class Template Reference

Baseline class for binary arrays.

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

### Public Member Functions

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

### Constructors

#### Parameters

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

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

*Move operator.*

- `BArrayDense< Cell_Type, Data_Type > & operator= (BArrayDense< Cell_Type, Data_Type > &&x)` noexcept

*Move assignment.*

- void `set_data` (Data\_Type \*data\_, bool delete\_data\_=false)

*Set the data object.*

- Data\_Type \* `D` ()
- const Data\_Type \* `D` () const

### Queries

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

#### Parameters

i,j	<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>

- BArrayDense< Cell\_Type, Data\_Type > & `operator+=` (const std::pair< uint, uint > &coords)
- BArrayDense< Cell\_Type, Data\_Type > & `operator-=` (const std::pair< uint, uint > &coords)
- BArrayCell< Cell\_Type, Data\_Type > `operator()` (uint i, uint j, bool check\_bounds=true)
- const BArrayCell< 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

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

### Public Attributes

- `bool` `visited` = false

### Friends

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

## 7.4.1 Detailed Description

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

Baseline class for binary arrays.

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

## 7.4.2 Constructor & Destructor Documentation



**7.4.2.1 BArrayDense()** [1/6]

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

Zero-size array.

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

**7.4.2.2 BArrayDense()** [2/6]

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

Empty array.

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

**7.4.2.3 BArrayDense()** [3/6]

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

Edgelist with data.

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

**7.4.2.4 BArrayDense()** [4/6]

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

Edgelist with no data (simpler)

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

#### 7.4.2.5 BArrayDense() [5/6]

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

Copy constructor.

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

#### 7.4.2.6 BArrayDense() [6/6]

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

Move operator.

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

#### 7.4.2.7 ~BArrayDense()

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

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

### 7.4.3 Member Function Documentation

#### 7.4.3.1 clear()

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

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

#### 7.4.3.2 col()

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

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

#### 7.4.3.3 D() [1/2]

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

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

#### 7.4.3.4 D() [2/2]

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

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

#### 7.4.3.5 default\_val()

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

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

#### 7.4.3.6 get\_cell()

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

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

**7.4.3.7 get\_col\_vec()** [1/2]

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

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

**7.4.3.8 get\_col\_vec()** [2/2]

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

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

**7.4.3.9 get\_entries()**

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

Get the edgelist.

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

Returns

`Entries<Cell_Type>`

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

**7.4.3.10 get\_row\_vec()** [1/2]

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

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

#### 7.4.3.11 get\_row\_vec() [2/2]

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

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

#### 7.4.3.12 insert\_cell() [1/3]

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

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

#### 7.4.3.13 insert\_cell() [2/3]

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

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

#### 7.4.3.14 insert\_cell() [3/3]

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

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

#### 7.4.3.15 is\_empty()

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

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

#### 7.4.3.16 ncol()

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

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

#### 7.4.3.17 nnozero()

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

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

#### 7.4.3.18 nrow()

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

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

#### 7.4.3.19 operator>() [1/2]

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

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

**7.4.3.20 operator>() [2/2]**

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

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

**7.4.3.21 operator\*=( )**

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

**7.4.3.22 operator+=( ) [1/3]**

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

**7.4.3.23 operator+=( ) [2/3]**

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

**7.4.3.24 operator+=( ) [3/3]**

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

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

**7.4.3.25 operator-=()** [1/3]

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

**7.4.3.26 operator-=()** [2/3]

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

**7.4.3.27 operator-=()** [3/3]

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

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

**7.4.3.28 operator/=()**

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

**7.4.3.29 operator=()** [1/2]

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

Move assignment.

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



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

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

Assignment constructor.

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

#### 7.4.3.31 operator==()

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

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

#### 7.4.3.32 out\_of\_range()

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

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

#### 7.4.3.33 print()

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

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

#### 7.4.3.34 reserve()

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

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

**7.4.3.35 resize()**

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

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

**7.4.3.36 rm\_cell()**

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

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

**7.4.3.37 row()**

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

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

**7.4.3.38 set\_data()**

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

Set the data object.

**Parameters**

<i>data_</i>	
<i>delete_ ↔ data_</i>	

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

#### 7.4.3.39 swap\_cells()

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

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

#### 7.4.3.40 swap\_cols()

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

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

#### 7.4.3.41 swap\_rows()

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

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

#### 7.4.3.42 toggle\_cell()

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

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

#### 7.4.3.43 toggle\_lock()

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

#### 7.4.3.44 transpose()

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

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

#### 7.4.3.45 zero\_col()

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

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

#### 7.4.3.46 zero\_row()

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

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

### 7.4.4 Friends And Related Function Documentation

#### 7.4.4.1 BArrayCell< Cell\_Type, Data\_Type >

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

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

#### 7.4.4.2 BArrayCell\_const< Cell\_Type, Data\_Type >

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

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

### 7.4.5 Member Data Documentation

#### 7.4.5.1 visited

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

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

Definition at line 44 of file `barraydense-bones.hpp`.

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

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

## 7.5 BArrayDenseCell< Cell\_Type, Data\_Type > Class Template Reference

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

### Public Member Functions

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

### 7.5.1 Detailed Description

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

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

### 7.5.2 Constructor & Destructor Documentation

#### 7.5.2.1 BArrayDenseCell()

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

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

#### 7.5.2.2 ~BArrayDenseCell()

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

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

### 7.5.3 Member Function Documentation

#### 7.5.3.1 operator Cell\_Type()

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

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

### 7.5.3.2 operator\*=( )

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

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

### 7.5.3.3 operator+=( )

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

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

### 7.5.3.4 operator-=( )

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

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

### 7.5.3.5 operator/=( )

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

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

### 7.5.3.6 operator=( )

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

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

### 7.5.3.7 operator==( )

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

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

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

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

## 7.6 BArrayDenseCell\_const< Cell\_Type, Data\_Type > Class Template Reference

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

### Public Member Functions

- `BArrayDenseCell_const` (const `BArrayDense`< Cell\_Type, Data\_Type > \*Array\_, `uint` i\_, `uint` j\_, bool check↔\_bounds=true)
- `~BArrayDenseCell_const` ( )
- `operator Cell_Type` ( ) const
- bool `operator==` (const Cell\_Type &val) const
- bool `operator!=` (const Cell\_Type &val) const
- bool `operator<` (const Cell\_Type &val) const
- bool `operator>` (const Cell\_Type &val) const
- bool `operator<=` (const Cell\_Type &val) const
- bool `operator>=` (const Cell\_Type &val) const

### 7.6.1 Detailed Description

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

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

### 7.6.2 Constructor & Destructor Documentation



### 7.6.2.1 BArrayDenseCell\_const()

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

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

### 7.6.2.2 ~BArrayDenseCell\_const()

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

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

## 7.6.3 Member Function Documentation

### 7.6.3.1 operator Cell\_Type()

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

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

### 7.6.3.2 operator"!=(

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

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

### 7.6.3.3 operator<()

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

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

#### 7.6.3.4 operator<=()

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

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

#### 7.6.3.5 operator==( )

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

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

#### 7.6.3.6 operator>()

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

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

#### 7.6.3.7 operator>=()

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

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

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

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

## 7.7 BArrayVector< Cell\_Type, Data\_Type > Class Template Reference

Row or column of a `BArray`

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

## Public Member Functions

- [BArrayVector](#) ([BArray](#)< Cell\_Type, Data\_Type > \*Array\_, [uint](#) &dim\_ [uint](#) &i\_, bool check\_bounds=true)  
Construct a new [BArrayVector](#) object.
- [~BArrayVector](#) ()
- bool [is\\_row](#) () const noexcept
- bool [is\\_col](#) () const noexcept
- [uint](#) [size](#) () const noexcept
- std::vector< Cell\_Type >::const\_iterator [begin](#) () noexcept
- std::vector< Cell\_Type >::const\_iterator [end](#) () noexcept
- void [operator=](#) (const Cell\_Type &val)
- void [operator+=](#) (const Cell\_Type &val)
- void [operator-=](#) (const Cell\_Type &val)
- void [operator\\*=](#) (const Cell\_Type &val)
- void [operator/=](#) (const Cell\_Type &val)
- [operator](#) std::vector< [Cell\\_Type](#) > () const
- bool [operator==](#) (const Cell\_Type &val) const

### 7.7.1 Detailed Description

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

Row or column of a [BArray](#)

Template Parameters

<i>Cell_Type</i>	
<i>Data_Type</i>	

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

### 7.7.2 Constructor & Destructor Documentation

#### 7.7.2.1 BArrayVector()

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

Construct a new [BArrayVector](#) object.

Parameters

<i>Array_</i>	Pointer to a <a href="#">BArray</a> object
<i>dim_</i>	Dimension. 0 means row and 1 means column.
<i>Generated by Doxygen</i>	Element to point.
<i>check_bounds</i>	When <code>true</code> , check boundaries.

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

#### 7.7.2.2 `~BArrayVector()`

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

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

### 7.7.3 Member Function Documentation

#### 7.7.3.1 `begin()`

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

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

#### 7.7.3.2 `end()`

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

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

#### 7.7.3.3 `is_col()`

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

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

#### 7.7.3.4 is\_row()

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

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

#### 7.7.3.5 operator std::vector< Cell\_Type >()

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

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

#### 7.7.3.6 operator\*=( )

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

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

#### 7.7.3.7 operator+=( )

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

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

#### 7.7.3.8 operator-=( )

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

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

#### 7.7.3.9 operator/=( )

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

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

#### 7.7.3.10 operator=( )

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

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

#### 7.7.3.11 operator==( )

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

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

#### 7.7.3.12 size( )

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

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

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

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

## 7.8 BArrayVector\_const< Cell\_Type, Data\_Type > Class Template Reference

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

## Public Member Functions

- [BArrayVector\\_const](#) (const [BArray](#)< Cell\_Type, Data\_Type > \*Array\_, [uint](#) &dim\_ [uint](#) &i\_, bool check\_↵ bounds=true)
- [~BArrayVector\\_const](#) ()
- bool [is\\_row](#) () const noexcept
- bool [is\\_col](#) () const noexcept
- [uint](#) [size](#) () const noexcept
- std::vector< Cell\_Type >::const\_iterator [begin](#) () noexcept
- std::vector< Cell\_Type >::const\_iterator [end](#) () noexcept
- [operator](#) std::vector< [Cell\\_Type](#) > () const
- bool [operator==](#) (const Cell\_Type &val) const
- bool [operator!=](#) (const Cell\_Type &val) const
- bool [operator<](#) (const Cell\_Type &val) const
- bool [operator>](#) (const Cell\_Type &val) const
- bool [operator<=](#) (const Cell\_Type &val) const
- bool [operator>=](#) (const Cell\_Type &val) const

### 7.8.1 Detailed Description

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

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

### 7.8.2 Constructor & Destructor Documentation

#### 7.8.2.1 BArrayVector\_const()

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

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

#### 7.8.2.2 ~BArrayVector\_const()

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

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

## 7.8.3 Member Function Documentation

### 7.8.3.1 begin()

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

### 7.8.3.2 end()

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

### 7.8.3.3 is\_col()

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

### 7.8.3.4 is\_row()

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

### 7.8.3.5 operator std::vector< Cell\_Type >()

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

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



#### 7.8.3.6 operator!=(())

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

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

#### 7.8.3.7 operator<()

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

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

#### 7.8.3.8 operator<=()

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

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

#### 7.8.3.9 operator==(())

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

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

#### 7.8.3.10 operator>()

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

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

### 7.8.3.11 operator>=()

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

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

### 7.8.3.12 size()

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

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

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

## 7.9 Cell< Cell\_Type > Class Template Reference

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

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

### Public Member Functions

- [Cell](#) ()
- [Cell](#) (Cell\_Type value\_, bool visited\_=false)
- [~Cell](#) ()
- [Cell](#) (const [Cell](#)< Cell\_Type > &arg)
- [Cell](#)< Cell\_Type > & [operator=](#) ([Cell](#)< Cell\_Type > &other)
- [Cell](#) ([Cell](#)< Cell\_Type > &&arg) noexcept
- [Cell](#)< Cell\_Type > & [operator=](#) ([Cell](#)< Cell\_Type > &&other) noexcept
- void [add](#) (Cell\_Type x)
- [operator Cell\\_Type](#) () const
- bool [operator==](#) (const [Cell](#)< Cell\_Type > &rhs) const
- bool [operator!=](#) (const [Cell](#)< Cell\_Type > &rhs) const
- void [add](#) (double x)
- void [add](#) (unsigned int x)
- void [add](#) (int x)
- [Cell](#) ()
- [Cell](#) ()
- [Cell](#) ()

### Public Attributes

- Cell\_Type [value](#)
- bool [visited](#)

### 7.9.1 Detailed Description

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

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

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

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

### 7.9.2 Constructor & Destructor Documentation

#### 7.9.2.1 Cell() [1/7]

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

#### 7.9.2.2 Cell() [2/7]

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

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

#### 7.9.2.3 ~Cell()

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

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

#### 7.9.2.4 Cell() [3/7]

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

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

#### 7.9.2.5 Cell() [4/7]

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

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

#### 7.9.2.6 Cell() [5/7]

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

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

#### 7.9.2.7 Cell() [6/7]

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

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

#### 7.9.2.8 Cell() [7/7]

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

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

### 7.9.3 Member Function Documentation

#### 7.9.3.1 add() [1/4]

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

#### 7.9.3.2 add() [2/4]

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

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

#### 7.9.3.3 add() [3/4]

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

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

#### 7.9.3.4 add() [4/4]

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

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

#### 7.9.3.5 operator Cell\_Type()

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

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

#### 7.9.3.6 operator"!=(

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

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

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

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

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

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

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

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

#### 7.9.3.9 operator==()

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

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

### 7.9.4 Member Data Documentation

#### 7.9.4.1 value

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

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

#### 7.9.4.2 visited

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

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

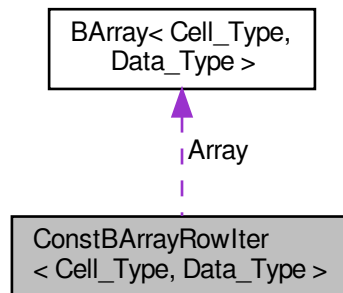
The documentation for this class was generated from the following files:

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

## 7.10 ConstBArrayRowIter< Cell\_Type, Data\_Type > Class Template Reference

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

Collaboration diagram for ConstBArrayRowIter< Cell\_Type, Data\_Type >:



### Public Member Functions

- [ConstBArrayRowIter](#) (const [BArray](#)< Cell\_Type, Data\_Type > \*Array\_)
- [~ConstBArrayRowIter](#) ()

### Public Attributes

- [uint](#) current\_row
- [uint](#) current\_col
- [Row\\_type](#)< Cell\_Type >::const\_iterator iter
- const [BArray](#)< Cell\_Type, Data\_Type > \* [Array](#)

#### 7.10.1 Detailed Description

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

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

#### 7.10.2 Constructor & Destructor Documentation

### 7.10.2.1 ConstBArrayRowIter()

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

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

### 7.10.2.2 ~ConstBArrayRowIter()

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

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

## 7.10.3 Member Data Documentation

### 7.10.3.1 Array

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

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

### 7.10.3.2 current\_col

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

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

### 7.10.3.3 current\_row

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

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



## 7.10.3.4 iter

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

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

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

- include/barry/barray-iterator.hpp

## 7.11 Counter&lt; Array\_Type, Data\_Type &gt; 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 <a href="#">BArray</a> as the needed variables (see <a href="#">BArray::data</a> ).
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](#) = ""

### 7.11.1 Detailed Description

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

A counter function based on change statistics.

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

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

### 7.11.2 Constructor & Destructor Documentation

#### 7.11.2.1 Counter() [1/4]

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

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

#### 7.11.2.2 Counter() [2/4]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Counter< Array_Type, Data_Type >::Counter (
    Counter\_fun\_type< Array_Type, Data_Type > count\_fun\_,
    Counter\_fun\_type< Array_Type, Data_Type > init\_fun\_ = nullptr,
    Data_Type * data\_ = nullptr,
    bool delete\_data\_ = false,
    std::string name\_ = "",
    std::string desc\_ = "" ) [inline]
```

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

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

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

Copy constructor.

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

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

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

Move constructor.

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

### 7.11.2.5 ~Counter()

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

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

## 7.11.3 Member Function Documentation

### 7.11.3.1 count()

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

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

### 7.11.3.2 init()

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

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

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

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

Copy assignment.

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

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

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

Move assignment.

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

## 7.11.4 Member Data Documentation

### 7.11.4.1 count\_fun

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

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

#### 7.11.4.2 data

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

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

#### 7.11.4.3 delete\_data

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

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

#### 7.11.4.4 desc

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

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

#### 7.11.4.5 init\_fun

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

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

#### 7.11.4.6 name

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

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

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

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

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

Vector of counters.

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

### Public Member Functions

- [Counters](#) ()
- [~Counters](#) ()
- [Counters](#) (const [Counters](#)< Array\_Type, Data\_Type > &counter\_)
- Copy constructor.*
- [Counters](#) ([Counters](#)< Array\_Type, Data\_Type > &&counters\_) noexcept
- Move constructor.*
- [Counters](#)< Array\_Type, Data\_Type > [operator=](#) (const [Counters](#)< Array\_Type, Data\_Type > &counter\_)
- Copy assignment constructor.*
- [Counters](#)< Array\_Type, Data\_Type > & [operator=](#) ([Counters](#)< Array\_Type, Data\_Type > &&counter\_) noexcept
- Move assignment constructor.*
- [Counter](#)< Array\_Type, Data\_Type > & [operator\[\]](#) (uint idx)
- Returns a pointer to a particular counter.*
- std::size\_t [size](#) () const noexcept
- Number of counters in the set.*
- void [add\\_counter](#) ([Counter](#)< Array\_Type, Data\_Type > &counter)
- void [add\\_counter](#) ([Counter](#)< Array\_Type, Data\_Type > \*counter)
- void [add\\_counter](#) ([Counter\\_fun\\_type](#)< Array\_Type, Data\_Type > count\_fun\_, [Counter\\_fun\\_type](#)< Array\_Type, Data\_Type > init\_fun\_=nullptr, Data\_Type \*data\_=nullptr, bool delete\_data\_=false, std::string name\_="", std::string desc\_="")
- void [clear](#) ()

### 7.12.1 Detailed Description

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

Vector of counters.

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

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

### 7.12.2 Constructor & Destructor Documentation

**7.12.2.1 Counters()** [1/3]

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

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

**7.12.2.2 ~Counters()**

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

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

**7.12.2.3 Counters()** [2/3]

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

Copy constructor.

**Parameters**

<i>counter_↔</i>	
—	

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

**7.12.2.4 Counters()** [3/3]

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

Move constructor.

**Parameters**

<i>counters_↔</i>	
—	

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

## 7.12.3 Member Function Documentation

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

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

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

### 7.12.3.2 add\_counter() [2/3]

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

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

### 7.12.3.3 add\_counter() [3/3]

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

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

### 7.12.3.4 clear()

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

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

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

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

Copy assignment constructor.



## Parameters

<i>counter</i> ↔	
—	

## Returns

Counters<Array\_Type,Data\_Type>

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

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

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

Move assignment constructor.

## Parameters

<i>counter</i> ↔	
—	

## Returns

Counters<Array\_Type,Data\_Type>&

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

## 7.12.3.7 operator[]()

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

Returns a pointer to a particular counter.

## Parameters

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

## Returns

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

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

### 7.12.3.8 size()

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

Number of counters in the set.

#### Returns

uint

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

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

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

## 7.13 Entries< Cell\_Type > Class Template Reference

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

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

### Public Member Functions

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

### Public Attributes

- std::vector< uint > `source`
- std::vector< uint > `target`
- std::vector< Cell\_Type > `val`

### 7.13.1 Detailed Description

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

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

## Template Parameters

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

Definition at line 59 of file typedefs.hpp.

## 7.13.2 Constructor & Destructor Documentation

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

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

Definition at line 65 of file typedefs.hpp.

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

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

Definition at line 66 of file typedefs.hpp.

### 7.13.2.3 ~Entries()

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

Definition at line 73 of file typedefs.hpp.

## 7.13.3 Member Function Documentation

### 7.13.3.1 resize()

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

Definition at line 75 of file typedefs.hpp.

### 7.13.4 Member Data Documentation

#### 7.13.4.1 source

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

Definition at line 61 of file typedefs.hpp.

#### 7.13.4.2 target

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

Definition at line 62 of file typedefs.hpp.

#### 7.13.4.3 val

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

Definition at line 63 of file typedefs.hpp.

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

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

## 7.14 Flock Class Reference

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

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

## Public Member Functions

- [Flock](#) ()
- [~Flock](#) ()
- unsigned int [add\\_data](#) (std::vector< std::vector< unsigned int > > &annotations, std::vector< unsigned int > &geneid, std::vector< int > &parent, std::vector< bool > &duplication)  
*Add a tree to the flock.*
- void [set\\_seed](#) (const unsigned int &s)  
*Set the seed of the model.*
- void [init](#) (bool verb=true)
- [phylocounters::PhyloCounters](#) \* [get\\_counters](#) ()
- [phylocounters::PhyloSupport](#) \* [get\\_support](#) ()
- double [likelihood\\_joint](#) (const std::vector< double > &par, bool as\_log=false, bool use\_reduced\_↵ sequence=true)  
*Returns the joint likelihood of the model.*
- [Geese](#) \* [operator\(\)](#) (unsigned int i, bool check\_bounds=true)  
*Access the i-th geese element.*

## Information about the model

- unsigned int [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
- unsigned int [support\\_size](#) () const noexcept
- std::vector< std::string > [colnames](#) () const
- unsigned int [parse\\_polytomies](#) (bool verb=true) const noexcept

## Public Attributes

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

### 7.14.1 Detailed Description

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

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

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

### 7.14.2 Constructor & Destructor Documentation

### 7.14.2.1 Flock()

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

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

### 7.14.2.2 ~Flock()

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

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

## 7.14.3 Member Function Documentation

### 7.14.3.1 add\_data()

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

Add a tree to the flock.

#### Parameters

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

#### Returns

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

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

### 7.14.3.2 colnames()

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

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

### 7.14.3.3 get\_counters()

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

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

### 7.14.3.4 get\_support()

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

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

### 7.14.3.5 init()

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

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

### 7.14.3.6 likelihood\_joint()

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

Returns the joint likelihood of the model.

#### Parameters

<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 82 of file flock-meet.hpp.

#### 7.14.3.7 nfuncs()

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

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

#### 7.14.3.8 nleafs()

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

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

#### 7.14.3.9 nnodes()

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

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

#### 7.14.3.10 nterms()

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

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

#### 7.14.3.11 ntrees()

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

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

#### 7.14.3.12 operator>()()

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

Access the i-th geese element.



## Parameters

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

## Returns

Geese\*

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

### 7.14.3.13 parse\_polytomies()

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

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

### 7.14.3.14 set\_seed()

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

Set the seed of the model.

## Parameters

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

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

### 7.14.3.15 support\_size()

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

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

## 7.14.4 Member Data Documentation

#### 7.14.4.1 dat

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

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

#### 7.14.4.2 initialized

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

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

#### 7.14.4.3 model

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

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

#### 7.14.4.4 nfunctions

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

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

#### 7.14.4.5 engine

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

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

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

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

## 7.15 FreqTable< T > Class Template Reference

Database of statistics.

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

## Public Member Functions

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

### 7.15.1 Detailed Description

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

Database of statistics.

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

Definition at line 16 of file statsdb.hpp.

### 7.15.2 Constructor & Destructor Documentation

#### 7.15.2.1 FreqTable()

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

Definition at line 28 of file statsdb.hpp.

#### 7.15.2.2 ~FreqTable()

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

Definition at line 29 of file statsdb.hpp.

### 7.15.3 Member Function Documentation

#### 7.15.3.1 add()

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

Definition at line 47 of file statsdb.hpp.

#### 7.15.3.2 as\_vector()

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

Definition at line 61 of file statsdb.hpp.

#### 7.15.3.3 clear()

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

Definition at line 83 of file statsdb.hpp.

#### 7.15.3.4 get\_data()

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

Definition at line 73 of file statsdb.hpp.

#### 7.15.3.5 get\_data\_ptr()

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

Definition at line 78 of file statsdb.hpp.

### 7.15.3.6 print()

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

Definition at line 102 of file statsdb.hpp.

### 7.15.3.7 reserve()

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

Definition at line 89 of file statsdb.hpp.

### 7.15.3.8 size()

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

Definition at line 126 of file statsdb.hpp.

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

- include/barry/statsdb.hpp

## 7.16 Geese Class Reference

Annotated Phyllo [Model](#).

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

### Public Member Functions

- [~Geese](#) ()
- void [init](#) (bool verb=true)
- void [inherit\\_support](#) (const [Geese](#) &model\_, bool delete\_support\_=false)
- void [calc\\_sequence](#) ([Node](#) \*n=nullptr)
- void [calc\\_reduced\\_sequence](#) ()
- double [likelihood](#) (const std::vector< double > &par, bool as\_log=false, bool use\_reduced\_sequence=true)
- double [likelihood\\_exhaust](#) (const std::vector< double > &par)
- std::vector< double > [get\\_probabilities](#) () const
- void [set\\_seed](#) (const unsigned int &s)
- std::vector< std::vector< unsigned int > > [simulate](#) (const std::vector< double > &par)
- std::vector< std::vector< double > > [observed\\_counts](#) ()
- void [print\\_observed\\_counts](#) ()
- void [init\\_node](#) ([Node](#) &n)
- void [update\\_annotations](#) (unsigned int nodeid, std::vector< unsigned int > newann)
- std::vector< std::vector< bool > > [get\\_states](#) () const
 

*Powerset of a gene's possible states.*
- std::vector< unsigned int > [get\\_annotated\\_nodes](#) () const
 

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

### Construct a new Geese object

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

*Parameters*

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

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

**Information about the model***Parameters*

verb	<i>When <code>true</code> it will print out information about the encountered polytomies.</i>
------	---

- unsigned int [nfuncs](#) () const noexcept  
*Number of functions analyzed.*
- unsigned int [nnodes](#) () const noexcept  
*Number of nodes (interior + leaf)*
- unsigned int [nleafs](#) () const noexcept  
*Number of leaf.*
- unsigned int [nterms](#) () const  
*Number of terms included.*
- unsigned int [support\\_size](#) () const noexcept  
*Number of unique sets of sufficient stats.*
- std::vector< std::string > [colnames](#) () const  
*Names of the terms in the model.*
- unsigned int [parse\\_polytomies](#) (bool verb=true) const noexcept  
*Check polytomies and return the largest.*

**Geese prediction**

*Calculate the conditional probability*

*Parameters*

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

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

**Returns**

*std::vector< double >* Returns the posterior probability

- `std::vector< std::vector< double > >` [predict](#) (const `std::vector< double >` &par, `std::vector< std::vector< double > >` \*res\_prob=nullptr, bool leave\_one\_out=false, bool only\_annotated=false, bool use\_reduced\_sequence=true)
- `std::vector< std::vector< double > >` [predict\\_backend](#) (const `std::vector< double >` &par, bool use\_reduced\_sequence, const `std::vector< uint >` &preorder)
- `std::vector< std::vector< double > >` [predict\\_exhaust\\_backend](#) (const `std::vector< double >` &par, const `std::vector< uint >` &preorder)
- `std::vector< std::vector< double > >` [predict\\_exhaust](#) (const `std::vector< double >` &par)
- `std::vector< std::vector< double > >` [predict\\_sim](#) (const `std::vector< double >` &par, bool only\_annotated=false, unsigned int nsims=10000u)

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

These functions provide direct access to some member objects that are shared by the nodes within [Geese](#).

**Returns**

[get\\_engine\(\)](#) returns the Pseudo-RNG engine used.

[get\\_counters\(\)](#) returns the vector of counters used.

[get\\_model\(\)](#) returns the [Model](#) object used.

[get\\_support\(\)](#) returns the computed support of the model.

- `std::mt19937` \* [get\\_engine\(\)](#)
- `phylocounters::PhyloCounters` \* [get\\_counters\(\)](#)
- `phylocounters::PhyloModel` \* [get\\_model\(\)](#)
- `phylocounters::PhyloSupport` \* [get\\_support\(\)](#)

**Public Attributes**

- unsigned int [nfunctions](#)
- `std::map< unsigned int, Node >` [nodes](#)
- `barry::MapVec_type< unsigned int >` [map\\_to\\_nodes](#)
- `std::vector< unsigned int >` [sequence](#)
- `std::vector< unsigned int >` [reduced\\_sequence](#)
- bool [initialized](#) = false
- bool [delete\\_engine](#) = false
- bool [delete\\_support](#) = false

**7.16.1 Detailed Description**

Annotated Phylo [Model](#).

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

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

**7.16.2 Constructor & Destructor Documentation**

#### 7.16.2.1 Geese() [1/4]

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

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

#### 7.16.2.2 Geese() [2/4]

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

Definition at line 20 of file geese-meat-constructors.hpp.

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

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

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

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

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

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

#### 7.16.2.5 ~Geese()

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

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

### 7.16.3 Member Function Documentation



### 7.16.3.1 calc\_reduced\_sequence()

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

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

### 7.16.3.2 calc\_sequence()

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

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

### 7.16.3.3 colnames()

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

Names of the terms in the model.

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

### 7.16.3.4 get\_annotated\_nodes()

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

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

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

### 7.16.3.5 get\_counters()

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

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

### 7.16.3.6 get\_model()

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

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

### 7.16.3.7 get\_probabilities()

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

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

### 7.16.3.8 get\_rengine()

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

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

### 7.16.3.9 get\_states()

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

Powerset of a gene's possible states.

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

#### Returns

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

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

### 7.16.3.10 get\_support()

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

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

### 7.16.3.11 inherit\_support()

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

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

### 7.16.3.12 init()

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

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

### 7.16.3.13 init\_node()

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

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

### 7.16.3.14 likelihood()

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

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

### 7.16.3.15 likelihood\_exhaust()

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

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

### 7.16.3.16 nfuncs()

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

Number of functions analyzed.

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

#### 7.16.3.17 nleafs()

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

Number of leaf.

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

#### 7.16.3.18 nnodes()

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

Number of nodes (interior + leaf)

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

#### 7.16.3.19 nterms()

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

Number of terms included.

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

#### 7.16.3.20 observed\_counts()

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

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

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

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

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

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

### 7.16.3.23 parse\_polytomies()

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

Check polytomies and return the largest.

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

### 7.16.3.24 predict()

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

Definition at line 210 of file geese-meat-predict.hpp.

### 7.16.3.25 predict\_backend()

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

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

### 7.16.3.26 predict\_exhaust()

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

Definition at line 5 of file geese-meat-predict\_exhaust.hpp.

### 7.16.3.27 predict\_exhaust\_backend()

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

Definition at line 47 of file geese-meat-predict\_exhaust.hpp.

#### 7.16.3.28 predict\_sim()

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

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

#### 7.16.3.29 print\_observed\_counts()

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

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

#### 7.16.3.30 set\_seed()

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

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

#### 7.16.3.31 simulate()

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

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

#### 7.16.3.32 support\_size()

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

Number of unique sets of sufficient stats.

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

### 7.16.3.33 update\_annotations()

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

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

## 7.16.4 Member Data Documentation

### 7.16.4.1 delete\_engine

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

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

### 7.16.4.2 delete\_support

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

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

### 7.16.4.3 initialized

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

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

### 7.16.4.4 map\_to\_nodes

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

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

#### 7.16.4.5 nfunctions

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

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

#### 7.16.4.6 nodes

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

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

#### 7.16.4.7 reduced\_sequence

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

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

#### 7.16.4.8 sequence

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

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

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

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

## 7.17 **Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type > Class Template Reference**

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

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



## Public Member Functions

- void [set\\_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 > &params={})
- Array\_Type [sample](#) (const [uint](#) &i, const std::vector< double > &params)
- 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](#) ()
- [Support](#)< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type > \* [get\\_support](#) ()

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

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

- void [add\\_counter](#) ([Counter](#)< Array\_Type, Data\_Counter\_Type > &counter)
- void [add\\_counter](#) ([Counter](#)< Array\_Type, Data\_Counter\_Type > \*counter)
- void [add\\_counter](#) ([Counter\\_fun\\_type](#)< Array\_Type, Data\_Counter\_Type > count\_fun\_, [Counter\\_fun\\_type](#)< Array\_Type, Data\_Counter\_Type > init\_fun\_=nullptr, Data\_Counter\_Type \*data\_=nullptr, bool delete\_data\_=false)
- void [set\\_counters](#) ([Counters](#)< Array\_Type, Data\_Counter\_Type > \*counters\_)

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

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

- void [add\\_rule](#) ([Rule](#)< Array\_Type, Data\_Rule\_Type > &rule)
- void [add\\_rule](#) ([Rule](#)< Array\_Type, Data\_Rule\_Type > \*rule)
- void [add\\_rule](#) ([Rule\\_fun\\_type](#)< Array\_Type, Data\_Rule\_Type > count\_fun\_, Data\_Rule\_Type \*data\_=nullptr, bool delete\_data\_=false)
- void [set\\_rules](#) ([Rules](#)< Array\_Type, Data\_Rule\_Type > \*rules\_)
- void [add\\_rule\\_dyn](#) ([Rule](#)< Array\_Type, Data\_Rule\_Dyn\_Type > &rule)
- void [add\\_rule\\_dyn](#) ([Rule](#)< Array\_Type, Data\_Rule\_Dyn\_Type > \*rule)
- void [add\\_rule\\_dyn](#) ([Rule\\_fun\\_type](#)< Array\_Type, Data\_Rule\_Dyn\_Type > count\_fun\_, Data\_Rule\_Dyn\_Type \*data\_=nullptr, bool delete\_data\_=false)
- void [set\\_rules\\_dyn](#) ([Rules](#)< Array\_Type, Data\_Rule\_Dyn\_Type > \*rules\_)

## Likelihood functions.

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

## Parameters

params	Vector of parameters
as_log	When <code>true</code> , the function returns the log-likelihood.

- double `likelihood` (const std::vector< double > &params, const uint &i, bool as\_log=false)
- double `likelihood` (const std::vector< double > &params, const Array\_Type &Array\_, int i=-1, bool as\_log=false)
- double `likelihood` (const std::vector< double > &params, const std::vector< double > &target\_, const uint &i, bool as\_log=false)
- double `likelihood_total` (const std::vector< double > &params, bool as\_log=false)

### Extract elements by index

#### Parameters

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

- double `get_norm_const` (const std::vector< double > &params, const uint &i, bool as\_log=false)
- const std::vector< Array\_Type > \* `get_pset` (const uint &i)
- const std::vector< std::vector< double > > \* `get_stats` (const uint &i)

### Size of the model

Number of different supports included in the model

This will return the size of `stats`.

#### Returns

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

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

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

- unsigned int `size()` const noexcept
- unsigned int `size_unique()` const noexcept
- unsigned int `nterms()` const noexcept
- unsigned int `support_size()` const noexcept
- std::vector< std::string > `colnames()` const

## 7.17.1 Detailed Description

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

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

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

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

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

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

## 7.17.2 Constructor & Destructor Documentation

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

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

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

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

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

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

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

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

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

#### 7.17.2.4 ~Model()

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

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

### 7.17.3 Member Function Documentation

#### 7.17.3.1 add\_array()

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

Adds an array to the support of not already included.

##### Parameters

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

##### Returns

The number of the array.

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

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

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

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

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

### 7.17.3.3 add\_counter() [2/3]

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

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

### 7.17.3.4 add\_counter() [3/3]

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

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

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

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

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

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

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

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

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

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

Definition at line 244 of file model-meat.hpp.

**7.17.3.8 add\_rule\_dyn() [1/3]**

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

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

**7.17.3.9 add\_rule\_dyn() [2/3]**

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

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

**7.17.3.10 add\_rule\_dyn() [3/3]**

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

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

### 7.17.3.11 colnames()

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

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

### 7.17.3.12 get\_counters()

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

Definition at line 725 of file model-meat.hpp.

### 7.17.3.13 get\_norm\_const()

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

Definition at line 567 of file model-meat.hpp.

### 7.17.3.14 get\_pset()

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

Definition at line 599 of file model-meat.hpp.

**7.17.3.15 get\_engine()**

