

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>5</b>
2.1 Modules	5
<b>3 Hierarchical Index</b>	<b>7</b>
3.1 Class Hierarchy	7
<b>4 Class Index</b>	<b>9</b>
4.1 Class List	9
<b>5 File Index</b>	<b>11</b>
5.1 File List	11
<b>6 Module Documentation</b>	<b>13</b>
6.1 Counting	13
6.1.1 Detailed Description	15
6.1.2 Macro Definition Documentation	15
6.1.2.1 IF_MATCHES	15
6.1.2.2 IF_NOTMATCHES	15
6.1.2.3 IS_DUPLICATION	15
6.1.2.4 IS_EITHER	16
6.1.2.5 IS_SPECIATION	16
6.1.2.6 MAKE_DEFM_HASHER	16
6.1.2.7 MAKE_DUPL_VARS	16
6.1.2.8 PHYLO_CHECK_MISSING	17
6.1.2.9 PHYLO_COUNTER_LAMBDA	17
6.1.2.10 PHYLO_RULE_DYN_LAMBDA	17
6.1.3 Function Documentation	17
6.1.3.1 counter_co_opt()	18
6.1.3.2 counter_cogain()	18
6.1.3.3 counter_gains()	18
6.1.3.4 counter_gains_from_0()	19
6.1.3.5 counter_gains_k_offspring()	19
6.1.3.6 counter_genes_changing()	19
6.1.3.7 counter_k_genes_changing()	19
6.1.3.8 counter_less_than_p_prop_genes_changing()	20
6.1.3.9 counter_longest()	20
6.1.3.10 counter_loss()	20
6.1.3.11 counter_maxfuns()	20
6.1.3.12 counter_neofun()	21
6.1.3.13 counter_neofun_a2b()	21
6.1.3.14 counter_overall_changes()	21
6.1.3.15 counter_overall_gains()	21

6.1.3.16 counter_overall_gains_from_0()	22
6.1.3.17 counter_overall_loss()	22
6.1.3.18 counter_pairwise_first_gain()	22
6.1.3.19 counter_pairwise_neofun_singlefun()	22
6.1.3.20 counter_pairwise_overall_change()	23
6.1.3.21 counter_pairwise_preserving()	23
6.1.3.22 counter_preserve_pseudogene()	23
6.1.3.23 counter_prop_genes_changing()	23
6.1.3.24 counter_subfun()	24
6.1.3.25 get_last_name()	24
6.2 Statistical Models	24
6.2.1 Detailed Description	24
6.3 Network counters	24
6.3.1 Detailed Description	26
6.3.2 Function Documentation	26
6.3.2.1 counter_absdiff()	27
6.3.2.2 counter_ctriads() [1/2]	27
6.3.2.3 counter_ctriads() [2/2]	27
6.3.2.4 counter_degree()	27
6.3.2.5 counter_density()	28
6.3.2.6 counter_diff()	28
6.3.2.7 counter_edges()	28
6.3.2.8 counter_fixed_effect()	28
6.3.2.9 counter_iddegree() [1/2]	29
6.3.2.10 counter_iddegree() [2/2]	29
6.3.2.11 counter_iddegree15() [1/2]	29
6.3.2.12 counter_iddegree15() [2/2]	29
6.3.2.13 counter_isolates() [1/2]	30
6.3.2.14 counter_isolates() [2/2]	30
6.3.2.15 counter_istar2() [1/2]	30
6.3.2.16 counter_istar2() [2/2]	30
6.3.2.17 counter_logit_intercept()	30
6.3.2.18 counter_mutual()	31
6.3.2.19 counter_nodecov()	31
6.3.2.20 counter_nodeicov()	31
6.3.2.21 counter_nodematch()	31
6.3.2.22 counter_nodeocov()	31
6.3.2.23 counter_odegree() [1/2]	32
6.3.2.24 counter_odegree() [2/2]	32
6.3.2.25 counter_odegree15() [1/2]	32
6.3.2.26 counter_odegree15() [2/2]	32
6.3.2.27 counter_ones()	32

6.3.2.28 counter_ostar2() [1/2]	33
6.3.2.29 counter_ostar2() [2/2]	33
6.3.2.30 counter_transition()	33
6.3.2.31 counter_transition_formula()	34
6.3.2.32 counter_ttriads() [1/2]	34
6.3.2.33 counter_ttriads() [2/2]	34
6.3.2.34 NETWORK_COUNTER()	34
6.3.2.35 rules_dont_become_zero()	35
6.3.2.36 rules_markov_fixed()	35
6.4 Phylo rules	35
6.4.1 Detailed Description	37
6.4.2 Typedef Documentation	37
6.4.2.1 DEFMCounter	37
6.4.2.2 DEFMCounters	37
6.4.2.3 DEFMModel	38
6.4.2.4 DEFMRule	38
6.4.2.5 DEFMRuleDyn	38
6.4.2.6 DEFMRules	38
6.4.2.7 DEFMRulesDyn	38
6.4.2.8 DEFMStatsCounter	38
6.4.2.9 DEFMSupport	39
6.4.3 Function Documentation	39
6.4.3.1 at()	39
6.4.3.2 DEFMCounterData() [1/2]	39
6.4.3.3 DEFMCounterData() [2/2]	39
6.4.3.4 DEFMData() [1/2]	39
6.4.3.5 DEFMData() [2/2]	39
6.4.3.6 DEFMRuleData() [1/3]	40
6.4.3.7 DEFMRuleData() [2/3]	40
6.4.3.8 DEFMRuleData() [3/3]	40
6.4.3.9 DEFMRuleDynData()	40
6.4.3.10 idx() [1/2]	41
6.4.3.11 idx() [2/2]	41
6.4.3.12 is_true() [1/2]	41
6.4.3.13 is_true() [2/2]	41
6.4.3.14 ncol()	41
6.4.3.15 nrow()	41
6.4.3.16 num() [1/2]	42
6.4.3.17 num() [2/2]	42
6.4.3.18 operator>()	42
6.4.3.19 print()	42
6.4.3.20 rule_dyn_limit_changes()	43

6.4.3.21 ~DEFMCounterData()	43
6.4.3.22 ~DEFMData()	43
6.4.3.23 ~DEFMRuleDynData()	43
6.4.4 Variable Documentation	44
6.4.4.1 array	44
6.4.4.2 counts	44
6.4.4.3 covar_sort	44
6.4.4.4 covar_used	44
6.4.4.5 covariates	44
6.4.4.6 indices [1/2]	45
6.4.4.7 indices [2/2]	45
6.4.4.8 init	45
6.4.4.9 is_motif	45
6.4.4.10 logical [1/2]	45
6.4.4.11 logical [2/2]	45
6.4.4.12 numbers [1/2]	46
6.4.4.13 numbers [2/2]	46
6.4.4.14 obs_start	46
6.4.4.15 X_ncol	46
6.4.4.16 X_nrow	46
6.5 Phylo counters	47
6.5.1 Detailed Description	48
6.5.2 Function Documentation	48
6.5.2.1 counter_co_opt()	48
6.5.2.2 counter_cogain()	49
6.5.2.3 counter_gains()	49
6.5.2.4 counter_gains_from_0()	49
6.5.2.5 counter_gains_k_offspring()	50
6.5.2.6 counter_genes_changing()	50
6.5.2.7 counter_k_genes_changing()	50
6.5.2.8 counter_less_than_p_prop_genes_changing()	50
6.5.2.9 counter_longest()	51
6.5.2.10 counter_loss()	51
6.5.2.11 counter_maxfuns()	51
6.5.2.12 counter_neofun()	51
6.5.2.13 counter_neofun_a2b()	52
6.5.2.14 counter_overall_changes()	52
6.5.2.15 counter_overall_gains()	52
6.5.2.16 counter_overall_gains_from_0()	52
6.5.2.17 counter_overall_loss()	53
6.5.2.18 counter_pairwise_first_gain()	53
6.5.2.19 counter_pairwise_neofun_singlefun()	53

6.5.2.20 counter_pairwise_overall_change()	53
6.5.2.21 counter_pairwise_preserving()	54
6.5.2.22 counter_preserve_pseudogene()	54
6.5.2.23 counter_prop_genes_changing()	54
6.5.2.24 counter_subfun()	54
<b>7 Namespace Documentation</b>	<b>55</b>
7.1 barry Namespace Reference	55
7.1.1 Detailed Description	55
7.2 barry::counters Namespace Reference	55
7.2.1 Detailed Description	55
7.3 barry::counters::network Namespace Reference	56
7.4 CHECK Namespace Reference	56
7.4.1 Detailed Description	56
7.4.2 Variable Documentation	56
7.4.2.1 BOTH	56
7.4.2.2 NONE	56
7.4.2.3 ONE	56
7.4.2.4 TWO	57
7.5 defm Namespace Reference	57
7.6 EXISTS Namespace Reference	57
7.6.1 Detailed Description	57
7.6.2 Variable Documentation	57
7.6.2.1 AS_ONE	57
7.6.2.2 AS_ZERO	57
7.6.2.3 BOTH	58
7.6.2.4 NONE	58
7.6.2.5 ONE	58
7.6.2.6 TWO	58
7.6.2.7 UNKNOWN	58
7.7 geese Namespace Reference	58
<b>8 Class Documentation</b>	<b>59</b>
8.1 BArray< Cell_Type, Data_Type > Class Template Reference	59
8.1.1 Detailed Description	61
8.1.2 Constructor & Destructor Documentation	62
8.1.2.1 BArray() [1/6]	62
8.1.2.2 BArray() [2/6]	62
8.1.2.3 BArray() [3/6]	62
8.1.2.4 BArray() [4/6]	63
8.1.2.5 BArray() [5/6]	63
8.1.2.6 BArray() [6/6]	63
8.1.2.7 ~BArray()	63

8.1.3 Member Function Documentation	63
8.1.3.1 clear()	63
8.1.3.2 col()	64
8.1.3.3 D() [1/2]	64
8.1.3.4 D() [2/2]	64
8.1.3.5 D_ptr() [1/2]	64
8.1.3.6 D_ptr() [2/2]	64
8.1.3.7 default_val()	64
8.1.3.8 flush_data()	64
8.1.3.9 get_cell()	65
8.1.3.10 get_col_vec() [1/2]	65
8.1.3.11 get_col_vec() [2/2]	65
8.1.3.12 get_entries()	65
8.1.3.13 get_row_vec() [1/2]	65
8.1.3.14 get_row_vec() [2/2]	66
8.1.3.15 insert_cell() [1/3]	66
8.1.3.16 insert_cell() [2/3]	66
8.1.3.17 insert_cell() [3/3]	66
8.1.3.18 is_dense()	66
8.1.3.19 is_empty()	67
8.1.3.20 ncol()	67
8.1.3.21 nnozero()	67
8.1.3.22 nrow()	67
8.1.3.23 operator>() [1/2]	67
8.1.3.24 operator>() [2/2]	67
8.1.3.25 operator*=( )	68
8.1.3.26 operator+=( ) [1/3]	68
8.1.3.27 operator+=( ) [2/3]	68
8.1.3.28 operator+=( ) [3/3]	68
8.1.3.29 operator-=( ) [1/3]	68
8.1.3.30 operator-=( ) [2/3]	68
8.1.3.31 operator-=( ) [3/3]	69
8.1.3.32 operator/=( )	69
8.1.3.33 operator=( ) [1/2]	69
8.1.3.34 operator=( ) [2/2]	69
8.1.3.35 operator==( )	69
8.1.3.36 out_of_range()	69
8.1.3.37 print()	70
8.1.3.38 print_n()	70
8.1.3.39 reserve()	70
8.1.3.40 resize()	70
8.1.3.41 rm_cell()	70



8.1.3.42 row()	71
8.1.3.43 set_data()	71
8.1.3.44 swap_cells()	71
8.1.3.45 swap_cols()	71
8.1.3.46 swap_rows()	72
8.1.3.47 toggle_cell()	72
8.1.3.48 toggle_lock()	72
8.1.3.49 transpose()	72
8.1.3.50 zero_col()	72
8.1.3.51 zero_row()	73
8.1.4 Friends And Related Function Documentation	73
8.1.4.1 BArrayCell< Cell_Type, Data_Type >	73
8.1.4.2 BArrayCell_const< Cell_Type, Data_Type >	73
8.1.5 Member Data Documentation	73
8.1.5.1 visited	73
8.2 BArrayCell< Cell_Type, Data_Type > Class Template Reference	74
8.2.1 Detailed Description	74
8.2.2 Constructor & Destructor Documentation	74
8.2.2.1 BArrayCell()	74
8.2.2.2 ~BArrayCell()	74
8.2.3 Member Function Documentation	75
8.2.3.1 operator Cell_Type()	75
8.2.3.2 operator*=( )	75
8.2.3.3 operator+=( )	75
8.2.3.4 operator-=( )	75
8.2.3.5 operator/=( )	75
8.2.3.6 operator=( )	76
8.2.3.7 operator==( )	76
8.3 BArrayCell_const< Cell_Type, Data_Type > Class Template Reference	76
8.3.1 Detailed Description	76
8.3.2 Constructor & Destructor Documentation	77
8.3.2.1 BArrayCell_const()	77
8.3.2.2 ~BArrayCell_const()	77
8.3.3 Member Function Documentation	77
8.3.3.1 operator Cell_Type()	77
8.3.3.2 operator!=( )	77
8.3.3.3 operator<( )	78
8.3.3.4 operator<=( )	78
8.3.3.5 operator==( )	78
8.3.3.6 operator>( )	78
8.3.3.7 operator>=( )	78
8.4 BArrayDense< Cell_Type, Data_Type > Class Template Reference	79

8.4.1 Detailed Description	81
8.4.2 Constructor & Destructor Documentation	82
8.4.2.1 BArrayDense() [1/6]	82
8.4.2.2 BArrayDense() [2/6]	82
8.4.2.3 BArrayDense() [3/6]	82
8.4.2.4 BArrayDense() [4/6]	83
8.4.2.5 BArrayDense() [5/6]	83
8.4.2.6 BArrayDense() [6/6]	83
8.4.2.7 ~BArrayDense()	83
8.4.3 Member Function Documentation	83
8.4.3.1 clear()	84
8.4.3.2 col() [1/2]	84
8.4.3.3 col() [2/2]	84
8.4.3.4 colsum()	84
8.4.3.5 D() [1/2]	84
8.4.3.6 D() [2/2]	85
8.4.3.7 D_ptr() [1/2]	85
8.4.3.8 D_ptr() [2/2]	85
8.4.3.9 default_val()	85
8.4.3.10 get_cell()	85
8.4.3.11 get_col_vec() [1/2]	86
8.4.3.12 get_col_vec() [2/2]	86
8.4.3.13 get_data()	86
8.4.3.14 get_entries()	86
8.4.3.15 get_row_vec() [1/2]	87
8.4.3.16 get_row_vec() [2/2]	87
8.4.3.17 insert_cell() [1/2]	87
8.4.3.18 insert_cell() [2/2]	87
8.4.3.19 is_dense()	88
8.4.3.20 is_empty()	88
8.4.3.21 ncol()	88
8.4.3.22 nnozero()	88
8.4.3.23 nrow()	88
8.4.3.24 operator>() [1/2]	89
8.4.3.25 operator>() [2/2]	89
8.4.3.26 operator*=( )	89
8.4.3.27 operator+=( ) [1/3]	89
8.4.3.28 operator+=( ) [2/3]	89
8.4.3.29 operator+=( ) [3/3]	90
8.4.3.30 operator-=( ) [1/3]	90
8.4.3.31 operator-=( ) [2/3]	90
8.4.3.32 operator-=( ) [3/3]	90

8.4.3.33 operator/=( )	90
8.4.3.34 operator=( ) [1/2]	91
8.4.3.35 operator=( ) [2/2]	91
8.4.3.36 operator==( )	91
8.4.3.37 out_of_range( )	91
8.4.3.38 print( )	91
8.4.3.39 reserve( )	92
8.4.3.40 resize( )	92
8.4.3.41 rm_cell( )	92
8.4.3.42 row( ) [1/2]	92
8.4.3.43 row( ) [2/2]	92
8.4.3.44 rowsum( )	93
8.4.3.45 set_data( )	93
8.4.3.46 swap_cells( )	93
8.4.3.47 swap_cols( )	93
8.4.3.48 swap_rows( )	94
8.4.3.49 toggle_cell( )	94
8.4.3.50 toggle_lock( )	94
8.4.3.51 transpose( )	94
8.4.3.52 zero_col( )	94
8.4.3.53 zero_row( )	95
8.4.4 Friends And Related Function Documentation	95
8.4.4.1 BArrayDenseCell< Cell_Type, Data_Type >	95
8.4.4.2 BArrayDenseCol< Cell_Type, Data_Type >	95
8.4.4.3 BArrayDenseCol_const< Cell_Type, Data_Type >	95
8.4.4.4 BArrayDenseRow< Cell_Type, Data_Type >	95
8.4.4.5 BArrayDenseRow_const< Cell_Type, Data_Type >	96
8.4.5 Member Data Documentation	96
8.4.5.1 visited	96
8.5 BArrayDenseCell< Cell_Type, Data_Type > Class Template Reference	96
8.5.1 Detailed Description	97
8.5.2 Constructor & Destructor Documentation	97
8.5.2.1 BArrayDenseCell( )	97
8.5.2.2 ~BArrayDenseCell( )	97
8.5.3 Member Function Documentation	97
8.5.3.1 operator Cell_Type( )	97
8.5.3.2 operator*=( )	98
8.5.3.3 operator+=( )	98
8.5.3.4 operator-=( )	98
8.5.3.5 operator/=( )	98
8.5.3.6 operator=( ) [1/2]	98
8.5.3.7 operator=( ) [2/2]	99

8.5.3.8 operator==( )	99
8.5.4 Friends And Related Function Documentation	99
8.5.4.1 BArrayDense< Cell_Type, Data_Type >	99
8.5.4.2 BArrayDenseCol< Cell_Type, Data_Type >	99
8.5.4.3 BArrayDenseCol_const< Cell_Type, Data_Type >	99
8.6 BArrayDenseCell_const< Cell_Type, Data_Type > Class Template Reference	100
8.6.1 Detailed Description	100
8.7 BArrayDenseCol< Cell_Type, Data_Type > Class Template Reference	100
8.7.1 Detailed Description	100
8.7.2 Constructor & Destructor Documentation	100
8.7.2.1 BArrayDenseCol()	101
8.7.3 Member Function Documentation	101
8.7.3.1 begin()	101
8.7.3.2 end()	101
8.7.3.3 operator()( )	101
8.7.3.4 size()	101
8.7.4 Friends And Related Function Documentation	102
8.7.4.1 BArrayDense< Cell_Type, Data_Type >	102
8.7.4.2 BArrayDenseCell< Cell_Type, Data_Type >	102
8.7.4.3 BArrayDenseCell_const< Cell_Type, Data_Type >	102
8.8 BArrayDenseCol_const< Cell_Type, Data_Type > Class Template Reference	102
8.8.1 Detailed Description	103
8.8.2 Constructor & Destructor Documentation	103
8.8.2.1 BArrayDenseCol_const()	103
8.8.3 Member Function Documentation	103
8.8.3.1 begin()	103
8.8.3.2 end()	103
8.8.3.3 operator()( )	104
8.8.3.4 size()	104
8.8.4 Friends And Related Function Documentation	104
8.8.4.1 BArrayDenseCell< Cell_Type, Data_Type >	104
8.8.4.2 BArrayDenseCell_const< Cell_Type, Data_Type >	104
8.9 BArrayDenseRow< Cell_Type, Data_Type > Class Template Reference	104
8.9.1 Detailed Description	105
8.9.2 Constructor & Destructor Documentation	105
8.9.2.1 BArrayDenseRow()	105
8.9.3 Member Function Documentation	105
8.9.3.1 begin()	105
8.9.3.2 end()	106
8.9.3.3 operator()( )	106
8.9.3.4 size()	106
8.9.4 Friends And Related Function Documentation	106

8.9.4.1 BArrayDense< Cell_Type, Data_Type > . . . . .	106
8.9.4.2 BArrayDenseCell< Cell_Type, Data_Type > . . . . .	106
8.9.4.3 BArrayDenseCell_const< Cell_Type, Data_Type > . . . . .	107
8.10 BArrayDenseRow_const< Cell_Type, Data_Type > Class Template Reference . . . . .	107
8.10.1 Detailed Description . . . . .	107
8.10.2 Constructor & Destructor Documentation . . . . .	107
8.10.2.1 BArrayDenseRow_const() . . . . .	108
8.10.3 Member Function Documentation . . . . .	108
8.10.3.1 begin() . . . . .	108
8.10.3.2 end() . . . . .	108
8.10.3.3 operator>() . . . . .	108
8.10.3.4 size() . . . . .	108
8.10.4 Friends And Related Function Documentation . . . . .	109
8.10.4.1 BArrayDenseCell< Cell_Type, Data_Type > . . . . .	109
8.10.4.2 BArrayDenseCell_const< Cell_Type, Data_Type > . . . . .	109
8.11 BArrayRow< Cell_Type, Data_Type > Class Template Reference . . . . .	109
8.11.1 Detailed Description . . . . .	109
8.11.2 Constructor & Destructor Documentation . . . . .	110
8.11.2.1 BArrayRow() . . . . .	110
8.11.2.2 ~BArrayRow() . . . . .	110
8.11.3 Member Function Documentation . . . . .	110
8.11.3.1 operator BArrayRow< Cell_Type, Data_Type >() . . . . .	110
8.11.3.2 operator*=( ) . . . . .	110
8.11.3.3 operator+=( ) . . . . .	110
8.11.3.4 operator-=( ) . . . . .	111
8.11.3.5 operator/=( ) . . . . .	111
8.11.3.6 operator=( ) . . . . .	111
8.11.3.7 operator==( ) . . . . .	111
8.12 BArrayRow_const< Cell_Type, Data_Type > Class Template Reference . . . . .	111
8.12.1 Detailed Description . . . . .	112
8.12.2 Constructor & Destructor Documentation . . . . .	112
8.12.2.1 BArrayRow_const() . . . . .	112
8.12.2.2 ~BArrayRow_const() . . . . .	112
8.12.3 Member Function Documentation . . . . .	112
8.12.3.1 operator BArrayRow_const< Cell_Type, Data_Type >() . . . . .	112
8.12.3.2 operator"!=( ) . . . . .	112
8.12.3.3 operator<() . . . . .	113
8.12.3.4 operator<=( ) . . . . .	113
8.12.3.5 operator==( ) . . . . .	113
8.12.3.6 operator>() . . . . .	113
8.12.3.7 operator>=( ) . . . . .	113
8.13 BArrayVector< Cell_Type, Data_Type > Class Template Reference . . . . .	113

8.13.1 Detailed Description	114
8.13.2 Constructor & Destructor Documentation	114
8.13.2.1 BArrayVector()	114
8.13.2.2 ~BArrayVector()	115
8.13.3 Member Function Documentation	115
8.13.3.1 begin()	115
8.13.3.2 end()	115
8.13.3.3 is_col()	115
8.13.3.4 is_row()	116
8.13.3.5 operator std::vector< Cell_Type >()	116
8.13.3.6 operator*=( )	116
8.13.3.7 operator+=( )	116
8.13.3.8 operator-=( )	116
8.13.3.9 operator/=( )	117
8.13.3.10 operator=( )	117
8.13.3.11 operator==( )	117
8.13.3.12 size()	117
8.14 BArrayVector_const< Cell_Type, Data_Type > Class Template Reference	117
8.14.1 Detailed Description	118
8.14.2 Constructor & Destructor Documentation	118
8.14.2.1 BArrayVector_const()	118
8.14.2.2 ~BArrayVector_const()	118
8.14.3 Member Function Documentation	119
8.14.3.1 begin()	119
8.14.3.2 end()	119
8.14.3.3 is_col()	119
8.14.3.4 is_row()	119
8.14.3.5 operator std::vector< Cell_Type >()	119
8.14.3.6 operator"!=( )	120
8.14.3.7 operator<()	120
8.14.3.8 operator<=( )	120
8.14.3.9 operator==( )	120
8.14.3.10 operator>()	120
8.14.3.11 operator>=( )	121
8.14.3.12 size()	121
8.15 Cell< Cell_Type > Class Template Reference	121
8.15.1 Detailed Description	122
8.15.2 Constructor & Destructor Documentation	122
8.15.2.1 Cell() [1/7]	122
8.15.2.2 Cell() [2/7]	122
8.15.2.3 ~Cell()	122
8.15.2.4 Cell() [3/7]	123