```
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_engine [inline]
```

Definition at line 719 of file model-meat.hpp.

**7.17.3.16 get\_rules()**

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

Definition at line 731 of file model-meat.hpp.

**7.17.3.17 get\_rules\_dyn()**

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

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

**7.17.3.18 get\_stats()**

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

Definition at line 612 of file model-meat.hpp.

**7.17.3.19 get\_support()**

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

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



### 7.17.3.20 likelihood() [1/3]

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

Definition at line 453 of file model-meat.hpp.

### 7.17.3.21 likelihood() [2/3]

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

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

### 7.17.3.22 likelihood() [3/3]

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

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

### 7.17.3.23 likelihood\_total()

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

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

**7.17.3.24 nterms()**

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

Definition at line 653 of file model-meat.hpp.

**7.17.3.25 operator=()**

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

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

**7.17.3.26 print\_stats()**

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

Definition at line 624 of file model-meat.hpp.

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

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

### 7.17.3.28 sample() [2/2]

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

Definition at line 683 of file model-meat.hpp.

### 7.17.3.29 set\_counters()

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

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

### 7.17.3.30 set\_keygen()

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

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

### 7.17.3.31 set\_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 * engine_,
    bool delete_ = false ) [inline]
```

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

**7.17.3.32 set\_rules()**

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

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

**7.17.3.33 set\_rules\_dyn()**

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

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

**7.17.3.34 set\_seed()**

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

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

**7.17.3.35 size()**

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

Definition at line 643 of file model-meat.hpp.

**7.17.3.36 size\_unique()**

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

Definition at line 648 of file model-meat.hpp.

### 7.17.3.37 store\_psets()

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

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

### 7.17.3.38 support\_size()

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

Definition at line 660 of file model-meat.hpp.

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

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

## 7.18 NetCounterData Class Reference

Data class used to store arbitrary uint or double vectors.

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

### Public Member Functions

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

### Public Attributes

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

### 7.18.1 Detailed Description

Data class used to store arbitrary uint or double vectors.

Definition at line 61 of file network.hpp.

## 7.18.2 Constructor & Destructor Documentation

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

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

Definition at line 67 of file network.hpp.

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

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

Definition at line 68 of file network.hpp.

### 7.18.2.3 ~NetCounterData()

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

Definition at line 73 of file network.hpp.

## 7.18.3 Member Data Documentation

### 7.18.3.1 indices

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

Definition at line 64 of file network.hpp.

### 7.18.3.2 numbers

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

Definition at line 65 of file network.hpp.

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

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

## 7.19 NetworkData Class Reference

Data class for Networks.

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

### Public Member Functions

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

### Public Attributes

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

#### 7.19.1 Detailed Description

Data class for Networks.

Details on the available counters for [NetworkData](#) can be found in the [Network counters](#) section.

This holds information about whether the graph is directed or not, and, if defined, vectors of node (vertex) attributes ([vertex\\_attr](#)).

Definition at line 24 of file [network.hpp](#).

#### 7.19.2 Constructor & Destructor Documentation

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

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

Definition at line 30 of file [network.hpp](#).

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

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

Constructor using a single attribute.

**Parameters**

<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.

**7.19.2.3 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.

**7.19.2.4 ~NetworkData()**

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

Definition at line 56 of file network.hpp.

**7.19.3 Member Data Documentation****7.19.3.1 directed**

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

Definition at line 27 of file network.hpp.



### 7.19.3.2 vertex\_attr

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

Definition at line 28 of file network.hpp.

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

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

## 7.20 Node Class Reference

A single node for the model.

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

Collaboration diagram for Node:



### Public Member Functions

- [~Node](#) ()
- int [get\\_parent](#) () const
- unsigned int [noffspring](#) () const noexcept
- bool [is\\_leaf](#) () const noexcept

### Construct a new Node object

- [Node](#) ()
- [Node](#) (unsigned int id\_, unsigned int ord\_, bool duplication\_)
- [Node](#) (unsigned int id\_, unsigned int ord\_, std::vector< unsigned int > annotations\_, bool duplication\_)
- [Node](#) ([Node](#) &&x) noexcept
- [Node](#) (const [Node](#) &x)

## Public Attributes

- unsigned int `id`  
*Id of the node (as specified in the input)*
- unsigned int `ord`  
*Order in which the node was created.*
- `phylocounters::PhyloArray` `array`
- `std::vector< unsigned int >` `annotations`  
*Observed annotations (only defined for [Geese](#))*
- bool `duplication`
- `std::vector< phylocounters::PhyloArray >` `arrays` = {}  
*Arrays given all possible states.*
- `Node *` `parent` = nullptr  
*Parent node.*
- `std::vector< Node \* >` `offspring` = {}  
*Offspring nodes.*
- `std::vector< unsigned int >` `narray` = {}  
*ID of the array in the model.*
- bool `visited` = false
- `std::vector< double >` `subtree_prob`  
*Induced subtree probabilities.*
- `std::vector< double >` `probability`  
*The probability of observing each state.*

### 7.20.1 Detailed Description

A single node for the model.

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

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

### 7.20.2 Constructor & Destructor Documentation

#### 7.20.2.1 `Node()` [1/5]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

### 7.20.2.6 ~Node()

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

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

## 7.20.3 Member Function Documentation

### 7.20.3.1 `get_parent()`

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

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

### 7.20.3.2 `is_leaf()`

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

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

### 7.20.3.3 `noffspring()`

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

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

## 7.20.4 Member Data Documentation

### 7.20.4.1 `annotations`

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

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

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

### 7.20.4.2 `array`

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

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

### 7.20.4.3 arrays

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

Arrays given all possible states.

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

### 7.20.4.4 duplication

```
bool Node::duplication
```

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

### 7.20.4.5 id

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

Id of the node (as specified in the input)

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

### 7.20.4.6 narray

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

ID of the array in the model.

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

### 7.20.4.7 offspring

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

Offspring nodes.

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

#### 7.20.4.8 ord

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

Order in which the node was created.

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

#### 7.20.4.9 parent

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

Parent node.

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

#### 7.20.4.10 probability

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

The probability of observing each state.

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

#### 7.20.4.11 subtree\_prob

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

Induced subtree probabilities.

Definition at line 27 of file geese-node-bones.hpp.

#### 7.20.4.12 visited

```
bool Node::visited = false
```

Definition at line 25 of file geese-node-bones.hpp.

The documentation for this class was generated from the following file:

- [include/barry/models/geese/geese-node-bones.hpp](#)

## 7.21 NodeData Class Reference

Data definition for the `PhyloArray` class.

```
#include <phylo.hpp>
```

### Public Member Functions

- [NodeData](#) (const std::vector< double > &blengths\_, const std::vector< bool > &states\_, bool duplication\_  
\_ = true)
- [~NodeData](#) ()

### Public Attributes

- std::vector< double > [blengths](#)
- std::vector< bool > [states](#)
- bool [duplication](#) = true

#### 7.21.1 Detailed Description

Data definition for the `PhyloArray` class.

Details about the available counters for `PhyloArray` objects can be found in the [Phylo counters](#) section.

This holds basic information about a given node.

Definition at line 23 of file `phylo.hpp`.

#### 7.21.2 Constructor & Destructor Documentation

##### 7.21.2.1 NodeData()

```
NodeData::NodeData (
    const std::vector< double > & blengths_,
    const std::vector< bool > & states_,
    bool duplication_ = true ) [inline]
```

Definition at line 43 of file `phylo.hpp`.

##### 7.21.2.2 ~NodeData()

```
NodeData::~NodeData ( ) [inline]
```

Definition at line 49 of file `phylo.hpp`.

## 7.21.3 Member Data Documentation

### 7.21.3.1 blengths

```
std::vector< double > NodeData::blengths
```

Branch length.

Definition at line 29 of file phylo.hpp.

### 7.21.3.2 duplication

```
bool NodeData::duplication = true
```

Definition at line 39 of file phylo.hpp.

### 7.21.3.3 states

```
std::vector< bool > NodeData::states
```

State of the parent node.

Definition at line 34 of file phylo.hpp.

The documentation for this class was generated from the following file:

- include/barry/counters/[phylo.hpp](#)

## 7.22 PhyloRuleDynData Class Reference

```
#include <phylo.hpp>
```

### Public Member Functions

- [PhyloRuleDynData](#) (const std::vector< double > \*counts\_, [uint](#) pos\_, [uint](#) lb\_, [uint](#) ub\_, bool duplication\_)
- [~PhyloRuleDynData](#) ()



## Public Attributes

- const std::vector< double > \* [counts](#)
- [uint pos](#)
- [uint lb](#)
- [uint ub](#)
- bool [duplication](#)

### 7.22.1 Detailed Description

Definition at line 1308 of file phylo.hpp.

### 7.22.2 Constructor & Destructor Documentation

#### 7.22.2.1 PhyloRuleDynData()

```
PhyloRuleDynData::PhyloRuleDynData (
    const std::vector< double > * counts_,
    uint pos_,
    uint lb_,
    uint ub_,
    bool duplication_ ) [inline]
```

Definition at line 1315 of file phylo.hpp.

#### 7.22.2.2 ~PhyloRuleDynData()

```
PhyloRuleDynData::~PhyloRuleDynData ( ) [inline]
```

Definition at line 1324 of file phylo.hpp.

### 7.22.3 Member Data Documentation

#### 7.22.3.1 counts

```
const std::vector< double >* PhyloRuleDynData::counts
```

Definition at line 1310 of file phylo.hpp.

### 7.22.3.2 duplication

```
bool PhyloRuleDynData::duplication
```

Definition at line 1314 of file phylo.hpp.

### 7.22.3.3 lb

```
uint PhyloRuleDynData::lb
```

Definition at line 1312 of file phylo.hpp.

### 7.22.3.4 pos

```
uint PhyloRuleDynData::pos
```

Definition at line 1311 of file phylo.hpp.

### 7.22.3.5 ub

```
uint PhyloRuleDynData::ub
```

Definition at line 1313 of file phylo.hpp.

The documentation for this class was generated from the following file:

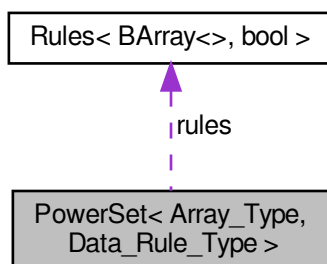
- include/barry/counters/[phylo.hpp](#)

## 7.23 PowerSet< Array\_Type, Data\_Rule\_Type > Class Template Reference

Powerset of a binary array.

```
#include <powerset-bones.hpp>
```

Collaboration diagram for PowerSet< Array\_Type, Data\_Rule\_Type >:



## Public Member Functions

- void [init\\_support](#) ()
- void [calc](#) ()
- void [reset](#) (uint N\_, uint M\_)

### Construct and destroy a PowerSet object

- [PowerSet](#) ()
- [PowerSet](#) (uint N\_, uint M\_)
- [PowerSet](#) (const Array\_Type &array)
- [~PowerSet](#) ()

### Wrappers for the `<tt>Rules</tt>` member.

*These will add rules to the model, which are shared by the support and the actual counter function.*

- void [add\\_rule](#) ([Rule](#)< Array\_Type, Data\_Rule\_Type > &rule)
- void [add\\_rule](#) ([Rule](#)< Array\_Type, Data\_Rule\_Type > \*rule)
- void [add\\_rule](#) ([Rule\\_fun\\_type](#)< Array\_Type, Data\_Rule\_Type > count\_fun\_, Data\_Rule\_Type \*data\_ ↵  
=nullptr, bool delete\_data\_=false)

### Getter functions

- const std::vector< Array\_Type > \* [get\\_data\\_ptr](#) () const
- std::vector< Array\_Type > [get\\_data](#) () const
- std::vector< Array\_Type >::iterator [begin](#) ()
- std::vector< Array\_Type >::iterator [end](#) ()
- std::size\_t [size](#) () const noexcept
- const Array\_Type & [operator\[\]](#) (const unsigned int &i) const

## Public Attributes

- Array\_Type [EmptyArray](#)
- std::vector< Array\_Type > [data](#)
- [Rules](#)< Array\_Type, Data\_Rule\_Type > \* [rules](#)
- uint [N](#)
- uint [M](#)
- bool [rules\\_deleted](#) = false
- std::vector< std::pair< uint, uint > > [coordinates\\_free](#)
- std::vector< std::pair< uint, uint > > [coordinates\\_locked](#)

### 7.23.1 Detailed Description

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
class PowerSet< Array_Type, Data_Rule_Type >
```

Powerset of a binary array.

#### Template Parameters

<i>Array_Type</i>	
<i>Data_Rule_Type</i>	

Definition at line 17 of file powerset-bones.hpp.

## 7.23.2 Constructor & Destructor Documentation

### 7.23.2.1 PowerSet() [1/3]

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
PowerSet< Array_Type, Data_Rule_Type >::PowerSet ( ) [inline]
```

Definition at line 39 of file powerset-bones.hpp.

### 7.23.2.2 PowerSet() [2/3]

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
PowerSet< Array_Type, Data_Rule_Type >::PowerSet (
    uint N_,
    uint M_ ) [inline]
```

Definition at line 41 of file powerset-bones.hpp.

### 7.23.2.3 PowerSet() [3/3]

```
template<typename Array_Type , typename Data_Rule_Type >
PowerSet< Array_Type, Data_Rule_Type >::PowerSet (
    const Array_Type & array ) [inline]
```

Definition at line 7 of file powerset-meat.hpp.

### 7.23.2.4 ~PowerSet()

```
template<typename Array_Type , typename Data_Rule_Type >
PowerSet< Array_Type, Data_Rule_Type >::~~PowerSet [inline]
```

Definition at line 15 of file powerset-meat.hpp.

## 7.23.3 Member Function Documentation

### 7.23.3.1 add\_rule() [1/3]

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::add_rule (
    Rule< Array_Type, Data_Rule_Type > & rule ) [inline]
```

Definition at line 113 of file powerset-meat.hpp.

### 7.23.3.2 add\_rule() [2/3]

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::add_rule (
    Rule< Array_Type, Data_Rule_Type > * rule ) [inline]
```

Definition at line 122 of file powerset-meat.hpp.

### 7.23.3.3 add\_rule() [3/3]

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::add_rule (
    Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_,
    Data_Rule_Type * data_ = nullptr,
    bool delete_data_ = false ) [inline]
```

Definition at line 132 of file powerset-meat.hpp.

### 7.23.3.4 begin()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type >::iterator PowerSet< Array_Type, Data_Rule_Type >::begin ( ) [inline]
```

Definition at line 73 of file powerset-bones.hpp.

### 7.23.3.5 calc()

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::calc [inline]
```

Definition at line 88 of file powerset-meat.hpp.

### 7.23.3.6 end()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type >::iterator PowerSet< Array_Type, Data_Rule_Type >::end ( ) [inline]
```

Definition at line 74 of file powerset-bones.hpp.

### 7.23.3.7 get\_data()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type > PowerSet< Array_Type, Data_Rule_Type >::get_data ( ) const [inline]
```

Definition at line 72 of file powerset-bones.hpp.

### 7.23.3.8 get\_data\_ptr()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
const std::vector< Array_Type >* PowerSet< Array_Type, Data_Rule_Type >::get_data_ptr ( )
const [inline]
```

Definition at line 71 of file powerset-bones.hpp.

### 7.23.3.9 init\_support()

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::init_support [inline]
```

Definition at line 21 of file powerset-meat.hpp.

### 7.23.3.10 operator[]()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
const Array_Type& PowerSet< Array_Type, Data_Rule_Type >::operator[] (
    const unsigned int & i ) const [inline]
```

Definition at line 76 of file powerset-bones.hpp.

### 7.23.3.11 reset()

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::reset (
    uint N_,
    uint M_ ) [inline]
```

Definition at line 101 of file powerset-meat.hpp.

### 7.23.3.12 size()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::size_t PowerSet< Array_Type, Data_Rule_Type >::size ( ) const [inline], [noexcept]
```

Definition at line 75 of file powerset-bones.hpp.

## 7.23.4 Member Data Documentation

### 7.23.4.1 coordinates\_free

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< std::pair<uint,uint> > PowerSet< Array_Type, Data_Rule_Type >::coordinates_free
```

Definition at line 31 of file powerset-bones.hpp.