8.15.2.5 Cell() [4/7]	123
8.15.2.6 Cell() [5/7]	123
8.15.2.7 Cell() [6/7]	123
8.15.2.8 Cell() [7/7]	123
8.15.3 Member Function Documentation	123
8.15.3.1 add() [1/4]	124
8.15.3.2 add() [2/4]	124
8.15.3.3 add() [3/4]	124
8.15.3.4 add() [4/4]	124
8.15.3.5 operator Cell_Type()	124
8.15.3.6 operator!=()	124
8.15.3.7 operator=() [1/2]	125
8.15.3.8 operator=() [2/2]	125
8.15.3.9 operator==()	125
8.15.4 Member Data Documentation	125
8.15.4.1 active	125
8.15.4.2 value	125
8.15.4.3 visited	126
8.16 Cell_const< Cell_Type > Class Template Reference	126
8.16.1 Detailed Description	126
8.17 ConstBArrayRowIter< Cell_Type, Data_Type > Class Template Reference	126
8.17.1 Detailed Description	127
8.17.2 Constructor & Destructor Documentation	127
8.17.2.1 ConstBArrayRowIter()	127
8.17.2.2 ~ConstBArrayRowIter()	127
8.17.3 Member Data Documentation	127
8.17.3.1 Array	128
8.17.3.2 current_col	128
8.17.3.3 current_row	128
8.17.3.4 iter	128
8.18 Counter< Array_Type, Data_Type > Class Template Reference	128
8.18.1 Detailed Description	129
8.18.2 Constructor & Destructor Documentation	130
8.18.2.1 Counter() [1/4]	130
8.18.2.2 Counter() [2/4]	130
8.18.2.3 Counter() [3/4]	130
8.18.2.4 Counter() [4/4]	130
8.18.2.5 ~Counter()	131
8.18.3 Member Function Documentation	131
8.18.3.1 count()	131
8.18.3.2 get_description()	131
8.18.3.3 get_hasher()	131

8.18.3.4 <code>get_name()</code> . . . . .	131
8.18.3.5 <code>init()</code> . . . . .	131
8.18.3.6 <code>operator=()</code> [1/2] . . . . .	132
8.18.3.7 <code>operator=()</code> [2/2] . . . . .	132
8.18.3.8 <code>set_hasher()</code> . . . . .	132
8.18.4 Member Data Documentation . . . . .	132
8.18.4.1 <code>count_fun</code> . . . . .	132
8.18.4.2 <code>data</code> . . . . .	133
8.18.4.3 <code>desc</code> . . . . .	133
8.18.4.4 <code>hasher_fun</code> . . . . .	133
8.18.4.5 <code>init_fun</code> . . . . .	133
8.18.4.6 <code>name</code> . . . . .	133
8.19 <code>Counters&lt; Array_Type, Data_Type &gt;</code> Class Template Reference . . . . .	134
8.19.1 Detailed Description . . . . .	134
8.19.2 Constructor & Destructor Documentation . . . . .	134
8.19.2.1 <code>Counters()</code> [1/3] . . . . .	135
8.19.2.2 <code>~Counters()</code> . . . . .	135
8.19.2.3 <code>Counters()</code> [2/3] . . . . .	135
8.19.2.4 <code>Counters()</code> [3/3] . . . . .	135
8.19.3 Member Function Documentation . . . . .	135
8.19.3.1 <code>add_counter()</code> [1/2] . . . . .	136
8.19.3.2 <code>add_counter()</code> [2/2] . . . . .	136
8.19.3.3 <code>add_hash()</code> . . . . .	136
8.19.3.4 <code>gen_hash()</code> . . . . .	136
8.19.3.5 <code>get_descriptions()</code> . . . . .	137
8.19.3.6 <code>get_names()</code> . . . . .	137
8.19.3.7 <code>operator=()</code> [1/2] . . . . .	137
8.19.3.8 <code>operator=()</code> [2/2] . . . . .	137
8.19.3.9 <code>operator[]()</code> . . . . .	138
8.19.3.10 <code>size()</code> . . . . .	138
8.20 DEFM Class Reference . . . . .	139
8.20.1 Detailed Description . . . . .	140
8.20.2 Constructor & Destructor Documentation . . . . .	140
8.20.2.1 <code>DEFM()</code> . . . . .	140
8.20.3 Member Function Documentation . . . . .	140
8.20.3.1 <code>get_column_major()</code> . . . . .	140
8.20.3.2 <code>get_ID()</code> . . . . .	140
8.20.3.3 <code>get_m_order()</code> . . . . .	141
8.20.3.4 <code>get_model()</code> . . . . .	141
8.20.3.5 <code>get_n_covars()</code> . . . . .	141
8.20.3.6 <code>get_n_obs()</code> . . . . .	141
8.20.3.7 <code>get_n_rows()</code> . . . . .	141



8.20.3.8 <code>get_n_y()</code>	141
8.20.3.9 <code>get_X()</code>	142
8.20.3.10 <code>get_X_names()</code>	142
8.20.3.11 <code>get_Y()</code>	142
8.20.3.12 <code>get_Y_names()</code>	142
8.20.3.13 <code>init()</code>	142
8.20.3.14 <code>is_motif()</code>	142
8.20.3.15 <code>logodds()</code>	143
8.20.3.16 <code>motif_census()</code>	143
8.20.3.17 <code>print()</code>	143
8.20.3.18 <code>set_names()</code>	143
8.20.3.19 <code>simulate()</code>	143
8.21 DEFMCounterData Class Reference	144
8.21.1 Detailed Description	144
8.22 DEFMDData Class Reference	144
8.22.1 Detailed Description	145
8.23 DEFMRuleData Class Reference	146
8.23.1 Detailed Description	146
8.24 DEFMRuleDynData Class Reference	147
8.24.1 Detailed Description	147
8.25 Entries< Cell_Type > Class Template Reference	148
8.25.1 Detailed Description	148
8.25.2 Constructor & Destructor Documentation	148
8.25.2.1 <code>Entries()</code> [1/2]	148
8.25.2.2 <code>Entries()</code> [2/2]	149
8.25.2.3 <code>~Entries()</code>	149
8.25.3 Member Function Documentation	149
8.25.3.1 <code>resize()</code>	149
8.25.4 Member Data Documentation	149
8.25.4.1 <code>source</code>	149
8.25.4.2 <code>target</code>	149
8.25.4.3 <code>val</code>	150
8.26 Flock Class Reference	150
8.26.1 Detailed Description	151
8.26.2 Constructor & Destructor Documentation	151
8.26.2.1 <code>Flock()</code>	151
8.26.2.2 <code>~Flock()</code>	151
8.26.3 Member Function Documentation	151
8.26.3.1 <code>add_data()</code>	151
8.26.3.2 <code>colnames()</code>	152
8.26.3.3 <code>get_counters()</code>	152
8.26.3.4 <code>get_model()</code>	152

8.26.3.5 <code>get_stats_support()</code>	152
8.26.3.6 <code>get_stats_target()</code>	153
8.26.3.7 <code>get_support_fun()</code>	153
8.26.3.8 <code>init()</code>	153
8.26.3.9 <code>likelihood_joint()</code>	153
8.26.3.10 <code>nfuncs()</code>	154
8.26.3.11 <code>nleaves()</code>	154
8.26.3.12 <code>nnodes()</code>	154
8.26.3.13 <code>nterms()</code>	154
8.26.3.14 <code>ntrees()</code>	154
8.26.3.15 <code>operator()</code>	154
8.26.3.16 <code>parse_polytomies()</code>	155
8.26.3.17 <code>print()</code>	155
8.26.3.18 <code>set_seed()</code>	155
8.26.3.19 <code>support_size()</code>	156
8.26.4 Member Data Documentation	156
8.26.4.1 <code>dat</code>	156
8.26.4.2 <code>initialized</code>	156
8.26.4.3 <code>model</code>	156
8.26.4.4 <code>nfunctions</code>	156
8.26.4.5 <code>engine</code>	157
8.27 <code>FreqTable&lt; T &gt;</code> Class Template Reference	157
8.27.1 Detailed Description	157
8.27.2 Constructor & Destructor Documentation	158
8.27.2.1 <code>FreqTable()</code>	158
8.27.2.2 <code>~FreqTable()</code>	158
8.27.3 Member Function Documentation	158
8.27.3.1 <code>add()</code>	158
8.27.3.2 <code>as_vector()</code>	158
8.27.3.3 <code>clear()</code>	159
8.27.3.4 <code>get_data()</code>	159
8.27.3.5 <code>get_index()</code>	159
8.27.3.6 <code>make_hash()</code>	159
8.27.3.7 <code>print()</code>	159
8.27.3.8 <code>reserve()</code>	160
8.27.3.9 <code>size()</code>	160
8.28 Geese Class Reference	160
8.28.1 Detailed Description	163
8.28.2 Constructor & Destructor Documentation	163
8.28.2.1 <code>Geese()</code> [1/4]	164
8.28.2.2 <code>Geese()</code> [2/4]	164
8.28.2.3 <code>Geese()</code> [3/4]	164

8.28.2.4 Geese() [ 4 / 4 ] . . . . .	164
8.28.2.5 ~Geese() . . . . .	164
8.28.3 Member Function Documentation . . . . .	164
8.28.3.1 calc_reduced_sequence() . . . . .	165
8.28.3.2 calc_sequence() . . . . .	165
8.28.3.3 colnames() . . . . .	165
8.28.3.4 get_annotated_nodes() . . . . .	165
8.28.3.5 get_annotations() . . . . .	165
8.28.3.6 get_counters() . . . . .	166
8.28.3.7 get_model() . . . . .	166
8.28.3.8 get_probabilities() . . . . .	166
8.28.3.9 get_engine() . . . . .	166
8.28.3.10 get_states() . . . . .	166
8.28.3.11 get_support_fun() . . . . .	167
8.28.3.12 inherit_support() . . . . .	167
8.28.3.13 init() . . . . .	167
8.28.3.14 init_node() . . . . .	167
8.28.3.15 likelihood() . . . . .	167
8.28.3.16 likelihood_exhaust() . . . . .	168
8.28.3.17 nannotations() . . . . .	168
8.28.3.18 nfuncs() . . . . .	168
8.28.3.19 nleaves() . . . . .	168
8.28.3.20 nnodes() . . . . .	168
8.28.3.21 nterms() . . . . .	169
8.28.3.22 observed_counts() . . . . .	169
8.28.3.23 operator=() [ 1 / 2 ] . . . . .	169
8.28.3.24 operator=() [ 2 / 2 ] . . . . .	169
8.28.3.25 parse_polytomies() . . . . .	169
8.28.3.26 predict() . . . . .	170
8.28.3.27 predict_backend() . . . . .	170
8.28.3.28 predict_exhaust() . . . . .	170
8.28.3.29 predict_exhaust_backend() . . . . .	170
8.28.3.30 predict_sim() . . . . .	170
8.28.3.31 print() . . . . .	171
8.28.3.32 print_nodes() . . . . .	171
8.28.3.33 print_observed_counts() . . . . .	171
8.28.3.34 set_seed() . . . . .	171
8.28.3.35 simulate() . . . . .	171
8.28.3.36 support_size() . . . . .	171
8.28.3.37 update_annotations() . . . . .	172
8.28.4 Member Data Documentation . . . . .	172
8.28.4.1 delete_engine . . . . .	172

8.28.4.2 delete_support . . . . .	172
8.28.4.3 etype_default . . . . .	172
8.28.4.4 etype_duplication . . . . .	172
8.28.4.5 etype_either . . . . .	173
8.28.4.6 etype_speciation . . . . .	173
8.28.4.7 initialized . . . . .	173
8.28.4.8 map_to_state_id . . . . .	173
8.28.4.9 nfunctions . . . . .	173
8.28.4.10 nodes . . . . .	173
8.28.4.11 pset_loc . . . . .	174
8.28.4.12 reduced_sequence . . . . .	174
8.28.4.13 sequence . . . . .	174
8.29 Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference . . . . .	175
8.29.1 Detailed Description . . . . .	179
8.29.2 Constructor & Destructor Documentation . . . . .	179
8.29.2.1 Model() [1/3] . . . . .	179
8.29.2.2 Model() [2/3] . . . . .	179
8.29.2.3 Model() [3/3] . . . . .	180
8.29.2.4 ~Model() . . . . .	180
8.29.3 Member Function Documentation . . . . .	180
8.29.3.1 add_array() . . . . .	180
8.29.3.2 add_counter() [1/2] . . . . .	181
8.29.3.3 add_counter() [2/2] . . . . .	181
8.29.3.4 add_hasher() . . . . .	181
8.29.3.5 add_rule() [1/2] . . . . .	181
8.29.3.6 add_rule() [2/2] . . . . .	181
8.29.3.7 add_rule_dyn() [1/2] . . . . .	182
8.29.3.8 add_rule_dyn() [2/2] . . . . .	182
8.29.3.9 colnames() . . . . .	182
8.29.3.10 conditional_prob() . . . . .	182
8.29.3.11 gen_key() . . . . .	183
8.29.3.12 get_arrays2support() . . . . .	183
8.29.3.13 get_counters() . . . . .	183
8.29.3.14 get_normalizing_constants() . . . . .	184
8.29.3.15 get_pset() . . . . .	184
8.29.3.16 get_pset_arrays() . . . . .	184
8.29.3.17 get_pset_probs() . . . . .	184
8.29.3.18 get_pset_stats() [1/2] . . . . .	184
8.29.3.19 get_pset_stats() [2/2] . . . . .	185
8.29.3.20 get_rengine() . . . . .	185
8.29.3.21 get_rules() . . . . .	185

8.29.3.22	<a href="#">get_rules_dyn()</a>	185
8.29.3.23	<a href="#">get_stats_support()</a>	185
8.29.3.24	<a href="#">get_stats_target()</a>	186
8.29.3.25	<a href="#">get_support_fun()</a>	186
8.29.3.26	<a href="#">likelihood()</a> [1/4]	186
8.29.3.27	<a href="#">likelihood()</a> [2/4]	186
8.29.3.28	<a href="#">likelihood()</a> [3/4]	187
8.29.3.29	<a href="#">likelihood()</a> [4/4]	187
8.29.3.30	<a href="#">likelihood_total()</a>	187
8.29.3.31	<a href="#">nrules()</a>	187
8.29.3.32	<a href="#">nrules_dyn()</a>	188
8.29.3.33	<a href="#">nterms()</a>	188
8.29.3.34	<a href="#">operator=()</a>	188
8.29.3.35	<a href="#">print()</a>	188
8.29.3.36	<a href="#">print_stats()</a>	189
8.29.3.37	<a href="#">sample()</a> [1/2]	189
8.29.3.38	<a href="#">sample()</a> [2/2]	189
8.29.3.39	<a href="#">set_counters()</a>	189
8.29.3.40	<a href="#">set_rengine()</a>	190
8.29.3.41	<a href="#">set_rules()</a>	190
8.29.3.42	<a href="#">set_rules_dyn()</a>	190
8.29.3.43	<a href="#">set_seed()</a>	190
8.29.3.44	<a href="#">set_transform_model()</a>	190
8.29.3.45	<a href="#">size()</a>	191
8.29.3.46	<a href="#">size_unique()</a>	191
8.29.3.47	<a href="#">store_psets()</a>	191
8.29.3.48	<a href="#">support_size()</a>	191
8.29.3.49	<a href="#">transform_model()</a>	192
8.29.3.50	<a href="#">update_normalizing_constants()</a>	192
8.29.4	<a href="#">Member Data Documentation</a>	192
8.29.4.1	<a href="#">arrays2support</a>	192
8.29.4.2	<a href="#">counter_fun</a>	192
8.29.4.3	<a href="#">counters</a>	193
8.29.4.4	<a href="#">delete_counters</a>	193
8.29.4.5	<a href="#">delete_rengine</a>	193
8.29.4.6	<a href="#">delete_rules</a>	193
8.29.4.7	<a href="#">delete_rules_dyn</a>	193
8.29.4.8	<a href="#">first_calc_done</a>	194
8.29.4.9	<a href="#">keys2support</a>	194
8.29.4.10	<a href="#">normalizing_constants</a>	194
8.29.4.11	<a href="#">params_last</a>	194
8.29.4.12	<a href="#">pset_arrays</a>	195

8.29.4.13 pset_probs	195
8.29.4.14 pset_stats	195
8.29.4.15 rengine	195
8.29.4.16 rules	196
8.29.4.17 rules_dyn	196
8.29.4.18 stats_support	196
8.29.4.19 stats_support_n_arrays	196
8.29.4.20 stats_target	197
8.29.4.21 support_fun	197
8.29.4.22 transform_model_fun	197
8.29.4.23 transform_model_term_names	198
8.29.4.24 with_pset	198
8.30 NetCounterData Class Reference	198
8.30.1 Detailed Description	198
8.30.2 Constructor & Destructor Documentation	199
8.30.2.1 NetCounterData() [1/2]	199
8.30.2.2 NetCounterData() [2/2]	199
8.30.2.3 ~NetCounterData()	199
8.30.3 Member Data Documentation	199
8.30.3.1 indices	199
8.30.3.2 numbers	199
8.31 NetworkData Class Reference	200
8.31.1 Detailed Description	200
8.31.2 Constructor & Destructor Documentation	200
8.31.2.1 NetworkData() [1/3]	200
8.31.2.2 NetworkData() [2/3]	200
8.31.2.3 NetworkData() [3/3]	201
8.31.2.4 ~NetworkData()	201
8.31.3 Member Data Documentation	201
8.31.3.1 directed	201
8.31.3.2 vertex_attr	202
8.32 Node Class Reference	202
8.32.1 Detailed Description	203
8.32.2 Constructor & Destructor Documentation	203
8.32.2.1 Node() [1/5]	203
8.32.2.2 Node() [2/5]	204
8.32.2.3 Node() [3/5]	204
8.32.2.4 Node() [4/5]	204
8.32.2.5 Node() [5/5]	204
8.32.2.6 ~Node()	204
8.32.3 Member Function Documentation	204
8.32.3.1 get_parent()	205

8.32.3.2 <code>is_leaf()</code>	205
8.32.3.3 <code>noffspring()</code>	205
8.32.4 Member Data Documentation	205
8.32.4.1 annotations	205
8.32.4.2 array	205
8.32.4.3 arrays	206
8.32.4.4 duplication	206
8.32.4.5 id	206
8.32.4.6 narray	206
8.32.4.7 offspring	206
8.32.4.8 ord	207
8.32.4.9 parent	207
8.32.4.10 probability	207
8.32.4.11 subtree_prob	207
8.32.4.12 visited	207
8.33 NodeData Class Reference	208
8.33.1 Detailed Description	208
8.33.2 Constructor & Destructor Documentation	208
8.33.2.1 <code>NodeData()</code>	208
8.33.3 Member Data Documentation	208
8.33.3.1 blengths	209
8.33.3.2 duplication	209
8.33.3.3 states	209
8.34 PhyloCounterData Class Reference	209
8.34.1 Detailed Description	210
8.34.2 Constructor & Destructor Documentation	210
8.34.2.1 <code>PhyloCounterData()</code> [1/2]	210
8.34.2.2 <code>PhyloCounterData()</code> [2/2]	210
8.34.3 Member Function Documentation	210
8.34.3.1 <code>at()</code>	210
8.34.3.2 <code>begin()</code>	210
8.34.3.3 <code>empty()</code>	211
8.34.3.4 <code>end()</code>	211
8.34.3.5 <code>get_counters()</code>	211
8.34.3.6 <code>operator&gt;()</code>	211
8.34.3.7 <code>operator[]()</code>	211
8.34.3.8 <code>push_back()</code>	211
8.34.3.9 <code>reserve()</code>	212
8.34.3.10 <code>shrink_to_fit()</code>	212
8.34.3.11 <code>size()</code>	212
8.35 PhyloRuleDynData Class Reference	212
8.35.1 Detailed Description	213

8.35.2 Constructor & Destructor Documentation	213
8.35.2.1 PhyloRuleDynData()	213
8.35.2.2 ~PhyloRuleDynData()	213
8.35.3 Member Function Documentation	213
8.35.3.1 operator>()	213
8.35.4 Member Data Documentation	213
8.35.4.1 counts	213
8.35.4.2 duplication	214
8.35.4.3 lb	214
8.35.4.4 pos	214
8.35.4.5 ub	214
8.36 PowerSet< Array_Type, Data_Rule_Type > Class Template Reference	214
8.36.1 Detailed Description	215
8.36.2 Constructor & Destructor Documentation	216
8.36.2.1 PowerSet() [1/3]	216
8.36.2.2 PowerSet() [2/3]	216
8.36.2.3 PowerSet() [3/3]	216
8.36.2.4 ~PowerSet()	216
8.36.3 Member Function Documentation	216
8.36.3.1 add_rule() [1/2]	217
8.36.3.2 add_rule() [2/2]	217
8.36.3.3 begin()	217
8.36.3.4 calc()	217
8.36.3.5 end()	217
8.36.3.6 get_data()	218
8.36.3.7 get_data_ptr()	218
8.36.3.8 init_support()	218
8.36.3.9 operator[]()	218
8.36.3.10 reset()	218
8.36.3.11 size()	219
8.36.4 Member Data Documentation	219
8.36.4.1 coordinates_free	219
8.36.4.2 coordinates_locked	219
8.36.4.3 data	219
8.36.4.4 EmptyArray	219
8.36.4.5 M	220
8.36.4.6 N	220
8.36.4.7 n_free	220
8.36.4.8 n_locked	220
8.36.4.9 rules	220
8.36.4.10 rules_deleted	221
8.37 Progress Class Reference	221



8.37.1 Detailed Description	221
8.37.2 Constructor & Destructor Documentation	221
8.37.2.1 Progress()	221
8.37.2.2 ~Progress()	222
8.37.3 Member Function Documentation	222
8.37.3.1 end()	222
8.37.3.2 next()	222
8.38 Rule< Array_Type, Data_Type > Class Template Reference	222
8.38.1 Detailed Description	223
8.38.2 Constructor & Destructor Documentation	223
8.38.2.1 Rule() [1/2]	223
8.38.2.2 Rule() [2/2]	223
8.38.2.3 ~Rule()	224
8.38.3 Member Function Documentation	224
8.38.3.1 D()	224
8.38.3.2 get_description() [1/2]	224
8.38.3.3 get_description() [2/2]	224
8.38.3.4 get_name() [1/2]	224
8.38.3.5 get_name() [2/2]	225
8.38.3.6 operator()()	225
8.39 Rules< Array_Type, Data_Type > Class Template Reference	225
8.39.1 Detailed Description	226
8.39.2 Constructor & Destructor Documentation	226
8.39.2.1 Rules() [1/2]	226
8.39.2.2 Rules() [2/2]	226
8.39.2.3 ~Rules()	227
8.39.3 Member Function Documentation	227
8.39.3.1 add_rule() [1/2]	227
8.39.3.2 add_rule() [2/2]	227
8.39.3.3 begin()	227
8.39.3.4 end()	227
8.39.3.5 get_descriptions()	228
8.39.3.6 get_names()	228
8.39.3.7 get_seq()	228
8.39.3.8 operator()()	228
8.39.3.9 operator=()	229
8.39.3.10 size()	229
8.40 StatsCounter< Array_Type, Data_Type > Class Template Reference	229
8.40.1 Detailed Description	230
8.40.2 Constructor & Destructor Documentation	230
8.40.2.1 StatsCounter() [1/3]	230
8.40.2.2 StatsCounter() [2/3]	231