### 7.23.4.2 coordinates\_locked

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< std::pair<uint,uint> > PowerSet< Array_Type, Data_Rule_Type >::coordinates_↵
locked
```

Definition at line 32 of file powerset-bones.hpp.

### 7.23.4.3 data

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type > PowerSet< Array_Type, Data_Rule_Type >::data
```

Definition at line 24 of file powerset-bones.hpp.

#### 7.23.4.4 EmptyArray

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
Array_Type PowerSet< Array_Type, Data_Rule_Type >::EmptyArray
```

Definition at line 23 of file powerset-bones.hpp.

#### 7.23.4.5 M

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
uint PowerSet< Array_Type, Data_Rule_Type >::M
```

Definition at line 27 of file powerset-bones.hpp.

#### 7.23.4.6 N

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
uint PowerSet< Array_Type, Data_Rule_Type >::N
```

Definition at line 27 of file powerset-bones.hpp.

#### 7.23.4.7 rules

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
Rules<Array_Type,Data_Rule_Type>* PowerSet< Array_Type, Data_Rule_Type >::rules
```

Definition at line 25 of file powerset-bones.hpp.

#### 7.23.4.8 rules\_deleted

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
bool PowerSet< Array_Type, Data_Rule_Type >::rules_deleted = false
```

Definition at line 28 of file powerset-bones.hpp.

The documentation for this class was generated from the following files:

- [include/barry/powerset-bones.hpp](#)
- [include/barry/powerset-meat.hpp](#)



## 7.24 Rule< Array\_Type, Data\_Type > Class Template Reference

[Rule](#) for determining if a cell should be included in a sequence.

```
#include <rules-bones.hpp>
```

### Public Member Functions

- [~Rule](#) ()
- Data\_Type \* [D](#) ()  
*Read/Write access to the data.*
- bool [operator\(\)](#) (const Array\_Type &a, [uint](#) i, [uint](#) j)

#### Construct a new Rule object

Construct a new [Rule](#) object

#### Parameters

fun_	A function of type <i>Rule_fun_type</i> .
dat_	Data pointer to be passed to <i>fun_</i>
delete_↔ dat_	When <i>true</i> , the <a href="#">Rule</a> destructor will delete the pointer, if defined.

- [Rule](#) ()
- [Rule](#) ([Rule\\_fun\\_type](#)< Array\_Type, Data\_Type > fun\_, Data\_Type \*dat\_=nullptr, bool delete\_dat\_=false)

### 7.24.1 Detailed Description

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
class Rule< Array_Type, Data_Type >
```

[Rule](#) for determining if a cell should be included in a sequence.

[Rules](#) can be used together with [Support](#) and [PowerSet](#) to determine which cells should be included when enumerating all possible realizations of a binary array.

#### Template Parameters

<i>Array_Type</i>	An object of class <a href="#">BArray</a> .
<i>Data_Type</i>	Any type.

Definition at line 23 of file rules-bones.hpp.

### 7.24.2 Constructor & Destructor Documentation

### 7.24.2.1 Rule() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Rule< Array_Type, Data_Type >::Rule ( ) [inline]
```

Definition at line 42 of file rules-bones.hpp.

### 7.24.2.2 Rule() [2/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Rule< Array_Type, Data_Type >::Rule (
    Rule_fun_type< Array_Type, Data_Type > fun_,
    Data_Type * dat_ = nullptr,
    bool delete_dat_ = false ) [inline]
```

Definition at line 43 of file rules-bones.hpp.

### 7.24.2.3 ~Rule()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Rule< Array_Type, Data_Type >::~Rule ( ) [inline]
```

Definition at line 50 of file rules-bones.hpp.

## 7.24.3 Member Function Documentation

### 7.24.3.1 D()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Data_Type* Rule< Array_Type, Data_Type >::D ( )
```

Read/Write access to the data.

### 7.24.3.2 operator()()

```
template<typename Array_Type , typename Data_Type >
bool Rule< Array_Type, Data_Type >::operator() (
    const Array_Type & a,
    uint i,
    uint j ) [inline]
```

Definition at line 63 of file rules-meat.hpp.

The documentation for this class was generated from the following files:

- include/barry/rules-bones.hpp
- include/barry/rules-meat.hpp

## 7.25 Rules< Array\_Type, Data\_Type > Class Template Reference

Vector of objects of class [Rule](#).

```
#include <rules-bones.hpp>
```

### Public Member Functions

- [Rules](#) ()
- [Rules](#) (const [Rules](#)< Array\_Type, Data\_Type > &rules\_)
- [Rules](#)< Array\_Type, Data\_Type > [operator=](#) (const [Rules](#)< Array\_Type, Data\_Type > &rules\_)
- [~Rules](#) ()
- [uint size](#) () const noexcept
- bool [operator\(\)](#) (const Array\_Type &a, [uint](#) i, [uint](#) j)  
*Check whether a given cell is free or locked.*
- void [clear](#) ()
- void [get\\_seq](#) (const Array\_Type &a, std::vector< std::pair< [uint](#), [uint](#) > > \*free, std::vector< std::pair< [uint](#), [uint](#) > > \*locked=nullptr)  
*Computes the sequence of free and locked cells in an [BArray](#).*

### Rule adding

#### Parameters

rule	
------	--

- void [add\\_rule](#) ([Rule](#)< Array\_Type, Data\_Type > &rule)
- void [add\\_rule](#) ([Rule](#)< Array\_Type, Data\_Type > \*rule)
- void [add\\_rule](#) ([Rule\\_fun\\_type](#)< Array\_Type, Data\_Type > rule\_, Data\_Type \*data\_=nullptr, bool delete↵\_data\_=false)

### 7.25.1 Detailed Description

```
template<typename Array_Type, typename Data_Type>
class Rules< Array_Type, Data_Type >
```

Vector of objects of class [Rule](#).

#### Template Parameters

<i>Array_Type</i>	An object of class <a href="#">BArray</a>
<i>Data_Type</i>	Any type.

Definition at line 69 of file rules-bones.hpp.

### 7.25.2 Constructor & Destructor Documentation

### 7.25.2.1 Rules() [1/2]

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::Rules ( ) [inline]
```

Definition at line 76 of file rules-bones.hpp.

### 7.25.2.2 Rules() [2/2]

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::Rules (
    const Rules< Array_Type, Data_Type > & rules_ ) [inline]
```

Definition at line 10 of file rules-meat.hpp.

### 7.25.2.3 ~Rules()

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::~Rules ( ) [inline]
```

Definition at line 81 of file rules-bones.hpp.

## 7.25.3 Member Function Documentation

### 7.25.3.1 add\_rule() [1/3]

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::add_rule (
    Rule< Array_Type, Data_Type > & rule ) [inline]
```

Definition at line 68 of file rules-meat.hpp.

### 7.25.3.2 add\_rule() [2/3]

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::add_rule (
    Rule< Array_Type, Data_Type > * rule ) [inline]
```

Definition at line 79 of file rules-meat.hpp.

**7.25.3.3 add\_rule()** [3/3]

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::add_rule (
    Rule_fun_type< Array_Type, Data_Type > rule_,
    Data_Type * data_ = nullptr,
    bool delete_data_ = false ) [inline]
```

Definition at line 89 of file rules-meat.hpp.

**7.25.3.4 clear()**

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::clear [inline]
```

Definition at line 127 of file rules-meat.hpp.

**7.25.3.5 get\_seq()**

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::get_seq (
    const Array_Type & a,
    std::vector< std::pair< uint, uint > > * free,
    std::vector< std::pair< uint, uint > > * locked = nullptr ) [inline]
```

Computes the sequence of free and locked cells in an [BArray](#).

**Parameters**

<i>a</i>	An object of class <a href="#">BArray</a> .
<i>free</i>	Pointer to a vector of pairs (i, j) listing the free cells.
<i>locked</i>	(optional) Pointer to a vector of pairs (i, j) listing the locked cells.

**Returns**

Nothing.

Definition at line 139 of file rules-meat.hpp.

**7.25.3.6 operator()**

```
template<typename Array_Type , typename Data_Type >
bool Rules< Array_Type, Data_Type >::operator() (
```

```
const Array_Type & a,  
uint i,  
uint j ) [inline]
```

Check whether a given cell is free or locked.

## Parameters

<i>a</i>	A <a href="#">BArray</a> object
<i>i</i>	row position
<i>j</i>	col position

## Returns

true If the cell is locked  
false If the cell is free

Definition at line 111 of file rules-meat.hpp.

## 7.25.3.7 operator=()

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type > Rules< Array_Type, Data_Type >::operator= (
    const Rules< Array_Type, Data_Type > & rules_ )
```

Definition at line 35 of file rules-meat.hpp.

## 7.25.3.8 size()

```
template<typename Array_Type , typename Data_Type >
uint Rules< Array_Type, Data_Type >::size ( ) const [inline], [noexcept]
```

Definition at line 86 of file rules-bones.hpp.

The documentation for this class was generated from the following files:

- include/barry/rules-bones.hpp
- include/barry/rules-meat.hpp

## 7.26 StatsCounter&lt; Array\_Type, Data\_Type &gt; Class Template Reference

Count stats for a single Array.

```
#include <statscounter-bones.hpp>
```

## Public Member Functions

- [StatsCounter](#) (const Array\_Type \*Array\_)  
*Creator of a [StatsCounter](#)*
- [StatsCounter](#) ()  
*Can be created without setting the array.*
- [~StatsCounter](#) ()
- void [reset\\_array](#) (const Array\_Type \*Array\_)  
*Changes the reference array for the counting.*
- void [add\\_counter](#) (Counter< Array\_Type, Data\_Type > \*f\_)
- void [add\\_counter](#) (Counter< Array\_Type, Data\_Type > f\_)
- void [set\\_counters](#) (Counters< Array\_Type, Data\_Type > \*counters\_)
- void [count\\_init](#) (uint i, uint j)  
*[Counter](#) functions This function recurses through the entries of *Array* and at each step of adding a new cell it uses the functions to list the statistics.*
- void [count\\_current](#) (uint i, uint j)
- std::vector< double > [count\\_all](#) ()
- Counters< Array\_Type, Data\_Type > \* [get\\_counters](#) ()

### 7.26.1 Detailed Description

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
class StatsCounter< Array_Type, Data_Type >
```

Count stats for a single Array.

Users can a list of functions that can be used with this. The baseline set of arguments is a pointer to a binary array and a dataset to add the counts to.

Definition at line 16 of file statscounter-bones.hpp.

### 7.26.2 Constructor & Destructor Documentation

#### 7.26.2.1 StatsCounter() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
StatsCounter< Array_Type, Data_Type >::StatsCounter (
    const Array_Type * Array_ ) [inline]
```

Creator of a [StatsCounter](#)

Parameters

<i>Array</i> ↔	A const pointer to a <a href="#">BArray</a> .
—	

Definition at line 36 of file statscounter-bones.hpp.



### 7.26.2.2 StatsCounter() [2/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
StatsCounter< Array_Type, Data_Type >::StatsCounter ( ) [inline]
```

Can be created without setting the array.

Definition at line 51 of file statscounter-bones.hpp.

### 7.26.2.3 ~StatsCounter()

```
template<typename Array_Type , typename Data_Type >
StatsCounter< Array_Type, Data_Type >::~~StatsCounter [inline]
```

Definition at line 7 of file statscounter-meat.hpp.

## 7.26.3 Member Function Documentation

### 7.26.3.1 add\_counter() [1/2]

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::add_counter (
    Counter< Array_Type, Data_Type > * f_ ) [inline]
```

Definition at line 25 of file statscounter-meat.hpp.

### 7.26.3.2 add\_counter() [2/2]

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::add_counter (
    Counter< Array_Type, Data_Type > f_ ) [inline]
```

Definition at line 35 of file statscounter-meat.hpp.

### 7.26.3.3 count\_all()

```
template<typename Array_Type , typename Data_Type >
std::vector< double > StatsCounter< Array_Type, Data_Type >::count_all [inline]
```

Definition at line 99 of file statscounter-meat.hpp.

### 7.26.3.4 count\_current()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::count_current (
    uint i,
    uint j ) [inline]
```

Definition at line 81 of file statscounter-meat.hpp.

### 7.26.3.5 count\_init()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::count_init (
    uint i,
    uint j ) [inline]
```

**Counter** functions This function recurses through the entries of `Array` and at each step of adding a new cell it uses the functions to list the statistics.

Definition at line 61 of file statscounter-meat.hpp.

### 7.26.3.6 get\_counters()

```
template<typename Array_Type , typename Data_Type >
Counters< Array_Type, Data_Type > * StatsCounter< Array_Type, Data_Type >::get_counters [inline]
```

Definition at line 139 of file statscounter-meat.hpp.

### 7.26.3.7 reset\_array()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::reset_array (
    const Array_Type * Array_ ) [inline]
```

Changes the reference array for the counting.

#### Parameters

<code>Array_↵</code> —	A pointer to an array of class <code>Array_Type</code> .
---------------------------	--

Definition at line 14 of file statscounter-meat.hpp.

#### 7.26.3.8 set\_counters()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::set_counters (
    Counters< Array_Type, Data_Type > * counters_ ) [inline]
```

Definition at line 46 of file statscounter-meat.hpp.

The documentation for this class was generated from the following files:

- include/barry/statscounter-bones.hpp
- include/barry/statscounter-meat.hpp

## 7.27 Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type > Class Template Reference

Compute the support of sufficient statistics.

```
#include <support-bones.hpp>
```

### Public Member Functions

- [Support](#) (const `Array_Type` &`Array_`)  
*Constructor passing a reference Array.*
- [Support](#) (uint `N_`, uint `M_`)  
*Constructor specifying the dimensions of the array (empty).*
- [Support](#) ()
- [~Support](#) ()
- void [init\\_support](#) (std::vector< `Array_Type` > \*`array_bank`=nullptr, std::vector< std::vector< double > > \*`stats_bank`=nullptr)
- void [calc](#) (std::vector< `Array_Type` > \*`array_bank`=nullptr, std::vector< std::vector< double > > \*`stats_↵` bank=nullptr, unsigned int `max_num_elements_`=0u)  
*Computes the entire support.*
- [Counts\\_type](#) [get\\_counts](#) () const
- const [MapVec\\_type](#) \* [get\\_counts\\_ptr](#) () const
- std::vector< double > \* [get\\_current\\_stats](#) ()  
*List current statistics.*
- void [print](#) () const
- const [FreqTable](#) & [get\\_data](#) () const
- [Counters](#)< `Array_Type`, `Data_Counter_Type` > \* [get\\_counters](#) ()

*Vector of 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](#)

### 7.27.1 Detailed Description

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename
Data_Rule_Dyn_Type = bool>
class Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >
```

Compute the support of sufficient statistics.

Given an array and a set of counters, this object iterates throughout the support set of the Array while at the same time computing the support of the sufficient 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.

### 7.27.2 Constructor & Destructor Documentation

#### 7.27.2.1 Support() [1/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Support (
    const Array_Type & Array_ ) [inline]
```

Constructor passing a reference Array.

Definition at line 69 of file support-bones.hpp.

#### 7.27.2.2 Support() [2/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Support (
    uint N_,
    uint M_ ) [inline]
```

Constructor specifying the dimensions of the array (empty).

Definition at line 78 of file support-bones.hpp.

### 7.27.2.3 Support() [3/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Support ( )
[inline]
```

Definition at line 85 of file support-bones.hpp.

### 7.27.2.4 ~Support()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::~~Support ( )
[inline]
```

Definition at line 92 of file support-bones.hpp.

## 7.27.3 Member Function Documentation

### 7.27.3.1 add\_counter() [1/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_counter
(
    Counter< Array_Type, Data_Counter_Type > * f_ ) [inline]
```

Definition at line 215 of file support-meat.hpp.

### 7.27.3.2 add\_counter() [2/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_counter
(
    Counter< Array_Type, Data_Counter_Type > f_ ) [inline]
```

Definition at line 225 of file support-meat.hpp.

**7.27.3.3 add\_rule() [1/2]**

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
    Rule< Array_Type, Data_Rule_Type > * f_ ) [inline]
```

Definition at line 252 of file support-meat.hpp.

**7.27.3.4 add\_rule() [2/2]**

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
    Rule< Array_Type, Data_Rule_Type > f_ ) [inline]
```

Definition at line 262 of file support-meat.hpp.

**7.27.3.5 add\_rule\_dyn() [1/2]**

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_↵
dyn (
    Rule< Array_Type, Data_Rule_Dyn_Type > * f_ ) [inline]
```

Definition at line 287 of file support-meat.hpp.

**7.27.3.6 add\_rule\_dyn() [2/2]**

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_↵
dyn (
    Rule< Array_Type, Data_Rule_Dyn_Type > f_ ) [inline]
```

Definition at line 297 of file support-meat.hpp.

**7.27.3.7 calc()**

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::calc (
    std::vector< Array_Type > * array_bank = nullptr,
    std::vector< std::vector< double > > * stats_bank = nullptr,
    unsigned int max_num_elements_ = 0u ) [inline]
```

Computes the entire support.

Not to be used by the user. Sets the starting point in the array (column-major).



#### Parameters

<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 189 of file support-meat.hpp.

#### 7.27.3.8 get\_counters()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
Counters< Array_Type, Data_Counter_Type > * Support< Array_Type, Data_Counter_Type, Data_↵
Rule_Type, Data_Rule_Dyn_Type >::get_counters [inline]
```

Vector of couter functions.

Definition at line 360 of file support-meat.hpp.

#### 7.27.3.9 get\_counts()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
Counts_type Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >↵
::get_counts [inline]
```

Definition at line 324 of file support-meat.hpp.

#### 7.27.3.10 get\_counts\_ptr()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
const MapVec_type * Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type
>::get_counts_ptr [inline]
```

Definition at line 331 of file support-meat.hpp.

#### 7.27.3.11 get\_current\_stats()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
std::vector< double > * Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_↵
Dyn_Type >::get_current_stats [inline]
```

List current statistics.

Definition at line 338 of file support-meat.hpp.

**7.27.3.12 get\_data()**

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
const FreqTable & Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type
>::get_data [inline]
```

Definition at line 355 of file support-meat.hpp.

**7.27.3.13 get\_rules()**

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
Rules< Array_Type, Data_Rule_Type > * Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_rules [inline]
```

Vector of static rules (cells to iterate).

Definition at line 365 of file support-meat.hpp.

**7.27.3.14 get\_rules\_dyn()**

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
Rules< Array_Type, Data_Rule_Dyn_Type > * Support< Array_Type, Data_Counter_Type, Data_Rule_↵
_Type, Data_Rule_Dyn_Type >::get_rules_dyn [inline]
```

Vector of dynamic rules (to include/exclude a realization).

Definition at line 370 of file support-meat.hpp.

**7.27.3.15 init\_support()**

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::init_↵
support (
    std::vector< Array_Type > * array_bank = nullptr,
    std::vector< std::vector< double > > * stats_bank = nullptr ) [inline]
```

Definition at line 7 of file support-meat.hpp.

#### 7.27.3.16 print()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::print
[inline]
```

Definition at line 343 of file support-meat.hpp.

#### 7.27.3.17 reset\_array() [1/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::reset_array
[inline]
```

Definition at line 84 of file support-meat.hpp.

#### 7.27.3.18 reset\_array() [2/2]

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::reset_array
(
    const Array_Type & Array_ ) [inline]
```

Definition at line 91 of file support-meat.hpp.

#### 7.27.3.19 set\_counters()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_↵
counters (
    Counters< Array_Type, Data_Counter_Type > * counters_ ) [inline]
```

Definition at line 235 of file support-meat.hpp.

#### 7.27.3.20 set\_rules()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_rules (
    Rules< Array_Type, Data_Rule_Type > * rules_ ) [inline]
```

Definition at line 272 of file support-meat.hpp.

### 7.27.3.21 set\_rules\_dyn()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_rules_dyn(
    Rules< Array_Type, Data_Rule_Dyn_Type > * rules_ ) [inline]
```

Definition at line 307 of file support-meat.hpp.

## 7.27.4 Member Data Documentation

### 7.27.4.1 change\_stats

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool,
typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector< double > > Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::change_stats
```

Definition at line 65 of file support-bones.hpp.

### 7.27.4.2 coordinates\_free

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool,
typename Data_Rule_Dyn_Type = bool>
std::vector< std::pair<uint, uint> > Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::coordinates_free
```

Definition at line 63 of file support-bones.hpp.

### 7.27.4.3 coordinates\_locked

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool,
typename Data_Rule_Dyn_Type = bool>
std::vector< std::pair<uint, uint> > Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::coordinates_locked
```

Definition at line 64 of file support-bones.hpp.

#### 7.27.4.4 current\_stats

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< double > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_↵
_Type >::current_stats
```

Definition at line 62 of file support-bones.hpp.

#### 7.27.4.5 delete\_counters

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↵
counters = true
```

Definition at line 56 of file support-bones.hpp.

#### 7.27.4.6 delete\_rules

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↵
rules = true
```

Definition at line 57 of file support-bones.hpp.

#### 7.27.4.7 delete\_rules\_dyn

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↵
rules_dyn = true
```

Definition at line 58 of file support-bones.hpp.

#### 7.27.4.8 M

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::M
```

Definition at line 55 of file support-bones.hpp.

#### 7.27.4.9 max\_num\_elements

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::max_num_elements = BARRY_MAX_NUM_ELEMENTS
```

Definition at line 59 of file support-bones.hpp.

#### 7.27.4.10 N

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::N
```

Definition at line 55 of file support-bones.hpp.

The documentation for this class was generated from the following files:

- [include/barry/support-bones.hpp](#)
- [include/barry/support-meat.hpp](#)

## 7.28 vecHasher< T > Struct Template Reference

```
#include <typedefs.hpp>
```

### Public Member Functions

- `std::size_t operator() (std::vector< T > const &dat) const` noexcept

#### 7.28.1 Detailed Description

```
template<typename T>
struct vecHasher< T >
```

Definition at line 86 of file typedefs.hpp.

#### 7.28.2 Member Function Documentation

##### 7.28.2.1 operator()()

```
template<typename T >
std::size_t vecHasher< T >::operator() (
    std::vector< T > const & dat ) const [inline], [noexcept]
```

Definition at line 87 of file typedefs.hpp.

The documentation for this struct was generated from the following file:

- [include/barry/typedefs.hpp](#)

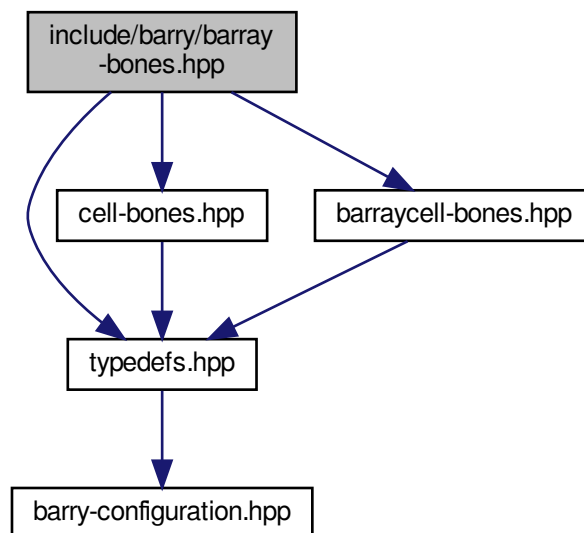
## Chapter 8

# File Documentation

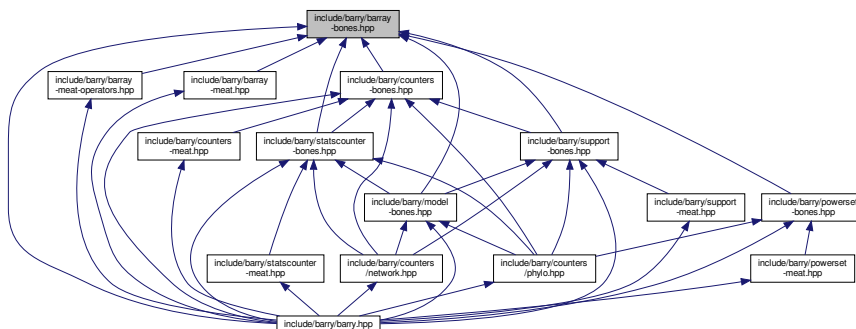
### 8.1 include/barry/barray-bones.hpp File Reference

```
#include "typedefs.hpp"  
#include "cell-bones.hpp"  
#include "barraycell-bones.hpp"
```

Include dependency graph for barray-bones.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [BArray< Cell\\_Type, Data\\_Type >](#)  
*Baseline class for binary arrays.*

## Macros

- `#define BARRAY\_BONES\_HPP 1`

### 8.1.1 Macro Definition Documentation

#### 8.1.1.1 BARRAY\_BONES\_HPP

```
#define BARRAY_BONES_HPP 1
```

Definition at line 8 of file `barray-bones.hpp`.

## 8.2 include/barry/barray-iterator.hpp File Reference

### Classes

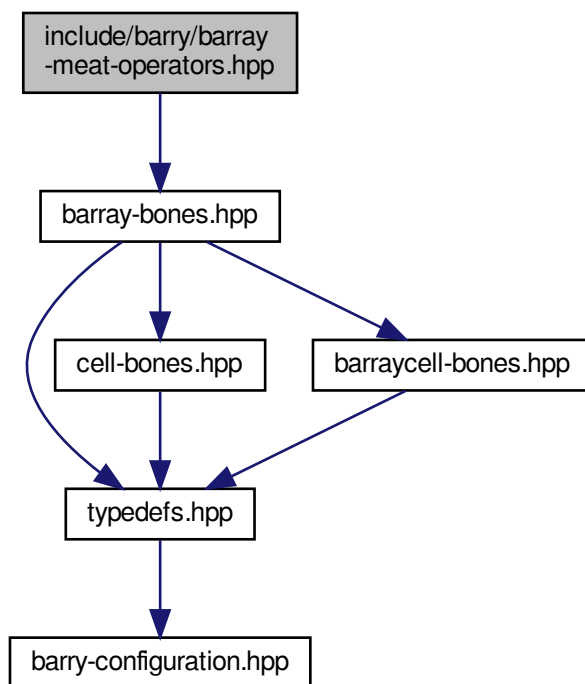
- class [ConstBArrayRowIter< Cell\\_Type, Data\\_Type >](#)



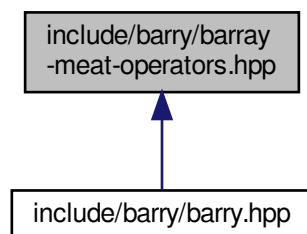
## 8.3 include/barry/barray-meat-operators.hpp File Reference

```
#include "barray-bones.hpp"
```

Include dependency graph for barray-meat-operators.hpp:



This graph shows which files directly or indirectly include this file:



### Macros

- `#define BARRY_BARRAY_MEAT_OPERATORS_HPP 1`
- `#define ROW(a) this->el_ij[a]`
- `#define COL(a) this->el_ji[a]`

## Functions

- `template<typename Cell_Type , typename Data_Type >`  
`void checkdim_ (const BArray< Cell_Type, Data_Type > &lhs, const BArray< Cell_Type, Data_Type > &rhs)`

### 8.3.1 Macro Definition Documentation

#### 8.3.1.1 BARRY\_BARRAY\_MEAT\_OPERATORS\_HPP

```
#define BARRY_BARRAY_MEAT_OPERATORS_HPP 1
```

Definition at line 5 of file `barray-meat-operators.hpp`.

#### 8.3.1.2 COL

```
#define COL(  
    a ) this->el_ji[a]
```

Definition at line 8 of file `barray-meat-operators.hpp`.

#### 8.3.1.3 ROW

```
#define ROW(  
    a ) this->el_ij[a]
```

Definition at line 7 of file `barray-meat-operators.hpp`.

### 8.3.2 Function Documentation

#### 8.3.2.1 checkdim\_()

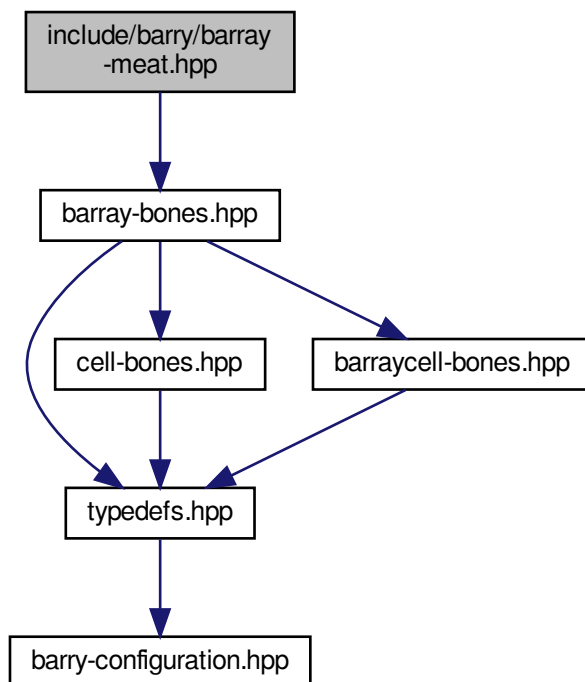
```
template<typename Cell_Type , typename Data_Type >  
void checkdim_ (  
    const BArray< Cell_Type, Data_Type > & lhs,  
    const BArray< Cell_Type, Data_Type > & rhs ) [inline]
```

Definition at line 11 of file `barray-meat-operators.hpp`.

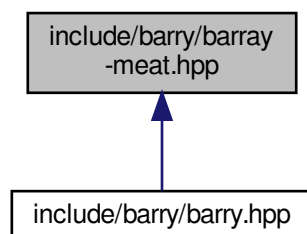
## 8.4 include/barry/barray-meat.hpp File Reference

```
#include "barray-bones.hpp"
```

Include dependency graph for barray-meat.hpp:



This graph shows which files directly or indirectly include this file:



### Macros

- `#define ROW(a) this->el_ij[a]`
- `#define COL(a) this->el_ji[a]`

## 8.4.1 Macro Definition Documentation

### 8.4.1.1 COL

```
#define COL(  
    a ) this->el_ji[a]
```

Definition at line 8 of file `barray-meat.hpp`.

### 8.4.1.2 ROW

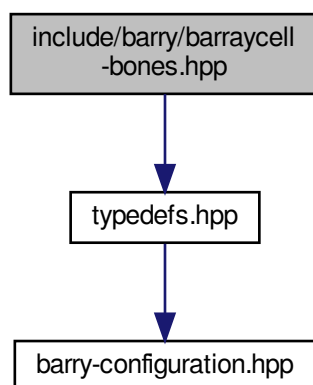
```
#define ROW(  
    a ) this->el_ij[a]
```

Definition at line 7 of file `barray-meat.hpp`.

## 8.5 `include/barry/barraycell-bones.hpp` File Reference

```
#include "typedefs.hpp"
```

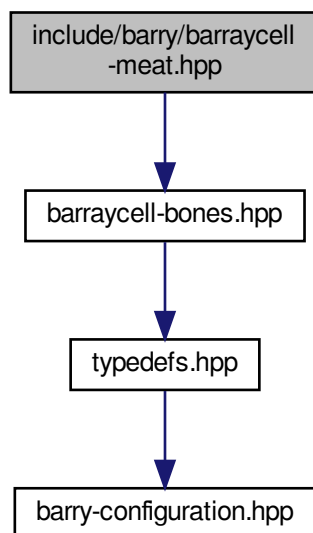
Include dependency graph for `barraycell-bones.hpp`:



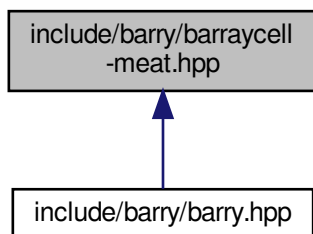
[illegible]

- class BArrayCell< Cell\_Type, Data\_Type >
- class BArrayCell\_const< Cell\_Type, Data\_Type >

```
#include "barraycell-bones.hpp"
Include dependency graph for barraycell-meat.hpp:
```



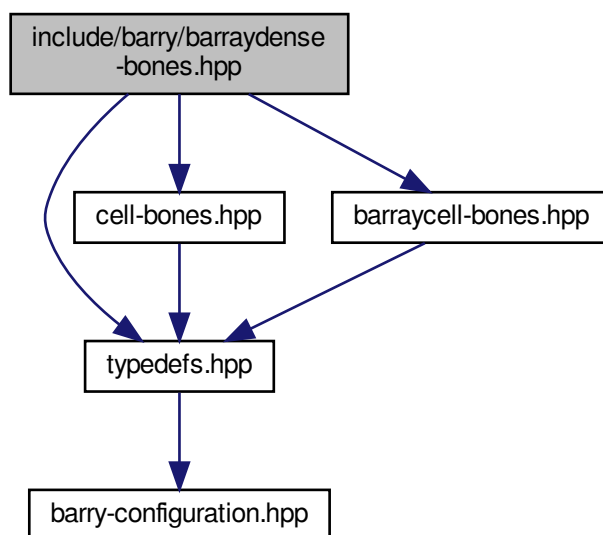
This graph shows which files directly or indirectly include this file:



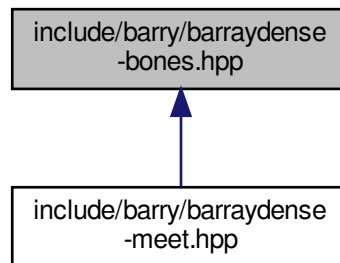
## 8.7 include/barry/barraydense-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "cell-bones.hpp"
#include "barraycell-bones.hpp"
```

Include dependency graph for barraydense-bones.hpp:



This graph shows which files directly or indirectly include this file:



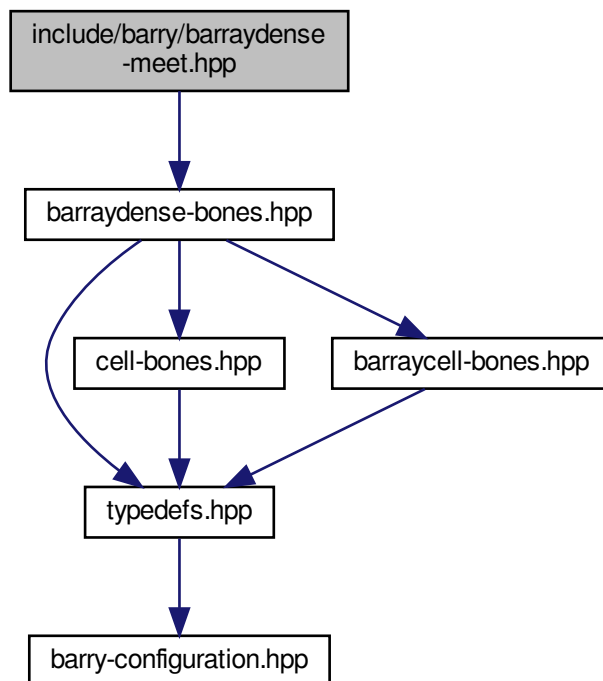
## Classes

- class [BArrayDense](#)< [Cell\\_Type](#), [Data\\_Type](#) >  
*Baseline class for binary arrays.*

## 8.8 include/barry/barraydense-meet.hpp File Reference

```
#include "barraydense-bones.hpp"
```

Include dependency graph for `barrydense-meet.hpp`:



## Macros

- `#define BARRY_BARRAYDENSE_MEAT_HPP`
- `#define ROW(a) this->el_ij[a]`
- `#define COL(a) this->el_ji[a]`
- `#define POS(a, b) (b)*N + (a)`
- `#define ZERO_CELL Cell< Cell_Type >(static_cast< Cell_Type >(0.0))`

## 8.8.1 Macro Definition Documentation

### 8.8.1.1 BARRY\_BARRAYDENSE\_MEAT\_HPP

```
#define BARRY_BARRAYDENSE_MEAT_HPP
```

Definition at line 5 of file `barrydense-meet.hpp`.



### 8.8.1.2 COL

```
#define COL(  
    a ) this->el_ji[a]
```

Definition at line 8 of file barraydense-meet.hpp.

### 8.8.1.3 POS

```
#define POS(  
    a,  
    b ) (b)*N + (a)
```

Definition at line 9 of file barraydense-meet.hpp.

### 8.8.1.4 ROW

```
#define ROW(  
    a ) this->el_ij[a]
```

Definition at line 7 of file barraydense-meet.hpp.

### 8.8.1.5 ZERO\_CELL

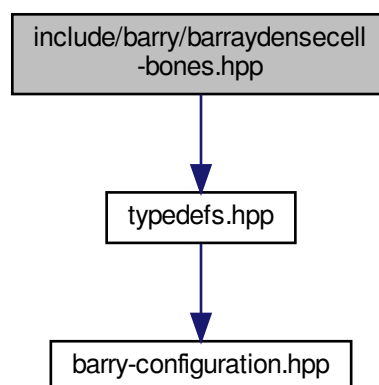
```
#define ZERO_CELL Cell< Cell_Type >(static_cast< Cell_Type >(0.0))
```

Definition at line 14 of file barraydense-meet.hpp.

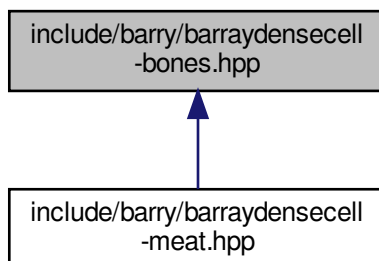
## 8.9 include/barry/barraydensecell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barraydensecell-bones.hpp:



This graph shows which files directly or indirectly include this file:



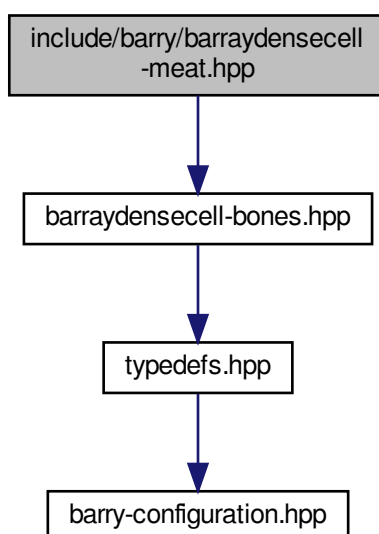
## Classes

- class [BArrayDenseCell< Cell\\_Type, Data\\_Type >](#)
- class [BArrayDenseCell\\_const< Cell\\_Type, Data\\_Type >](#)

## 8.10 include/barry/barraydensecell-meat.hpp File Reference

```
#include "barraydensecell-bones.hpp"
```

Include dependency graph for `barraydensecell-meat.hpp`:



## Macros

- #define `BARRY_BARRAYDENSECELL_MEAT_HPP` 1
- #define `POS(a, b)`  $(a) + (b) * \text{Array->N}$

### 8.10.1 Macro Definition Documentation

#### 8.10.1.1 BARRY\_BARRAYDENSECELL\_MEAT\_HPP

```
#define BARRY_BARRAYDENSECELL_MEAT_HPP 1
```

Definition at line 4 of file `barraydensecell-meat.hpp`.

#### 8.10.1.2 POS

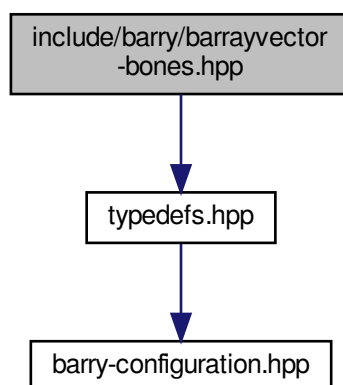
```
#define POS(  
    a,  
    b ) (a) + (b) * Array->N
```

Definition at line 6 of file `barraydensecell-meat.hpp`.

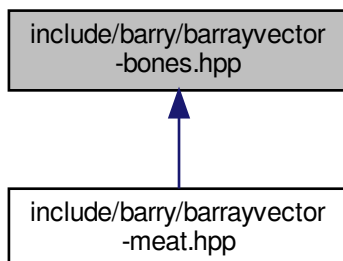
## 8.11 include/barry/barrayvector-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for `barrayvector-bones.hpp`:



This graph shows which files directly or indirectly include this file:



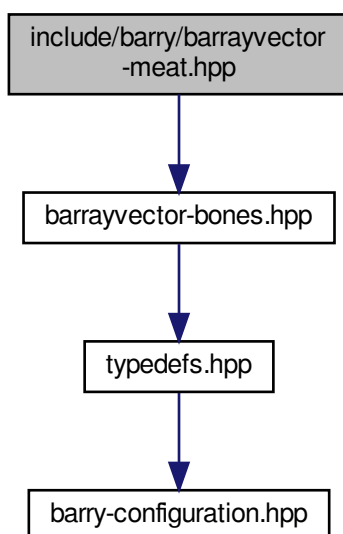
## Classes

- class [BArrayVector< Cell\\_Type, Data\\_Type >](#)  
*Row or column of a [BArray](#)*
- class [BArrayVector\\_const< Cell\\_Type, Data\\_Type >](#)

## 8.12 include/barry/barrayvector-meat.hpp File Reference

```
#include "barrayvector-bones.hpp"
```

Include dependency graph for `barrayvector-meat.hpp`:



## Macros

- `#define BARRY_BARRAYVECTOR_MEAT_HPP` 1

### 8.12.1 Macro Definition Documentation

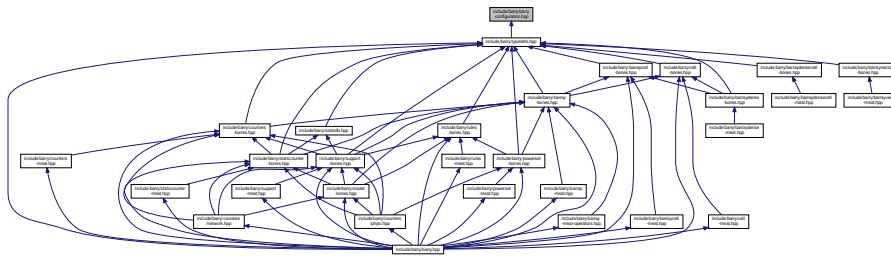
#### 8.12.1.1 BARRY\_BARRAYVECTOR\_MEAT\_HPP

```
#define BARRY_BARRAYVECTOR_MEAT_HPP 1
```

Definition at line 4 of file `barryvector-meat.hpp`.

## 8.13 include/barry/barry-configuration.hpp File Reference

This graph shows which files directly or indirectly include this file:



## Configuration MACROS

These are mostly related to performance. The definitions follow:

- `BARRY_USE_UNORDERED_MAP` If specified, then barry is compiled using `std::unordered_map`. Otherwise it will use `std::map` for the arrays.
- `BARRY_USE_SAFE_EXP` When specified, it will multiply all likelihoods in `Model` by  $(1/100)/(1/100)$  so that numerical overflows are avoided.
- `BARRY_USE_ISFINITE` When specified, it will introduce a macro that checks whether the likelihood is finite or not.
- `printf_barry` If not specified, will be defined as `printf`.
- `#define BARRY_SAFE_EXP -100.0`
- `#define BARRY_ISFINITE(a)`
- `#define BARRY_CHECK_SUPPORT(x, maxs)`
- `#define printf_barry printf`
- `#define BARRY_MAX_NUM_ELEMENTS static_cast< unsigned int >(UINT_MAX/2u)`
- `template<typename Ta, typename Tb >`  
`using Map = std::map< Ta, Tb >`

## 8.13.1 Macro Definition Documentation

### 8.13.1.1 BARRY\_CHECK\_SUPPORT

```
#define BARRY_CHECK_SUPPORT(  
    x,  
    maxs )
```

Definition at line 45 of file barry-configuration.hpp.

### 8.13.1.2 BARRY\_ISFINITE

```
#define BARRY_ISFINITE(  
    a )
```

Definition at line 38 of file barry-configuration.hpp.

### 8.13.1.3 BARRY\_MAX\_NUM\_ELEMENTS

```
#define BARRY_MAX_NUM_ELEMENTS static_cast< unsigned int >(UINT_MAX/2u)
```

Definition at line 53 of file barry-configuration.hpp.

### 8.13.1.4 BARRY\_SAFE\_EXP

```
#define BARRY_SAFE_EXP -100.0
```

Definition at line 31 of file barry-configuration.hpp.

### 8.13.1.5 printf\_barry

```
#define printf_barry printf
```

Definition at line 49 of file barry-configuration.hpp.

## 8.13.2 Typedef Documentation

### 8.13.2.1 Map

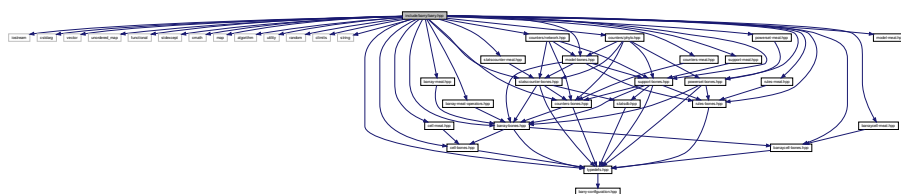
```
template<typename Ta , typename Tb >
using Map = std::map<Ta,Tb>
```

Definition at line 25 of file barry-configuration.hpp.

## 8.14 include/barry/barry.hpp File Reference

```
#include <iostream>
#include <cstdint>
#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 BARRY_VERSION 0.1`
- `#define COUNTER_FUNCTION(a)`
- `#define COUNTER_LAMBDA(a)`
- `#define RULE_FUNCTION(a)`
- `#define RULE_LAMBDA(a)`

### 8.14.1 Macro Definition Documentation

#### 8.14.1.1 BARRY\_HPP

```
#define BARRY_HPP
```

Definition at line 20 of file `barry.hpp`.

#### 8.14.1.2 BARRY\_VERSION

```
#define BARRY_VERSION 0.1
```

Definition at line 22 of file `barry.hpp`.

#### 8.14.1.3 COUNTER\_FUNCTION

```
#define COUNTER_FUNCTION(  
    a )
```

##### Value:

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \  
inline double (a) (const Array_Type & Array, uint i, uint j, Data_Type * data) \  
{
```

Definition at line 73 of file `barry.hpp`.



#### 8.14.1.4 COUNTER\_LAMBDA

```
#define COUNTER_LAMBDA(  
    a )
```

**Value:**

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \  
Counter_fun_type<Array_Type, Data_Type> a = \  
[] (const Array_Type & Array, uint i, uint j, Data_Type * data)
```

Definition at line 76 of file barry.hpp.

#### 8.14.1.5 RULE\_FUNCTION

```
#define RULE_FUNCTION(  
    a )
```

**Value:**

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \  
inline bool (a) (const Array_Type & Array, uint i, uint j, Data_Type * data) \  
{
```

Definition at line 80 of file barry.hpp.

#### 8.14.1.6 RULE\_LAMBDA

```
#define RULE_LAMBDA(  
    a )
```

**Value:**

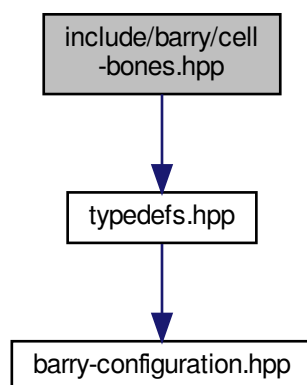
```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \  
Rule_fun_type<Array_Type, Data_Type> a = \  
[] (const Array_Type & Array, uint i, uint j, Data_Type * data)
```

Definition at line 83 of file barry.hpp.

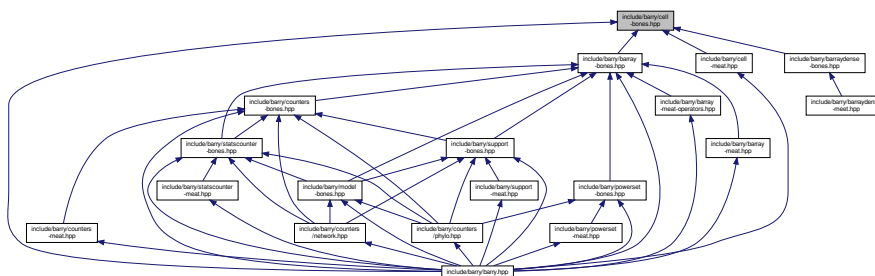
## 8.15 include/barry/cell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for cell-bones.hpp:



This graph shows which files directly or indirectly include this file:



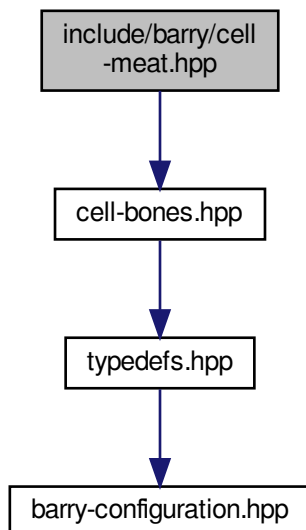
## Classes

- class `Cell< Cell_Type >`  
*Entries in BArray. For now, it only has two members:*

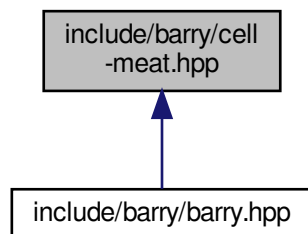
## 8.16 include/barry/cell-meat.hpp File Reference

```
#include "cell-bones.hpp"
```

Include dependency graph for cell-meat.hpp:



This graph shows which files directly or indirectly include this file:



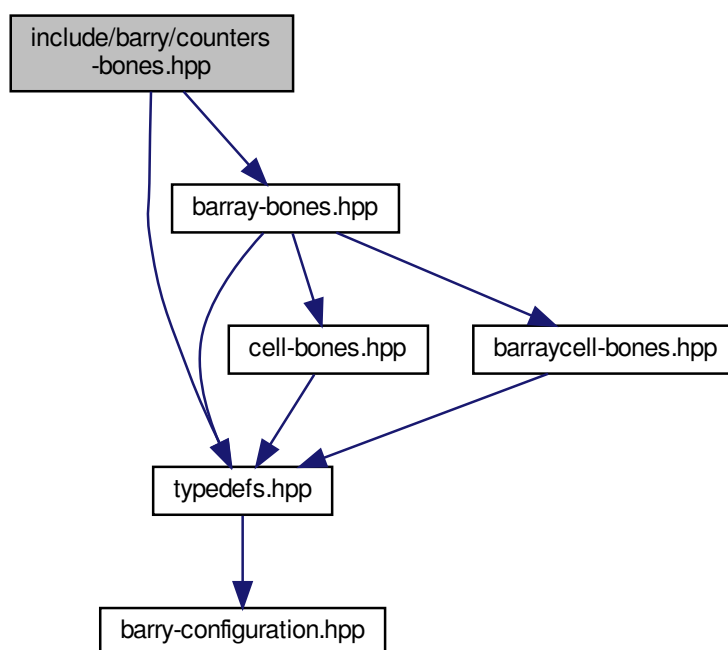
## 8.17 include/barry/col-bones.hpp File Reference

## 8.18 include/barry/counters-bones.hpp File Reference

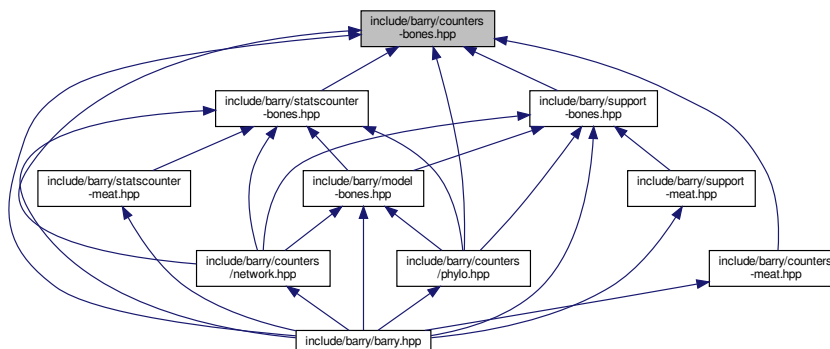
```
#include "typedefs.hpp"
```