8.40.2.3 StatsCounter() [3/3]	231
8.40.2.4 ~StatsCounter()	231
8.40.3 Member Function Documentation	231
8.40.3.1 add_counter()	231
8.40.3.2 count_all()	232
8.40.3.3 count_current()	232
8.40.3.4 count_init()	232
8.40.3.5 get_counters()	232
8.40.3.6 get_descriptions()	232
8.40.3.7 get_names()	232
8.40.3.8 reset_array()	232
8.40.3.9 set_counters()	233
8.40.3.10 size()	233
8.41 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference	233
8.41.1 Detailed Description	235
8.41.2 Constructor & Destructor Documentation	235
8.41.2.1 Support() [1/3]	235
8.41.2.2 Support() [2/3]	236
8.41.2.3 Support() [3/3]	236
8.41.2.4 ~Support()	236
8.41.3 Member Function Documentation	236
8.41.3.1 add_counter()	236
8.41.3.2 add_rule() [1/2]	237
8.41.3.3 add_rule() [2/2]	237
8.41.3.4 add_rule_dyn() [1/2]	237
8.41.3.5 add_rule_dyn() [2/2]	237
8.41.3.6 calc()	237
8.41.3.7 eval_rules_dyn()	238
8.41.3.8 get_counters()	238
8.41.3.9 get_counts()	238
8.41.3.10 get_current_stats()	239
8.41.3.11 get_data()	239
8.41.3.12 get_rules()	239
8.41.3.13 get_rules_dyn()	239
8.41.3.14 init_support()	240
8.41.3.15 print()	240
8.41.3.16 reset_array() [1/2]	240
8.41.3.17 reset_array() [2/2]	240
8.41.3.18 set_counters()	241
8.41.3.19 set_rules()	241
8.41.3.20 set_rules_dyn()	241

8.41.4 Member Data Documentation	241
8.41.4.1 change_stats	241
8.41.4.2 coordiantes_n_free	242
8.41.4.3 coordiantes_n_locked	242
8.41.4.4 coordinates_free	242
8.41.4.5 coordinates_locked	242
8.41.4.6 current_stats	242
8.41.4.7 delete_counters	243
8.41.4.8 delete_rules	243
8.41.4.9 delete_rules_dyn	243
8.41.4.10 hashes	243
8.41.4.11 hashes_initialized	243
8.41.4.12 M	244
8.41.4.13 max_num_elements	244
8.41.4.14 N	244
8.41.4.15 n_counters	244
8.42 vecHasher< T > Struct Template Reference	244
8.42.1 Detailed Description	245
8.42.2 Member Function Documentation	245
8.42.2.1 operator>()	245
<b>9 File Documentation</b>	<b>247</b>
9.1 include/barry/barray-bones.hpp File Reference	247
9.2 include/barry/barray-iterator.hpp File Reference	247
9.3 include/barry/barray-meat-operators.hpp File Reference	248
9.3.1 Macro Definition Documentation	248
9.3.1.1 BARRAY_TEMPLATE	249
9.3.1.2 BARRAY_TEMPLATE_ARGS	249
9.3.1.3 BARRAY_TYPE	249
9.3.1.4 COL	249
9.3.1.5 ROW	249
9.3.2 Function Documentation	249
9.3.2.1 BARRAY_TEMPLATE() [1/6]	250
9.3.2.2 BARRAY_TEMPLATE() [2/6]	250
9.3.2.3 BARRAY_TEMPLATE() [3/6]	250
9.3.2.4 BARRAY_TEMPLATE() [4/6]	250
9.3.2.5 BARRAY_TEMPLATE() [5/6]	250
9.3.2.6 BARRAY_TEMPLATE() [6/6]	251
9.3.2.7 BARRAY_TEMPLATE_ARGS()	251
9.3.2.8 BARRAY_TYPE()	251
9.3.2.9 for()	251
9.3.2.10 operator>()	251

9.3.3 Variable Documentation . . . . .	251
9.3.3.1 rhs . . . . .	252
9.3.3.2 this . . . . .	252
9.4 include/barry/barray-meat.hpp File Reference . . . . .	252
9.4.1 Macro Definition Documentation . . . . .	252
9.4.1.1 COL . . . . .	253
9.4.1.2 ROW . . . . .	253
9.5 include/barry/barraycell-bones.hpp File Reference . . . . .	253
9.6 include/barry/barraycell-meat.hpp File Reference . . . . .	254
9.7 include/barry/barraydense-bones.hpp File Reference . . . . .	254
9.8 include/barry/barraydense-meat-operators.hpp File Reference . . . . .	255
9.8.1 Macro Definition Documentation . . . . .	255
9.8.1.1 BDENSE_TEMPLATE . . . . .	255
9.8.1.2 BDENSE_TEMPLATE_ARGS . . . . .	256
9.8.1.3 BDENSE_TYPE . . . . .	256
9.8.1.4 COL . . . . .	256
9.8.1.5 POS . . . . .	256
9.8.1.6 POS_N . . . . .	256
9.8.1.7 ROW . . . . .	257
9.8.2 Function Documentation . . . . .	257
9.8.2.1 BDENSE_TEMPLATE() [1/4] . . . . .	257
9.8.2.2 BDENSE_TEMPLATE() [2/4] . . . . .	257
9.8.2.3 BDENSE_TEMPLATE() [3/4] . . . . .	257
9.8.2.4 BDENSE_TEMPLATE() [4/4] . . . . .	257
9.8.2.5 BDENSE_TEMPLATE_ARGS() . . . . .	258
9.8.2.6 BDENSE_TYPE() . . . . .	258
9.9 include/barry/barraydense-meat.hpp File Reference . . . . .	258
9.9.1 Macro Definition Documentation . . . . .	258
9.9.1.1 COL . . . . .	258
9.9.1.2 POS . . . . .	259
9.9.1.3 POS_N . . . . .	259
9.9.1.4 ROW . . . . .	259
9.9.1.5 ZERO_CELL . . . . .	259
9.10 include/barry/barraydensecell-bones.hpp File Reference . . . . .	259
9.10.1 Macro Definition Documentation . . . . .	260
9.10.1.1 POS . . . . .	260
9.11 include/barry/barraydensecell-meat.hpp File Reference . . . . .	260
9.11.1 Macro Definition Documentation . . . . .	260
9.11.1.1 POS . . . . .	261
9.12 include/barry/barraydensecol-bones.hpp File Reference . . . . .	261
9.12.1 Macro Definition Documentation . . . . .	261
9.12.1.1 POS . . . . .	261

9.12.1.2 POS_N	262
9.12.1.3 ZERO_CELL	262
9.13 include/barry/barraydenserow-bones.hpp File Reference	262
9.13.1 Macro Definition Documentation	263
9.13.1.1 POS	263
9.13.1.2 POS_N	263
9.13.1.3 ZERO_CELL	263
9.14 include/barry/barrayrow-bones.hpp File Reference	263
9.15 include/barry/barrayrow-meat.hpp File Reference	263
9.15.1 Macro Definition Documentation	264
9.15.1.1 BROW_TEMPLATE	264
9.15.1.2 BROW_TEMPLATE_ARGS	264
9.15.1.3 BROW_TYPE	264
9.15.2 Function Documentation	264
9.15.2.1 BROW_TEMPLATE() [1/5]	264
9.15.2.2 BROW_TEMPLATE() [2/5]	265
9.15.2.3 BROW_TEMPLATE() [3/5]	265
9.15.2.4 BROW_TEMPLATE() [4/5]	265
9.15.2.5 BROW_TEMPLATE() [5/5]	265
9.16 include/barry/barrayvector-bones.hpp File Reference	265
9.17 include/barry/barrayvector-meat.hpp File Reference	266
9.18 include/barry/barry-configuration.hpp File Reference	266
9.18.1 Macro Definition Documentation	267
9.18.1.1 BARRY_CHECK_SUPPORT	267
9.18.1.2 BARRY_ISFINITE	267
9.18.1.3 BARRY_MAX_NUM_ELEMENTS	267
9.18.1.4 BARRY_SAFE_EXP	267
9.18.1.5 printf_barry	267
9.18.2 Typedef Documentation	268
9.18.2.1 Map	268
9.19 include/barry/barry-debug.hpp File Reference	268
9.19.1 Macro Definition Documentation	268
9.19.1.1 BARRY_DEBUG_LEVEL	268
9.20 include/barry/barry-macros.hpp File Reference	269
9.20.1 Macro Definition Documentation	269
9.20.1.1 BARRY_NCORES_ARG	269
9.20.1.2 BARRY_ONE	269
9.20.1.3 BARRY_ONE_DENSE	270
9.20.1.4 BARRY_UNUSED	270
9.20.1.5 BARRY_ZERO	270
9.20.1.6 BARRY_ZERO_DENSE	270
9.21 include/barry/barry.hpp File Reference	270

9.21.1 Macro Definition Documentation	272
9.21.1.1 BARRY_HPP	272
9.21.1.2 BARRY_VERSION	272
9.21.1.3 BARRY_VERSION_MAYOR	272
9.21.1.4 BARRY_VERSION_MINOR	272
9.21.1.5 COUNTER_FUNCTION	272
9.21.1.6 COUNTER_LAMBDA	273
9.21.1.7 RULE_FUNCTION	273
9.21.1.8 RULE_LAMBDA	273
9.22 include/barry/cell-bones.hpp File Reference	273
9.23 include/barry/cell-meat.hpp File Reference	274
9.24 include/barry/col-bones.hpp File Reference	274
9.25 include/barry/counters-bones.hpp File Reference	274
9.26 include/barry/counters-meat.hpp File Reference	275
9.26.1 Macro Definition Documentation	276
9.26.1.1 COUNTER_TEMPLATE	276
9.26.1.2 COUNTER_TEMPLATE_ARGS	277
9.26.1.3 COUNTER_TYPE	277
9.26.1.4 COUNTERS_TEMPLATE	277
9.26.1.5 COUNTERS_TEMPLATE_ARGS	277
9.26.1.6 COUNTERS_TYPE	277
9.26.1.7 TMP_HASHER_CALL	277
9.26.2 Function Documentation	278
9.26.2.1 count_fun()	278
9.26.2.2 COUNTER_TEMPLATE() [1/9]	278
9.26.2.3 COUNTER_TEMPLATE() [2/9]	278
9.26.2.4 COUNTER_TEMPLATE() [3/9]	278
9.26.2.5 COUNTER_TEMPLATE() [4/9]	278
9.26.2.6 COUNTER_TEMPLATE() [5/9]	279
9.26.2.7 COUNTER_TEMPLATE() [6/9]	279
9.26.2.8 COUNTER_TEMPLATE() [7/9]	279
9.26.2.9 COUNTER_TEMPLATE() [8/9]	279
9.26.2.10 COUNTER_TEMPLATE() [9/9]	279
9.26.2.11 COUNTERS_TEMPLATE() [1/9]	280
9.26.2.12 COUNTERS_TEMPLATE() [2/9]	280
9.26.2.13 COUNTERS_TEMPLATE() [3/9]	280
9.26.2.14 COUNTERS_TEMPLATE() [4/9]	280
9.26.2.15 COUNTERS_TEMPLATE() [5/9]	280
9.26.2.16 COUNTERS_TEMPLATE() [6/9]	280
9.26.2.17 COUNTERS_TEMPLATE() [7/9]	281
9.26.2.18 COUNTERS_TEMPLATE() [8/9]	281
9.26.2.19 COUNTERS_TEMPLATE() [9/9]	281

9.26.2.20 data()	281
9.26.2.21 desc()	281
9.26.2.22 for()	281
9.26.2.23 hasher() [1/2]	282
9.26.2.24 hasher() [2/2]	282
9.26.2.25 hasher_fun() [1/2]	282
9.26.2.26 hasher_fun() [2/2]	282
9.26.2.27 if() [1/3]	282
9.26.2.28 if() [2/3]	282
9.26.2.29 if() [3/3]	283
9.26.2.30 init_fun() [1/3]	283
9.26.2.31 init_fun() [2/3]	283
9.26.2.32 init_fun() [3/3]	283
9.26.2.33 name()	283
9.26.3 Variable Documentation	283
9.26.3.1 add_dims	283
9.26.3.2 count_fun_	284
9.26.3.3 counter	284
9.26.3.4 counter_	284
9.26.3.5 data_	284
9.26.3.6 desc_	285
9.26.3.7 fun	285
9.26.3.8 fun_	285
9.26.3.9 hasher_fun_	285
9.26.3.10 i	286
9.26.3.11 init_fun_	286
9.26.3.12 j	286
9.26.3.13 name_	286
9.26.3.14 noexcept	286
9.26.3.15 res	287
9.26.3.16 return	287
9.27 include/barry/counters/network-css.hpp File Reference	287
9.27.1 Macro Definition Documentation	288
9.27.1.1 CSS_APPEND	289
9.27.1.2 CSS_CASE_ELSE	289
9.27.1.3 CSS_CASE_PERCEIVED	289
9.27.1.4 CSS_CASE_TRUTH	289
9.27.1.5 CSS_CHECK_SIZE	289
9.27.1.6 CSS_CHECK_SIZE_INIT	290
9.27.1.7 CSS_MATCH_TYPE	290
9.27.1.8 CSS_NET_COUNTER_LAMBDA_INIT	290
9.27.1.9 CSS_PERCEIVED_CELLS	290

9.27.1.10	CSS_SIZE	291
9.27.1.11	CSS_TRUE_CELLS	291
9.27.2	Function Documentation	291
9.27.2.1	counter_css_census01()	291
9.27.2.2	counter_css_census02()	291
9.27.2.3	counter_css_census03()	292
9.27.2.4	counter_css_census04()	292
9.27.2.5	counter_css_census05()	292
9.27.2.6	counter_css_census06()	292
9.27.2.7	counter_css_census07()	293
9.27.2.8	counter_css_census08()	293
9.27.2.9	counter_css_census09()	293
9.27.2.10	counter_css_census10()	293
9.27.2.11	counter_css_completely_false_recip_comiss()	294
9.27.2.12	counter_css_completely_false_recip_omiss()	294
9.27.2.13	counter_css_mixed_recip()	294
9.27.2.14	counter_css_partially_false_recip_commi()	294
9.27.2.15	counter_css_partially_false_recip_omiss()	295
9.28	include/barry/counters/network.hpp File Reference	295
9.28.1	Macro Definition Documentation	298
9.28.1.1	BARRY_ZERO_NETWORK	298
9.28.1.2	BARRY_ZERO_NETWORK_DENSE	299
9.28.1.3	NET_C_DATA_IDX	299
9.28.1.4	NET_C_DATA_NUM	299
9.28.1.5	NETWORK_COUNTER	299
9.28.1.6	NETWORK_COUNTER_LAMBDA	299
9.28.1.7	NETWORK_RULE	300
9.28.1.8	NETWORK_RULE_LAMBDA	300
9.28.1.9	NETWORKDENSE_COUNTER_LAMBDA	300
9.28.2	Typedef Documentation	300
9.28.2.1	NetCounter	300
9.28.2.2	NetCounters	301
9.28.2.3	NetModel	301
9.28.2.4	NetRule	301
9.28.2.5	NetRules	301
9.28.2.6	NetStatsCounter	301
9.28.2.7	NetSupport	301
9.28.2.8	Network	302
9.28.2.9	NetworkDense	302
9.28.3	Function Documentation	302
9.28.3.1	rules_zerodiag()	302
9.29	include/barry/freqtable.hpp File Reference	302



9.30 include/barry/model-bones.hpp File Reference . . . . .	303
9.31 include/barry/model-meat.hpp File Reference . . . . .	303
9.31.1 Function Documentation . . . . .	304
9.31.1.1 likelihood_() . . . . .	304
9.31.1.2 update_normalizing_constant() . . . . .	304
9.32 include/barry/models/defm.hpp File Reference . . . . .	304
9.33 include/barry/models/defm/counters.hpp File Reference . . . . .	305
9.33.1 Macro Definition Documentation . . . . .	306
9.33.1.1 DEFM_COUNTER . . . . .	306
9.33.1.2 DEFM_COUNTER_LAMBDA . . . . .	307
9.33.1.3 DEFM_RULE . . . . .	307
9.33.1.4 DEFM_RULE_LAMBDA . . . . .	307
9.33.1.5 DEFM_RULEDYN_LAMBDA . . . . .	308
9.33.1.6 UNI_SUB . . . . .	308
9.34 include/barry/models/geese/counters.hpp File Reference . . . . .	308
9.35 include/barry/models/defm/defm-bones.hpp File Reference . . . . .	310
9.36 include/barry/models/defm/defm-meat.hpp File Reference . . . . .	311
9.36.1 Macro Definition Documentation . . . . .	311
9.36.1.1 DEFM_LOOP_ARRAYS . . . . .	311
9.36.1.2 DEFM_RANGES . . . . .	311
9.36.2 Function Documentation . . . . .	312
9.36.2.1 keygen_defm() . . . . .	312
9.37 include/barry/models/defm/defm-types.hpp File Reference . . . . .	312
9.37.1 Typedef Documentation . . . . .	313
9.37.1.1 DEFMArray . . . . .	313
9.38 include/barry/models/defm/formula.hpp File Reference . . . . .	313
9.38.1 Function Documentation . . . . .	313
9.38.1.1 defm_motif_parser() . . . . .	314
9.39 include/barry/models/geese.hpp File Reference . . . . .	315
9.40 include/barry/models/geese/flock-bones.hpp File Reference . . . . .	315
9.41 include/barry/models/geese/flock-meat.hpp File Reference . . . . .	316
9.42 include/barry/models/geese/geese-bones.hpp File Reference . . . . .	316
9.42.1 Macro Definition Documentation . . . . .	317
9.42.1.1 INITIALIZED . . . . .	317
9.42.2 Function Documentation . . . . .	317
9.42.2.1 keygen_full() . . . . .	317
9.42.2.2 RULE_FUNCTION() . . . . .	317
9.42.2.3 vec_diff() . . . . .	318
9.42.2.4 vector_caster() . . . . .	318
9.43 include/barry/models/geese/geese-meat-constructors.hpp File Reference . . . . .	318
9.44 include/barry/models/geese/geese-meat-likelihood.hpp File Reference . . . . .	319
9.45 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference . . . . .	320

9.46 include/barry/models/geese/geese-meat-predict.hpp File Reference	320
9.47 include/barry/models/geese/geese-meat-predict_exhaust.hpp File Reference	321
9.48 include/barry/models/geese/geese-meat-predict_sim.hpp File Reference	321
9.49 include/barry/models/geese/geese-meat-simulate.hpp File Reference	322
9.50 include/barry/models/geese/geese-meat.hpp File Reference	322
9.51 include/barry/models/geese/geese-node-bones.hpp File Reference	323
9.52 include/barry/models/geese/geese-types.hpp File Reference	323
9.52.1 Macro Definition Documentation	324
9.52.1.1 POS	324
9.52.2 Typedef Documentation	324
9.52.2.1 PhyloArray	324
9.52.2.2 PhyloCounter	325
9.52.2.3 PhyloCounters	325
9.52.2.4 PhyloModel	325
9.52.2.5 PhyloPowerSet	325
9.52.2.6 PhyloRule	325
9.52.2.7 PhyloRuleData	325
9.52.2.8 PhyloRuleDyn	326
9.52.2.9 PhyloRules	326
9.52.2.10 PhyloRulesDyn	326
9.52.2.11 PhyloStatsCounter	326
9.52.2.12 PhyloSupport	326
9.53 include/barry/powerset-bones.hpp File Reference	326
9.54 include/barry/powerset-meat.hpp File Reference	327
9.55 include/barry/progress.hpp File Reference	327
9.55.1 Macro Definition Documentation	328
9.55.1.1 BARRY_PROGRESS_BAR_WIDTH	328
9.56 include/barry/rules-bones.hpp File Reference	328
9.56.1 Function Documentation	329
9.56.1.1 rule_fun_default()	329
9.57 include/barry/rules-meat.hpp File Reference	329
9.58 include/barry/statscounter-bones.hpp File Reference	329
9.59 include/barry/statscounter-meat.hpp File Reference	330
9.59.1 Macro Definition Documentation	331
9.59.1.1 STATSCOUNTER_TEMPLATE	331
9.59.1.2 STATSCOUNTER_TEMPLATE_ARGS	331
9.59.1.3 STATSCOUNTER_TYPE	331
9.59.2 Function Documentation	331
9.59.2.1 clear()	331
9.59.2.2 for()	332
9.59.2.3 resize()	332
9.59.2.4 STATSCOUNTER_TEMPLATE() [1/9]	332

9.59.2.5 STATSCOUNTER_TEMPLATE() [2/9]	332
9.59.2.6 STATSCOUNTER_TEMPLATE() [3/9]	332
9.59.2.7 STATSCOUNTER_TEMPLATE() [4/9]	332
9.59.2.8 STATSCOUNTER_TEMPLATE() [5/9]	333
9.59.2.9 STATSCOUNTER_TEMPLATE() [6/9]	333
9.59.2.10 STATSCOUNTER_TEMPLATE() [7/9]	333
9.59.2.11 STATSCOUNTER_TEMPLATE() [8/9]	333
9.59.2.12 STATSCOUNTER_TEMPLATE() [9/9]	333
9.59.3 Variable Documentation	333
9.59.3.1 counter	334
9.59.3.2 counter_deleted	334
9.59.3.3 counters	334
9.59.3.4 counters_	334
9.59.3.5 current_stats	334
9.59.3.6 EmptyArray	335
9.59.3.7 f_	335
9.59.3.8 j	335
9.59.3.9 return	335
9.60 include/barry/support-bones.hpp File Reference	335
9.61 include/barry/support-meat.hpp File Reference	336
9.61.1 Macro Definition Documentation	336
9.61.1.1 BARRY_SUPPORT_MEAT_HPP	336
9.62 include/barry/typedefs.hpp File Reference	337
9.62.1 Typedef Documentation	339
9.62.1.1 Col_type	339
9.62.1.2 Counter_fun_type	339
9.62.1.3 Counts_type	339
9.62.1.4 Hasher_fun_type	339
9.62.1.5 MapVec_type	340
9.62.1.6 Row_type	340
9.62.1.7 Rule_fun_type	340
9.62.2 Function Documentation	340
9.62.2.1 sort_array()	340
9.62.2.2 vec_equal()	341
9.62.2.3 vec_equal_approx()	341
9.62.2.4 vec_inner_prod() [1/2]	342
9.62.2.5 vec_inner_prod() [2/2]	342
9.63 README.md File Reference	342



# 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. Its primary goal is to provide a general framework for building discrete exponential-family models. A particular example is Exponential Random Graph Models (ERGMs), but we can use `barry` to deal with non-square arrays.

Among the key features included in `barry`, we have:

- Sparse arrays.
- User-defined count statistics.
- User-defined constrain of the support set.
- Powerset generation of binary arrays.
- Discrete Exponential Family Models module (DEFMs).
- Pooled DEFMs.

To use `barry`, you can either download the entire repository or, since it is header-only, the single header version `barry.hpp`.