```
#include "barray-bones.hpp"
```

Include dependency graph for counters-bones.hpp:



This graph shows which files directly or indirectly include this file:



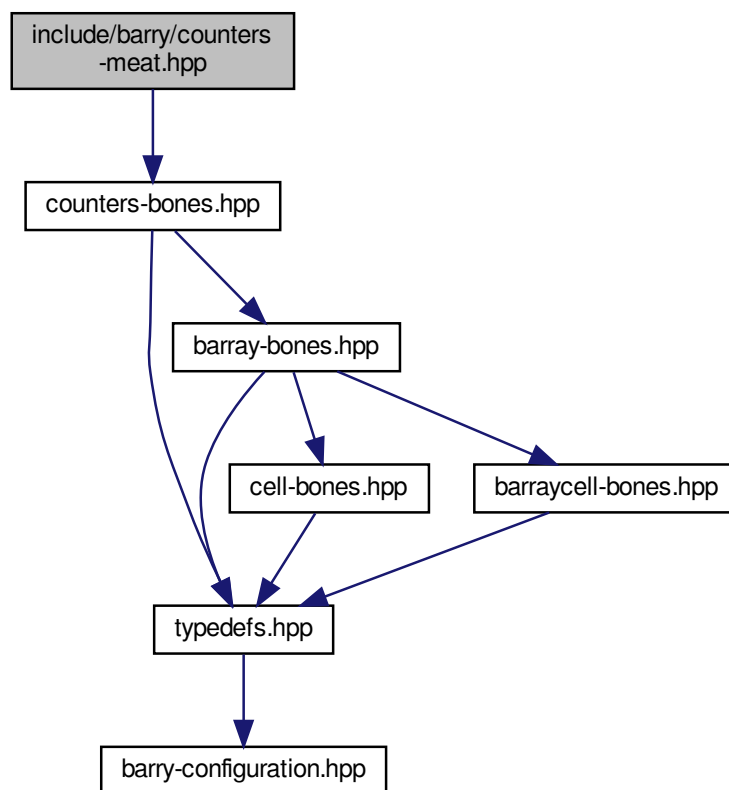
## Classes

- class [Counter< Array\\_Type, Data\\_Type >](#)  
*A counter function based on change statistics.*
- class [Counters< Array\\_Type, Data\\_Type >](#)  
*Vector of counters.*

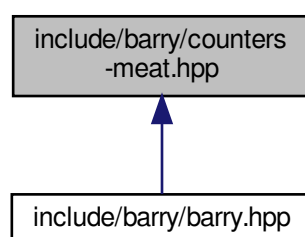
## 8.19 include/barry/counters-meat.hpp File Reference

```
#include "counters-bones.hpp"
```

Include dependency graph for counters-meat.hpp:



This graph shows which files directly or indirectly include this file:



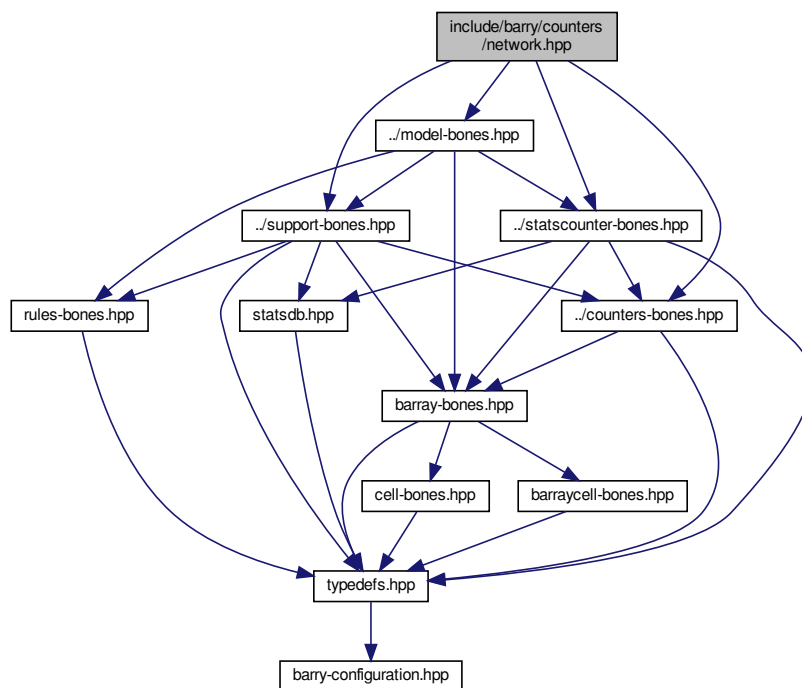
## 8.20 include/barry/counters/network.hpp File Reference

```
#include "../counters-bones.hpp"
#include "../support-bones.hpp"
```

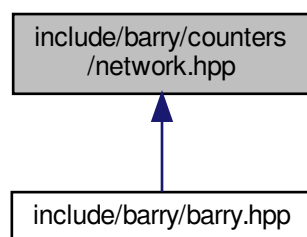
```
#include "../statscounter-bones.hpp"
```

```
#include "../model-bones.hpp"
```

Include dependency graph for network.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [NetworkData](#)  
*Data class for Networks.*
- class [NetCounterData](#)  
*Data class used to store arbitrary uint or double vectors.*

## Macros

- #define `NET_C_DATA_IDX(i)` (`data->indices[i]`)
- #define `NET_C_DATA_NUM(i)` (`data->numbers[i]`)

### Macros for defining counters

- #define `NETWORK_COUNTER(a)`
- #define `NETWORK_COUNTER_LAMBDA(a)`

### Macros for defining rules

- #define `NETWORK_RULE(a)`
- #define `NETWORK_RULE_LAMBDA(a)`

## Typedefs

### Convenient typedefs for network objects.

- typedef `BArray< double, NetworkData > Network`
- typedef `Counter< Network, NetCounterData > NetCounter`
- typedef `Counters< Network, NetCounterData > NetCounters`
- typedef `Support< Network, NetCounterData > NetSupport`
- typedef `StatsCounter< Network, NetCounterData > NetStatsCounter`
- typedef `Model< Network, NetCounterData > NetModel`
- typedef `Rule< Network, bool > NetRule`
- typedef `Rules< Network, bool > NetRules`

## Functions

- void `counter_edges` (`NetCounters *counters`)  
*Number of edges.*
- void `counter_isolates` (`NetCounters *counters`)  
*Number of isolated vertices.*
- void `counter_mutual` (`NetCounters *counters`)  
*Number of mutual ties.*
- void `counter_istar2` (`NetCounters *counters`)
- void `counter_ostar2` (`NetCounters *counters`)
- void `counter_ttriads` (`NetCounters *counters`)
- void `counter_ctriads` (`NetCounters *counters`)
- void `counter_density` (`NetCounters *counters`)
- void `counter_idegree15` (`NetCounters *counters`)
- void `counter_odegree15` (`NetCounters *counters`)
- void `counter_absdiff` (`NetCounters *counters`, `uint attr_id`, `double alpha=1.0`)  
*Sum of absolute attribute difference between ego and alter.*
- void `counter_diff` (`NetCounters *counters`, `uint attr_id`, `double alpha=1.0`, `double tail_head=true`)  
*Sum of attribute difference between ego and alter to pow(alpha)*
- `NETWORK_COUNTER` (`init_single_attr`)
- void `counter_nodeicov` (`NetCounters *counters`, `uint attr_id`)
- void `counter_nodeocov` (`NetCounters *counters`, `uint attr_id`)

- void `counter_nodecov` (`NetCounters` \*counters, `uint` attr\_id)
- void `counter_nodematch` (`NetCounters` \*counters, `uint` attr\_id)
- void `counter_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&lt;Network, bool&gt;</code> ).
-------	--

- void `rules_zerodiag` (`NetRules` \*rules)  
Number of edges.

## 8.20.1 Macro Definition Documentation

### 8.20.1.1 NET\_C\_DATA\_IDX

```
#define NET_C_DATA_IDX(  
    i ) (data->indices[i])
```

Definition at line 79 of file network.hpp.

### 8.20.1.2 NET\_C\_DATA\_NUM

```
#define NET_C_DATA_NUM(  
    i ) (data->numbers[i])
```

Definition at line 80 of file network.hpp.

### 8.20.1.3 NETWORK\_COUNTER

```
#define NETWORK_COUNTER(  
    a )
```

#### Value:

```
inline double (a) \  
(const Network & Array, uint i, uint j, NetCounterData * data)
```

Function for definition of a network counter function

Definition at line 101 of file network.hpp.



#### 8.20.1.4 NETWORK\_COUNTER\_LAMBDA

```
#define NETWORK_COUNTER_LAMBDA(  
    a )
```

**Value:**

```
Counter_fun_type<Network, NetCounterData> a = \  
[] (const Network & Array, uint i, uint j, NetCounterData * data)
```

Lambda function for definition of a network counter function

Definition at line 104 of file network.hpp.

#### 8.20.1.5 NETWORK\_RULE

```
#define NETWORK_RULE(  
    a )
```

**Value:**

```
inline bool (a) \  
(const Network & Array, uint i, uint j, bool * data)
```

Function for definition of a network counter function

Definition at line 113 of file network.hpp.

#### 8.20.1.6 NETWORK\_RULE\_LAMBDA

```
#define NETWORK_RULE_LAMBDA(  
    a )
```

**Value:**

```
Rule_fun_type<Network, bool> a = \  
[] (const Network & Array, uint i, uint j, bool * data)
```

Lambda function for definition of a network counter function

Definition at line 116 of file network.hpp.

### 8.20.2 Typedef Documentation

#### 8.20.2.1 NetCounter

```
typedef Counter<Network, NetCounterData > NetCounter
```

Definition at line 88 of file network.hpp.

#### 8.20.2.2 NetCounters

```
typedef Counters< Network, NetCounterData> NetCounters
```

Definition at line 89 of file network.hpp.

#### 8.20.2.3 NetModel

```
typedef Model<Network, NetCounterData> NetModel
```

Definition at line 92 of file network.hpp.

#### 8.20.2.4 NetRule

```
typedef Rule<Network, bool> NetRule
```

Definition at line 93 of file network.hpp.

#### 8.20.2.5 NetRules

```
typedef Rules<Network, bool> NetRules
```

Definition at line 94 of file network.hpp.

#### 8.20.2.6 NetStatsCounter

```
typedef StatsCounter<Network, NetCounterData> NetStatsCounter
```

Definition at line 91 of file network.hpp.

#### 8.20.2.7 NetSupport

```
typedef Support<Network, NetCounterData > NetSupport
```

Definition at line 90 of file network.hpp.

### 8.20.2.8 Network

```
typedef BArray<double, NetworkData> Network
```

Definition at line 87 of file network.hpp.

## 8.20.3 Function Documentation

### 8.20.3.1 rules\_zerodiag()

```
void rules_zerodiag (
    NetRules * rules ) [inline]
```

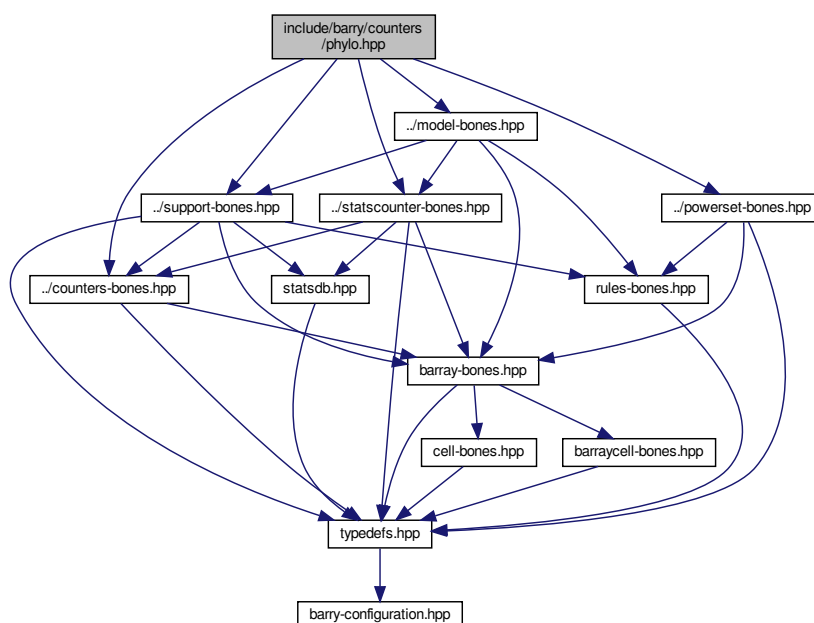
Number of edges.

Definition at line 742 of file network.hpp.

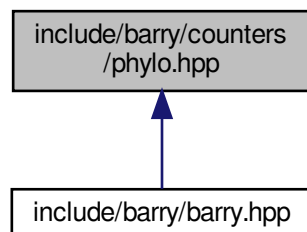
## 8.21 include/barry/counters/phylo.hpp File Reference

```
#include "../counters-bones.hpp"
#include "../support-bones.hpp"
#include "../statscounter-bones.hpp"
#include "../model-bones.hpp"
#include "../powerset-bones.hpp"
```

Include dependency graph for phylo.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [NodeData](#)  
*Data definition for the `PhyloArray` class.*
- class [PhyloRuleDynData](#)

## Macros

- `#define` [PHYLO\\_COUNTER\\_LAMBDA\(a\)](#)  
*Extension of a simple counter.*
- `#define` [PHYLO\\_RULE\\_DYN\\_LAMBDA\(a\)](#)
- `#define` [PHYLO\\_CHECK\\_MISSING\(\)](#)

## Typedefs

- `typedef` `std::vector< uint >` [PhyloCounterData](#)
- `typedef` `std::vector< std::pair< uint, uint > >` [PhyloRuleData](#)

### Convenient typedefs for Node objects.

- `typedef` [BArray< \[uint\]\(#\), \[NodeData\]\(#\) >](#) [PhyloArray](#)
- `typedef` [Counter< \[PhyloArray\]\(#\), \[PhyloCounterData\]\(#\) >](#) [PhyloCounter](#)
- `typedef` [Counters< \[PhyloArray\]\(#\), \[PhyloCounterData\]\(#\) >](#) [PhyloCounters](#)
- `typedef` [Rule< \[PhyloArray\]\(#\), \[PhyloRuleData\]\(#\) >](#) [PhyloRule](#)
- `typedef` [Rules< \[PhyloArray\]\(#\), \[PhyloRuleData\]\(#\) >](#) [PhyloRules](#)
- `typedef` [Rule< \[PhyloArray\]\(#\), \[PhyloRuleDynData\]\(#\) >](#) [PhyloRuleDyn](#)
- `typedef` [Rules< \[PhyloArray\]\(#\), \[PhyloRuleDynData\]\(#\) >](#) [PhyloRulesDyn](#)
- `typedef` [Support< \[PhyloArray\]\(#\), \[PhyloCounterData\]\(#\), \[PhyloRuleData\]\(#\), \[PhyloRuleDynData\]\(#\) >](#) [PhyloSupport](#)
- `typedef` [StatsCounter< \[PhyloArray\]\(#\), \[PhyloCounterData\]\(#\) >](#) [PhyloStatsCounter](#)
- `typedef` [Model< \[PhyloArray\]\(#\), \[PhyloCounterData\]\(#\), \[PhyloRuleData\]\(#\), \[PhyloRuleDynData\]\(#\) >](#) [PhyloModel](#)
- `typedef` [PowerSet< \[PhyloArray\]\(#\), \[PhyloRuleData\]\(#\) >](#) [PhyloPowerSet](#)

## Functions

- std::string `get_last_name` (bool d)
- void `counter_overall_gains` (PhyloCounters \*counters, bool duplication=true)  
*Overall functional gains.*
- void `counter_gains` (PhyloCounters \*counters, std::vector< uint > nfun, bool duplication=true)  
*Functional gains for a specific function (nfun).*
- void `counter_gains_k_offspring` (PhyloCounters \*counters, std::vector< uint > nfun, uint k=1u, bool duplication=true)  
*k genes gain function nfun*
- void `counter_genes_changing` (PhyloCounters \*counters, bool duplication=true)  
*Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)*
- void `counter_prop_genes_changing` (PhyloCounters \*counters, bool duplication=true)  
*Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)*
- void `counter_overall_loss` (PhyloCounters \*counters, bool duplication=true)  
*Overall functional loss.*
- void `counter_maxfuns` (PhyloCounters \*counters, uint lb, uint ub, bool duplication=true)  
*Cap the number of functions per gene.*
- void `counter_loss` (PhyloCounters \*counters, std::vector< uint > nfun, bool duplication=true)  
*Total count of losses for an specific function.*
- void `counter_overall_changes` (PhyloCounters \*counters, bool duplication=true)  
*Total number of changes. Use this statistic to account for "preservation".*
- void `counter_subfun` (PhyloCounters \*counters, uint nfunA, uint nfunB, bool duplication=true)  
*Total count of Sub-functionalization events.*
- void `counter_cogain` (PhyloCounters \*counters, uint nfunA, uint nfunB, bool duplication=true)  
*Co-evolution (joint gain or loss)*
- void `counter_longest` (PhyloCounters \*counters)  
*Longest branch mutates (either by gain or by loss)*
- void `counter_neofun` (PhyloCounters \*counters, uint nfunA, uint nfunB, bool duplication=true)  
*Total number of neofunctionalization events.*
- void `counter_neofun_a2b` (PhyloCounters \*counters, uint nfunA, uint nfunB, bool duplication=true)  
*Total number of neofunctionalization events.*
- void `counter_co_opt` (PhyloCounters \*counters, uint nfunA, uint nfunB, bool duplication=true)  
*Function co-opting.*
- void `rule_dyn_limit_changes` (PhyloSupport \*support, uint pos, uint lb, uint ub, bool duplication=true)  
*Overall functional gains.*

### 8.21.1 Macro Definition Documentation

#### 8.21.1.1 PHYLO\_CHECK\_MISSING

```
#define PHYLO_CHECK_MISSING( )
```

##### Value:

```
if (Array.D() == nullptr) \
throw std::logic_error("The array data is nullptr."); \
if (data == nullptr) \
throw std::logic_error("The counter/rule data is nullptr.")
```

Definition at line 94 of file phylo.hpp.

### 8.21.1.2 PHYLO\_COUNTER\_LAMBDA

```
#define PHYLO_COUNTER_LAMBDA(  
    a )
```

**Value:**

```
Counter_fun_type<PhyloArray, PhyloCounterData> a = \  
[] (const PhyloArray & Array, uint i, uint j, PhyloCounterData * data)
```

Extension of a simple counter.

It allows specifying extra arguments, in particular, the corresponding sets of rows to which this statistic may be relevant. This could be important in the case of, for example, counting correlation type statistics between function 1 and 2, and between function 1 and 3.

Definition at line 88 of file phylo.hpp.

### 8.21.1.3 PHYLO\_RULE\_DYN\_LAMBDA

```
#define PHYLO_RULE_DYN_LAMBDA(  
    a )
```

**Value:**

```
Rule_fun_type<PhyloArray, PhyloRuleDynData> a = \  
[] (const PhyloArray & Array, uint i, uint j, PhyloRuleDynData * data)
```

Definition at line 91 of file phylo.hpp.

## 8.21.2 Typedef Documentation

### 8.21.2.1 PhyloArray

```
typedef BArray<uint, NodeData> PhyloArray
```

Definition at line 61 of file phylo.hpp.

### 8.21.2.2 PhyloCounter

```
typedef Counter<PhyloArray, PhyloCounterData > PhyloCounter
```

Definition at line 62 of file phylo.hpp.

### 8.21.2.3 PhyloCounterData

```
typedef std::vector< uint > PhyloCounterData
```

Definition at line 53 of file phylo.hpp.

### 8.21.2.4 PhyloCounters

```
typedef Counters< PhyloArray, PhyloCounterData> PhyloCounters
```

Definition at line 63 of file phylo.hpp.

### 8.21.2.5 PhyloModel

```
typedef Model<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel
```

Definition at line 73 of file phylo.hpp.

### 8.21.2.6 PhyloPowerSet

```
typedef PowerSet<PhyloArray, PhyloRuleData> PhyloPowerSet
```

Definition at line 74 of file phylo.hpp.

### 8.21.2.7 PhyloRule

```
typedef Rule<PhyloArray,PhyloRuleData> PhyloRule
```

Definition at line 65 of file phylo.hpp.

### 8.21.2.8 PhyloRuleData

```
typedef std::vector< std::pair< uint, uint > > PhyloRuleData
```

Definition at line 54 of file phylo.hpp.

### 8.21.2.9 PhyloRuleDyn

```
typedef Rule<PhyloArray,PhyloRuleDynData> PhyloRuleDyn
```

Definition at line 68 of file phylo.hpp.

### 8.21.2.10 PhyloRules

```
typedef Rules<PhyloArray,PhyloRuleData> PhyloRules
```

Definition at line 66 of file phylo.hpp.

### 8.21.2.11 PhyloRulesDyn

```
typedef Rules<PhyloArray,PhyloRuleDynData> PhyloRulesDyn
```

Definition at line 69 of file phylo.hpp.

### 8.21.2.12 PhyloStatsCounter

```
typedef StatsCounter<PhyloArray, PhyloCounterData> PhyloStatsCounter
```

Definition at line 72 of file phylo.hpp.

### 8.21.2.13 PhyloSupport

```
typedef Support<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport
```

Definition at line 71 of file phylo.hpp.

## 8.21.3 Function Documentation

### 8.21.3.1 get\_last\_name()

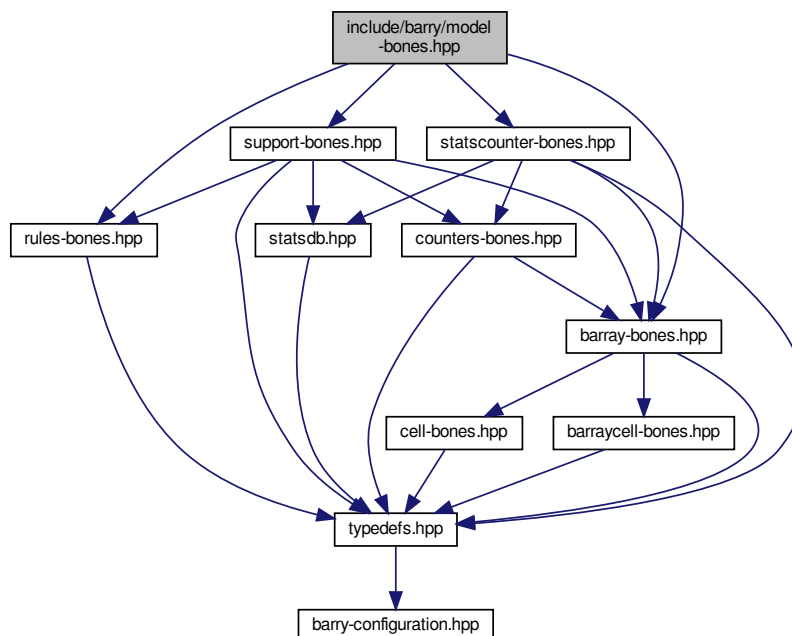
```
std::string get_last_name (
    bool d ) [inline]
```

Definition at line 99 of file phylo.hpp.

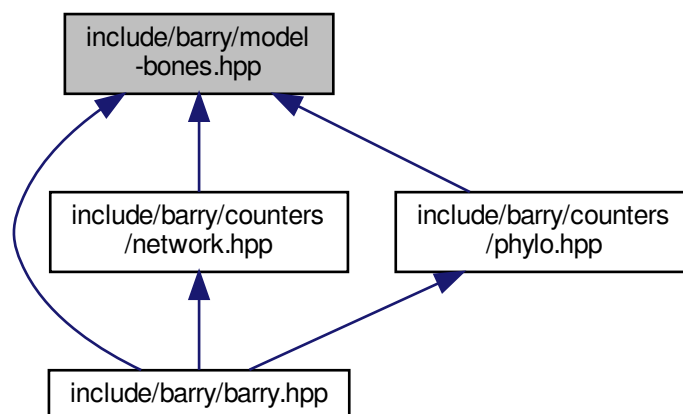


## 8.22 include/barry/model-bones.hpp File Reference

```
#include "barray-bones.hpp"
#include "support-bones.hpp"
#include "statscounter-bones.hpp"
#include "rules-bones.hpp"
Include dependency graph for model-bones.hpp:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [Model< Array\\_Type, Data\\_Counter\\_Type, Data\\_Rule\\_Type, Data\\_Rule\\_Dyn\\_Type >](#)

*General framework for discrete exponential models. This class allows generating discrete exponential models in the form of a linear exponential model:*

## Functions

- double [update\\_normalizing\\_constant](#) (const std::vector< double > &params, const [Counts\\_type](#) &support)
- double [likelihood\\_](#) (const std::vector< double > &target\_stats, const std::vector< double > &params, const double normalizing\_constant, bool log\_ = false)
- template<typename Array\_Type >  
std::vector< double > [keygen\\_default](#) (const Array\_Type &Array\_)

*Array Hasher class (used for computing support)*

### 8.22.1 Function Documentation

#### 8.22.1.1 keygen\_default()

```
template<typename Array_Type >
std::vector< double > keygen_default (
    const Array_Type & Array_ ) [inline]
```

Array Hasher class (used for computing support)

Definition at line 69 of file model-bones.hpp.

#### 8.22.1.2 likelihood\_()

```
double likelihood_ (
    const std::vector< double > & target_stats,
    const std::vector< double > & params,
    const double normalizing_constant,
    bool log_ = false ) [inline]
```

Definition at line 40 of file model-bones.hpp.

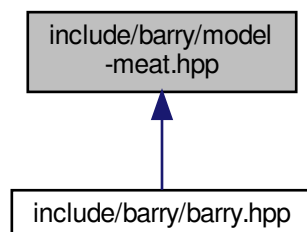
#### 8.22.1.3 update\_normalizing\_constant()

```
double update_normalizing_constant (
    const std::vector< double > & params,
    const Counts\_type & support ) [inline]
```

Definition at line 16 of file model-bones.hpp.

## 8.23 include/barry/model-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



## 8.24 include/barry/models/geese.hpp File Reference

```

#include "geese/geese-node-bones.hpp"
#include "geese/geese-bones.hpp"
#include "geese/geese-meat.hpp"
#include "geese/geese-meat-constructors.hpp"
#include "geese/geese-meat-likelihood.hpp"
#include "geese/geese-meat-likelihood_exhaust.hpp"
#include "geese/geese-meat-simulate.hpp"
#include "geese/geese-meat-predict.hpp"
#include "geese/geese-meat-predict_exhaust.hpp"
#include "geese/geese-meat-predict_sim.hpp"
#include "geese/flock-bones.hpp"
#include "geese/flock-meet.hpp"

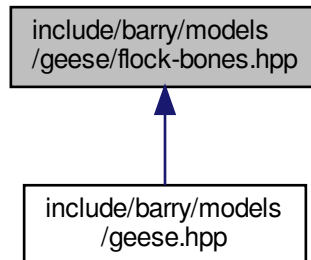
```

Include dependency graph for geese.hpp:



## 8.25 include/barry/models/geese/flock-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:

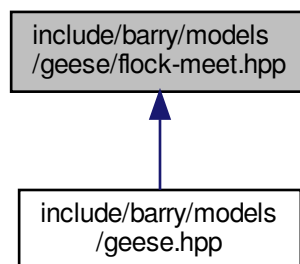


### Classes

- class [Flock](#)  
*A [Flock](#) is a group of [Geese](#).*

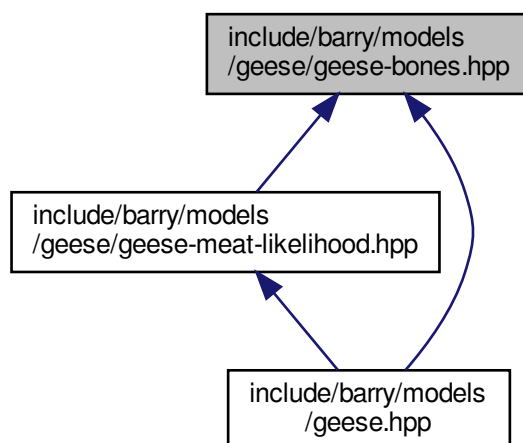
## 8.26 include/barry/models/geese/flock-meet.hpp File Reference

This graph shows which files directly or indirectly include this file:



## 8.27 include/barry/models/geese/geese-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



### Classes

- class [Geese](#)  
*Annotated Phylo [Model](#).*

### Macros

- `#define` [INITIALIZED\(\)](#)

### Functions

- `template<typename Ta , typename Tb >`  
`std::vector< Ta > vector\_caster (const std::vector< Tb > &x)`
- [RULE\\_FUNCTION](#) (`rule_empty_free`)
- `std::vector< double > keygen\_full (const phylocounters::PhyloArray &array)`
- `bool vec\_diff (const std::vector< unsigned int > &s, const std::vector< unsigned int > &a)`

#### 8.27.1 Macro Definition Documentation

### 8.27.1.1 INITIALIZED

```
#define INITIALIZED( )
```

**Value:**

```
if (!this->initialized) \  
    throw std::logic_error("The model has not been initialized yet.");
```

Definition at line 22 of file geese-bones.hpp.

## 8.27.2 Function Documentation

### 8.27.2.1 keygen\_full()

```
std::vector< double > keygen_full ( \  
    const phylocounters::PhyloArray & array ) [inline]
```

Definition at line 35 of file geese-bones.hpp.

### 8.27.2.2 RULE\_FUNCTION()

```
RULE_FUNCTION ( \  
    rule_empty_free )
```

Definition at line 26 of file geese-bones.hpp.

### 8.27.2.3 vec\_diff()

```
bool vec_diff ( \  
    const std::vector< unsigned int > & s, \  
    const std::vector< unsigned int > & a ) [inline]
```

Definition at line 55 of file geese-bones.hpp.

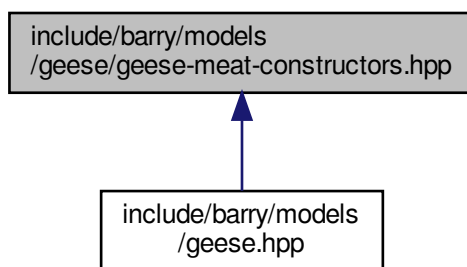
### 8.27.2.4 vector\_caster()

```
template<typename Ta , typename Tb > \  
std::vector< Ta > vector_caster ( \  
    const std::vector< Tb > & x ) [inline]
```

Definition at line 10 of file geese-bones.hpp.

## 8.28 include/barry/models/geese/geese-meat-constructors.hpp File Reference

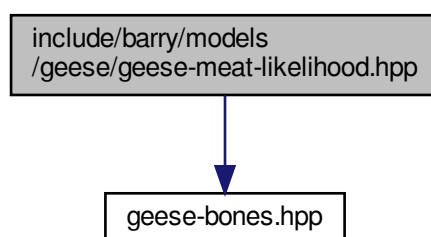
This graph shows which files directly or indirectly include this file:



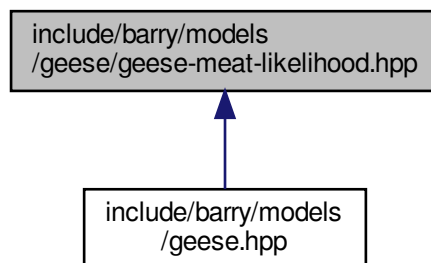
## 8.29 include/barry/models/geese/geese-meat-likelihood.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat-likelihood.hpp:

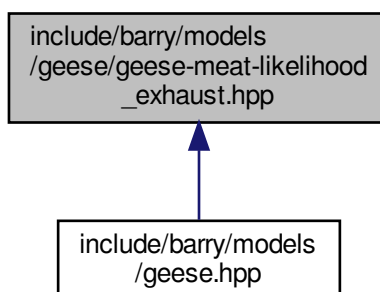


This graph shows which files directly or indirectly include this file:



### 8.30 `include/barry/models/geese/geese-meat-likelihood_exhaust.hpp` File Reference

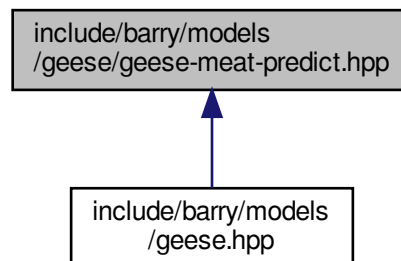
This graph shows which files directly or indirectly include this file:





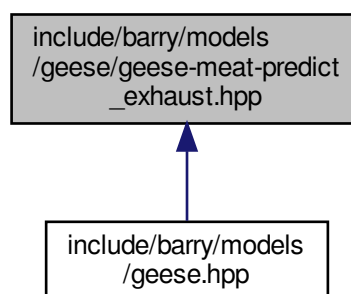
## 8.31 include/barry/models/geese/geese-meat-predict.hpp File Reference

This graph shows which files directly or indirectly include this file:



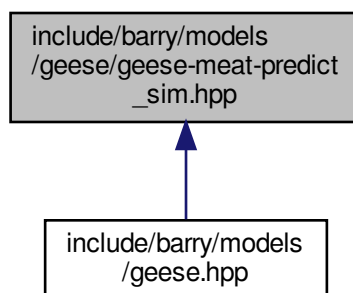
## 8.32 include/barry/models/geese/geese-meat-predict\_exhaust.hpp File Reference

This graph shows which files directly or indirectly include this file:



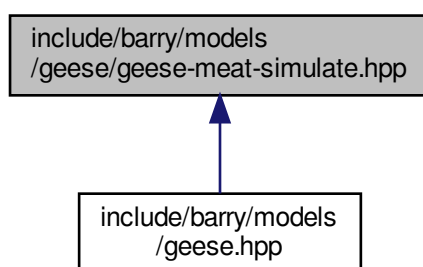
### 8.33 `include/barry/models/geese/geese-meat-predict_sim.hpp` File Reference

This graph shows which files directly or indirectly include this file:



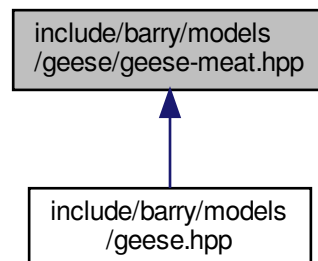
### 8.34 `include/barry/models/geese/geese-meat-simulate.hpp` File Reference

This graph shows which files directly or indirectly include this file:



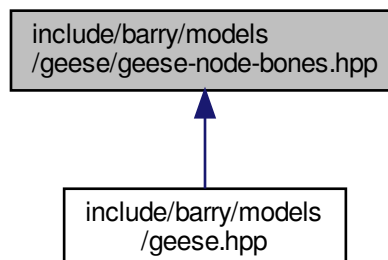
## 8.35 include/barry/models/geese/geese-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



## 8.36 include/barry/models/geese/geese-node-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



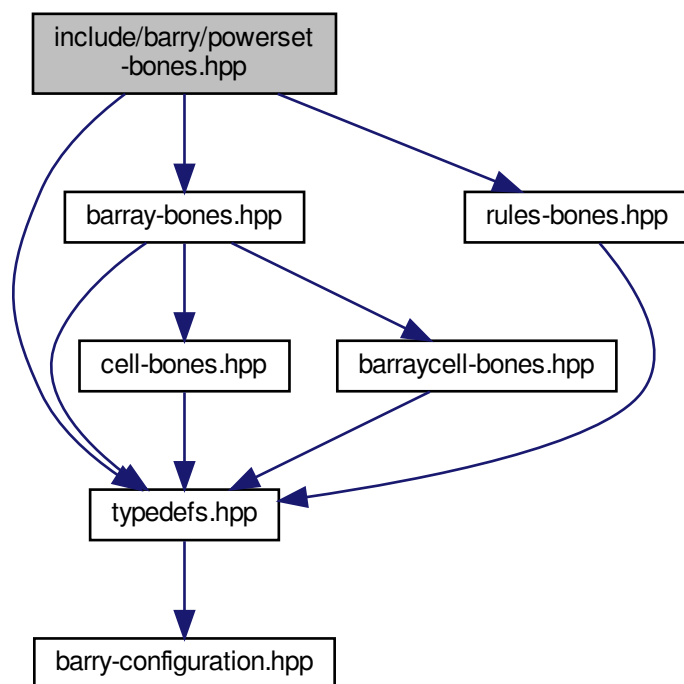
### Classes

- class [Node](#)  
*A single node for the model.*

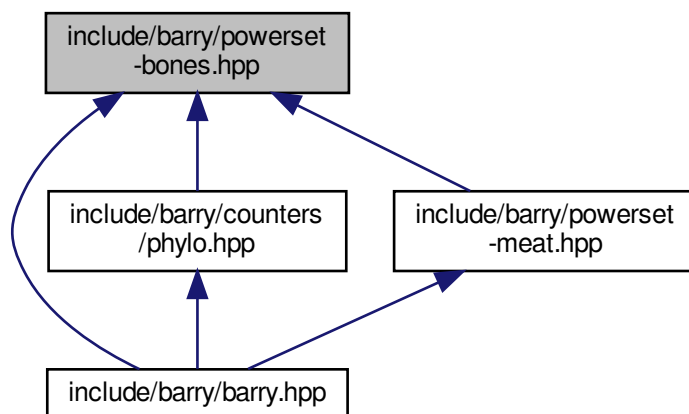
### 8.37 include/barry/powerset-bones.hpp File Reference

```
#include "typedefs.hpp"  
#include "barray-bones.hpp"  
#include "rules-bones.hpp"
```

Include dependency graph for powerset-bones.hpp:



This graph shows which files directly or indirectly include this file:



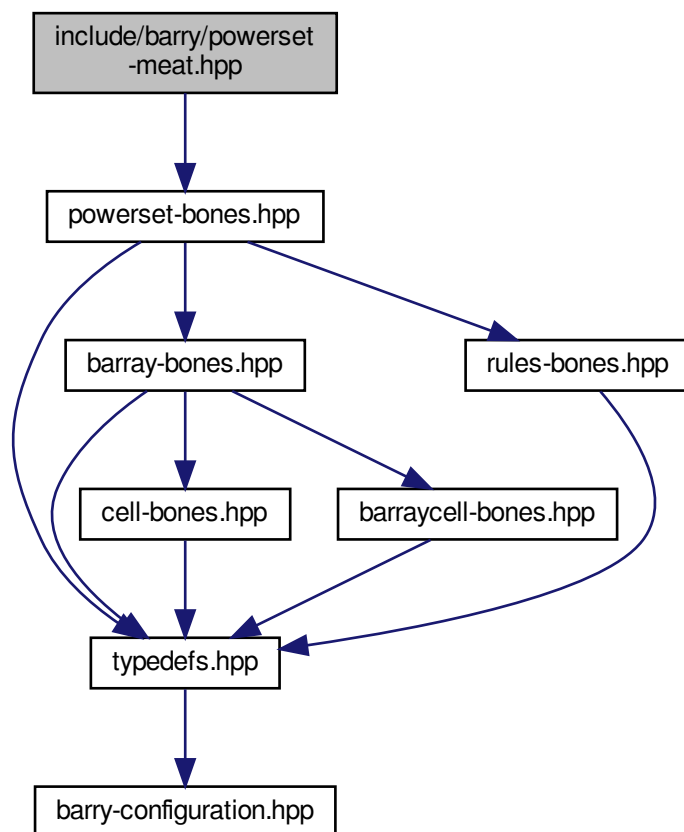
## Classes

- class [PowerSet< Array\\_Type, Data\\_Rule\\_Type >](#)  
*PowerSet of a binary array.*

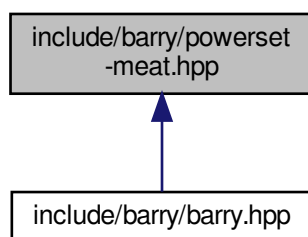
## 8.38 include/barry/powerset-meat.hpp File Reference

```
#include "powerset-bones.hpp"
```

Include dependency graph for powerset-meat.hpp:



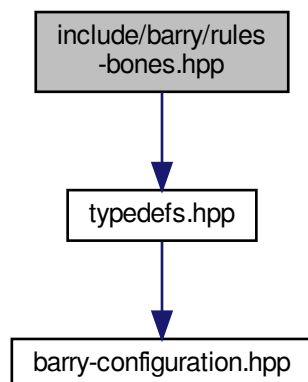
This graph shows which files directly or indirectly include this file:



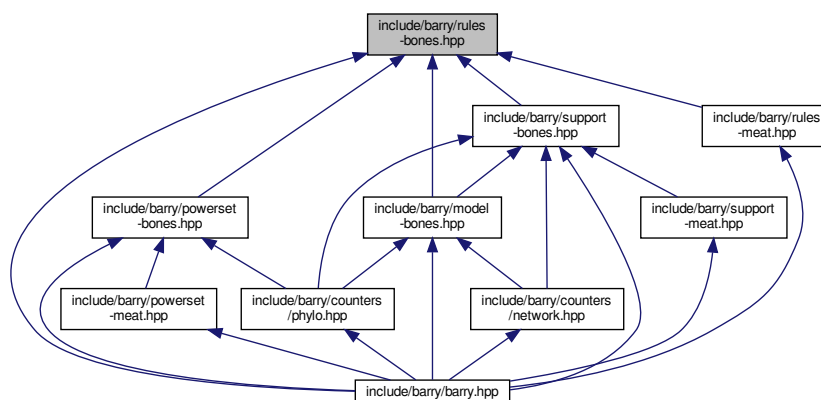
## 8.39 include/barry/rules-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for rules-bones.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [Rule< Array\\_Type, Data\\_Type >](#)  
*Rule for determining if a cell should be included in a sequence.*
- class [Rules< Array\\_Type, Data\\_Type >](#)  
*Vector of objects of class Rule.*

## Functions

- `template<typename Array_Type , typename Data_Type >`  
`bool rule_fun_default (const Array_Type *array, uint i, uint j, Data_Type *dat)`

### 8.39.1 Function Documentation

#### 8.39.1.1 rule\_fun\_default()

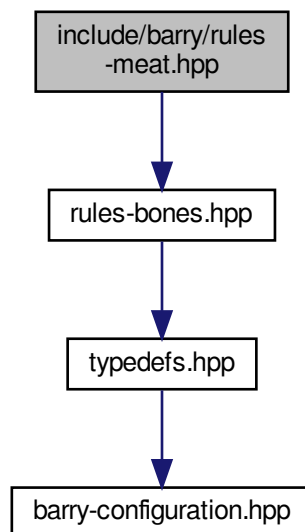
```
template<typename Array_Type , typename Data_Type >
bool rule_fun_default (
    const Array_Type * array,
    uint i,
    uint j,
    Data_Type * dat )
```

Definition at line 10 of file rules-bones.hpp.

## 8.40 include/barry/rules-meat.hpp File Reference

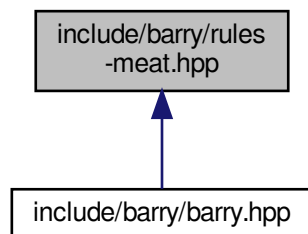
```
#include "rules-bones.hpp"
```

Include dependency graph for rules-meat.hpp:





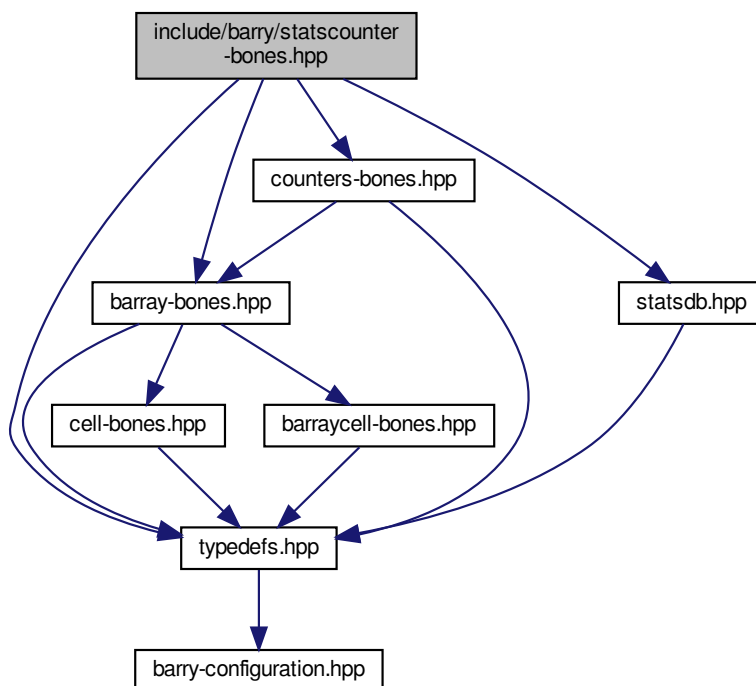
This graph shows which files directly or indirectly include this file:



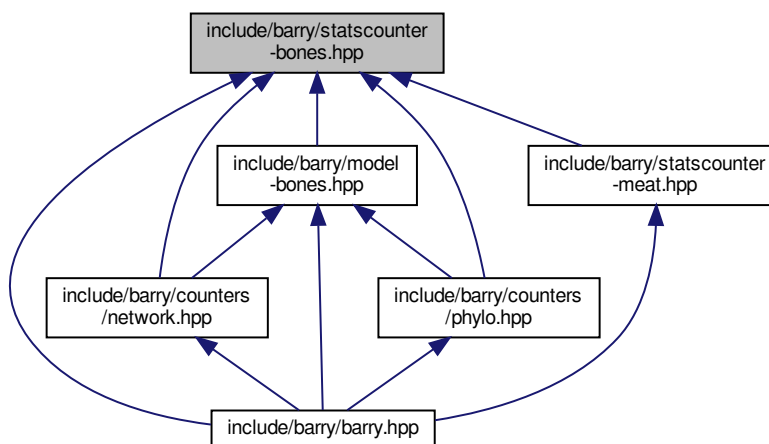
## 8.41 include/barry/statscounter-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "barray-bones.hpp"
#include "statsdb.hpp"
#include "counters-bones.hpp"
```

Include dependency graph for statscounter-bones.hpp:



This graph shows which files directly or indirectly include this file:



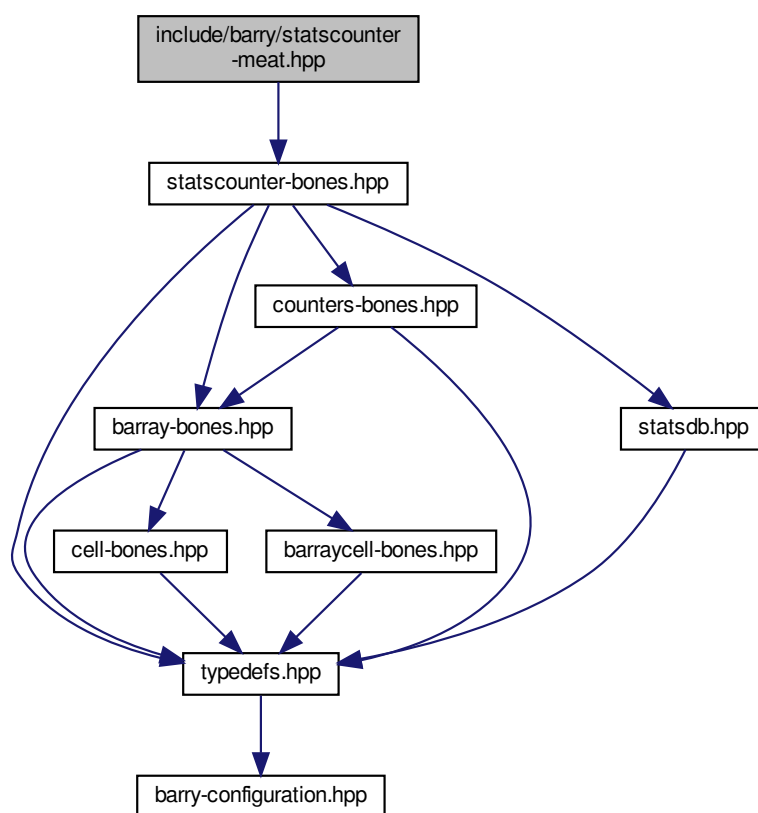
## Classes

- class [StatsCounter](#)< [Array\\_Type](#), [Data\\_Type](#) >  
*Count stats for a single Array.*

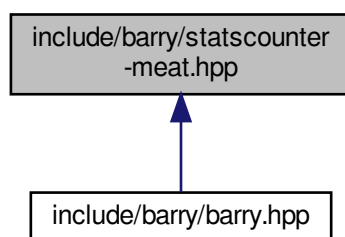
## 8.42 include/barry/statscounter-meat.hpp File Reference

```
#include "statscounter-bones.hpp"
```

Include dependency graph for statscounter-meat.hpp:



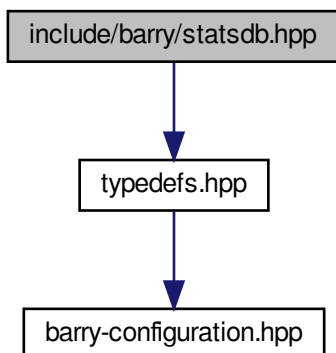
This graph shows which files directly or indirectly include this file:



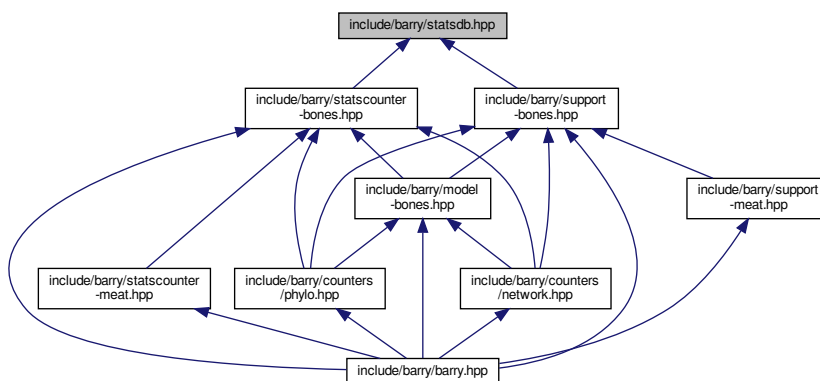
## 8.43 include/barry/statsdb.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for statsdb.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [FreqTable< T >](#)  
*Database of statistics.*

## 8.44 include/barry/support-bones.hpp File Reference

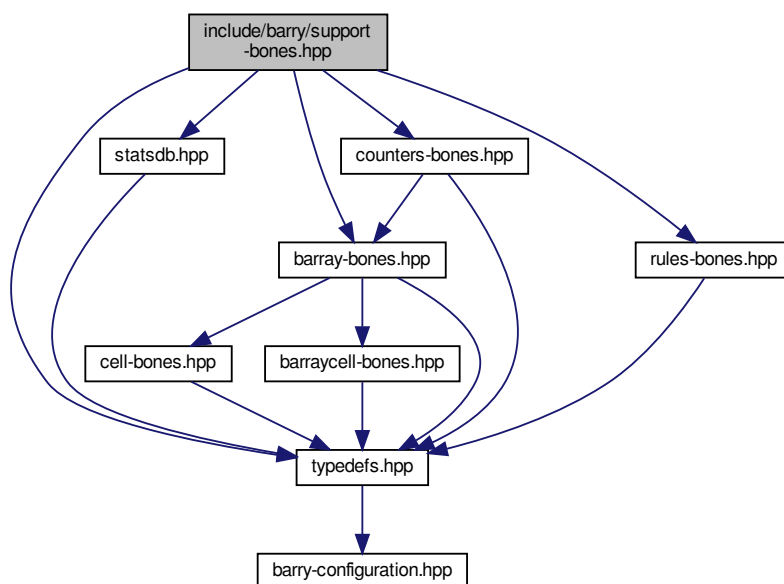
```

#include "typedefs.hpp"
#include "barray-bones.hpp"
#include "statsdb.hpp"
#include "counters-bones.hpp"