This library was created and maintained by Dr. George G. Vega Yon as part of his doctoral dissertation "Essays on Bioinformatics and Social Network Analysis: Statistical and Computational Methods for Complex Systems."

## 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?
    net.print("Current view");

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

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

    // 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
```

## Features

### Efficient memory usage

One of the key features of `barry` is that it will handle memory efficiently. In the case of pooled-data models, the module for statistical models avoids double-counting support when possible by keeping track of what datasets (networks, for instance) share the same.

## Documentation

More information can be found in the Doxygen website [here](#) and in the PDF version of the documentation [here](#).

## 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 . . . . .	13
Statistical Models . . . . .	24
Network counters . . . . .	24
Phylo counters . . . . .	47
Phylo rules . . . . .	35



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

BArray< Cell_Type, Data_Type > . . . . .	59
BArray< bool, bool > . . . . .	59
BArray< Cell_Type, Data_Type > . . . . .	59
BArrayCell< Cell_Type, Data_Type > . . . . .	74
BArrayCell_const< Cell_Type, Data_Type > . . . . .	76
BArrayDense< Cell_Type, Data_Type > . . . . .	79
BArrayDense< bool, bool > . . . . .	79
BArrayDenseCell< Cell_Type, Data_Type > . . . . .	96
BArrayDenseCell_const< Cell_Type, Data_Type > . . . . .	100
BArrayDenseCol< Cell_Type, Data_Type > . . . . .	100
BArrayDenseCol_const< Cell_Type, Data_Type > . . . . .	102
BArrayDenseRow< Cell_Type, Data_Type > . . . . .	104
BArrayDenseRow_const< Cell_Type, Data_Type > . . . . .	107
BArrayRow< Cell_Type, Data_Type > . . . . .	109
BArrayRow_const< Cell_Type, Data_Type > . . . . .	111
BArrayVector< Cell_Type, Data_Type > . . . . .	113
BArrayVector_const< Cell_Type, Data_Type > . . . . .	117
Cell< Cell_Type > . . . . .	121
Cell< bool > . . . . .	121
Cell_const< Cell_Type > . . . . .	126
ConstBArrayRowIter< Cell_Type, Data_Type > . . . . .	126
Counter< Array_Type, Data_Type > . . . . .	128
Counters< Array_Type, Data_Type > . . . . .	134
Counters< Array_Type, Data_Type > . . . . .	134
Counters< BArray< bool, bool >, bool > . . . . .	134
Counters< BArray<>, bool > . . . . .	134
DEFMCounterData . . . . .	144
DEFMData . . . . .	144
DEFMModel . . . . .	
DEFM . . . . .	139
DEFMRuleData . . . . .	146
DEFMRuleDynData . . . . .	147
Entries< Cell_Type > . . . . .	148
Flock . . . . .	150
FreqTable< T > . . . . .	157

Geese . . . . .	160
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > . . . . .	175
NetCounterData . . . . .	198
NetworkData . . . . .	200
Node . . . . .	202
NodeData . . . . .	208
PhyloCounterData . . . . .	209
PhyloRuleDynData . . . . .	212
PowerSet< Array_Type, Data_Rule_Type > . . . . .	214
Progress . . . . .	221
Rule< Array_Type, Data_Type > . . . . .	222
Rules< Array_Type, Data_Type > . . . . .	225
Rules< BArray< bool, bool >, bool > . . . . .	225
Rules< BArray<>, bool > . . . . .	225
StatsCounter< Array_Type, Data_Type > . . . . .	229
StatsCounter< BArray<>, bool > . . . . .	229
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > . . . . .	233
Support< BArray<>, bool, bool, bool > . . . . .	233
vecHasher< T > . . . . .	244

## Chapter 4

# Class Index

### 4.1 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	59
<a href="#">BArrayCell&lt; Cell_Type, Data_Type &gt;</a>	74
<a href="#">BArrayCell_const&lt; Cell_Type, Data_Type &gt;</a>	76
<a href="#">BArrayDense&lt; Cell_Type, Data_Type &gt;</a>	
Baseline class for binary arrays	79
<a href="#">BArrayDenseCell&lt; Cell_Type, Data_Type &gt;</a>	96
<a href="#">BArrayDenseCell_const&lt; Cell_Type, Data_Type &gt;</a>	100
<a href="#">BArrayDenseCol&lt; Cell_Type, Data_Type &gt;</a>	100
<a href="#">BArrayDenseCol_const&lt; Cell_Type, Data_Type &gt;</a>	102
<a href="#">BArrayDenseRow&lt; Cell_Type, Data_Type &gt;</a>	104
<a href="#">BArrayDenseRow_const&lt; Cell_Type, Data_Type &gt;</a>	107
<a href="#">BArrayRow&lt; Cell_Type, Data_Type &gt;</a>	109
<a href="#">BArrayRow_const&lt; Cell_Type, Data_Type &gt;</a>	111
<a href="#">BArrayVector&lt; Cell_Type, Data_Type &gt;</a>	
Row or column of a <a href="#">BArray</a>	113
<a href="#">BArrayVector_const&lt; Cell_Type, Data_Type &gt;</a>	117
<a href="#">Cell&lt; Cell_Type &gt;</a>	
Entries in <a href="#">BArray</a> . For now, it only has two members:	121
<a href="#">Cell_const&lt; Cell_Type &gt;</a>	126
<a href="#">ConstBArrayRowIter&lt; Cell_Type, Data_Type &gt;</a>	126
<a href="#">Counter&lt; Array_Type, Data_Type &gt;</a>	
A counter function based on change statistics	128
<a href="#">Counters&lt; Array_Type, Data_Type &gt;</a>	
Vector of counters	134
<a href="#">DEFM</a>	139
<a href="#">DEFMCounterData</a>	
Data class used to store arbitrary size_t or double vectors	144
<a href="#">DEFMData</a>	
Data class for <a href="#">DEFM</a> arrays	144
<a href="#">DEFMRuleData</a>	146
<a href="#">DEFMRuleDynData</a>	147
<a href="#">Entries&lt; Cell_Type &gt;</a>	
A wrapper class to store source, target, val from a <a href="#">BArray</a> object	148
<a href="#">Flock</a>	
A <a href="#">Flock</a> is a group of <a href="#">Geese</a>	150

<a href="#">FreqTable&lt; T &gt;</a>	
Frequency table of vectors	157
<a href="#">Geese</a>	
Annotated Phylo <a href="#">Model</a>	160
<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:	175
<a href="#">NetCounterData</a>	
Data class used to store arbitrary size_t or double vectors	198
<a href="#">NetworkData</a>	
Data class for Networks	200
<a href="#">Node</a>	
A single node for the model	202
<a href="#">NodeData</a>	
Data definition for the <a href="#">PhyloArray</a> class	208
<a href="#">PhyloCounterData</a>	209
<a href="#">PhyloRuleDynData</a>	212
<a href="#">PowerSet&lt; Array_Type, Data_Rule_Type &gt;</a>	
Powerset of a binary array	214
<a href="#">Progress</a>	
A simple progress bar	221
<a href="#">Rule&lt; Array_Type, Data_Type &gt;</a>	
Rule for determining if a cell should be included in a sequence	222
<a href="#">Rules&lt; Array_Type, Data_Type &gt;</a>	
Vector of objects of class <a href="#">Rule</a>	225
<a href="#">StatsCounter&lt; Array_Type, Data_Type &gt;</a>	
Count stats for a single Array	229
<a href="#">Support&lt; Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type &gt;</a>	
Compute the support of sufficient statistics	233
<a href="#">vecHasher&lt; T &gt;</a>	244

## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

include/barry/barray-bones.hpp	247
include/barry/barray-iterator.hpp	247
include/barry/barray-meat-operators.hpp	248
include/barry/barray-meat.hpp	252
include/barry/barraycell-bones.hpp	253
include/barry/barraycell-meat.hpp	254
include/barry/barraydense-bones.hpp	254
include/barry/barraydense-meat-operators.hpp	255
include/barry/barraydense-meat.hpp	258
include/barry/barraydensecell-bones.hpp	259
include/barry/barraydensecell-meat.hpp	260
include/barry/barraydensecol-bones.hpp	261
include/barry/barraydenserow-bones.hpp	262
include/barry/barrayrow-bones.hpp	263
include/barry/barrayrow-meat.hpp	263
include/barry/barrayvector-bones.hpp	265
include/barry/barrayvector-meat.hpp	266
include/barry/barray-configuration.hpp	266
include/barry/barray-debug.hpp	268
include/barry/barray-macros.hpp	269
include/barry/barray.hpp	270
include/barry/cell-bones.hpp	273
include/barry/cell-meat.hpp	274
include/barry/col-bones.hpp	274
include/barry/counters-bones.hpp	274
include/barry/counters-meat.hpp	275
include/barry/freqtable.hpp	302
include/barry/model-bones.hpp	303
include/barry/model-meat.hpp	303
include/barry/powerset-bones.hpp	326
include/barry/powerset-meat.hpp	327
include/barry/progress.hpp	327
include/barry/rules-bones.hpp	328
include/barry/rules-meat.hpp	329
include/barry/statscounter-bones.hpp	329

include/barry/statscounter-meat.hpp	330
include/barry/support-bones.hpp	335
include/barry/support-meat.hpp	336
include/barry/typedefs.hpp	337
include/barry/counters/network-css.hpp	287
include/barry/counters/network.hpp	295
include/barry/models/defm.hpp	304
include/barry/models/geese.hpp	315
include/barry/models/defm/counters.hpp	305
include/barry/models/defm/defm-bones.hpp	310
include/barry/models/defm/defm-meat.hpp	311
include/barry/models/defm/defm-types.hpp	312
include/barry/models/defm/formula.hpp	313
include/barry/models/geese/counters.hpp	308
include/barry/models/geese/flock-bones.hpp	315
include/barry/models/geese/flock-meat.hpp	316
include/barry/models/geese/geese-bones.hpp	316
include/barry/models/geese/geese-meat-constructors.hpp	318
include/barry/models/geese/geese-meat-likelihood.hpp	319
include/barry/models/geese/geese-meat-likelihood_exhaust.hpp	320
include/barry/models/geese/geese-meat-predict.hpp	320
include/barry/models/geese/geese-meat-predict_exhaust.hpp	321
include/barry/models/geese/geese-meat-predict_sim.hpp	321
include/barry/models/geese/geese-meat-simulate.hpp	322
include/barry/models/geese/geese-meat.hpp	322
include/barry/models/geese/geese-node-bones.hpp	323
include/barry/models/geese/geese-types.hpp	323



## Chapter 6

# Module Documentation

### 6.1 Counting

#### Classes

- class [NetworkData](#)  
*Data class for Networks.*
- class [Counter](#)< [Array\\_Type](#), [Data\\_Type](#) >  
*A counter function based on change statistics.*

#### Macros

- `#define MAKE\_DEFM\_HASHER(hasher, a, cov)`
- `#define MAKE\_DUPL\_VARS()`
- `#define IS\_EITHER() ((DATA_AT == Geese::etype\_either)`
- `#define IS\_DUPLICATION() ((DATA_AT == Geese::etype\_duplication) & (DPL))`
- `#define IS\_SPECIATION() ((DATA_AT == Geese::etype\_speciation) & (!DPL))`
- `#define IF\_MATCHES()`
- `#define IF\_NOTMATCHES()`
- `#define PHYLO\_COUNTER\_LAMBDA(a)`  
*Extension of a simple counter.*
- `#define PHYLO\_RULE\_DYN\_LAMBDA(a)`
- `#define PHYLO\_CHECK\_MISSING()`
- `std::string get\_last\_name (size_t d)`
- `void counter\_overall\_gains (PhyloCounters *counters, size_t duplication=Geese::etype\_default)`  
*Overall functional gains.*
- `void counter\_gains (PhyloCounters *counters, std::vector< size_t > nfun, size_t duplication=Geese::etype\_default)`  
*Functional gains for a specific function (nfun).*
- `void counter\_gains\_k\_offspring (PhyloCounters *counters, std::vector< size_t > nfun, size_t k=1u, size_t duplication=Geese::etype\_default)`  
*k genes gain function nfun*
- `void counter\_genes\_changing (PhyloCounters *counters, size_t duplication=Geese::etype\_default)`  
*Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)*

- void `counter_preserve_pseudogene` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=Geese::etype_default`)  
*Keeps track of how many pairs of genes preserve pseudostate.*
- void `counter_prop_genes_changing` (`PhyloCounters *counters`, `size_t duplication=Geese::etype_default`)  
*Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)*
- void `counter_overall_loss` (`PhyloCounters *counters`, `size_t duplication=Geese::etype_default`)  
*Overall functional loss.*
- void `counter_maxfuns` (`PhyloCounters *counters`, `size_t lb`, `size_t ub`, `size_t duplication=Geese::etype_default`)  
*Cap the number of functions per gene.*
- void `counter_loss` (`PhyloCounters *counters`, `std::vector< size_t > nfun`, `size_t duplication=Geese::etype_default`)  
*Total count of losses for an specific function.*
- void `counter_overall_changes` (`PhyloCounters *counters`, `size_t duplication=Geese::etype_default`)  
*Total number of changes. Use this statistic to account for "preservation".*
- void `counter_subfun` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=Geese::etype_default`)  
*Total count of Sub-functionalization events.*
- void `counter_cogain` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=Geese::etype_default`)  
*Co-evolution (joint gain or loss)*
- void `counter_longest` (`PhyloCounters *counters`, `size_t duplication=Geese::etype_default`)  
*Longest branch mutates (either by gain or by loss)*
- void `counter_neofun` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=Geese::etype_default`)  
*Total number of neofunctionalization events.*
- void `counter_pairwise_neofun_singlefun` (`PhyloCounters *counters`, `size_t nfunA`, `size_t duplication=Geese::etype_default`)  
*Total number of neofunctionalization events  $\sum_u \sum_{w < u} [x(u,a)*(1 - x(w,a)) + (1 - x(u,a)) * x(w,a)]$  change  
stat:  $\delta\{x(u,a): 0 \rightarrow 1\} = 1 - 2 * x(w,a)$*
- void `counter_neofun_a2b` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=Geese::etype_default`)  
*Total number of neofunctionalization events.*
- void `counter_co_opt` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=Geese::etype_default`)  
*Function co-opting.*
- void `counter_k_genes_changing` (`PhyloCounters *counters`, `size_t k`, `size_t duplication=Geese::etype_default`)  
*Indicator function. Equals to one if k genes changed and zero otherwise.*
- void `counter_less_than_p_prop_genes_changing` (`PhyloCounters *counters`, `double p`, `size_t duplication=Geese::etype_default`)  
*Indicator function. Equals to one if k genes changed and zero otherwise.*
- void `counter_gains_from_0` (`PhyloCounters *counters`, `std::vector< size_t > nfun`, `size_t duplication=Geese::etype_default`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `counter_overall_gains_from_0` (`PhyloCounters *counters`, `size_t duplication=Geese::etype_default`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `counter_pairwise_overall_change` (`PhyloCounters *counters`, `size_t duplication=Geese::etype_default`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `counter_pairwise_preserving` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=Geese::etype_default`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `counter_pairwise_first_gain` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=Geese::etype_default`)  
*Used when all the functions are in 0 (like the root node prob.)*

### 6.1.1 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.

### 6.1.2 Macro Definition Documentation

#### 6.1.2.1 IF\_MATCHES

```
#define IF_MATCHES( )
```

**Value:**

```
MAKE_DUPL_VARS() \
if (IS_EITHER() || IS_DUPLICATION() || IS_SPECIATION())
```

Definition at line 20 of file `counters.hpp`.

#### 6.1.2.2 IF\_NOTMATCHES

```
#define IF_NOTMATCHES( )
```

**Value:**

```
MAKE_DUPL_VARS() \
if (!IS_EITHER() && !IS_DUPLICATION() && !IS_SPECIATION())
```

Definition at line 22 of file `counters.hpp`.

#### 6.1.2.3 IS\_DUPLICATION

```
#define IS_DUPLICATION( ) ((DATA_AT == Geese::etype_duplication) & (DPL))
```

Definition at line 17 of file `counters.hpp`.

#### 6.1.2.4 IS\_EITHER

```
#define IS_EITHER( ) (DATA_AT == Geese::etype_either)
```

Definition at line 16 of file counters.hpp.

#### 6.1.2.5 IS\_SPECIATION

```
#define IS_SPECIATION( ) ((DATA_AT == Geese::etype_speciation) & (!DPL))
```

Definition at line 18 of file counters.hpp.

#### 6.1.2.6 MAKE\_DEFM\_HASHER

```
#define MAKE_DEFM_HASHER(  
    hasher,  
    a,  
    cov )
```

##### Value:

```
barry::Hasher_fun_type<DEFMArray,DEFMCounterData> \  
hasher = [cov](const DEFMArray & array, DEFMCounterData * d) { \  
    std::vector< double > res; \  
    /* Adding the column feature */ \  
    for (size_t i = 0u; i < array.nrow(); ++i) \  
        res.push_back(array.D()(i, cov)); \  
    /* Adding the fixed dims */ \  
    for (size_t i = 0u; i < (array.nrow() - 1); ++i) \  
        for (size_t j = 0u; j < array.ncol(); ++j) \  
            res.push_back(array(i, j)); \  
    return res; \  
};
```

Details on the available counters for DEFMworkData can be found in the [Network counters](#) section.

Definition at line 21 of file counters.hpp.

#### 6.1.2.7 MAKE\_DUPL\_VARS

```
#define MAKE_DUPL_VARS( )
```

##### Value:

```
bool DPL = Array.D_ptr()->duplication; \  
size_t DATA_AT = data[0u];
```

Details about the available counters for PhyloArray objects can be found in the [Phylo counters](#) section.

Definition at line 12 of file counters.hpp.

### 6.1.2.8 PHYLO\_CHECK\_MISSING

```
#define PHYLO_CHECK_MISSING( )
```

**Value:**

```
if (Array.D_ptr() == nullptr) \
    throw std::logic_error("The array data is nullptr."); \
```

Definition at line 42 of file counters.hpp.

### 6.1.2.9 PHYLO\_COUNTER\_LAMBDA

```
#define PHYLO_COUNTER_LAMBDA(  
    a )
```

**Value:**

```
barry::Counter_fun_type<PhyloArray, PhyloCounterData> a = \
    [](const PhyloArray & Array, size_t i, size_t 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 36 of file counters.hpp.

### 6.1.2.10 PHYLO\_RULE\_DYN\_LAMBDA

```
#define PHYLO_RULE_DYN_LAMBDA(  
    a )
```

**Value:**

```
barry::Rule_fun_type<PhyloArray, PhyloRuleDynData> a = \
    [](const PhyloArray & Array, size_t i, size_t j, PhyloRuleDynData & data)
```

Definition at line 39 of file counters.hpp.

## 6.1.3 Function Documentation

### 6.1.3.1 counter\_co\_opt()

```
void counter_co_opt (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = Geese::etype_default ) [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 1296 of file counters.hpp.

### 6.1.3.2 counter\_cogain()

```
void counter_cogain (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = Geese::etype_default ) [inline]
```

Co-evolution (joint gain or loss)

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

Definition at line 791 of file counters.hpp.

### 6.1.3.3 counter\_gains()

```
void counter_gains (
    PhyloCounters * counters,
    std::vector< size_t > nfun,
    size_t duplication = Geese::etype_default ) [inline]
```

Functional gains for a specific function (nfun).

Definition at line 96 of file counters.hpp.

#### 6.1.3.4 counter\_gains\_from\_0()

```
void counter_gains_from_0 (
    PhyloCounters * counters,
    std::vector< size_t > nfun,
    size_t duplication = Geese::etype_default ) [inline]
```

Used when all the functions are in 0 (like the root node prob.)

Needs to specify function a.

Definition at line 1630 of file counters.hpp.

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

```
void counter_gains_k_offspring (
    PhyloCounters * counters,
    std::vector< size_t > nfun,
    size_t k = 1u,
    size_t duplication = Geese::etype_default ) [inline]
```

k genes gain function nfun

Definition at line 156 of file counters.hpp.

#### 6.1.3.6 counter\_genes\_changing()

```
void counter_genes_changing (
    PhyloCounters * counters,
    size_t duplication = Geese::etype_default ) [inline]
```

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

Definition at line 228 of file counters.hpp.

#### 6.1.3.7 counter\_k\_genes\_changing()

```
void counter_k_genes_changing (
    PhyloCounters * counters,
    size_t k,
    size_t duplication = Geese::etype_default ) [inline]
```

Indicator function. Equals to one if  $k$  genes changed and zero otherwise.

Definition at line 1394 of file counters.hpp.

#### 6.1.3.8 counter\_less\_than\_p\_prop\_genes\_changing()

```
void counter_less_than_p_prop_genes_changing (
    PhyloCounters * counters,
    double p,
    size_t duplication = Geese::etype_default ) [inline]
```

Indicator function. Equals to one if  $k$  genes changed and zero otherwise.

< How many genes diverge the parent

Definition at line 1514 of file counters.hpp.

#### 6.1.3.9 counter\_longest()

```
void counter_longest (
    PhyloCounters * counters,
    size_t duplication = Geese::etype_default ) [inline]
```

Longest branch mutates (either by gain or by loss)

Definition at line 848 of file counters.hpp.

#### 6.1.3.10 counter\_loss()

```
void counter_loss (
    PhyloCounters * counters,
    std::vector< size_t > nfun,
    size_t duplication = Geese::etype_default ) [inline]
```

Total count of losses for an specific function.

Definition at line 591 of file counters.hpp.

#### 6.1.3.11 counter\_maxfuns()

```
void counter_maxfuns (
    PhyloCounters * counters,
    size_t lb,
    size_t ub,
    size_t duplication = Geese::etype_default ) [inline]
```

Cap the number of functions per gene.

Definition at line 529 of file counters.hpp.



#### 6.1.3.12 counter\_neofun()

```
void counter_neofun (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = Geese::etype_default ) [inline]
```

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 1018 of file counters.hpp.

#### 6.1.3.13 counter\_neofun\_a2b()

```
void counter_neofun_a2b (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = Geese::etype_default ) [inline]
```

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 1163 of file counters.hpp.

#### 6.1.3.14 counter\_overall\_changes()

```
void counter_overall_changes (
    PhyloCounters * counters,
    size_t duplication = Geese::etype_default ) [inline]
```

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

Definition at line 643 of file counters.hpp.

#### 6.1.3.15 counter\_overall\_gains()

```
void counter_overall_gains (
    PhyloCounters * counters,
    size_t duplication = Geese::etype_default ) [inline]
```

Overall functional gains.

Total number of gains (irrespective of the function).

Definition at line 58 of file counters.hpp.

#### 6.1.3.16 counter\_overall\_gains\_from\_0()

```
void counter_overall_gains_from_0 (
    PhyloCounters * counters,
    size_t duplication = Geese::etype_default ) [inline]
```

Used when all the functions are in 0 (like the root node prob.)

Needs to specify function a.

Definition at line 1696 of file counters.hpp.

#### 6.1.3.17 counter\_overall\_loss()

```
void counter_overall_loss (
    PhyloCounters * counters,
    size_t duplication = Geese::etype_default ) [inline]
```

Overall functional loss.

Definition at line 481 of file counters.hpp.

#### 6.1.3.18 counter\_pairwise\_first\_gain()

```
void counter_pairwise_first_gain (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = Geese::etype_default ) [inline]
```

Used when all the functions are in 0 (like the root node prob.)

Needs to specify function a.  $\sum x(a)^3(1-x(b))^3 + x(b)^3(1-x(a))^3 + x(a)^3 * x(b)^3 + (1-x(a))^3 * (1-x(b))^3$

Definition at line 1948 of file counters.hpp.

#### 6.1.3.19 counter\_pairwise\_neofun\_singlefun()