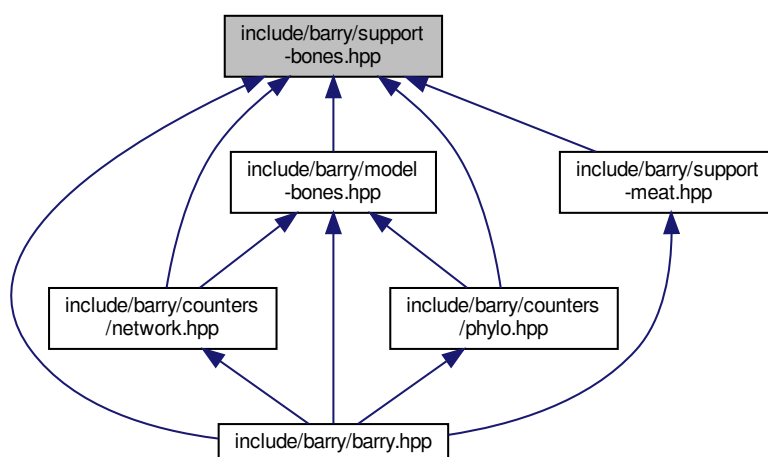
```

```
#include "rules-bones.hpp"
```

Include dependency graph for support-bones.hpp:



This graph shows which files directly or indirectly include this file:



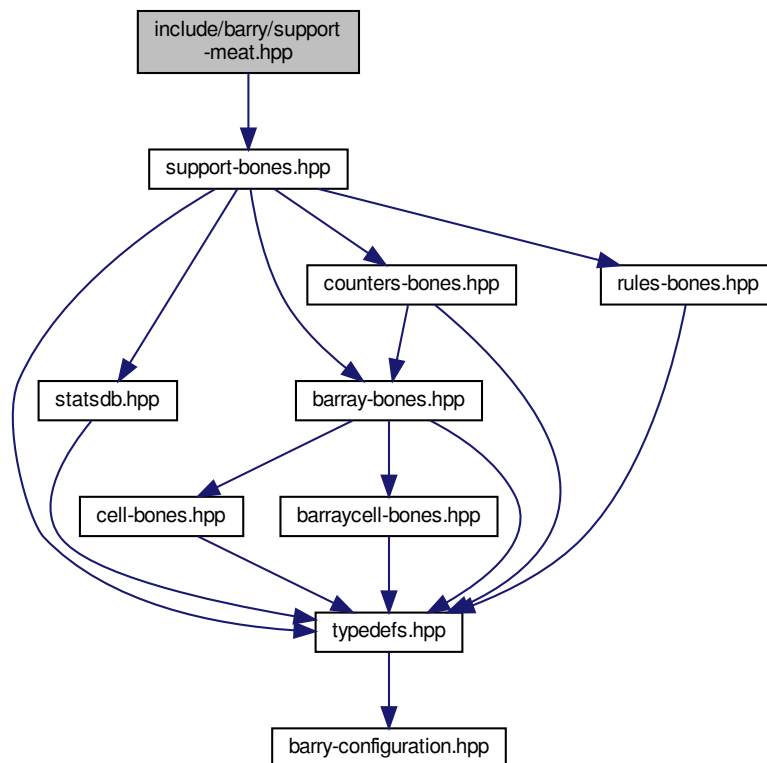
## Classes

- class `Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >`  
Compute the support of sufficient statistics.

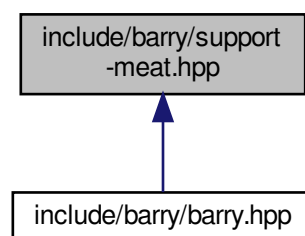
## 8.45 include/barry/support-meat.hpp File Reference

```
#include "support-bones.hpp"
```

Include dependency graph for support-meat.hpp:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define BARRY_SUPPORT_MEAT_HPP` 1

## 8.45.1 Macro Definition Documentation

### 8.45.1.1 BARRY\_SUPPORT\_MEAT\_HPP

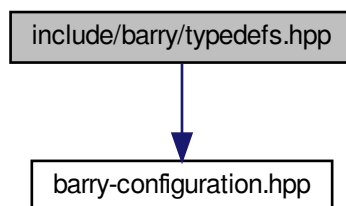
```
#define BARRY_SUPPORT_MEAT_HPP 1
```

Definition at line 4 of file support-meat.hpp.

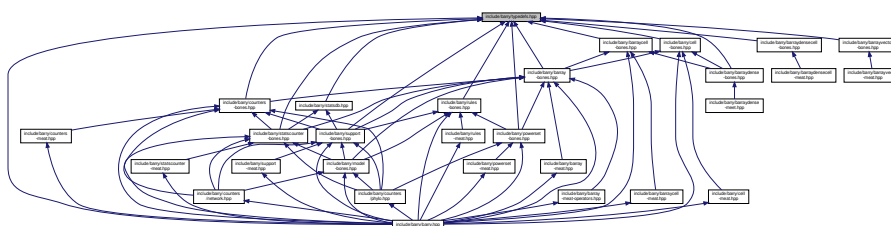
## 8.46 include/barry/typedefs.hpp File Reference

```
#include "barry-configuration.hpp"
```

Include dependency graph for typedefs.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [Entries< Cell\\_Type >](#)

*A wrapper class to store source, target, val from a [BArray](#) object.*

- struct [vecHasher< T >](#)

## Namespaces

- [CHECK](#)  
*Integer constants used to specify which cell should be check.*
- [EXISTS](#)  
*Integer constants used to specify which cell should be check to exist or not.*

## Typedefs

- typedef unsigned int [uint](#)
- typedef std::vector< std::pair< std::vector< double >, [uint](#) > > [Counts\\_type](#)
- template<typename Cell\_Type >  
using [Row\\_type](#) = [Map](#)< [uint](#), [Cell](#)< Cell\_Type > >
- template<typename Cell\_Type >  
using [Col\\_type](#) = [Map](#)< [uint](#), [Cell](#)< Cell\_Type > \* >
- template<typename Ta = double, typename Tb = uint>  
using [MapVec\\_type](#) = std::unordered\_map< std::vector< Ta >, Tb, [vecHasher](#)< Ta > >
- template<typename Array\_Type , typename Data\_Type >  
using [Counter\\_fun\\_type](#) = std::function< double(const Array\_Type &, [uint](#), [uint](#), Data\_Type \*)>  
*[Counter](#) and rule functions.*
- template<typename Array\_Type , typename Data\_Type >  
using [Rule\\_fun\\_type](#) = std::function< bool(const Array\_Type &, [uint](#), [uint](#), Data\_Type \*)>

## Functions

- template<typename T >  
T [vec\\_inner\\_prod](#) (const std::vector< T > &a, const std::vector< T > &b)
- template<typename T >  
bool [vec\\_equal](#) (const std::vector< T > &a, const std::vector< T > &b)  
*Compares if -a- and -b- are equal.*
- template<typename T >  
bool [vec\\_equal\\_approx](#) (const std::vector< T > &a, const std::vector< T > &b, double eps=1e-10)

## Variables

- const int [CHECK::BOTH](#) = -1
- const int [CHECK::NONE](#) = 0
- const int [CHECK::ONE](#) = 1
- const int [CHECK::TWO](#) = 2
- const int [EXISTS::BOTH](#) = -1
- const int [EXISTS::NONE](#) = 0
- const int [EXISTS::ONE](#) = 1
- const int [EXISTS::TWO](#) = 1
- const int [EXISTS::UNKNOWN](#) = -1
- const int [EXISTS::AS\\_ZERO](#) = 0
- const int [EXISTS::AS\\_ONE](#) = 1



## 8.46.1 Typedef Documentation

### 8.46.1.1 Col\_type

```
template<typename Cell_Type >
using Col_type = Map< uint, Cell<Cell_Type>* >
```

Definition at line 51 of file typedefs.hpp.

### 8.46.1.2 Counter\_fun\_type

```
template<typename Array_Type , typename Data_Type >
using Counter_fun_type = std::function<double(const Array_Type &, uint, uint, Data_Type *)>
```

[Counter](#) and rule functions.

#### Parameters

<i>Array_Type</i>	a <a href="#">BArray</a>
<i>unit,uint</i>	Focal cell
<i>Data_Type</i>	Data associated with the function, for example, id of the attribute in the Array.

#### Returns

[Counter\\_fun\\_type](#) a double (the change statistic)

[Rule\\_fun\\_type](#) a bool. True if the cell is blocked.

Definition at line 123 of file typedefs.hpp.

### 8.46.1.3 Counts\_type

```
typedef std::vector< std::pair< std::vector<double>, uint > > Counts_type
```

Definition at line 44 of file typedefs.hpp.

### 8.46.1.4 MapVec\_type

```
template<typename Ta = double, typename Tb = uint>
using MapVec_type = std::unordered_map< std::vector< Ta >, Tb, vecHasher<Ta> >
```

Definition at line 105 of file typedefs.hpp.

### 8.46.1.5 Row\_type

```
template<typename Cell_Type >
using Row_type = Map< uint, Cell<Cell_Type> >
```

Definition at line 48 of file typedefs.hpp.

### 8.46.1.6 Rule\_fun\_type

```
template<typename Array_Type , typename Data_Type >
using Rule_fun_type = std::function<bool(const Array_Type &, uint, uint, Data_Type *)>
```

Definition at line 126 of file typedefs.hpp.

### 8.46.1.7 uint

```
typedef unsigned int uint
```

Definition at line 10 of file typedefs.hpp.

## 8.46.2 Function Documentation

### 8.46.2.1 vec\_equal()

```
template<typename T >
bool vec_equal (
    const std::vector< T > & a,
    const std::vector< T > & b ) [inline]
```

Compares if -a- and -b- are equal.

#### Parameters

<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.

### 8.46.2.2 `vec_equal_approx()`

```
template<typename T >
bool vec_equal_approx (
    const std::vector< T > & a,
    const std::vector< T > & b,
    double eps = 1e-10 ) [inline]
```

Definition at line 155 of file typedefs.hpp.

### 8.46.2.3 `vec_inner_prod()`