```
void counter_pairwise_neofun_singlefun (
    PhyloCounters * counters,
    size_t nfunA,
    size_t duplication = Geese::etype_default ) [inline]
```

Total number of neofunctionalization events  $\sum_u \sum_{w < u} [x(u,a)*(1-x(w,a)) + (1-x(u,a)) * x(w,a)]$  change stat:  $\Delta\{x(u,a): 0 \rightarrow 1\} = 1 - 2 * x(w,a)$

Definition at line 1099 of file counters.hpp.

**6.1.3.20 counter\_pairwise\_overall\_change()**

```
void counter_pairwise_overall_change (
    PhyloCounters * counters,
    size_t duplication = Geese::etype_default ) [inline]
```

Used when all the functions are in 0 (like the root node prob.)

Needs to specify function a.

Definition at line 1744 of file counters.hpp.

**6.1.3.21 counter\_pairwise\_preserving()**

```
void counter_pairwise_preserving (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = Geese::etype_default ) [inline]
```

Used when all the functions are in 0 (like the root node prob.)

Needs to specify function a.  $\sum x(a)^3(1-x(b))^3 + x(b)^3(1-x(a))^3 + x(a)^3 * x(b)^3 + (1-x(a))^3 * (1-x(b))^3$

Definition at line 1809 of file counters.hpp.

**6.1.3.22 counter\_preserve\_pseudogene()**

```
void counter_preserve_pseudogene (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = Geese::etype_default ) [inline]
```

Keeps track of how many pairs of genes preserve pseudostate.

Definition at line 297 of file counters.hpp.

**6.1.3.23 counter\_prop\_genes\_changing()**

```
void counter_prop_genes_changing (
    PhyloCounters * counters,
    size_t duplication = Geese::etype_default ) [inline]
```

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

Definition at line 379 of file counters.hpp.

#### 6.1.3.24 counter\_subfun()

```
void counter_subfun (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = Geese::etype_default ) [inline]
```

Total count of Sub-functionalization events.

It requires to specify data = {funA, funB}

Definition at line 702 of file counters.hpp.

#### 6.1.3.25 get\_last\_name()

```
std::string get_last_name (
    size_t d ) [inline]
```

Definition at line 45 of file counters.hpp.

## 6.2 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](#).*

#### 6.2.1 Detailed Description

Statistical models available in `barry`.

## 6.3 Network counters

[Counters](#) for network models.

## Functions

- `template<typename Tnet = Network>`  
`void counter_edges (NetCounters< Tnet > *counters)`  
*Number of edges.*
- `template<typename Tnet = Network>`  
`void counter_isolates (NetCounters< Tnet > *counters)`  
*Number of isolated vertices.*
- `template<> void counter_isolates (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter_mutual (NetCounters< Tnet > *counters)`  
*Number of mutual ties.*
- `template<typename Tnet = Network>`  
`void counter_istar2 (NetCounters< Tnet > *counters)`
- `template<> void counter_istar2 (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter_ostar2 (NetCounters< Tnet > *counters)`
- `template<> void counter_ostar2 (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter_ttriads (NetCounters< Tnet > *counters)`
- `template<> void counter_ttriads (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter_ctriads (NetCounters< Tnet > *counters)`
- `template<> void counter_ctriads (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter_density (NetCounters< Tnet > *counters)`
- `template<typename Tnet = Network>`  
`void counter_idegree15 (NetCounters< Tnet > *counters)`
- `template<> void counter_idegree15 (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter_odegree15 (NetCounters< Tnet > *counters)`
- `template<> void counter_odegree15 (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter_absdiff (NetCounters< Tnet > *counters, size_t attr_id, double alpha=1.0)`  
*Sum of absolute attribute difference between ego and alter.*
- `template<typename Tnet = Network>`  
`void counter_diff (NetCounters< Tnet > *counters, size_t 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)
- `template<typename Tnet = Network>`  
`void counter_nodeicov (NetCounters< Tnet > *counters, size_t attr_id)`
- `template<typename Tnet = Network>`  
`void counter_nodeocov (NetCounters< Tnet > *counters, size_t attr_id)`
- `template<typename Tnet = Network>`  
`void counter_nodcov (NetCounters< Tnet > *counters, size_t attr_id)`
- `template<typename Tnet = Network>`  
`void counter_nodematch (NetCounters< Tnet > *counters, size_t attr_id)`
- `template<typename Tnet = Network>`  
`void counter_idegree (NetCounters< Tnet > *counters, std::vector< size_t > d)`  
*Counts number of vertices with a given in-degree.*
- `template<> void counter_idegree (NetCounters< NetworkDense > *counters, std::vector< size_t > d)`
- `template<typename Tnet = Network>`  
`void counter_odegree (NetCounters< Tnet > *counters, std::vector< size_t > d)`  
*Counts number of vertices with a given out-degree.*
- `template<> void counter_odegree (NetCounters< NetworkDense > *counters, std::vector< size_t > d)`

- `template<typename Tnet = Network>`  
`void counter_degree (NetCounters< Tnet > *counters, std::vector< size_t > d)`  
*Counts number of vertices with a given out-degree.*
- `void counter_ones (DEFMCounters *counters, int covar_index=-1, std::string vname="", const std::vector< std::string > *x_names=nullptr)`  
*Prevalence of ones.*
- `void counter_logit_intercept (DEFMCounters *counters, size_t n_y, std::vector< size_t > which={}, int covar_index=-1, std::string vname="", const std::vector< std::string > *x_names=nullptr, const std::vector< std::string > *y_names=nullptr)`
- `void counter_transition (DEFMCounters *counters, std::vector< size_t > coords, std::vector< bool > signs, size_t m_order, size_t n_y, int covar_index=-1, std::string vname="", const std::vector< std::string > *x_names=nullptr, const std::vector< std::string > *y_names=nullptr)`  
*Prevalence of ones.*
- `void counter_transition_formula (DEFMCounters *counters, std::string formula, size_t m_order, size_t n_y, int covar_index=-1, std::string vname="", const std::vector< std::string > *x_names=nullptr, const std::vector< std::string > *y_names=nullptr)`  
*Prevalence of ones.*
- `void counter_fixed_effect (DEFMCounters *counters, int covar_index, double k, std::string vname="", const std::vector< std::string > *x_names=nullptr)`  
*Prevalence of ones.*

## Returns true if the cell is free

### Parameters

<i>rules</i>	A pointer to a DEFMRules object ( <code>Rules&lt;DEFMArray, bool&gt;</code> ).
--------------	--

- `void rules_markov_fixed (DEFMRules *rules, size_t markov_order)`  
*Number of edges.*
- `void rules_dont_become_zero (DEFMSupport *support, std::vector< size_t > ids)`  
*Blocks switching a one to zero.*

## 6.3.1 Detailed Description

`Counters` for network models.

### Parameters

<i>counters</i>	A pointer to a NetCounters object ( <code>Counters&lt;Network, NetCounterData&gt;</code> ).
<i>counters</i>	A pointer to a DEFMCounters object ( <code>Counters&lt;DEFMArray, DEFMCOUNTERData&gt;</code> ).

## 6.3.2 Function Documentation

### 6.3.2.1 counter\_absdiff()

```
template<typename Tnet = Network>
void counter_absdiff (
    NetCounters< Tnet > * counters,
    size_t attr_id,
    double alpha = 1.0 ) [inline]
```

Sum of absolute attribute difference between ego and alter.

Definition at line 908 of file network.hpp.

### 6.3.2.2 counter\_ctriads() [1/2]

```
template<>
void counter_ctriads (
    NetCounters< NetworkDense > * counters ) [inline]
```

Definition at line 665 of file network.hpp.

### 6.3.2.3 counter\_ctriads() [2/2]

```
template<typename Tnet = Network>
void counter_ctriads (
    NetCounters< Tnet > * counters ) [inline]
```

Definition at line 610 of file network.hpp.

### 6.3.2.4 counter\_degree()

```
template<typename Tnet = Network>
void counter_degree (
    NetCounters< Tnet > * counters,
    std::vector< size_t > d ) [inline]
```

Counts number of vertices with a given out-degree.

Definition at line 1326 of file network.hpp.

### 6.3.2.5 counter\_density()

```
template<typename Tnet = Network>
void counter_density (
    NetCounters< Tnet > * counters ) [inline]
```

Definition at line 729 of file network.hpp.

### 6.3.2.6 counter\_diff()

```
template<typename Tnet = Network>
void counter_diff (
    NetCounters< Tnet > * counters,
    size_t 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 953 of file network.hpp.

### 6.3.2.7 counter\_edges()

```
template<typename Tnet = Network>
void counter_edges (
    NetCounters< Tnet > * counters ) [inline]
```

Number of edges.

Definition at line 152 of file network.hpp.

### 6.3.2.8 counter\_fixed\_effect()

```
void counter_fixed_effect (
    DEFMCounters * counters,
    int covar_index,
    double k,
    std::string vname = "",
    const std::vector< std::string > * x_names = nullptr ) [inline]
```

Prevalence of ones.

#### Parameters

<i>counters</i>	Pointer ot a vector of counters
<i>covar_index</i>	If >= than 0, then the interaction



Definition at line 610 of file counters.hpp.

#### 6.3.2.9 counter\_iddegree() [1/2]

```
template<>
void counter_iddegree (
    NetCounters< NetworkDense > * counters,
    std::vector< size_t > d ) [inline]
```

Definition at line 1170 of file network.hpp.

#### 6.3.2.10 counter\_iddegree() [2/2]

```
template<typename Tnet = Network>
void counter_iddegree (
    NetCounters< Tnet > * counters,
    std::vector< size_t > d ) [inline]
```

Counts number of vertices with a given in-degree.

Definition at line 1123 of file network.hpp.

#### 6.3.2.11 counter\_iddegree15() [1/2]

```
template<>
void counter_iddegree15 (
    NetCounters< NetworkDense > * counters ) [inline]
```

Definition at line 785 of file network.hpp.

#### 6.3.2.12 counter\_iddegree15() [2/2]

```
template<typename Tnet = Network>
void counter_iddegree15 (
    NetCounters< Tnet > * counters ) [inline]
```

Definition at line 757 of file network.hpp.

**6.3.2.13 counter\_isolates() [1/2]**

```
template<>
void counter_isolates (
    NetCounters< NetworkDense > * counters ) [inline]
```

Definition at line 215 of file network.hpp.

**6.3.2.14 counter\_isolates() [2/2]**

```
template<typename Tnet = Network>
void counter_isolates (
    NetCounters< Tnet > * counters ) [inline]
```

Number of isolated vertices.

Definition at line 175 of file network.hpp.

**6.3.2.15 counter\_istar2() [1/2]**

```
template<>
void counter_istar2 (
    NetCounters< NetworkDense > * counters ) [inline]
```

Definition at line 338 of file network.hpp.

**6.3.2.16 counter\_istar2() [2/2]**

```
template<typename Tnet = Network>
void counter_istar2 (
    NetCounters< Tnet > * counters ) [inline]
```

Definition at line 312 of file network.hpp.

**6.3.2.17 counter\_logit\_intercept()**

```
void counter_logit_intercept (
    DEFMCounters * counters,
    size_t n_y,
    std::vector< size_t > which = {},
    int covar_index = -1,
    std::string vname = "",
    const std::vector< std::string > * x_names = nullptr,
    const std::vector< std::string > * y_names = nullptr ) [inline]
```

Definition at line 151 of file counters.hpp.

#### 6.3.2.18 counter\_mutual()

```
template<typename Tnet = Network>
void counter_mutual (
    NetCounters< Tnet > * counters ) [inline]
```

Number of mutual ties.

Definition at line 256 of file network.hpp.

#### 6.3.2.19 counter\_nodecov()

```
template<typename Tnet = Network>
void counter_nodecov (
    NetCounters< Tnet > * counters,
    size_t attr_id ) [inline]
```

Definition at line 1066 of file network.hpp.

#### 6.3.2.20 counter\_nodeicov()

```
template<typename Tnet = Network>
void counter_nodeicov (
    NetCounters< Tnet > * counters,
    size_t attr_id ) [inline]
```

Definition at line 1016 of file network.hpp.

#### 6.3.2.21 counter\_nodematch()

```
template<typename Tnet = Network>
void counter_nodematch (
    NetCounters< Tnet > * counters,
    size_t attr_id ) [inline]
```

Definition at line 1091 of file network.hpp.

#### 6.3.2.22 counter\_nodeocov()

```
template<typename Tnet = Network>
void counter_nodeocov (
    NetCounters< Tnet > * counters,
    size_t attr_id ) [inline]
```

Definition at line 1041 of file network.hpp.

**6.3.2.23 counter\_odegree()** [1/2]

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

Definition at line 1271 of file network.hpp.

**6.3.2.24 counter\_odegree()** [2/2]

```
template<typename Tnet = Network>
void counter_odegree (
    NetCounters< Tnet > * counters,
    std::vector< size_t > d ) [inline]
```

Counts number of vertices with a given out-degree.

Definition at line 1223 of file network.hpp.

**6.3.2.25 counter\_odegree15()** [1/2]

```
template<>
void counter_odegree15 (
    NetCounters< NetworkDense > * counters ) [inline]
```

Definition at line 862 of file network.hpp.

**6.3.2.26 counter\_odegree15()** [2/2]

```
template<typename Tnet = Network>
void counter_odegree15 (
    NetCounters< Tnet > * counters ) [inline]
```

Definition at line 834 of file network.hpp.

**6.3.2.27 counter\_ones()**

```
void counter_ones (
    DEFMCounters * counters,
    int covar_index = -1,
    std::string vname = "",
    const std::vector< std::string > * x_names = nullptr ) [inline]
```

Prevalence of ones.

## Parameters

<i>counters</i>	Pointer ot a vector of counters
<i>covar_index</i>	If $\geq$ than 0, then the interaction

Definition at line 81 of file counters.hpp.

**6.3.2.28 counter\_ostar2() [1/2]**

```
template<>
void counter_ostar2 (
    NetCounters< NetworkDense > * counters ) [inline]
```

Definition at line 404 of file network.hpp.

**6.3.2.29 counter\_ostar2() [2/2]**

```
template<typename Tnet = Network>
void counter_ostar2 (
    NetCounters< Tnet > * counters ) [inline]
```

Definition at line 376 of file network.hpp.

**6.3.2.30 counter\_transition()**

```
void counter_transition (
    DEFMCounters * counters,
    std::vector< size_t > coords,
    std::vector< bool > signs,
    size_t m_order,
    size_t n_y,
    int covar_index = -1,
    std::string vname = "",
    const std::vector< std::string > * x_names = nullptr,
    const std::vector< std::string > * y_names = nullptr ) [inline]
```

Prevalence of ones.

## Parameters

<i>counters</i>	Pointer ot a vector of counters
<i>covar_index</i>	If $\geq$ than 0, then the interaction

Definition at line 270 of file counters.hpp.

### 6.3.2.31 counter\_transition\_formula()

```
void counter_transition_formula (
    DEFMCounters * counters,
    std::string formula,
    size_t m_order,
    size_t n_y,
    int covar_index = -1,
    std::string vname = "",
    const std::vector< std::string > * x_names = nullptr,
    const std::vector< std::string > * y_names = nullptr ) [inline]
```

Prevalence of ones.

#### Parameters

<i>counters</i>	Pointer of a vector of counters
<i>covar_index</i>	If >= than 0, then the interaction

Definition at line 579 of file counters.hpp.

### 6.3.2.32 counter\_ttriads() [1/2]

```
template<>
void counter_ttriads (
    NetCounters< NetworkDense > * counters ) [inline]
```

Definition at line 531 of file network.hpp.

### 6.3.2.33 counter\_ttriads() [2/2]

```
template<typename Tnet = Network>
void counter_ttriads (
    NetCounters< Tnet > * counters ) [inline]
```

Definition at line 441 of file network.hpp.

### 6.3.2.34 NETWORK\_COUNTER()

```
NETWORK_COUNTER (
    init_single_attr )
```

Definition at line 997 of file network.hpp.

### 6.3.2.35 rules\_dont\_become\_zero()

```
void rules_dont_become_zero (
    DEFMSupport * support,
    std::vector< size_t > ids ) [inline]
```

Blocks switching a one to zero.

#### Parameters

<i>rules</i>	
<i>ids</i>	Ids of the variables that will follow this rule.

Definition at line 678 of file counters.hpp.

### 6.3.2.36 rules\_markov\_fixed()

```
void rules_markov_fixed (
    DEFMRules * rules,
    size_t markov_order ) [inline]
```

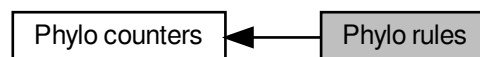
Number of edges.

Definition at line 653 of file counters.hpp.

## 6.4 Phylo rules

[Rules](#) for phylogenetic modeling.

Collaboration diagram for Phylo rules:



## Classes

- class [DEFMRuleDynData](#)

## Functions

- [DEFMData::DEFMData](#) ()  
*Vector indicating which covariates are included in the model.*
- [DEFMData::DEFMData](#) ([DEFMArray](#) \*array\_, const double \*covariates\_, size\_t obs\_start\_, size\_t X\_ncol\_, size\_t X\_nrow\_)  
*Constructor.*
- double [DEFMData::operator\(\)](#) (size\_t i, size\_t j) const  
*Access to the row (i) column (j) data.*
- double [DEFMData::at](#) (size\_t i, size\_t j) const
- size\_t [DEFMData::ncol](#) () const
- size\_t [DEFMData::nrow](#) () const
- void [DEFMData::print](#) () const
- [DEFMData::~~DEFMData](#) ()
- [DEFMCounterData::DEFMCounterData](#) ()
- [DEFMCounterData::DEFMCounterData](#) (const std::vector< size\_t > indices\_, const std::vector< double > numbers\_, const std::vector< bool > logical\_, bool is\_motif\_=true)
- size\_t [DEFMCounterData::idx](#) (size\_t i) const
- double [DEFMCounterData::num](#) (size\_t i) const
- bool [DEFMCounterData::is\\_true](#) (size\_t i) const
- [DEFMCounterData::~~DEFMCounterData](#) ()
- double [DEFMRuleData::num](#) (size\_t i) const
- size\_t [DEFMRuleData::idx](#) (size\_t i) const
- bool [DEFMRuleData::is\\_true](#) (size\_t i) const
- [DEFMRuleData::DEFMRuleData](#) ()
- [DEFMRuleData::DEFMRuleData](#) (std::vector< double > numbers\_, std::vector< size\_t > indices\_, std::vector< bool > logical\_)
- [DEFMRuleData::DEFMRuleData](#) (std::vector< double > numbers\_, std::vector< size\_t > indices\_)
- [DEFMRuleDynData::DEFMRuleDynData](#) (const std::vector< double > \*counts\_, std::vector< double > numbers\_= {}, std::vector< size\_t > indices\_= {}, std::vector< bool > logical\_= {})
- [DEFMRuleDynData::~~DEFMRuleDynData](#) ()
- void [rule\\_dyn\\_limit\\_changes](#) ([PhyloSupport](#) \*support, size\_t pos, size\_t lb, size\_t ub, size\_t duplication=[Geese::etype\\_default](#))  
*Overall functional gains.*

## Variables

- [DEFMArray](#) \* [DEFMData::array](#)
- const double \* [DEFMData::covariates](#)  
*Vector of covariates (complete vector)*
- size\_t [DEFMData::obs\\_start](#)  
*Index of the observation in the data.*
- size\_t [DEFMData::X\\_ncol](#)  
*Number of columns in the array of covariates.*
- size\_t [DEFMData::X\\_nrow](#)  
*Number of rows in the array of covariates.*
- std::vector< size\_t > [DEFMData::covar\\_sort](#)
- std::vector< size\_t > [DEFMData::covar\\_used](#)  
*Value where the sorting of the covariates is stored.*
- std::vector< size\_t > [DEFMCounterData::indices](#)
- std::vector< double > [DEFMCounterData::numbers](#)
- std::vector< bool > [DEFMCounterData::logical](#)
- bool [DEFMCounterData::is\\_motif](#)



*If false, then is a logit intercept.*

- `std::vector< double > DEFMRuleData::numbers`
- `std::vector< size_t > DEFMRuleData::indices`
- `std::vector< bool > DEFMRuleData::logical`
- `bool DEFMRuleData::init = false`
- `const std::vector< double > * DEFMRuleDynData::counts`

### Convenient typedefs for network objects.

- `typedef barry::Counter< DEFMArray, DEFMCounterData > DEFMCounter`
- `typedef barry::Counters< DEFMArray, DEFMCounterData > DEFMCounters`
- `typedef barry::Support< DEFMArray, DEFMCounterData, DEFMRuleData, DEFMRuleDynData > DEFMSupport`
- `typedef barry::StatsCounter< DEFMArray, DEFMCounterData > DEFMStatsCounter`
- `typedef barry::Model< DEFMArray, DEFMCounterData, DEFMRuleData, DEFMRuleDynData > DEFMModel`
- `typedef barry::Rule< DEFMArray, DEFMRuleData > DEFMRule`
- `typedef barry::Rules< DEFMArray, DEFMRuleData > DEFMRules`
- `typedef barry::Rule< DEFMArray, DEFMRuleDynData > DEFMRuleDyn`
- `typedef barry::Rules< DEFMArray, DEFMRuleDynData > DEFMRulesDyn`

#### 6.4.1 Detailed Description

[Rules](#) for phylogenetic modeling.

Parameters

<i>rules</i>	A pointer to a <code>PhyloRules</code> object ( <code>Rules&lt;PhyloArray, PhyloRuleData&gt;</code> ).
--------------	--

#### 6.4.2 Typedef Documentation

##### 6.4.2.1 DEFMCounter

```
typedef barry::Counter<DEFMArray, DEFMCounterData > DEFMCounter
```

Definition at line 171 of file `defm-types.hpp`.

##### 6.4.2.2 DEFMCounters

```
typedef barry::Counters<DEFMArray, DEFMCounterData> DEFMCounters
```

Definition at line 172 of file `defm-types.hpp`.

#### 6.4.2.3 DEFMMModel

```
typedef barry::Model<DEFMArray, DEFMCounterData, DEFMRuleData, DEFMRuleDynData> DEFMMModel
```

Definition at line 175 of file defm-types.hpp.

#### 6.4.2.4 DEFMRule

```
typedef barry::Rule<DEFMArray, DEFMRuleData> DEFMRule
```

Definition at line 178 of file defm-types.hpp.

#### 6.4.2.5 DEFMRuleDyn

```
typedef barry::Rule<DEFMArray, DEFMRuleDynData> DEFMRuleDyn
```

Definition at line 180 of file defm-types.hpp.

#### 6.4.2.6 DEFMRules

```
typedef barry::Rules<DEFMArray, DEFMRuleData> DEFMRules
```

Definition at line 179 of file defm-types.hpp.

#### 6.4.2.7 DEFMRulesDyn

```
typedef barry::Rules<DEFMArray, DEFMRuleDynData> DEFMRulesDyn
```

Definition at line 181 of file defm-types.hpp.

#### 6.4.2.8 DEFMStatsCounter

```
typedef barry::StatsCounter<DEFMArray, DEFMCounterData> DEFMStatsCounter
```

Definition at line 174 of file defm-types.hpp.

### 6.4.2.9 DEFMSupport

```
typedef barry::Support<DEFMArray, DEFMCounterData, DEFMRuleData,DEFMRuleDynData> DEFMSupport
```

Definition at line 173 of file defm-types.hpp.

## 6.4.3 Function Documentation

### 6.4.3.1 at()

```
double DEFMData::at (
    size_t i,
    size_t j ) const
```

### 6.4.3.2 DEFMCounterData() [1/2]

```
DEFMCounterData::DEFMCounterData ( ) [inline]
```

Definition at line 72 of file defm-types.hpp.

### 6.4.3.3 DEFMCounterData() [2/2]

```
DEFMCounterData::DEFMCounterData (
    const std::vector< size_t > indices_,
    const std::vector< double > numbers_,
    const std::vector< bool > logical_,
    bool is_motif_ = true ) [inline]
```

Definition at line 73 of file defm-types.hpp.

### 6.4.3.4 DEFMDData() [1/2]

```
DEFMDData::DEFMDData ( ) [inline]
```

Vector indicating which covariates are included in the model.

Definition at line 27 of file defm-types.hpp.

### 6.4.3.5 DEFMDData() [2/2]

```
DEFMDData::DEFMDData (
    DEFMArray * array_,
    const double * covariates_,
    size_t obs_start_,
    size_t X_ncol_,
    size_t X_nrow_ ) [inline]
```

Constructor.

## Parameters

<i>covariates</i> ↔ —	Pointer to the attribute data.
<i>obs_</i> ↔ <i>start_</i>	Location of the current observation in the covariates vector
<i>X_ncol_</i>	Number of columns (covariates.)

Definition at line 36 of file defm-types.hpp.

#### 6.4.3.6 DEFMRuleData() [1/3]

```
DEFMRuleData::DEFMRuleData ( ) [inline]
```

Definition at line 102 of file defm-types.hpp.

#### 6.4.3.7 DEFMRuleData() [2/3]

```
DEFMRuleData::DEFMRuleData (
    std::vector< double > numbers_,
    std::vector< size_t > indices_ ) [inline]
```

Definition at line 110 of file defm-types.hpp.

#### 6.4.3.8 DEFMRuleData() [3/3]

```
DEFMRuleData::DEFMRuleData (
    std::vector< double > numbers_,
    std::vector< size_t > indices_,
    std::vector< bool > logical_ ) [inline]
```

Definition at line 104 of file defm-types.hpp.

#### 6.4.3.9 DEFMRuleDynData()

```
DEFMRuleDynData::DEFMRuleDynData (
    const std::vector< double > * counts_,
    std::vector< double > numbers_ = {},
    std::vector< size_t > indices_ = {},
    std::vector< bool > logical_ = {} ) [inline]
```

Definition at line 156 of file defm-types.hpp.

**6.4.3.10 idx()** [1/2]

```
size_t DEFMCOUNTERData::idx (
    size_t i ) const [inline]
```

Definition at line 81 of file defm-types.hpp.

**6.4.3.11 idx()** [2/2]

```
size_t DEFMRULEData::idx (
    size_t i ) const [inline]
```

Definition at line 99 of file defm-types.hpp.

**6.4.3.12 is\_true()** [1/2]

```
bool DEFMCOUNTERData::is_true (
    size_t i ) const [inline]
```

Definition at line 83 of file defm-types.hpp.

**6.4.3.13 is\_true()** [2/2]

```
bool DEFMRULEData::is_true (
    size_t i ) const [inline]
```

Definition at line 100 of file defm-types.hpp.

**6.4.3.14 ncol()**

```
size_t DEFMDATA::ncol ( ) const [inline]
```

Definition at line 123 of file defm-types.hpp.

**6.4.3.15 nrow()**

```
size_t DEFMDATA::nrow ( ) const [inline]
```

Definition at line 127 of file defm-types.hpp.

**6.4.3.16 num()** [1/2]

```
double DEFMCOUNTERData::num (
    size_t i ) const [inline]
```

Definition at line 82 of file defm-types.hpp.

**6.4.3.17 num()** [2/2]

```
double DEFMRULEData::num (
    size_t i ) const [inline]
```

Definition at line 98 of file defm-types.hpp.

**6.4.3.18 operator()()**

```
double DEFMDATA::operator() (
    size_t i,
    size_t j ) const [inline]
```

Access to the row (i) column (j) data.

**Parameters**

<i>i</i>	
<i>j</i>	

**Returns**

double

Definition at line 118 of file defm-types.hpp.

**6.4.3.19 print()**

```
void DEFMDATA::print ( ) const [inline]
```

Definition at line 131 of file defm-types.hpp.

**6.4.3.20 rule\_dyn\_limit\_changes()**

```
void rule_dyn_limit_changes (
    PhyloSupport * support,
    size_t pos,
    size_t lb,
    size_t ub,
    size_t duplication = Geese::etype_default ) [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 2058 of file counters.hpp.

**6.4.3.21 ~DEFMCounterData()**

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

Definition at line 85 of file defm-types.hpp.

**6.4.3.22 ~DEFMData()**

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

Definition at line 58 of file defm-types.hpp.

**6.4.3.23 ~DEFMRuleDynData()**

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

Definition at line 163 of file defm-types.hpp.

## 6.4.4 Variable Documentation

### 6.4.4.1 array

```
DEFMArray* DEFMDData::array
```

Definition at line 19 of file defm-types.hpp.

### 6.4.4.2 counts

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

Definition at line 154 of file defm-types.hpp.

### 6.4.4.3 covar\_sort

```
std::vector< size_t > DEFMDData::covar_sort
```

Definition at line 24 of file defm-types.hpp.

### 6.4.4.4 covar\_used

```
std::vector< size_t > DEFMDData::covar_used
```

Value where the sorting of the covariates is stored.

Definition at line 25 of file defm-types.hpp.

### 6.4.4.5 covariates

```
const double* DEFMDData::covariates
```

Vector of covariates (complete vector)

Definition at line 20 of file defm-types.hpp.



#### 6.4.4.6 indices [1/2]

```
std::vector< size_t > DEFMCOUNTERData::indices
```

Definition at line 67 of file defm-types.hpp.

#### 6.4.4.7 indices [2/2]

```
std::vector< size_t > DEFMRULEData::indices
```

Definition at line 93 of file defm-types.hpp.

#### 6.4.4.8 init

```
bool DEFMRULEData::init = false
```

Definition at line 96 of file defm-types.hpp.

#### 6.4.4.9 is\_motif

```
bool DEFMCOUNTERData::is_motif
```

If false, then is a logit intercept.

Definition at line 70 of file defm-types.hpp.

#### 6.4.4.10 logical [1/2]

```
std::vector< bool > DEFMCOUNTERData::logical
```

Definition at line 69 of file defm-types.hpp.

#### 6.4.4.11 logical [2/2]

```
std::vector< bool > DEFMRULEData::logical
```

Definition at line 94 of file defm-types.hpp.

#### 6.4.4.12 numbers [1/2]

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

Definition at line 68 of file defm-types.hpp.

#### 6.4.4.13 numbers [2/2]

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

Definition at line 92 of file defm-types.hpp.

#### 6.4.4.14 obs\_start

```
size_t DEFMDData::obs_start
```

Index of the observation in the data.

Definition at line 21 of file defm-types.hpp.

#### 6.4.4.15 X\_ncol

```
size_t DEFMDData::X_ncol
```

Number of columns in the array of covariates.

Definition at line 22 of file defm-types.hpp.

#### 6.4.4.16 X\_nrow

```
size_t DEFMDData::X_nrow
```

Number of rows in the array of covariates.

Definition at line 23 of file defm-types.hpp.

## 6.5 Phylo counters

[Counters](#) for phylogenetic modeling.

Collaboration diagram for Phylo counters:



### Modules

- [Phylo rules](#)

*Rules for phylogenetic modeling.*

- void [counter\\_overall\\_gains](#) ([PhyloCounters](#) \*counters, size\_t duplication=[Geese::etype\\_default](#))  
*Overall functional gains.*
- void [counter\\_gains](#) ([PhyloCounters](#) \*counters, std::vector< size\_t > nfun, size\_t duplication=[Geese::etype\\_default](#))  
*Functional gains for a specific function (nfun).*
- void [counter\\_gains\\_k\\_offspring](#) ([PhyloCounters](#) \*counters, std::vector< size\_t > nfun, size\_t k=1u, size\_t duplication=[Geese::etype\\_default](#))  
*k genes gain function nfun*
- void [counter\\_genes\\_changing](#) ([PhyloCounters](#) \*counters, size\_t duplication=[Geese::etype\\_default](#))  
*Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)*
- void [counter\\_preserve\\_pseudogene](#) ([PhyloCounters](#) \*counters, size\_t nfunA, size\_t nfunB, size\_t duplication=[Geese::etype\\_default](#))  
*Keeps track of how many pairs of genes preserve pseudostate.*
- void [counter\\_prop\\_genes\\_changing](#) ([PhyloCounters](#) \*counters, size\_t duplication=[Geese::etype\\_default](#))  
*Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)*
- void [counter\\_overall\\_loss](#) ([PhyloCounters](#) \*counters, size\_t duplication=[Geese::etype\\_default](#))  
*Overall functional loss.*
- void [counter\\_maxfuns](#) ([PhyloCounters](#) \*counters, size\_t lb, size\_t ub, size\_t duplication=[Geese::etype\\_default](#))  
*Cap the number of functions per gene.*
- void [counter\\_loss](#) ([PhyloCounters](#) \*counters, std::vector< size\_t > nfun, size\_t duplication=[Geese::etype\\_default](#))  
*Total count of losses for an specific function.*
- void [counter\\_overall\\_changes](#) ([PhyloCounters](#) \*counters, size\_t duplication=[Geese::etype\\_default](#))  
*Total number of changes. Use this statistic to account for "preservation".*
- void [counter\\_subfun](#) ([PhyloCounters](#) \*counters, size\_t nfunA, size\_t nfunB, size\_t duplication=[Geese::etype\\_default](#))  
*Total count of Sub-functionalization events.*
- void [counter\\_cogain](#) ([PhyloCounters](#) \*counters, size\_t nfunA, size\_t nfunB, size\_t duplication=[Geese::etype\\_default](#))  
*Co-evolution (joint gain or loss)*
- void [counter\\_longest](#) ([PhyloCounters](#) \*counters, size\_t duplication=[Geese::etype\\_default](#))  
*Longest branch mutates (either by gain or by loss)*
- void [counter\\_neofun](#) ([PhyloCounters](#) \*counters, size\_t nfunA, size\_t nfunB, size\_t duplication=[Geese::etype\\_default](#))  
*Total number of neofunctionalization events.*

- void `counter_pairwise_neofun_singlefun` (`PhyloCounters *counters`, `size_t nfunA`, `size_t duplication=Geese::etype_default`)  
*Total number of neofunctionalization events  $\sum_u \sum_{\{w < u\}} [x(u,a)*(1 - x(w,a)) + (1 - x(u,a)) * x(w,a)]$  change stat:  $\delta\{x(u,a): 0 \rightarrow 1\} = 1 - 2 * x(w,a)$*
- void `counter_neofun_a2b` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=Geese::etype_default`)  
*Total number of neofunctionalization events.*
- void `counter_co_opt` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=Geese::etype_default`)  
*Function co-opting.*
- void `counter_k_genes_changing` (`PhyloCounters *counters`, `size_t k`, `size_t duplication=Geese::etype_default`)  
*Indicator function. Equals to one if  $k$  genes changed and zero otherwise.*
- void `counter_less_than_p_prop_genes_changing` (`PhyloCounters *counters`, `double p`, `size_t duplication=Geese::etype_default`)  
*Indicator function. Equals to one if  $k$  genes changed and zero otherwise.*
- void `counter_gains_from_0` (`PhyloCounters *counters`, `std::vector< size_t > nfun`, `size_t duplication=Geese::etype_default`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `counter_overall_gains_from_0` (`PhyloCounters *counters`, `size_t duplication=Geese::etype_default`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `counter_pairwise_overall_change` (`PhyloCounters *counters`, `size_t duplication=Geese::etype_default`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `counter_pairwise_preserving` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=Geese::etype_default`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `counter_pairwise_first_gain` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=Geese::etype_default`)  
*Used when all the functions are in 0 (like the root node prob.)*

## 6.5.1 Detailed Description

`Counters` for phylogenetic modeling.

### Parameters

<code>counters</code>	A pointer to a <code>PhyloCounters</code> object ( <code>Counters&lt;PhyloArray, PhyloCounterData&gt;</code> ).
-----------------------	---

## 6.5.2 Function Documentation

### 6.5.2.1 counter\_co\_opt()

```
void counter_co_opt (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = Geese::etype_default ) [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 1296 of file counters.hpp.

### 6.5.2.2 counter\_cogain()

```
void counter_cogain (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = Geese::etype_default ) [inline]
```

Co-evolution (joint gain or loss)

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

Definition at line 791 of file counters.hpp.

### 6.5.2.3 counter\_gains()

```
void counter_gains (
    PhyloCounters * counters,
    std::vector< size_t > nfun,
    size_t duplication = Geese::etype_default ) [inline]
```

Functional gains for a specific function (nfun).

Definition at line 96 of file counters.hpp.

### 6.5.2.4 counter\_gains\_from\_0()

```
void counter_gains_from_0 (
    PhyloCounters * counters,
    std::vector< size_t > nfun,
    size_t duplication = Geese::etype_default ) [inline]
```

Used when all the functions are in 0 (like the root node prob.)

Needs to specify function a.

Definition at line 1630 of file counters.hpp.

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

```
void counter_gains_k_offspring (
    PhyloCounters * counters,
    std::vector< size_t > nfun,
    size_t k = 1u,
    size_t duplication = Geese::etype_default ) [inline]
```

k genes gain function nfun

Definition at line 156 of file counters.hpp.

### 6.5.2.6 counter\_genes\_changing()

```
void counter_genes_changing (
    PhyloCounters * counters,
    size_t duplication = Geese::etype_default ) [inline]
```

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

Definition at line 228 of file counters.hpp.

### 6.5.2.7 counter\_k\_genes\_changing()

```
void counter_k_genes_changing (
    PhyloCounters * counters,
    size_t k,
    size_t duplication = Geese::etype_default ) [inline]
```

Indicator function. Equals to one if  $k$  genes changed and zero otherwise.

Definition at line 1394 of file counters.hpp.

### 6.5.2.8 counter\_less\_than\_p\_prop\_genes\_changing()

```
void counter_less_than_p_prop_genes_changing (
    PhyloCounters * counters,
    double p,
    size_t duplication = Geese::etype_default ) [inline]
```

Indicator function. Equals to one if  $k$  genes changed and zero otherwise.

< How many genes diverge the parent

Definition at line 1514 of file counters.hpp.

### 6.5.2.9 counter\_longest()

```
void counter_longest (
    PhyloCounters * counters,
    size_t duplication = Geese::etype_default ) [inline]
```

Longest branch mutates (either by gain or by loss)

Definition at line 848 of file counters.hpp.

### 6.5.2.10 counter\_loss()

```
void counter_loss (
    PhyloCounters * counters,
    std::vector< size_t > nfun,
    size_t duplication = Geese::etype_default ) [inline]
```

Total count of losses for an specific function.

Definition at line 591 of file counters.hpp.

### 6.5.2.11 counter\_maxfuns()

```
void counter_maxfuns (
    PhyloCounters * counters,
    size_t lb,
    size_t ub,
    size_t duplication = Geese::etype_default ) [inline]
```

Cap the number of functions per gene.

Definition at line 529 of file counters.hpp.

### 6.5.2.12 counter\_neofun()

```
void counter_neofun (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = Geese::etype_default ) [inline]
```

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 1018 of file counters.hpp.

#### 6.5.2.13 counter\_neofun\_a2b()

```
void counter_neofun_a2b (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = Geese::etype_default ) [inline]
```

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 1163 of file counters.hpp.

#### 6.5.2.14 counter\_overall\_changes()

```
void counter_overall_changes (
    PhyloCounters * counters,
    size_t duplication = Geese::etype_default ) [inline]
```

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

Definition at line 643 of file counters.hpp.

#### 6.5.2.15 counter\_overall\_gains()

```
void counter_overall_gains (
    PhyloCounters * counters,
    size_t duplication = Geese::etype_default ) [inline]
```

Overall functional gains.

Total number of gains (irrespective of the function).

Definition at line 58 of file counters.hpp.

#### 6.5.2.16 counter\_overall\_gains\_from\_0()

```
void counter_overall_gains_from_0 (
    PhyloCounters * counters,
    size_t duplication = Geese::etype_default ) [inline]
```

Used when all the functions are in 0 (like the root node prob.)

Needs to specify function a.

Definition at line 1696 of file counters.hpp.



**6.5.2.17 counter\_overall\_loss()**

```
void counter_overall_loss (
    PhyloCounters * counters,
    size_t duplication = Geese::etype_default ) [inline]
```

Overall functional loss.

Definition at line 481 of file counters.hpp.

**6.5.2.18 counter\_pairwise\_first\_gain()**

```
void counter_pairwise_first_gain (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = Geese::etype_default ) [inline]
```

Used when all the functions are in 0 (like the root node prob.)

Needs to specify function a.  $\sum x(a)^3(1-x(b))^3 + x(b)^3(1-x(a))^3 + x(a)^3 * x(b)^3 + (1-x(a))^3 * (1-x(b))^3$

Definition at line 1948 of file counters.hpp.

**6.5.2.19 counter\_pairwise\_neofun\_singlefun()**

```
void counter_pairwise_neofun_singlefun (
    PhyloCounters * counters,
    size_t nfunA,
    size_t duplication = Geese::etype_default ) [inline]
```

Total number of neofunctionalization events  $\sum_u \sum_{w < u} [x(u,a)*(1-x(w,a)) + (1-x(u,a)) * x(w,a)]$  change stat:  $\Delta\{x(u,a): 0 \rightarrow 1\} = 1 - 2 * x(w,a)$

Definition at line 1099 of file counters.hpp.

**6.5.2.20 counter\_pairwise\_overall\_change()**

```
void counter_pairwise_overall_change (
    PhyloCounters * counters,
    size_t duplication = Geese::etype_default ) [inline]
```

Used when all the functions are in 0 (like the root node prob.)

Needs to specify function a.

Definition at line 1744 of file counters.hpp.

#### 6.5.2.21 counter\_pairwise\_preserving()

```
void counter_pairwise_preserving (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = Geese::etype_default ) [inline]
```

Used when all the functions are in 0 (like the root node prob.)

Needs to specify function a.  $\sum x(a)^3(1-x(b))^3 + x(b)^3(1-x(a))^3 + x(a)^3 * x(b)^3 + (1-x(a))^3 * (1-x(b))^3$

Definition at line 1809 of file counters.hpp.

#### 6.5.2.22 counter\_preserve\_pseudogene()

```
void counter_preserve_pseudogene (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = Geese::etype_default ) [inline]
```

Keeps track of how many pairs of genes preserve pseudostate.

Definition at line 297 of file counters.hpp.

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

```
void counter_prop_genes_changing (
    PhyloCounters * counters,
    size_t duplication = Geese::etype_default ) [inline]
```

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

Definition at line 379 of file counters.hpp.

#### 6.5.2.24 counter\_subfun()

```
void counter_subfun (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = Geese::etype_default ) [inline]
```

Total count of Sub-functionalization events.

It requires to specify data = {funA, funB}

Definition at line 702 of file counters.hpp.

## Chapter 7

# Namespace Documentation

### 7.1 `barry` Namespace Reference

`barry`: Your go-to motif accountant

#### Namespaces

- [counters](#)

*Tree class and Treeliterator class.*

#### 7.1.1 Detailed Description

`barry`: Your go-to motif accountant

### 7.2 `barry::counters` Namespace Reference

Tree class and Treeliterator class.

#### Namespaces

- [network](#)

#### 7.2.1 Detailed Description

Tree class and Treeliterator class.

## 7.3 barry::counters::network Namespace Reference

## 7.4 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

### 7.4.1 Detailed Description

Integer constants used to specify which cell should be check.

### 7.4.2 Variable Documentation

#### 7.4.2.1 BOTH

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

Definition at line 27 of file typedefs.hpp.

#### 7.4.2.2 NONE

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

Definition at line 28 of file typedefs.hpp.

#### 7.4.2.3 ONE

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

Definition at line 29 of file typedefs.hpp.

#### 7.4.2.4 TWO

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

Definition at line 30 of file typedefs.hpp.

## 7.5 defm Namespace Reference

## 7.6 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

### 7.6.1 Detailed Description

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

### 7.6.2 Variable Documentation

#### 7.6.2.1 AS\_ONE

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

Definition at line 45 of file typedefs.hpp.

#### 7.6.2.2 AS\_ZERO

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

Definition at line 44 of file typedefs.hpp.

#### 7.6.2.3 BOTH

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

Definition at line 38 of file typedefs.hpp.

#### 7.6.2.4 NONE

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

Definition at line 39 of file typedefs.hpp.

#### 7.6.2.5 ONE

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

Definition at line 40 of file typedefs.hpp.

#### 7.6.2.6 TWO

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

Definition at line 41 of file typedefs.hpp.

#### 7.6.2.7 UNKNOWN

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

Definition at line 43 of file typedefs.hpp.

## 7.7 geese Namespace Reference

## Chapter 8

# Class Documentation

### 8.1 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` (size\_t `i`, size\_t `j`) const
- `Cell_Type` `get_cell` (size\_t `i`, size\_t `j`, bool `check_bounds=true`) const
- std::vector< `Cell_Type` > `get_col_vec` (size\_t `i`, bool `check_bounds=true`) const
- std::vector< `Cell_Type` > `get_row_vec` (size\_t `i`, bool `check_bounds=true`) const
- void `get_col_vec` (std::vector< `Cell_Type` > \*`x`, size\_t `i`, bool `check_bounds=true`) const
- void `get_row_vec` (std::vector< `Cell_Type` > \*`x`, size\_t `i`, bool `check_bounds=true`) const
- const `Row_type`< `Cell_Type` > & `row` (size\_t `i`, bool `check_bounds=true`) const
- const `Col_type`< `Cell_Type` > & `col` (size\_t `i`, bool `check_bounds=true`) const
- `Entries`< `Cell_Type` > `get_entries` () const

*Get the edgelist.*

- void `transpose` ()
- void `clear` (bool `hard=true`)
- void `resize` (size\_t `N_`, size\_t `M_`)
- void `reserve` ()
- void `print` (const char \*`fmt=nullptr`,...) const
- void `print_n` (size\_t `nrow`, size\_t `ncol`, const char \*`fmt=nullptr`,...) const
- bool `is_dense` () const `noexcept`

#### Constructors

##### Parameters

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

Generated by Doxygen

- **BArray** ()  
*Zero-size array.*
  - **BArray** (size\_t N\_, size\_t M\_)  
*Empty array.*
  - **BArray** (size\_t N\_, size\_t M\_, const std::vector< size\_t > &source, const std::vector< size\_t > &target, const std::vector< Cell\_Type > &value, bool add=true)  
*Edgelist with data.*
  - **BArray** (size\_t N\_, size\_t M\_, const std::vector< size\_t > &source, const std::vector< size\_t > &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\_ptr** ()
  - const Data\_Type \* **D\_ptr** () const
  - Data\_Type & **D** ()
  - const Data\_Type & **D** () const
  - void **flush\_data** ()

### Queries

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

#### Parameters

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

- bool **is\_empty** (size\_t i, size\_t j, bool check\_bounds=true) const
- size\_t **nrow** () const **noexcept**
- size\_t **ncol** () const **noexcept**
- size\_t **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< size\_t, size\_t > &coords)
- `BArray< Cell_Type, Data_Type > & operator-=` (const std::pair< size\_t, size\_t > &coords)
- `BArrayCell< Cell_Type, Data_Type > operator()` (size\_t i, size\_t j, bool check\_bounds=true)
- `const Cell_Type operator()` (size\_t i, size\_t j, bool check\_bounds=true) const
- `void rm_cell` (size\_t i, size\_t j, bool check\_bounds=true, bool check\_exists=true)
- `void insert_cell` (size\_t i, size\_t j, const Cell< Cell\_Type > &v, bool check\_bounds, bool check\_exists)
- `void insert_cell` (size\_t i, size\_t j, Cell< Cell\_Type > &&v, bool check\_bounds, bool check\_exists)
- `void insert_cell` (size\_t i, size\_t j, Cell\_Type v, bool check\_bounds, bool check\_exists)
- `void swap_cells` (size\_t i0, size\_t j0, size\_t i1, size\_t j1, bool check\_bounds=true, int check\_exists=CHECK::BOTH, int \*report=nullptr)
- `void toggle_cell` (size\_t i, size\_t j, bool check\_bounds=true, int check\_exists=EXISTS::UNKNOWN)
- `void toggle_lock` (size\_t i, size\_t j, bool check\_bounds=true)

### Column/row wise interchange

- `void swap_rows` (size\_t i0, size\_t i1, bool check\_bounds=true)
- `void swap_cols` (size\_t j0, size\_t j1, bool check\_bounds=true)
- `void zero_row` (size\_t i, bool check\_bounds=true)
- `void zero_col` (size\_t 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 >`

## 8.1.1 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<size_t, Cell_Type> >`.

#### Template Parameters

<code>Cell_Type</code>	Type of cell (any type).
<code>Data_Type</code>	Data type of the array (bool default).