```
template<typename T >
T vec_inner_prod (
    const std::vector< T > & a,
    const std::vector< T > & b ) [inline]
```

Definition at line 175 of file typedefs.hpp.

## 8.47 README.md File Reference



# Index

- ~BArray
  - BArray< Cell\_Type, Data\_Type >, [31](#)
- ~BArrayCell
  - BArrayCell< Cell\_Type, Data\_Type >, [41](#)
- ~BArrayCell\_const
  - BArrayCell\_const< Cell\_Type, Data\_Type >, [44](#)
- ~BArrayDense
  - BArrayDense< Cell\_Type, Data\_Type >, [50](#)
- ~BArrayDenseCell
  - BArrayDenseCell< Cell\_Type, Data\_Type >, [62](#)
- ~BArrayDenseCell\_const
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, [65](#)
- ~BArrayVector
  - BArrayVector< Cell\_Type, Data\_Type >, [68](#)
- ~BArrayVector\_const
  - BArrayVector\_const< Cell\_Type, Data\_Type >, [71](#)
- ~Cell
  - Cell< Cell\_Type >, [75](#)
- ~ConstBArrayRowIter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [80](#)
- ~Counter
  - Counter< Array\_Type, Data\_Type >, [83](#)
- ~Counters
  - Counters< Array\_Type, Data\_Type >, [87](#)
- ~Entries
  - Entries< Cell\_Type >, [91](#)
- ~Flock
  - Flock, [94](#)
- ~FreqTable
  - FreqTable< T >, [99](#)
- ~Geese
  - Geese, [104](#)
- ~Model
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [115](#)
- ~NetCounterData
  - NetCounterData, [126](#)
- ~NetworkData
  - NetworkData, [128](#)
- ~Node
  - Node, [131](#)
- ~NodeData
  - NodeData, [135](#)
- ~PhyloRuleDynData
  - PhyloRuleDynData, [137](#)
- ~PowerSet
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [140](#)
- ~Rule
  - Rule< Array\_Type, Data\_Type >, [146](#)
- ~Rules
  - Rules< Array\_Type, Data\_Type >, [148](#)
- ~StatsCounter
  - StatsCounter< Array\_Type, Data\_Type >, [153](#)
- ~Support
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [159](#)
- add
  - Cell< Cell\_Type >, [76, 77](#)
  - FreqTable< T >, [99](#)
- add\_array
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [116](#)
- add\_counter
  - Counters< Array\_Type, Data\_Type >, [88](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [116, 117](#)
  - StatsCounter< Array\_Type, Data\_Type >, [153](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [159](#)
- add\_data
  - Flock, [94](#)
- add\_rule
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [117](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [140, 141](#)
  - Rules< Array\_Type, Data\_Type >, [148](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [159, 160](#)
- add\_rule\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [118](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [160](#)
- annotations
  - Node, [132](#)
- Array
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [80](#)

- array
  - Node, 132
- arrays
  - Node, 132
- AS\_ONE
  - EXISTS, 25
- as\_vector
  - FreqTable< T >, 100
- 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\_vec, 32
  - get\_entries, 33
  - get\_row\_vec, 33
  - insert\_cell, 33, 34
  - is\_empty, 34
  - ncol, 34
  - nnozero, 34
  - nrow, 34
  - operator\*=: 35
  - operator(), 35
  - operator+=, 35
  - operator-=, 36
  - operator/=: 36
  - operator=, 36
  - operator==, 37
  - out\_of\_range, 37
  - print, 37
  - reserve, 37
  - resize, 37
  - rm\_cell, 37
  - row, 38
  - set\_data, 38
  - swap\_cells, 38
  - swap\_cols, 38
  - swap\_rows, 39
  - toggle\_cell, 39
  - toggle\_lock, 39
  - transpose, 39
  - visited, 40
  - zero\_col, 39
  - zero\_row, 40
- barray-bones.hpp
  - BARRAY\_BONES\_HPP, 168
- barray-meat-operators.hpp
  - BARRY\_BARRAY\_MEAT\_OPERATORS\_HPP, 170
- checkdim\_, 170
- COL, 170
- ROW, 170
- barray-meat.hpp
  - COL, 172
  - ROW, 172
- BARRAY\_BONES\_HPP
  - barray-bones.hpp, 168
- BArrayCell
  - BArrayCell< Cell\_Type, Data\_Type >, 41
- BArrayCell< Cell\_Type, Data\_Type >, 41
  - ~BArrayCell, 41
  - BArray< Cell\_Type, Data\_Type >, 40
  - BArrayCell, 41
  - BArrayDense< Cell\_Type, Data\_Type >, 60
  - operator Cell\_Type, 42
  - operator\*=: 42
  - operator+=, 42
  - operator-=, 42
  - operator/=: 42
  - operator=, 42
  - operator==, 43
- BArrayCell\_const
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 44
- BArrayCell\_const< Cell\_Type, Data\_Type >, 43
  - ~BArrayCell\_const, 44
  - BArray< Cell\_Type, Data\_Type >, 40
  - BArrayCell\_const, 44
  - BArrayDense< Cell\_Type, Data\_Type >, 60
  - operator Cell\_Type, 44
  - operator!=, 44
  - operator<, 44
  - operator<=: 45
  - operator>, 45
  - operator>=: 45
  - operator==, 45
- BArrayDense
  - BArrayDense< Cell\_Type, Data\_Type >, 48–50
- BArrayDense< Cell\_Type, Data\_Type >, 46
  - ~BArrayDense, 50
  - BArrayCell< Cell\_Type, Data\_Type >, 60
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 60
  - BArrayDense, 48–50
  - clear, 50
  - col, 50
  - D, 51
  - default\_val, 51
  - get\_cell, 51
  - get\_col\_vec, 51, 52
  - get\_entries, 52
  - get\_row\_vec, 52
  - insert\_cell, 53
  - is\_empty, 53
  - ncol, 54
  - nnozero, 54
  - nrow, 54
  - operator\*=: 55
  - operator(), 54

- operator+=, 55
- operator-=, 55, 56
- operator/=: 56
- operator=: 56
- operator==, 57
- out\_of\_range, 57
- print, 57
- reserve, 57
- resize, 57
- rm\_cell, 58
- row, 58
- set\_data, 58
- swap\_cells, 59
- swap\_cols, 59
- swap\_rows, 59
- toggle\_cell, 59
- toggle\_lock, 59
- transpose, 60
- visited, 61
- zero\_col, 60
- zero\_row, 60
- barrydense-meet.hpp
  - BARRY\_BARRAYDENSE\_MEAT\_HPP, 176
  - COL, 176
  - POS, 177
  - ROW, 177
  - ZERO\_CELL, 177
- BArrayDenseCell
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 62
- BArrayDenseCell< Cell\_Type, Data\_Type >, 61
  - ~BArrayDenseCell, 62
  - BArrayDenseCell, 62
  - operator Cell\_Type, 62
  - operator\*=, 62
  - operator+=, 63
  - operator-=, 63
  - operator/=: 63
  - operator=: 63
  - operator==, 63
- barrydensecell-meat.hpp
  - BARRY\_BARRAYDENSECELL\_MEAT\_HPP, 179
  - POS, 179
- BArrayDenseCell\_const
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 64
- BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 64
  - ~BArrayDenseCell\_const, 65
  - BArrayDenseCell\_const, 64
  - operator Cell\_Type, 65
  - operator!=, 65
  - operator<, 65
  - operator<=: 65
  - operator>, 66
  - operator>=: 66
  - operator==, 66
- BArrayVector
  - BArrayVector< Cell\_Type, Data\_Type >, 67
- BArrayVector< Cell\_Type, Data\_Type >, 66
  - ~BArrayVector, 68
  - BArrayVector, 67
  - begin, 68
  - end, 68
  - is\_col, 68
  - is\_row, 68
  - operator std::vector< Cell\_Type >, 69
  - operator\*=, 69
  - operator+=, 69
  - operator-=, 69
  - operator/=: 69
  - operator=: 70
  - operator==, 70
  - size, 70
- barryvector-meat.hpp
  - BARRY\_BARRAYVECTOR\_MEAT\_HPP, 181
- BArrayVector\_const
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 71
- BArrayVector\_const< Cell\_Type, Data\_Type >, 70
  - ~BArrayVector\_const, 71
  - BArrayVector\_const, 71
  - begin, 72
  - end, 72
  - is\_col, 72
  - is\_row, 72
  - operator std::vector< Cell\_Type >, 72
  - operator!=, 72
  - operator<, 73
  - operator<=: 73
  - operator>, 73
  - operator>=: 73
  - operator==, 73
  - size, 74
- barry, 23
- barry-configuration.hpp
  - BARRY\_CHECK\_SUPPORT, 182
  - BARRY\_ISFINITE, 182
  - BARRY\_MAX\_NUM\_ELEMENTS, 182
  - BARRY\_SAFE\_EXP, 182
  - Map, 182
  - printf\_barry, 182
- barry.hpp
  - BARRY\_HPP, 184
  - BARRY\_VERSION, 184
  - COUNTER\_FUNCTION, 184
  - COUNTER\_LAMBDA, 184
  - RULE\_FUNCTION, 185
  - RULE\_LAMBDA, 185
- barry::counters, 23
- barry::counters::network, 24
- barry::counters::phylo, 24
- BARRY\_BARRAY\_MEAT\_OPERATORS\_HPP
  - barry-meat-operators.hpp, 170
- BARRY\_BARRAYDENSE\_MEAT\_HPP
  - barrydense-meet.hpp, 176
- BARRY\_BARRAYDENSECELL\_MEAT\_HPP
  - barrydensecell-meat.hpp, 179
- BARRY\_BARRAYVECTOR\_MEAT\_HPP

- barrayvector-meat.hpp, 181
- BARRY\_CHECK\_SUPPORT
  - barry-configuration.hpp, 182
- BARRY\_HPP
  - barry.hpp, 184
- BARRY\_ISFINITE
  - barry-configuration.hpp, 182
- BARRY\_MAX\_NUM\_ELEMENTS
  - barry-configuration.hpp, 182
- BARRY\_SAFE\_EXP
  - barry-configuration.hpp, 182
- BARRY\_SUPPORT\_MEAT\_HPP
  - support-meat.hpp, 223
- BARRY\_VERSION
  - barry.hpp, 184
- begin
  - BArrayVector< Cell\_Type, Data\_Type >, 68
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 72
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 141
- blengths
  - NodeData, 136
- BOTH
  - CHECK, 24
  - EXISTS, 25
- calc
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 141
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 160
- calc\_reduced\_sequence
  - Geese, 104
- calc\_sequence
  - Geese, 105
- Cell
  - Cell< Cell\_Type >, 75, 76
- Cell< Cell\_Type >, 74
  - ~Cell, 75
  - add, 76, 77
  - Cell, 75, 76
  - operator Cell\_Type, 77
  - operator!=, 77
  - operator=, 77, 78
  - operator==, 78
  - value, 78
  - visited, 78
- change\_stats
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 164
- CHECK, 24
  - BOTH, 24
  - NONE, 24
  - ONE, 24
  - TWO, 24
- checkdim\_
  - barray-meat-operators.hpp, 170
- clear
  - BArray< Cell\_Type, Data\_Type >, 31
- BArrayDense< Cell\_Type, Data\_Type >, 50
- Counters< Array\_Type, Data\_Type >, 88
- FreqTable< T >, 100
- Rules< Array\_Type, Data\_Type >, 149
- COL
  - barray-meat-operators.hpp, 170
  - barray-meat.hpp, 172
  - barraydense-meet.hpp, 176
- col
  - BArray< Cell\_Type, Data\_Type >, 31
  - BArrayDense< Cell\_Type, Data\_Type >, 50
- Col\_type
  - typedefs.hpp, 225
- colnames
  - Flock, 94
  - Geese, 105
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 118
- ConstBArrayRowIter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, 79
- ConstBArrayRowIter< Cell\_Type, Data\_Type >, 79
  - ~ConstBArrayRowIter, 80
  - Array, 80
  - ConstBArrayRowIter, 79
  - current\_col, 80
  - current\_row, 80
  - iter, 80
- coordinates\_free
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 143
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 164
- coordinates\_locked
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 143
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 164
- count
  - Counter< Array\_Type, Data\_Type >, 83
- count\_all
  - StatsCounter< Array\_Type, Data\_Type >, 153
- count\_current
  - StatsCounter< Array\_Type, Data\_Type >, 154
- count\_fun
  - Counter< Array\_Type, Data\_Type >, 84
- count\_init
  - StatsCounter< Array\_Type, Data\_Type >, 154
- Counter
  - Counter< Array\_Type, Data\_Type >, 82, 83
- Counter< Array\_Type, Data\_Type >, 81
  - ~Counter, 83
  - count, 83
  - count\_fun, 84
  - Counter, 82, 83
  - data, 84
  - delete\_data, 85
  - desc, 85



- init, [83](#)
- init\_fun, [85](#)
- name, [85](#)
- operator=, [84](#)
- 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, [225](#)
- COUNTER\_FUNCTION
  - barry.hpp, [184](#)
- counter\_gains
  - Phylo counters, [16](#)
- counter\_gains\_k\_offspring
  - Phylo counters, [17](#)
- counter\_genes\_changing
  - Phylo counters, [17](#)
- counter\_idegree
  - Network counters, [12](#)
- counter\_idegree15
  - Network counters, [12](#)
- counter\_isolates
  - Network counters, [12](#)
- counter\_istar2
  - Network counters, [12](#)
- COUNTER\_LAMBDA
  - barry.hpp, [184](#)
- 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\_nodecov
  - 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\_prop\_genes\_changing
  - Phylo counters, [19](#)
- counter\_subfun
  - Phylo counters, [19](#)
- counter\_ttriads
  - Network counters, [14](#)
- Counters
  - Counters< Array\_Type, Data\_Type >, [86](#), [87](#)
- Counters< Array\_Type, Data\_Type >, [86](#)
  - ~Counters, [87](#)
  - add\_counter, [88](#)
  - clear, [88](#)
  - Counters, [86](#), [87](#)
  - operator=, [88](#), [89](#)
  - operator[], [89](#)
  - size, [90](#)
- Counting, [9](#)
- counts
  - PhyloRuleDynData, [137](#)
- Counts\_type
  - typedefs.hpp, [225](#)
- current\_col
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [80](#)
- current\_row
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [80](#)
- current\_stats
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [164](#)
- D
  - BArray< Cell\_Type, Data\_Type >, [32](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [51](#)
  - Rule< Array\_Type, Data\_Type >, [146](#)
- dat
  - Flock, [97](#)
- data
  - Counter< Array\_Type, Data\_Type >, [84](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [143](#)
- default\_val
  - BArray< Cell\_Type, Data\_Type >, [32](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [51](#)
- delete\_counters

- Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
165
- delete\_data
  - Counter< Array\_Type, Data\_Type >, 85
- delete\_engine
  - Geese, 111
- delete\_rules
  - Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
165
- delete\_rules\_dyn
  - Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
165
- delete\_support
  - Geese, 111
- desc
  - Counter< Array\_Type, Data\_Type >, 85
- directed
  - NetworkData, 128
- duplication
  - Node, 133
  - NodeData, 136
  - PhyloRuleDynData, 137
- EmptyArray
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 143
- end
  - BArrayVector< Cell\_Type, Data\_Type >, 68
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 72
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 141
- Entries
  - Entries< Cell\_Type >, 91
- Entries< Cell\_Type >, 90
  - ~Entries, 91
  - Entries, 91
  - resize, 91
  - source, 92
  - target, 92
  - val, 92
- EXISTS, 25
  - AS\_ONE, 25
  - AS\_ZERO, 25
  - BOTH, 25
  - NONE, 26
  - ONE, 26
  - TWO, 26
  - UNKNOWN, 26
- Flock, 92
  - ~Flock, 94
  - add\_data, 94
  - colnames, 94
  - dat, 97
  - Flock, 93
  - get\_counters, 94
  - get\_support, 95
  - init, 95
  - initialized, 98
  - likelihood\_joint, 95
  - model, 98
  - nfunctions, 98
  - nfuncs, 95
  - nleaves, 96
  - nnodes, 96
  - nterms, 96
  - ntrees, 96
  - operator(), 96
  - parse\_polytomies, 97
  - rengine, 98
  - set\_seed, 97
  - support\_size, 97
- FreqTable
  - FreqTable< T >, 99
- FreqTable< T >, 98
  - ~FreqTable, 99
  - add, 99
  - as\_vector, 100
  - clear, 100
  - FreqTable, 99
  - get\_data, 100
  - get\_data\_ptr, 100
  - print, 100
  - reserve, 101
  - size, 101
- Geese, 101
  - ~Geese, 104
  - calc\_reduced\_sequence, 104
  - calc\_sequence, 105
  - colnames, 105
  - delete\_engine, 111
  - delete\_support, 111
  - Geese, 103, 104
  - get\_annotated\_nodes, 105
  - get\_counters, 105
  - get\_model, 105
  - get\_probabilities, 105
  - get\_rengine, 106
  - get\_states, 106
  - get\_support, 106
  - inherit\_support, 106
  - init, 106
  - init\_node, 107
  - initialized, 111
  - likelihood, 107
  - likelihood\_exhaust, 107
  - map\_to\_nodes, 111
  - nfunctions, 111
  - nfuncs, 107
  - nleaves, 107
  - nnodes, 108
  - nodes, 112
  - nterms, 108
  - observed\_counts, 108
  - operator=, 108
  - parse\_polytomies, 108

- predict, [109](#)
- predict\_backend, [109](#)
- predict\_exhaust, [109](#)
- predict\_exhaust\_backend, [109](#)
- predict\_sim, [109](#)
- print\_observed\_counts, [110](#)
- reduced\_sequence, [112](#)
- sequence, [112](#)
- set\_seed, [110](#)
- simulate, [110](#)
- support\_size, [110](#)
- update\_annotations, [110](#)
- geese-bones.hpp
  - INITIALIZED, [205](#)
  - keygen\_full, [206](#)
  - RULE\_FUNCTION, [206](#)
  - vec\_diff, [206](#)
  - vector\_caster, [206](#)
- get\_annotated\_nodes
  - Geese, [105](#)
- get\_cell
  - BArray< Cell\_Type, Data\_Type >, [32](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [51](#)
- get\_col\_vec
  - BArray< Cell\_Type, Data\_Type >, [32](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [51](#), [52](#)
- get\_counters
  - Flock, [94](#)
  - Geese, [105](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [119](#)
  - StatsCounter< Array\_Type, Data\_Type >, [154](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [161](#)
- get\_counts
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [161](#)
- get\_counts\_ptr
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [161](#)
- get\_current\_stats
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [161](#)
- get\_data
  - FreqTable< T >, [100](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [142](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [161](#)
- get\_data\_ptr
  - FreqTable< T >, [100](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [142](#)
- get\_entries
  - BArray< Cell\_Type, Data\_Type >, [33](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [52](#)
- get\_last\_name
  - phylo.hpp, [200](#)
- get\_model
  - Geese, [105](#)
- get\_norm\_const
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [119](#)
- get\_parent
  - Node, [131](#)
- get\_probabilities
  - Geese, [105](#)
- get\_pset
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [119](#)
- get\_rengine
  - Geese, [106](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [119](#)
- get\_row\_vec
  - BArray< Cell\_Type, Data\_Type >, [33](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [52](#)
- get\_rules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [120](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [162](#)
- get\_rules\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [120](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [162](#)
- get\_seq
  - Rules< Array\_Type, Data\_Type >, [149](#)
- get\_states
  - Geese, [106](#)
- get\_stats
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [120](#)
- get\_support
  - Flock, [95](#)
  - Geese, [106](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [120](#)
- id
  - Node, [133](#)
- include/barry/barray-bones.hpp, [167](#)
- include/barry/barray-iterator.hpp, [168](#)

- include/barry/barray-meat-operators.hpp, 169
- include/barry/barray-meat.hpp, 171
- include/barry/barraycell-bones.hpp, 172
- include/barry/barraycell-meat.hpp, 173
- include/barry/barraydense-bones.hpp, 174
- include/barry/barraydense-meet.hpp, 175
- include/barry/barraydensecell-bones.hpp, 177
- include/barry/barraydensecell-meat.hpp, 178
- include/barry/barrayvector-bones.hpp, 179
- include/barry/barrayvector-meat.hpp, 180
- include/barry/barry-configuration.hpp, 181
- include/barry/barry.hpp, 183
- include/barry/cell-bones.hpp, 185
- include/barry/cell-meat.hpp, 186
- include/barry/col-bones.hpp, 187
- include/barry/counters-bones.hpp, 187
- include/barry/counters-meat.hpp, 188
- include/barry/counters/network.hpp, 189
- include/barry/counters/phylo.hpp, 195
- include/barry/model-bones.hpp, 201
- include/barry/model-meat.hpp, 203
- include/barry/models/geese.hpp, 203
- include/barry/models/geese/flock-bones.hpp, 204
- include/barry/models/geese/flock-meet.hpp, 204
- include/barry/models/geese/geese-bones.hpp, 205
- include/barry/models/geese/geese-meat-constructors.hpp, 207
- include/barry/models/geese/geese-meat-likelihood.hpp, 207
- include/barry/models/geese/geese-meat-likelihood\_exhaust.hpp, 208
- include/barry/models/geese/geese-meat-predict.hpp, 209
- include/barry/models/geese/geese-meat-predict\_exhaust.hpp, 209
- include/barry/models/geese/geese-meat-predict\_sim.hpp, 210
- include/barry/models/geese/geese-meat-simulate.hpp, 210
- include/barry/models/geese/geese-meat.hpp, 211
- include/barry/models/geese/geese-node-bones.hpp, 211
- include/barry/powerset-bones.hpp, 212
- include/barry/powerset-meat.hpp, 213
- include/barry/rules-bones.hpp, 215
- include/barry/rules-meat.hpp, 216
- include/barry/statscounter-bones.hpp, 217
- include/barry/statscounter-meat.hpp, 218
- include/barry/statsdb.hpp, 219
- include/barry/support-bones.hpp, 220
- include/barry/support-meat.hpp, 222
- include/barry/typedefs.hpp, 223
- indices
  - NetCounterData, 126
- inherit\_support
  - Geese, 106
- init
  - Counter< Array\_Type, Data\_Type >, 83
  - Flock, 95
  - Geese, 106
- init\_fun
  - Counter< Array\_Type, Data\_Type >, 85
- init\_node
  - Geese, 107
- init\_support
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 142
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 162
- INITIALIZED
  - geese-bones.hpp, 205
- initialized
  - Flock, 98
  - Geese, 111
- insert\_cell
  - BArray< Cell\_Type, Data\_Type >, 33, 34
  - BArrayDense< Cell\_Type, Data\_Type >, 53
- is\_col
  - BArrayVector< Cell\_Type, Data\_Type >, 68
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 72
- is\_empty
  - BArray< Cell\_Type, Data\_Type >, 34
  - BArrayDense< Cell\_Type, Data\_Type >, 53
- is\_leaf
  - Node, 132
- is\_row
  - BArrayVector< Cell\_Type, Data\_Type >, 68
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 72
- iter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, 80
- keygen\_default
  - model-bones.hpp, 202
- keygen\_full
  - geese-bones.hpp, 206
- lb
  - PhyloRuleDynData, 138
- likelihood
  - Geese, 107
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 120, 121
- likelihood\_
  - model-bones.hpp, 202
- likelihood\_exhaust
  - Geese, 107
- likelihood\_joint
  - Flock, 95
- likelihood\_total
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 121
- M
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 144

- Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
165
- Map
  - barry-configuration.hpp, 182
- map\_to\_nodes
  - Geese, 111
- MapVec\_type
  - typedefs.hpp, 225
- max\_num\_elements
  - Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
165
- Model
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
115
- model
  - Flock, 98
- Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type,  
Data\_Rule\_Dyn\_Type >, 112
  - ~Model, 115
  - add\_array, 116
  - add\_counter, 116, 117
  - add\_rule, 117
  - add\_rule\_dyn, 118
  - colnames, 118
  - get\_counters, 119
  - get\_norm\_const, 119
  - get\_pset, 119
  - get\_rengine, 119
  - get\_rules, 120
  - get\_rules\_dyn, 120
  - get\_stats, 120
  - get\_support, 120
  - likelihood, 120, 121
  - likelihood\_total, 121
  - Model, 115
  - nterms, 121
  - operator=, 122
  - print\_stats, 122
  - sample, 122
  - set\_counters, 123
  - set\_keygen, 123
  - set\_rengine, 123
  - set\_rules, 123
  - set\_rules\_dyn, 124
  - set\_seed, 124
  - size, 124
  - size\_unique, 124
  - store\_psets, 124
  - support\_size, 125
- model-bones.hpp
  - keygen\_default, 202
  - likelihood\_, 202
  - update\_normalizing\_constant, 202
- N
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 144
- Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
166
- name
  - Counter< Array\_Type, Data\_Type >, 85
- narray
  - Node, 133
- ncol
  - BArray< Cell\_Type, Data\_Type >, 34
  - BArrayDense< Cell\_Type, Data\_Type >, 54
- NET\_C\_DATA\_IDX
  - network.hpp, 192
- NET\_C\_DATA\_NUM
  - network.hpp, 192
- NetCounter
  - network.hpp, 193
- NetCounterData, 125
  - ~NetCounterData, 126
  - indices, 126
- NetCounterData, 126
  - numbers, 126
- NetCounters
  - network.hpp, 193
- NetModel
  - network.hpp, 194
- NetRule
  - network.hpp, 194
- NetRules
  - network.hpp, 194
- NetStatsCounter
  - network.hpp, 194
- NetSupport
  - network.hpp, 194
- Network
  - network.hpp, 194
- Network counters, 10
  - counter\_absdiff, 11
  - counter\_ctriads, 11
  - counter\_degree, 11
  - counter\_density, 11
  - counter\_diff, 11
  - counter\_edges, 12
  - counter\_idegree, 12
  - counter\_idegree15, 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, 192

- NET\_C\_DATA\_NUM, 192
- NetCounter, 193
- NetCounters, 193
- NetModel, 194
- NetRule, 194
- NetRules, 194
- NetStatsCounter, 194
- NetSupport, 194
- Network, 194
- NETWORK\_COUNTER, 192
- NETWORK\_COUNTER\_LAMBDA, 192
- NETWORK\_RULE, 193
- NETWORK\_RULE\_LAMBDA, 193
- rules\_zerodiag, 195
- NETWORK\_COUNTER
  - Network counters, 14
  - network.hpp, 192
- NETWORK\_COUNTER\_LAMBDA
  - network.hpp, 192
- NETWORK\_RULE
  - network.hpp, 193
- NETWORK\_RULE\_LAMBDA
  - network.hpp, 193
- NetworkData, 127
  - ~NetworkData, 128
  - directed, 128
  - NetworkData, 127, 128
  - vertex\_attr, 128
- nfunctions
  - Flock, 98
  - Geese, 111
- nfuncs
  - Flock, 95
  - Geese, 107
- nleafs
  - Flock, 96
  - Geese, 107
- nnodes
  - Flock, 96
  - Geese, 108
- nnozero
  - BArray< Cell\_Type, Data\_Type >, 34
  - BArrayDense< Cell\_Type, Data\_Type >, 54
- Node, 129
  - ~Node, 131
  - annotations, 132
  - array, 132
  - arrays, 132
  - duplication, 133
  - get\_parent, 131
  - id, 133
  - is\_leaf, 132
  - narray, 133
  - Node, 130, 131
  - noffspring, 132
  - offspring, 133
  - ord, 133
  - parent, 134
  - probability, 134
  - subtree\_prob, 134
  - visited, 134
- NodeData, 135
  - ~NodeData, 135
  - blengths, 136
  - duplication, 136
  - NodeData, 135
  - states, 136
- nodes
  - Geese, 112
- noffspring
  - Node, 132
- NONE
  - CHECK, 24
  - EXISTS, 26
- nrow
  - BArray< Cell\_Type, Data\_Type >, 34
  - BArrayDense< Cell\_Type, Data\_Type >, 54
- nterms
  - Flock, 96
  - Geese, 108
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 121
- ntrees
  - Flock, 96
- numbers
  - NetCounterData, 126
- observed\_counts
  - Geese, 108
- offspring
  - Node, 133
- ONE
  - CHECK, 24
  - EXISTS, 26
- operator Cell\_Type
  - BArrayCell< Cell\_Type, Data\_Type >, 42
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 44
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 62
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 65
  - Cell< Cell\_Type >, 77
- operator std::vector< Cell\_Type >
  - BArrayVector< Cell\_Type, Data\_Type >, 69
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 72
- operator!=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 44
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 65
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 72
  - Cell< Cell\_Type >, 77
- operator<
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 44
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 65
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 73
- operator<=

- BArrayCell\_const< Cell\_Type, Data\_Type >, [45](#)
- BArrayDenseCell\_const< Cell\_Type, Data\_Type >, [65](#)
- BArrayVector\_const< Cell\_Type, Data\_Type >, [73](#)
- operator>
  - BArrayCell\_const< Cell\_Type, Data\_Type >, [45](#)
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, [66](#)
  - BArrayVector\_const< Cell\_Type, Data\_Type >, [73](#)
- operator>=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, [45](#)
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, [66](#)
  - BArrayVector\_const< Cell\_Type, Data\_Type >, [73](#)
- operator\*=
  - BArray< Cell\_Type, Data\_Type >, [35](#)
  - BArrayCell< Cell\_Type, Data\_Type >, [42](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [55](#)
  - BArrayDenseCell< Cell\_Type, Data\_Type >, [62](#)
  - BArrayVector< Cell\_Type, Data\_Type >, [69](#)
- operator()
  - BArray< Cell\_Type, Data\_Type >, [35](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [54](#)
  - Flock, [96](#)
  - Rule< Array\_Type, Data\_Type >, [146](#)
  - Rules< Array\_Type, Data\_Type >, [149](#)
  - vecHasher< T >, [166](#)
- operator+=
  - BArray< Cell\_Type, Data\_Type >, [35](#)
  - BArrayCell< Cell\_Type, Data\_Type >, [42](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [55](#)
  - BArrayDenseCell< Cell\_Type, Data\_Type >, [63](#)
  - BArrayVector< Cell\_Type, Data\_Type >, [69](#)
- operator-=
  - BArray< Cell\_Type, Data\_Type >, [36](#)
  - BArrayCell< Cell\_Type, Data\_Type >, [42](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [55](#), [56](#)
  - BArrayDenseCell< Cell\_Type, Data\_Type >, [63](#)
  - BArrayVector< Cell\_Type, Data\_Type >, [69](#)
- operator/=
  - BArray< Cell\_Type, Data\_Type >, [36](#)
  - BArrayCell< Cell\_Type, Data\_Type >, [42](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [56](#)
  - BArrayDenseCell< Cell\_Type, Data\_Type >, [63](#)
  - BArrayVector< Cell\_Type, Data\_Type >, [69](#)
- operator=
  - BArray< Cell\_Type, Data\_Type >, [36](#)
  - BArrayCell< Cell\_Type, Data\_Type >, [42](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [56](#)
  - BArrayDenseCell< Cell\_Type, Data\_Type >, [63](#)
  - BArrayVector< Cell\_Type, Data\_Type >, [70](#)
  - Cell< Cell\_Type >, [77](#), [78](#)
  - Counter< Array\_Type, Data\_Type >, [84](#)
  - Counters< Array\_Type, Data\_Type >, [88](#), [89](#)
  - Geese, [108](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [122](#)
  - Rules< Array\_Type, Data\_Type >, [151](#)
- operator==
  - BArray< Cell\_Type, Data\_Type >, [37](#)
  - BArrayCell< Cell\_Type, Data\_Type >, [43](#)
  - BArrayCell\_const< Cell\_Type, Data\_Type >, [45](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [57](#)
  - BArrayDenseCell< Cell\_Type, Data\_Type >, [63](#)
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, [66](#)
  - BArrayVector< Cell\_Type, Data\_Type >, [70](#)
  - BArrayVector\_const< Cell\_Type, Data\_Type >, [73](#)
  - Cell< Cell\_Type >, [78](#)
- operator[]
  - Counters< Array\_Type, Data\_Type >, [89](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [142](#)
- ord
  - Node, [133](#)
- out\_of\_range
  - BArray< Cell\_Type, Data\_Type >, [37](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [57](#)
- parent
  - Node, [134](#)
- parse\_polytomies
  - Flock, [97](#)
  - Geese, [108](#)
- Phylo counters, [15](#)
  - counter\_co\_opt, [16](#)
  - counter\_cogain, [16](#)
  - counter\_gains, [16](#)
  - counter\_gains\_k\_offspring, [17](#)
  - counter\_genes\_changing, [17](#)
  - counter\_longest, [17](#)
  - counter\_loss, [17](#)
  - counter\_maxfun, [18](#)
  - counter\_neofun, [18](#)
  - counter\_neofun\_a2b, [18](#)
  - counter\_overall\_changes, [18](#)
  - counter\_overall\_gains, [19](#)
  - counter\_overall\_loss, [19](#)
  - counter\_prop\_genes\_changing, [19](#)
  - counter\_subfun, [19](#)
- Phylo rules, [20](#)
  - rule\_dyn\_limit\_changes, [20](#)
- phylo.hpp
  - get\_last\_name, [200](#)
  - PHYLO\_CHECK\_MISSING, [197](#)
  - PHYLO\_COUNTER\_LAMBDA, [197](#)
  - PHYLO\_RULE\_DYN\_LAMBDA, [198](#)
  - PhyloArray, [198](#)
  - PhyloCounter, [198](#)
  - PhyloCounterData, [198](#)
  - PhyloCounters, [199](#)
  - PhyloModel, [199](#)
  - PhyloPowerSet, [199](#)
  - PhyloRule, [199](#)
  - PhyloRuleData, [199](#)
  - PhyloRuleDyn, [199](#)
  - PhyloRules, [200](#)



- PhyloRulesDyn, 200
- PhyloStatsCounter, 200
- PhyloSupport, 200
- PHYLO\_CHECK\_MISSING
  - phylo.hpp, 197
- PHYLO\_COUNTER\_LAMBDA
  - phylo.hpp, 197
- PHYLO\_RULE\_DYN\_LAMBDA
  - phylo.hpp, 198
- PhyloArray
  - phylo.hpp, 198
- PhyloCounter
  - phylo.hpp, 198
- PhyloCounterData
  - phylo.hpp, 198
- PhyloCounters
  - phylo.hpp, 199
- PhyloModel
  - phylo.hpp, 199
- PhyloPowerSet
  - phylo.hpp, 199
- PhyloRule
  - phylo.hpp, 199
- PhyloRuleData
  - phylo.hpp, 199
- PhyloRuleDyn
  - phylo.hpp, 199
- PhyloRuleDynData, 136
  - ~PhyloRuleDynData, 137
  - counts, 137
  - duplication, 137
  - lb, 138
  - PhyloRuleDynData, 137
  - pos, 138
  - ub, 138
- PhyloRules
  - phylo.hpp, 200
- PhyloRulesDyn
  - phylo.hpp, 200
- PhyloStatsCounter
  - phylo.hpp, 200
- PhyloSupport
  - phylo.hpp, 200
- POS
  - barraydense-meet.hpp, 177
  - barraydensecell-meet.hpp, 179
- pos
  - PhyloRuleDynData, 138
- PowerSet
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 140
- PowerSet< Array\_Type, Data\_Rule\_Type >, 138
  - ~PowerSet, 140
  - add\_rule, 140, 141
  - begin, 141
  - calc, 141
  - coordinates\_free, 143
  - coordinates\_locked, 143
  - data, 143
- EmptyArray, 143
- end, 141
- get\_data, 142
- get\_data\_ptr, 142
- init\_support, 142
- M, 144
- N, 144
- operator[], 142
- PowerSet, 140
- reset, 142
- rules, 144
- rules\_deleted, 144
- size, 143
- predict
  - Geese, 109
- predict\_backend
  - Geese, 109
- predict\_exhaust
  - Geese, 109
- predict\_exhaust\_backend
  - Geese, 109
- predict\_sim
  - Geese, 109
- print
  - BArray< Cell\_Type, Data\_Type >, 37
  - BArrayDense< Cell\_Type, Data\_Type >, 57
  - FreqTable< T >, 100
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 162
- print\_observed\_counts
  - Geese, 110
- print\_stats
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 122
- printf\_barry
  - barry-configuration.hpp, 182
- probability
  - Node, 134
- README.md, 227
- reduced\_sequence
  - Geese, 112
- engine
  - Flock, 98
- reserve
  - BArray< Cell\_Type, Data\_Type >, 37
  - BArrayDense< Cell\_Type, Data\_Type >, 57
  - FreqTable< T >, 101
- reset
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 142
- reset\_array
  - StatsCounter< Array\_Type, Data\_Type >, 154
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 163
- resize
  - BArray< Cell\_Type, Data\_Type >, 37



- BArrayDense< Cell\_Type, Data\_Type >, 57
- Entries< Cell\_Type >, 91
- rm\_cell
  - BArray< Cell\_Type, Data\_Type >, 37
  - BArrayDense< Cell\_Type, Data\_Type >, 58
- ROW
  - barray-meat-operators.hpp, 170
  - barray-meat.hpp, 172
  - barraydense-meet.hpp, 177
- row
  - BArray< Cell\_Type, Data\_Type >, 38
  - BArrayDense< Cell\_Type, Data\_Type >, 58
- Row\_type
  - typedefs.hpp, 225
- Rule
  - Rule< Array\_Type, Data\_Type >, 145, 146
- Rule< Array\_Type, Data\_Type >, 145
  - ~Rule, 146
  - D, 146
  - operator(), 146
  - Rule, 145, 146
- rule\_dyn\_limit\_changes
  - Phylo rules, 20
- rule\_fun\_default
  - rules-bones.hpp, 216
- Rule\_fun\_type
  - typedefs.hpp, 226
- RULE\_FUNCTION
  - barry.hpp, 185
  - geese-bones.hpp, 206
- RULE\_LAMBDA
  - barry.hpp, 185
- Rules
  - Rules< Array\_Type, Data\_Type >, 147, 148
- rules
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 144
- Rules< Array\_Type, Data\_Type >, 147
  - ~Rules, 148
  - add\_rule, 148
  - clear, 149
  - get\_seq, 149
  - operator(), 149
  - operator=, 151
  - Rules, 147, 148
  - size, 151
- rules-bones.hpp
  - rule\_fun\_default, 216
- rules\_deleted
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 144
- rules\_zerodiag
  - network.hpp, 195
- sample
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 122
- sequence
  - Geese, 112
- set\_counters
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 123
- StatsCounter< Array\_Type, Data\_Type >, 155
- Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 163
- set\_data
  - BArray< Cell\_Type, Data\_Type >, 38
  - BArrayDense< Cell\_Type, Data\_Type >, 58
- set\_keygen
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 123
- set\_rengine
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 123
- set\_rules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 123
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 163
- set\_rules\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 124
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 163
- set\_seed
  - Flock, 97
  - Geese, 110
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 124
- simulate
  - Geese, 110
- size
  - BArrayVector< Cell\_Type, Data\_Type >, 70
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 74
  - Counters< Array\_Type, Data\_Type >, 90
  - FreqTable< T >, 101
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 124
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 143
  - Rules< Array\_Type, Data\_Type >, 151
- size\_unique
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 124
- source
  - Entries< Cell\_Type >, 92
- states
  - NodeData, 136

- Statistical Models, 9
- StatsCounter
  - StatsCounter< Array\_Type, Data\_Type >, 152, 153
- StatsCounter< Array\_Type, Data\_Type >, 151
  - ~StatsCounter, 153
  - add\_counter, 153
  - count\_all, 153
  - count\_current, 154
  - count\_init, 154
  - get\_counters, 154
  - reset\_array, 154
  - set\_counters, 155
  - StatsCounter, 152, 153
- store\_psets
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 124
- subtree\_prob
  - Node, 134
- Support
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 158
- Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 155
  - ~Support, 159
  - add\_counter, 159
  - add\_rule, 159, 160
  - add\_rule\_dyn, 160
  - calc, 160
  - change\_stats, 164
  - coordinates\_free, 164
  - coordinates\_locked, 164
  - current\_stats, 164
  - delete\_counters, 165
  - delete\_rules, 165
  - delete\_rules\_dyn, 165
  - get\_counters, 161
  - get\_counts, 161
  - get\_counts\_ptr, 161
  - get\_current\_stats, 161
  - get\_data, 161
  - get\_rules, 162
  - get\_rules\_dyn, 162
  - init\_support, 162
  - M, 165
  - max\_num\_elements, 165
  - N, 166
  - print, 162
  - reset\_array, 163
  - set\_counters, 163
  - set\_rules, 163
  - set\_rules\_dyn, 163
  - Support, 158
- support-meat.hpp
  - BARRY\_SUPPORT\_MEAT\_HPP, 223
- support\_size
  - Flock, 97
  - Geese, 110
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 125
  - swap\_cells
    - BArray< Cell\_Type, Data\_Type >, 38
    - BArrayDense< Cell\_Type, Data\_Type >, 59
  - swap\_cols
    - BArray< Cell\_Type, Data\_Type >, 38
    - BArrayDense< Cell\_Type, Data\_Type >, 59
  - swap\_rows
    - BArray< Cell\_Type, Data\_Type >, 39
    - BArrayDense< Cell\_Type, Data\_Type >, 59
  - target
    - Entries< Cell\_Type >, 92
  - toggle\_cell
    - BArray< Cell\_Type, Data\_Type >, 39
    - BArrayDense< Cell\_Type, Data\_Type >, 59
  - toggle\_lock
    - BArray< Cell\_Type, Data\_Type >, 39
    - BArrayDense< Cell\_Type, Data\_Type >, 59
  - transpose
    - BArray< Cell\_Type, Data\_Type >, 39
    - BArrayDense< Cell\_Type, Data\_Type >, 60
  - TWO
    - CHECK, 24
    - EXISTS, 26
  - typedefs.hpp
    - Col\_type, 225
    - Counter\_fun\_type, 225
    - Counts\_type, 225
    - MapVec\_type, 225
    - Row\_type, 225
    - Rule\_fun\_type, 226
    - uint, 226
    - vec\_equal, 226
    - vec\_equal\_approx, 226
    - vec\_inner\_prod, 227
  - ub
    - PhyloRuleDynData, 138
  - uint
    - typedefs.hpp, 226
  - UNKNOWN
    - EXISTS, 26
  - update\_annotations
    - Geese, 110
  - update\_normalizing\_constant
    - model-bones.hpp, 202
  - val
    - Entries< Cell\_Type >, 92
  - value
    - Cell< Cell\_Type >, 78
  - vec\_diff
    - geese-bones.hpp, 206
  - vec\_equal

- typedefs.hpp, [226](#)
- vec\_equal\_approx
  - typedefs.hpp, [226](#)
- vec\_inner\_prod
  - typedefs.hpp, [227](#)
- vecHasher< T >, [166](#)
  - operator(), [166](#)
- vector\_caster
  - geese-bones.hpp, [206](#)
- vertex\_attr
  - NetworkData, [128](#)
- visited
  - BArray< Cell\_Type, Data\_Type >, [40](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [61](#)
  - Cell< Cell\_Type >, [78](#)
  - Node, [134](#)
- ZERO\_CELL
  - barraydense-meet.hpp, [177](#)
- zero\_col
  - BArray< Cell\_Type, Data\_Type >, [39](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [60](#)
- zero\_row
  - BArray< Cell\_Type, Data\_Type >, [40](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [60](#)