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

## 8.1.2 Constructor & Destructor Documentation

### 8.1.2.1 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 69 of file `barray-bones.hpp`.

### 8.1.2.2 BArray() [2/6]

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

Empty array.

Definition at line 72 of file `barray-bones.hpp`.

### 8.1.2.3 BArray() [3/6]

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

Edgelist with data.

#### 8.1.2.4 BArray() [4/6]

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

Edgelist with no data (simpler)

#### 8.1.2.5 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.

#### 8.1.2.6 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.

#### 8.1.2.7 ~BArray()

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

### 8.1.3 Member Function Documentation

#### 8.1.3.1 clear()

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

#### 8.1.3.2 col()

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

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

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

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

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

#### 8.1.3.5 D\_ptr() [1/2]

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

#### 8.1.3.6 D\_ptr() [2/2]

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

#### 8.1.3.7 default\_val()

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

#### 8.1.3.8 flush\_data()

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

**8.1.3.9 get\_cell()**

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

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

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

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

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

**8.1.3.12 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>`

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

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

**8.1.3.14 get\_row\_vec() [2/2]**

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

**8.1.3.15 insert\_cell() [1/3]**

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

**8.1.3.16 insert\_cell() [2/3]**

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

**8.1.3.17 insert\_cell() [3/3]**

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

**8.1.3.18 is\_dense()**

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

Definition at line 241 of file `barray-bones.hpp`.

**8.1.3.19 is\_empty()**

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

**8.1.3.20 ncol()**

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

**8.1.3.21 nnozero()**

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

**8.1.3.22 nrow()**

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

**8.1.3.23 operator>() [1/2]**

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

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

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

**8.1.3.25 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 )
```

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

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

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

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

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

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

**8.1.3.29 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 )
```

**8.1.3.30 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 )
```



**8.1.3.31 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< size_t, size_t > & coords )
```

**8.1.3.32 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 )
```

**8.1.3.33 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.

**8.1.3.34 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.

**8.1.3.35 operator==( )**

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

**8.1.3.36 out\_of\_range( )**

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

#### 8.1.3.37 print()

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

#### 8.1.3.38 print\_n()

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

#### 8.1.3.39 reserve()

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

#### 8.1.3.40 resize()

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

#### 8.1.3.41 rm\_cell()

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

**8.1.3.42 row()**

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

**8.1.3.43 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>	

**8.1.3.44 swap\_cells()**

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

**8.1.3.45 swap\_cols()**

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

#### 8.1.3.46 swap\_rows()

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

#### 8.1.3.47 toggle\_cell()

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

#### 8.1.3.48 toggle\_lock()

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

#### 8.1.3.49 transpose()

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

#### 8.1.3.50 zero\_col()

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

#### 8.1.3.51 zero\_row()

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

### 8.1.4 Friends And Related Function Documentation

#### 8.1.4.1 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.

#### 8.1.4.2 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.

### 8.1.5 Member Data Documentation

#### 8.1.5.1 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 54 of file barray-bones.hpp.

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

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

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

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

### Public Member Functions

- [BArrayCell](#) ([BArray](#)< Cell\_Type, Data\_Type > \*Array\_, size\_t i\_, size\_t 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

### 8.2.1 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.

### 8.2.2 Constructor & Destructor Documentation

#### 8.2.2.1 BArrayCell()

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

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

#### 8.2.2.2 ~BArrayCell()

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

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

## 8.2.3 Member Function Documentation

### 8.2.3.1 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.

### 8.2.3.2 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.

### 8.2.3.3 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.

### 8.2.3.4 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.

### 8.2.3.5 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.

### 8.2.3.6 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.

### 8.2.3.7 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/barray-bones.hpp
- include/barry/barraycell-bones.hpp
- include/barry/barraycell-meat.hpp
- include/barry/barrayrow-meat.hpp

## 8.3 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\_, size\_t i\_, size\_t 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

### 8.3.1 Detailed Description

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

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



## 8.3.2 Constructor & Destructor Documentation

### 8.3.2.1 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_,
    size_t i_,
    size_t j_,
    bool check_bounds = true ) [inline]
```

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

### 8.3.2.2 ~BArrayCell\_const()

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

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

## 8.3.3 Member Function Documentation

### 8.3.3.1 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.

### 8.3.3.2 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.

### 8.3.3.3 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 bararraycell-meat.hpp.

### 8.3.3.4 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 bararraycell-meat.hpp.

### 8.3.3.5 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 bararraycell-meat.hpp.

### 8.3.3.6 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 bararraycell-meat.hpp.

### 8.3.3.7 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 bararraycell-meat.hpp.

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

- [include/barry/bararray-bones.hpp](#)
- [include/barry/bararraycell-bones.hpp](#)
- [include/barry/bararraycell-meat.hpp](#)
- [include/barry/bararrayrow-meat.hpp](#)

## 8.4 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` (size\_t i, size\_t j) const
- Cell\_Type `get_cell` (size\_t i, size\_t j, bool check\_bounds=true) const
- std::vector< Cell\_Type > `get_col_vec` (size\_t i, bool check\_bounds=true) const
- std::vector< Cell\_Type > `get_row_vec` (size\_t i, bool check\_bounds=true) const
- void `get_col_vec` (std::vector< Cell\_Type > \*x, size\_t i, bool check\_bounds=true) const
- void `get_row_vec` (std::vector< Cell\_Type > \*x, size\_t i, bool check\_bounds=true) const
- BArrayDenseRow< Cell\_Type, Data\_Type > & `row` (size\_t i, bool check\_bounds=true)
- const BArrayDenseRow\_const< Cell\_Type, Data\_Type > `row` (size\_t i, bool check\_bounds=true) const
- BArrayDenseCol< Cell\_Type, Data\_Type > & `col` (size\_t j, bool check\_bounds=true)
- const BArrayDenseCol\_const< Cell\_Type, Data\_Type > `col` (size\_t j, bool check\_bounds=true) const
- Entries< Cell\_Type > `get_entries` () const
- *Get the edgelist.*
- void `transpose` ()
- void `clear` (bool hard=true)
- void `resize` (size\_t N\_, size\_t M\_)
- void `reserve` ()
- void `print` (const char \*fmt=nullptr,...) const
- bool `is_dense` () const `noexcept`
- const std::vector< Cell\_Type > & `get_data` () const
- const Cell\_Type `rowsum` (size\_t i) const
- const Cell\_Type `colsum` (size\_t i) const

### Constructors

#### Parameters

N_	Number of rows
M_	Number of columns
source	An unsigned vector ranging from 0 to N_
target	An size_t vector ranging from 0 to M_
target	When <code>true</code> tries to add repeated observations.
value	Cell_Type default fill-in value (zero, by default.)

- BArrayDense ()
- *Zero-size array.*
- BArrayDense (size\_t N\_, size\_t M\_, Cell\_Type value=static\_cast< Cell\_Type >(0))
- *Empty array.*
- BArrayDense (size\_t N\_, size\_t M\_, const std::vector< size\_t > &source, const std::vector< size\_t > &target, const std::vector< Cell\_Type > &value, bool add=true)
- *Edgelist with data.*
- BArrayDense (size\_t N\_, size\_t M\_, const std::vector< size\_t > &source, const std::vector< size\_t > &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\_ptr** ()
- const Data\_Type \* **D\_ptr** () const
- 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** (size\_t i, size\_t j, bool check\_bounds=true) const
- size\_t **nrow** () const **noexcept**
- size\_t **ncol** () const **noexcept**
- size\_t **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< size\_t, size\_t > &coords)
- **BArrayDense**< Cell\_Type, Data\_Type > & **operator-=** (const std::pair< size\_t, size\_t > &coords)
- **BArrayDenseCell**< Cell\_Type, Data\_Type > **operator()** (size\_t i, size\_t j, bool check\_bounds=true)
- const Cell\_Type **operator()** (size\_t i, size\_t j, bool check\_bounds=true) const
- void **rm\_cell** (size\_t i, size\_t j, bool check\_bounds=true, bool check\_exists=true)
- void **insert\_cell** (size\_t i, size\_t j, const **Cell**< Cell\_Type > &v, bool check\_bounds, bool)
- void **insert\_cell** (size\_t i, size\_t j, Cell\_Type v, bool check\_bounds, bool)

- void `swap_cells` (size\_t i0, size\_t j0, size\_t i1, size\_t j1, bool check\_bounds=true, int check\_exists=CHECK::BOTH, int \*report=nullptr)
- void `toggle_cell` (size\_t i, size\_t j, bool check\_bounds=true, int check\_exists=EXISTS::UNKNOWN)
- void `toggle_lock` (size\_t i, size\_t j, bool check\_bounds=true)

#### Column/row wise interchange

- void `swap_rows` (size\_t i0, size\_t i1, bool check\_bounds=true)
- void `swap_cols` (size\_t j0, size\_t j1, bool check\_bounds=true)
- void `zero_row` (size\_t i, bool check\_bounds=true)
- void `zero_col` (size\_t 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 `BArrayDenseCell< Cell_Type, Data_Type >`
- class `BArrayDenseCol< Cell_Type, Data_Type >`
- class `BArrayDenseCol_const< Cell_Type, Data_Type >`
- class `BArrayDenseRow< Cell_Type, Data_Type >`
- class `BArrayDenseRow_const< Cell_Type, Data_Type >`

### 8.4.1 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 dense-arrays. The data is stored internally in the `el` member, which can be accessed using the member function `get_data()`, by column.

#### 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 33 of file `barraydense-bones.hpp`.

## 8.4.2 Constructor & Destructor Documentation

### 8.4.2.1 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 79 of file `barraydense-bones.hpp`.

### 8.4.2.2 BArrayDense() [2/6]

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArrayDense< Cell_Type, Data_Type >::BArrayDense (
    size_t N_,
    size_t M_,
    Cell_Type value = static_cast<Cell_Type>(0) ) [inline]
```

Empty array.

Definition at line 82 of file `barraydense-bones.hpp`.

### 8.4.2.3 BArrayDense() [3/6]

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

Edgelist with data.

Definition at line 35 of file `barraydense-meat.hpp`.

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

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

Edgelist with no data (simpler)

Definition at line 86 of file `barraydense-meat.hpp`.

**8.4.2.5 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 135 of file `barraydense-meat.hpp`.

**8.4.2.6 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 229 of file `barraydense-meat.hpp`.

**8.4.2.7 ~BArrayDense()**

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

Definition at line 310 of file `barraydense-meat.hpp`.

**8.4.3 Member Function Documentation**

#### 8.4.3.1 clear()

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

Definition at line 905 of file `barraydense-meat.hpp`.

#### 8.4.3.2 col() [1/2]

```
template<typename Cell_Type , typename Data_Type >
BArrayDenseCol< Cell_Type, Data_Type > & BArrayDense< Cell_Type, Data_Type >::col (
    size_t j,
    bool check_bounds = true ) [inline]
```

Definition at line 498 of file `barraydense-meat.hpp`.

#### 8.4.3.3 col() [2/2]

```
template<typename Cell_Type , typename Data_Type >
const BArrayDenseCol_const< Cell_Type, Data_Type > BArrayDense< Cell_Type, Data_Type >::col (
    size_t j,
    bool check_bounds = true ) const [inline]
```

Definition at line 484 of file `barraydense-meat.hpp`.

#### 8.4.3.4 colsum()

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

Definition at line 1008 of file `barraydense-meat.hpp`.

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

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

Definition at line 345 of file `barraydense-meat.hpp`.



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

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

Definition at line 350 of file `barraydense-meat.hpp`.

#### 8.4.3.7 D\_ptr() [1/2]

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

Definition at line 335 of file `barraydense-meat.hpp`.

#### 8.4.3.8 D\_ptr() [2/2]

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

Definition at line 340 of file `barraydense-meat.hpp`.

#### 8.4.3.9 default\_val()

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

Definition at line 571 of file `barraydense-meat.hpp`.

#### 8.4.3.10 get\_cell()

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

Definition at line 376 of file `barraydense-meat.hpp`.

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

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

Definition at line 424 of file `barraydense-meat.hpp`.

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

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

Definition at line 442 of file `barraydense-meat.hpp`.

**8.4.3.13 get\_data()**

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

Definition at line 998 of file `barraydense-meat.hpp`.

**8.4.3.14 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 510 of file `barraydense-meat.hpp`.

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

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

Definition at line 391 of file `barraydense-meat.hpp`.

**8.4.3.16 get\_row\_vec() [2/2]**

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

Definition at line 409 of file `barraydense-meat.hpp`.

**8.4.3.17 insert\_cell() [1/2]**

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

Definition at line 697 of file `barraydense-meat.hpp`.

**8.4.3.18 insert\_cell() [2/2]**

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

Definition at line 663 of file `barraydense-meat.hpp`.

#### 8.4.3.19 is\_dense()

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

Definition at line 256 of file `barraydense-bones.hpp`.

#### 8.4.3.20 is\_empty()

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

Definition at line 539 of file `barraydense-meat.hpp`.

#### 8.4.3.21 ncol()

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

Definition at line 556 of file `barraydense-meat.hpp`.

#### 8.4.3.22 nnozero()

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

Definition at line 560 of file `barraydense-meat.hpp`.

#### 8.4.3.23 nrow()

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

Definition at line 552 of file `barraydense-meat.hpp`.

**8.4.3.24 operator>() [1/2]**

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

Definition at line 615 of file `barraydense-meat.hpp`.

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

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

Definition at line 626 of file `barraydense-meat.hpp`.

**8.4.3.26 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 )
```

**8.4.3.27 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 )
```

**8.4.3.28 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 )
```

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

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

Definition at line 576 of file barraydense-meat.hpp.

**8.4.3.30 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 )
```

**8.4.3.31 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 )
```

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

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

Definition at line 595 of file barraydense-meat.hpp.

**8.4.3.33 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 )
```

**8.4.3.34 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 247 of file `barraydense-meat.hpp`.

**8.4.3.35 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 179 of file `barraydense-meat.hpp`.

**8.4.3.36 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 291 of file `barraydense-meat.hpp`.

**8.4.3.37 out\_of\_range()**

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

Definition at line 355 of file `barraydense-meat.hpp`.

**8.4.3.38 print()**

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::print (
    const char * fmt = nullptr,
    ... ) const [inline]
```

Definition at line 964 of file `barraydense-meat.hpp`.

**8.4.3.39 reserve()**

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

Definition at line 955 of file barraydense-meat.hpp.

**8.4.3.40 resize()**

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

Definition at line 919 of file barraydense-meat.hpp.

**8.4.3.41 rm\_cell()**

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

Definition at line 640 of file barraydense-meat.hpp.

**8.4.3.42 row() [1/2]**

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

Definition at line 470 of file barraydense-meat.hpp.

**8.4.3.43 row() [2/2]**

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

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



**8.4.3.44 rowsum()**

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

Definition at line 1003 of file barraydense-meat.hpp.

**8.4.3.45 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 319 of file barraydense-meat.hpp.

**8.4.3.46 swap\_cells()**

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

Definition at line 728 of file barraydense-meat.hpp.

**8.4.3.47 swap\_cols()**

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

Definition at line 812 of file barraydense-meat.hpp.

**8.4.3.48 swap\_rows()**

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

Definition at line 784 of file `barraydense-meat.hpp`.

**8.4.3.49 toggle\_cell()**

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

Definition at line 765 of file `barraydense-meat.hpp`.

**8.4.3.50 toggle\_lock()**

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

**8.4.3.51 transpose()**

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

Definition at line 877 of file `barraydense-meat.hpp`.

**8.4.3.52 zero\_col()**

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

Definition at line 858 of file `barraydense-meat.hpp`.

#### 8.4.3.53 zero\_row()

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

Definition at line 839 of file barraydense-meat.hpp.

### 8.4.4 Friends And Related Function Documentation

#### 8.4.4.1 BArrayDenseCell< Cell\_Type, Data\_Type >

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

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

#### 8.4.4.2 BArrayDenseCol< Cell\_Type, Data\_Type >

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

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

#### 8.4.4.3 BArrayDenseCol\_const< Cell\_Type, Data\_Type >

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

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

#### 8.4.4.4 BArrayDenseRow< Cell\_Type, Data\_Type >

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

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

#### 8.4.4.5 BArrayDenseRow\_const< Cell\_Type, Data\_Type >

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

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

### 8.4.5 Member Data Documentation

#### 8.4.5.1 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 63 of file `barraydense-bones.hpp`.

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

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

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

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

### Public Member Functions

- `BArrayDenseCell` (`BArrayDense< Cell_Type, Data_Type > *Array_`, `size_t i_`, `size_t j_`, `bool check_`↵  
    `bounds=true`)
- `BArrayDenseCell< Cell_Type, Data_Type > & operator=` (`const BArrayDenseCell< Cell_Type, Data_Type >`  
    `&other`)
- `~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`

## Friends

- class [BArrayDense< Cell\\_Type, Data\\_Type >](#)
- class [BArrayDenseCol< Cell\\_Type, Data\\_Type >](#)
- class [BArrayDenseCol\\_const< Cell\\_Type, Data\\_Type >](#)

### 8.5.1 Detailed Description

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

Definition at line 18 of file `barraydensecell-bones.hpp`.

### 8.5.2 Constructor & Destructor Documentation

#### 8.5.2.1 BArrayDenseCell()

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

Definition at line 30 of file `barraydensecell-bones.hpp`.

#### 8.5.2.2 ~BArrayDenseCell()

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

Definition at line 56 of file `barraydensecell-bones.hpp`.

### 8.5.3 Member Function Documentation

#### 8.5.3.1 operator Cell\_Type()

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

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

### 8.5.3.2 operator\*=( )

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

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

### 8.5.3.3 operator+=( )

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

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

### 8.5.3.4 operator-=( )

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

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

### 8.5.3.5 operator/=( )

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

Definition at line 95 of file barraydensecell-meat.hpp.

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

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

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

**8.5.3.7 operator=()** [2/2]

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

Definition at line 31 of file `barraydensecell-meat.hpp`.

**8.5.3.8 operator==()**

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

Definition at line 117 of file `barraydensecell-meat.hpp`.

**8.5.4 Friends And Related Function Documentation****8.5.4.1 BArrayDense< Cell\_Type, Data\_Type >**

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

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

**8.5.4.2 BArrayDenseCol< Cell\_Type, Data\_Type >**

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

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

**8.5.4.3 BArrayDenseCol\_const< Cell\_Type, Data\_Type >**

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

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

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

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

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

### 8.6.1 Detailed Description

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

Definition at line 20 of file `barraydense-bones.hpp`.

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

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

## 8.7 BArrayDenseCol< Cell\_Type, Data\_Type > Class Template Reference

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

### Public Member Functions

- [BArrayDenseCol](#) ([BArrayDense](#)< Cell\_Type, Data\_Type > &array\_, size\_t j)
- [Col\\_type](#)< Cell\_Type >::iterator & [begin](#) ()
- [Col\\_type](#)< Cell\_Type >::iterator & [end](#) ()
- size\_t [size](#) () const [noexcept](#)
- std::pair< size\_t, Cell\_Type \* > & [operator\(\)](#) (size\_t i)

### Friends

- class [BArrayDense](#)< Cell\_Type, Data\_Type >
- class [BArrayDenseCell](#)< Cell\_Type, Data\_Type >
- class [BArrayDenseCell\\_const](#)< Cell\_Type, Data\_Type >

### 8.7.1 Detailed Description

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

Definition at line 9 of file `barraydensecol-bones.hpp`.

### 8.7.2 Constructor & Destructor Documentation



### 8.7.2.1 BArrayDenseCol()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArrayDenseCol< Cell_Type, Data_Type >::BArrayDenseCol (
    BArrayDense< Cell_Type, Data_Type > & array_,
    size_t j ) [inline]
```

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

## 8.7.3 Member Function Documentation

### 8.7.3.1 begin()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
Col_type<Cell_Type>::iterator& BArrayDenseCol< Cell_Type, Data_Type >::begin ( ) [inline]
```

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

### 8.7.3.2 end()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
Col_type<Cell_Type>::iterator& BArrayDenseCol< Cell_Type, Data_Type >::end ( ) [inline]
```

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

### 8.7.3.3 operator()()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
std::pair<size_t,Cell_Type*>& BArrayDenseCol< Cell_Type, Data_Type >::operator() (
    size_t i ) [inline]
```

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

### 8.7.3.4 size()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
size_t BArrayDenseCol< Cell_Type, Data_Type >::size ( ) const [inline], [noexcept]
```

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

## 8.7.4 Friends And Related Function Documentation

### 8.7.4.1 BArrayDense< Cell\_Type, Data\_Type >

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

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

### 8.7.4.2 BArrayDenseCell< Cell\_Type, Data\_Type >

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

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

### 8.7.4.3 BArrayDenseCell\_const< Cell\_Type, Data\_Type >

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

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

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

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

## 8.8 BArrayDenseCol\_const< Cell\_Type, Data\_Type > Class Template Reference

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

### Public Member Functions

- `BArrayDenseCol_const` (const `BArrayDense`< `Cell_Type`, `Data_Type` > &array\_, size\_t j)
- `Col_type`< `Cell_Type` >::iterator `begin` ()
- `Col_type`< `Cell_Type` >::iterator `end` ()
- `size_t` `size` () const `noexcept`
- const std::pair< size\_t, `Cell_Type` \* > `operator()` (size\_t i) const

## Friends

- class [BArrayDenseCell< Cell\\_Type, Data\\_Type >](#)
- class [BArrayDenseCell\\_const< Cell\\_Type, Data\\_Type >](#)

### 8.8.1 Detailed Description

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

Definition at line 71 of file `barraydensecol-bones.hpp`.

### 8.8.2 Constructor & Destructor Documentation

#### 8.8.2.1 BArrayDenseCol\_const()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArrayDenseCol_const< Cell_Type, Data_Type >::BArrayDenseCol_const (
    const BArrayDense< Cell_Type, Data_Type > & array_,
    size_t j ) [inline]
```

Definition at line 80 of file `barraydensecol-bones.hpp`.

### 8.8.3 Member Function Documentation

#### 8.8.3.1 begin()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
Col_type<Cell_Type>::iterator BArrayDenseCol_const< Cell_Type, Data_Type >::begin ( ) [inline]
```

Definition at line 96 of file `barraydensecol-bones.hpp`.

#### 8.8.3.2 end()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
Col_type<Cell_Type>::iterator BArrayDenseCol_const< Cell_Type, Data_Type >::end ( ) [inline]
```

Definition at line 101 of file `barraydensecol-bones.hpp`.

### 8.8.3.3 operator()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
const std::pair<size_t,Cell_Type*> BArrayDenseCol_const< Cell_Type, Data_Type >::operator() (
    size_t i ) const [inline]
```

Definition at line 112 of file `barraydensecol-bones.hpp`.

### 8.8.3.4 size()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
size_t BArrayDenseCol_const< Cell_Type, Data_Type >::size ( ) const [inline], [noexcept]
```

Definition at line 107 of file `barraydensecol-bones.hpp`.

## 8.8.4 Friends And Related Function Documentation

### 8.8.4.1 BArrayDenseCell< Cell\_Type, Data\_Type >

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

Definition at line 62 of file `barraydensecol-bones.hpp`.

### 8.8.4.2 BArrayDenseCell\_const< Cell\_Type, Data\_Type >

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

Definition at line 62 of file `barraydensecol-bones.hpp`.

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

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

## 8.9 BArrayDenseRow< Cell\_Type, Data\_Type > Class Template Reference

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

## Public Member Functions

- [BArrayDenseRow](#) ([BArrayDense](#)< Cell\_Type, Data\_Type > &array\_, size\_t i)
- [Row\\_type](#)< Cell\_Type >::iterator & [begin](#) ()
- [Row\\_type](#)< Cell\_Type >::iterator & [end](#) ()
- size\_t [size](#) () const [noexcept](#)
- std::pair< size\_t, [Cell](#)< Cell\_Type > > & [operator\(\)](#) (size\_t i)

## Friends

- class [BArrayDense](#)< Cell\_Type, Data\_Type >
- class [BArrayDenseCell](#)< Cell\_Type, Data\_Type >
- class [BArrayDenseCell\\_const](#)< Cell\_Type, Data\_Type >

### 8.9.1 Detailed Description

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

Definition at line 9 of file `barraydenserow-bones.hpp`.

### 8.9.2 Constructor & Destructor Documentation

#### 8.9.2.1 BArrayDenseRow()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArrayDenseRow< Cell_Type, Data_Type >::BArrayDenseRow (
    BArrayDense< Cell_Type, Data_Type > & array_,
    size_t i ) [inline]
```

Definition at line 40 of file `barraydenserow-bones.hpp`.

### 8.9.3 Member Function Documentation

#### 8.9.3.1 begin()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
Row_type<Cell_Type>::iterator& BArrayDenseRow< Cell_Type, Data_Type >::begin ( ) [inline]
```

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

### 8.9.3.2 end()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
Row_type<Cell_Type>::iterator& BArrayDenseRow< Cell_Type, Data_Type >::end ( ) [inline]
```

Definition at line 53 of file barraydenserow-bones.hpp.

### 8.9.3.3 operator()()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
std::pair<size_t, Cell<Cell_Type> >& BArrayDenseRow< Cell_Type, Data_Type >::operator() (
    size_t i ) [inline]
```

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

### 8.9.3.4 size()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
size_t BArrayDenseRow< Cell_Type, Data_Type >::size ( ) const [inline], [noexcept]
```

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

## 8.9.4 Friends And Related Function Documentation

### 8.9.4.1 BArrayDense< Cell\_Type, Data\_Type >

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

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

### 8.9.4.2 BArrayDenseCell< Cell\_Type, Data\_Type >

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

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

### 8.9.4.3 BArrayDenseCell\_const< Cell\_Type, Data\_Type >

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

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

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

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

## 8.10 BArrayDenseRow\_const< Cell\_Type, Data\_Type > Class Template Reference

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

### Public Member Functions

- [BArrayDenseRow\\_const](#) (const [BArrayDense](#)< Cell\_Type, Data\_Type > &array\_, size\_t i)
- [Row\\_type](#)< Cell\_Type >::const\_iterator [begin](#) () const
- [Row\\_type](#)< Cell\_Type >::const\_iterator [end](#) () const
- size\_t [size](#) () const [noexcept](#)
- const std::pair< size\_t, [Cell](#)< Cell\_Type > > [operator\(\)](#) (size\_t i) const

### Friends

- class [BArrayDenseCell](#)< Cell\_Type, Data\_Type >
- class [BArrayDenseCell\\_const](#)< Cell\_Type, Data\_Type >

### 8.10.1 Detailed Description

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

Definition at line 80 of file `barraydenserow-bones.hpp`.

### 8.10.2 Constructor & Destructor Documentation

### 8.10.2.1 BArrayDenseRow\_const()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArrayDenseRow_const< Cell_Type, Data_Type >::BArrayDenseRow_const (
    const BArrayDense< Cell_Type, Data_Type > & array_,
    size_t i ) [inline]
```

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

## 8.10.3 Member Function Documentation

### 8.10.3.1 begin()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
Row_type< Cell_Type >::const_iterator BArrayDenseRow_const< Cell_Type, Data_Type >::begin ( )
const [inline]
```

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

### 8.10.3.2 end()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
Row_type< Cell_Type >::const_iterator BArrayDenseRow_const< Cell_Type, Data_Type >::end ( )
const [inline]
```

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

### 8.10.3.3 operator()()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
const std::pair<size_t, Cell<Cell_Type> > BArrayDenseRow_const< Cell_Type, Data_Type >↵
::operator() (
    size_t i ) const [inline]
```

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

### 8.10.3.4 size()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
size_t BArrayDenseRow_const< Cell_Type, Data_Type >::size ( ) const [inline], [noexcept]
```

Definition at line 118 of file barraydenserow-bones.hpp.



### 8.10.4 Friends And Related Function Documentation

#### 8.10.4.1 BArrayDenseCell< Cell\_Type, Data\_Type >

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

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

#### 8.10.4.2 BArrayDenseCell\_const< Cell\_Type, Data\_Type >

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

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

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

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

## 8.11 BArrayRow< Cell\_Type, Data\_Type > Class Template Reference

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

### Public Member Functions

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

#### 8.11.1 Detailed Description

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

Definition at line 5 of file `barrayrow-bones.hpp`.

## 8.11.2 Constructor & Destructor Documentation

### 8.11.2.1 BArrayRow()

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

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

### 8.11.2.2 ~BArrayRow()

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

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

## 8.11.3 Member Function Documentation

### 8.11.3.1 operator BArrayRow< Cell\_Type, Data\_Type >()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArrayRow< Cell_Type, Data_Type >::operator BArrayRow< Cell_Type, Data_Type > ( ) const
```

### 8.11.3.2 operator\*=( )

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArrayRow< Cell_Type, Data_Type >::operator*= (
    const BArrayRow< Cell_Type, Data_Type > & val )
```

### 8.11.3.3 operator+=( )

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArrayRow< Cell_Type, Data_Type >::operator+= (
    const BArrayRow< Cell_Type, Data_Type > & val )
```

#### 8.11.3.4 operator-=( )

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArrayRow< Cell_Type, Data_Type >::operator-= (
    const BArrayRow< Cell_Type, Data_Type > & val )
```

#### 8.11.3.5 operator/=( )

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArrayRow< Cell_Type, Data_Type >::operator/= (
    const BArrayRow< Cell_Type, Data_Type > & val )
```

#### 8.11.3.6 operator=( )

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArrayRow< Cell_Type, Data_Type >::operator= (
    const BArrayRow< Cell_Type, Data_Type > & val )
```

#### 8.11.3.7 operator==( )

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

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

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

## 8.12 BArrayRow\_const< Cell\_Type, Data\_Type > Class Template Reference

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

### Public Member Functions

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

### 8.12.1 Detailed Description

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

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

### 8.12.2 Constructor & Destructor Documentation

#### 8.12.2.1 BArrayRow\_const()

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

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

#### 8.12.2.2 ~BArrayRow\_const()

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

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

### 8.12.3 Member Function Documentation

#### 8.12.3.1 operator BArrayRow\_const< Cell\_Type, Data\_Type >()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArrayRow_const< Cell_Type, Data_Type >::operator BArrayRow_const< Cell_Type, Data_Type > ( )
const
```

#### 8.12.3.2 operator"!="()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
bool BArrayRow_const< Cell_Type, Data_Type >::operator!= (
    const BArrayRow_const< Cell_Type, Data_Type > & val ) const
```

**8.12.3.3 operator<()**

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

**8.12.3.4 operator<=()**

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

**8.12.3.5 operator==(())**

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

**8.12.3.6 operator>()**

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

**8.12.3.7 operator>=()**

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

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

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

**8.13 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\_, size\_t &dim\_ size\_t &i\_, bool check\_bounds=true)  
Construct a new [BArrayVector](#) object.
- [~BArrayVector](#) ()
- bool [is\\_row](#) () const [noexcept](#)
- bool [is\\_col](#) () const [noexcept](#)
- size\_t [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

### 8.13.1 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 11 of file [barrayvector-bones.hpp](#).

### 8.13.2 Constructor & Destructor Documentation

#### 8.13.2.1 BArrayVector()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArrayVector< Cell_Type, Data_Type >::BArrayVector (
    BArray< Cell_Type, Data_Type > * Array_,
    size_t &dim_ size_t & 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>i_</i>	Element to point.
<i>check_bounds</i>	When <code>true</code> , check boundaries.

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

#### 8.13.2.2 ~BArrayVector()

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

Definition at line 53 of file barrayvector-bones.hpp.

### 8.13.3 Member Function Documentation

#### 8.13.3.1 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 50 of file barrayvector-meat.hpp.

#### 8.13.3.2 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 64 of file barrayvector-meat.hpp.

#### 8.13.3.3 is\_col()

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

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

#### 8.13.3.4 is\_row()

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

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

#### 8.13.3.5 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 175 of file barrayvector-meat.hpp.

#### 8.13.3.6 operator\*=( )

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

Definition at line 133 of file barrayvector-meat.hpp.

#### 8.13.3.7 operator+=( )

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

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

#### 8.13.3.8 operator-=( )

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

Definition at line 112 of file barrayvector-meat.hpp.



#### 8.13.3.9 operator/=( )

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

Definition at line 154 of file `barrayvector-meat.hpp`.

#### 8.13.3.10 operator=( )

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

Definition at line 69 of file `barrayvector-meat.hpp`.

#### 8.13.3.11 operator==( )

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

Definition at line 185 of file `barrayvector-meat.hpp`.

#### 8.13.3.12 size( )

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

Definition at line 39 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`

## 8.14 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\_, size\_t &dim\_size\_t &i\_, bool check↵\_bounds=true)
- [~BArrayVector\\_const](#) ()
- bool [is\\_row](#) () const [noexcept](#)
- bool [is\\_col](#) () const [noexcept](#)
- size\_t [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

### 8.14.1 Detailed Description

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

Definition at line 73 of file `barrayvector-bones.hpp`.

### 8.14.2 Constructor & Destructor Documentation

#### 8.14.2.1 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_,
    size_t &dim_size_t & i_,
    bool check_bounds = true ) [inline]
```

Definition at line 86 of file `barrayvector-bones.hpp`.

#### 8.14.2.2 ~BArrayVector\_const()

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

Definition at line 108 of file `barrayvector-bones.hpp`.

### 8.14.3 Member Function Documentation

#### 8.14.3.1 begin()

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

#### 8.14.3.2 end()

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

#### 8.14.3.3 is\_col()

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

#### 8.14.3.4 is\_row()

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

#### 8.14.3.5 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 212 of file barrayvector-meat.hpp.

#### 8.14.3.6 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 249 of file barrayvector-meat.hpp.

#### 8.14.3.7 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 254 of file barrayvector-meat.hpp.

#### 8.14.3.8 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 281 of file barrayvector-meat.hpp.

#### 8.14.3.9 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 222 of file barrayvector-meat.hpp.

#### 8.14.3.10 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 308 of file barrayvector-meat.hpp.

**8.14.3.11 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 315 of file barrayvector-meat.hpp.

**8.14.3.12 size()**

```
template<typename Cell_Type = bool, typename Data_Type = bool>
size_t 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

**8.15 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, bool active\_=true)
- [~Cell](#) ()
- [Cell](#) (const [Cell](#)< Cell\_Type > &arg)
- [Cell](#)< Cell\_Type > & [operator=](#) (const [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](#) (size\_t x)
- void [add](#) (int x)
- [Cell](#) ()
- [Cell](#) ()
- [Cell](#) ()

**Public Attributes**

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

### 8.15.1 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 10 of file cell-bones.hpp.

### 8.15.2 Constructor & Destructor Documentation

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

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

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

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

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

#### 8.15.2.3 ~Cell()

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

### 8.15.3 Member Function Documentation

**8.15.3.1 add() [1/4]**

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

**8.15.3.2 add() [2/4]**

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

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

**8.15.3.3 add() [3/4]**

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

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

**8.15.3.4 add() [4/4]**

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

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

**8.15.3.5 operator Cell\_Type()**

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

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

**8.15.3.6 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.



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

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

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

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

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

Definition at line 5 of file cell-meat.hpp.

### 8.15.3.9 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.

## 8.15.4 Member Data Documentation

### 8.15.4.1 active

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

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

### 8.15.4.2 value

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

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

### 8.15.4.3 visited

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

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

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

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

## 8.16 Cell\_const< Cell\_Type > Class Template Reference

### 8.16.1 Detailed Description

```
template<typename Cell_Type>
class Cell_const< Cell_Type >
```

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

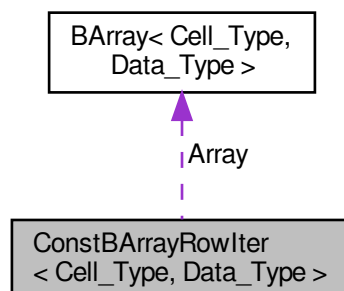
The documentation for this class was generated from the following file:

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

## 8.17 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

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

### 8.17.1 Detailed Description

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

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

### 8.17.2 Constructor & Destructor Documentation

#### 8.17.2.1 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`.

#### 8.17.2.2 ~ConstBArrayRowIter()

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

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

### 8.17.3 Member Data Documentation

### 8.17.3.1 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`.

### 8.17.3.2 current\_col

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

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

### 8.17.3.3 current\_row

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

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

### 8.17.3.4 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`

## 8.18 Counter< Array\_Type, Data\_Type > Class Template Reference

A counter function based on change statistics.

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

### Public Member Functions

- `~Counter` ()
- double `count` (Array\_Type &Array, size\_t `i`, size\_t `j`)
- double `init` (Array\_Type &Array, size\_t `i`, size\_t `j`)
- std::string `get_name` () const
- std::string `get_description` () const

**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](#), [Hasher\\_fun\\_type](#)< Array\_Type, Data\_Type > [hasher\\_fun](#), Data\_Type [data](#), std::string [name](#)\_="", std::string [desc](#)\_="")
- [Counter](#) (const [Counter](#)< Array\_Type, Data\_Type > &[counter](#)\_)  
Copy constructor.
- [Counter](#) ([Counter](#)< Array\_Type, Data\_Type > &&[counter](#)\_) **noexcept**  
Move constructor.
- [Counter](#)< Array\_Type, Data\_Type > **operator=** (const [Counter](#)< Array\_Type, Data\_Type > &[counter](#)\_)  
Copy assignment.
- [Counter](#)< Array\_Type, Data\_Type > & **operator=** ([Counter](#)< Array\_Type, Data\_Type > &&[counter](#)\_)  
**noexcept**  
Move assignment.
- void [set\\_hasher](#) ([Hasher\\_fun\\_type](#)< Array\_Type, Data\_Type > [fun](#))  
Get and set the hasher function.
- [Hasher\\_fun\\_type](#)< Array\_Type, Data\_Type > [get\\_hasher](#) ()

**Public Attributes**

- [Counter\\_fun\\_type](#)< Array\_Type, Data\_Type > [count\\_fun](#)
- [Counter\\_fun\\_type](#)< Array\_Type, Data\_Type > [init\\_fun](#)
- [Hasher\\_fun\\_type](#)< Array\_Type, Data\_Type > [hasher\\_fun](#)
- Data\_Type [data](#)
- std::string [name](#) = ""
- std::string [desc](#) = ""

**8.18.1 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 35 of file `counters-bones.hpp`.

## 8.18.2 Constructor & Destructor Documentation

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

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

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

### 8.18.2.2 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_,
    Hasher_fun_type< Array_Type, Data_Type > hasher_fun_,
    Data_Type data_,
    std::string name_ = "",
    std::string desc_ = "" ) [inline]
```

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

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

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

Copy constructor.

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

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

Move constructor.

### 8.18.2.5 ~Counter()

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

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

## 8.18.3 Member Function Documentation

### 8.18.3.1 count()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
double Counter< Array_Type, Data_Type >::count (
    Array_Type & Array,
    size_t i,
    size_t j )
```

### 8.18.3.2 get\_description()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::string Counter< Array_Type, Data_Type >::get_description ( ) const
```

### 8.18.3.3 get\_hasher()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Hasher_fun_type<Array_Type,Data_Type> Counter< Array_Type, Data_Type >::get_hasher ( )
```

### 8.18.3.4 get\_name()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::string Counter< Array_Type, Data_Type >::get_name ( ) const
```

### 8.18.3.5 init()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
double Counter< Array_Type, Data_Type >::init (
    Array_Type & Array,
    size_t i,
    size_t j )
```

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

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

Copy assignment.

**8.18.3.7 operator=() [2/2]**

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

Move assignment.

**8.18.3.8 set\_hasher()**

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
void Counter< Array_Type, Data_Type >::set_hasher (
    Hasher_fun_type< Array_Type, Data_Type > fun )
```

Get and set the hasher function.

The hasher function is used to characterize the support of the array. This way, if possible, the support enumeration is recycled.

**Parameters**

<i>fun</i>	
------------	--

**8.18.4 Member Data Documentation****8.18.4.1 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 38 of file counters-bones.hpp.



#### 8.18.4.2 data

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

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

#### 8.18.4.3 desc

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

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

#### 8.18.4.4 hasher\_fun

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Hasher_fun_type<Array_Type,Data_Type> Counter< Array_Type, Data_Type >::hasher_fun
```

Definition at line 40 of file counters-bones.hpp.

#### 8.18.4.5 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 39 of file counters-bones.hpp.

#### 8.18.4.6 name

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

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

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

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

## 8.19 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\_)
- Move assignment constructor.*
- [Counter](#)< Array\_Type, Data\_Type > & [operator\[\]](#) (size\_t 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\\_fun\\_type](#)< Array\_Type, Data\_Type > count\_fun\_, [Counter\\_fun\\_type](#)< Array\_Type, Data\_Type > init\_fun\_, [Hasher\\_fun\\_type](#)< Array\_Type, Data\_Type > hasher\_fun\_, Data\_Type data\_, std::string name\_="", std::string desc\_="")
- std::vector< std::string > [get\\_names](#) () const
- std::vector< std::string > [get\\_descriptions](#) () const
- std::vector< double > [gen\\_hash](#) (const Array\_Type &array, bool add\_dims=true)
- Generates a hash for the given array according to the counters.*
- void [add\\_hash](#) ([Hasher\\_fun\\_type](#)< Array\_Type, Data\_Type > fun\_)

### 8.19.1 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 108 of file counters-bones.hpp.

### 8.19.2 Constructor & Destructor Documentation

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

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

**8.19.2.2 ~Counters()**

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

Definition at line 120 of file counters-bones.hpp.

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

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

Copy constructor.

**Parameters**

<i>counter_↔</i>	
—	

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

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

Move constructor.

**Parameters**

<i>counters_↔</i>	
—	

**8.19.3 Member Function Documentation**

**8.19.3.1 add\_counter() [1/2]**

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
void Counters< Array_Type, Data_Type >::add_counter (
    Counter< Array_Type, Data_Type > counter )
```

**8.19.3.2 add\_counter() [2/2]**

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
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_,
    Hasher_fun_type< Array_Type, Data_Type > hasher_fun_,
    Data_Type data_,
    std::string name_ = "",
    std::string desc_ = "" )
```

**8.19.3.3 add\_hash()**

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
void Counters< Array_Type, Data_Type >::add_hash (
    Hasher_fun_type< Array_Type, Data_Type > fun_ )
```

**8.19.3.4 gen\_hash()**

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::vector< double > Counters< Array_Type, Data_Type >::gen_hash (
    const Array_Type & array,
    bool add_dims = true )
```

Generates a hash for the given array according to the counters.

**Parameters**

<i>array</i>	
<i>add_dims</i>	When <code>true</code> (default) the dimension of the array will be added to the hash.

**Returns**

`std::vector< double >` That can be hashed later.

**8.19.3.5 get\_descriptions()**

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::vector< std::string > Counters< Array_Type, Data_Type >::get_descriptions ( ) const
```

**8.19.3.6 get\_names()**

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::vector< std::string > Counters< Array_Type, Data_Type >::get_names ( ) const
```

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

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
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>](#)

**8.19.3.8 operator=() [2/2]**

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

Move assignment constructor.

**Parameters**

<i>counter_↔</i>	
—	

## Returns

[Counters](#)<[Array\\_Type](#),[Data\\_Type](#)>&

**8.19.3.9 operator[]()**

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Counter<Array\_Type,Data\_Type>& Counters< Array\_Type, Data\_Type >::operator[] (
    size_t idx )
```

Returns a pointer to a particular counter.

## Parameters

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

## Returns

[Counter](#)<[Array\\_Type](#),[Data\\_Type](#)>\*

**8.19.3.10 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

size\_t

Definition at line 164 of file counters-bones.hpp.

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

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

## 8.20 DEFM Class Reference

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

Inheritance diagram for DEFM:



Collaboration diagram for DEFM:



### Public Member Functions

- [DEFM](#) (int \*id, int \*y, double \*x, size\_t id\_length, size\_t y\_ncol, size\_t x\_ncol, size\_t m\_order, bool copy\_↔ data=true, bool column\_major=true)
- [DEFMMModel](#) & [get\\_model](#) ()
- void [init](#) ()
- void [simulate](#) (std::vector< double > par, int \*y\_out)
- size\_t [get\\_n\\_y](#) () const
- size\_t [get\\_n\\_obs](#) () const
- size\_t [get\\_n\\_covars](#) () const
- size\_t [get\\_m\\_order](#) () const
- size\_t [get\\_n\\_rows](#) () const
- const int \* [get\\_Y](#) () const
- const int \* [get\\_ID](#) () const
- const double \* [get\\_X](#) () const
- [barry::FreqTable](#)< int > [motif\\_census](#) (std::vector< size\_t > idx)

- `std::vector< double > logodds` (`const std::vector< double > &par`, `size_t i`, `size_t j`)
- `void set_names` (`std::vector< std::string > Y_names_`, `std::vector< std::string > X_names_`)
- `const std::vector< std::string > & get_Y_names` () `const`
- `const std::vector< std::string > & get_X_names` () `const`
- `void print` () `const`
- `std::vector< bool > is_motif` ()
- `bool get_column_major` () `const` `noexcept`

### 8.20.1 Detailed Description

Definition at line 4 of file `defm-bones.hpp`.

### 8.20.2 Constructor & Destructor Documentation

#### 8.20.2.1 DEFM()

```
DEFM::DEFM (
    int * id,
    int * y,
    double * x,
    size_t id_length,
    size_t y_ncol,
    size_t x_ncol,
    size_t m_order,
    bool copy_data = true,
    bool column_major = true ) [inline]
```

Definition at line 105 of file `defm-meat.hpp`.

### 8.20.3 Member Function Documentation

#### 8.20.3.1 get\_column\_major()

```
bool DEFM::get_column_major ( ) const [inline], [noexcept]
```

Definition at line 449 of file `defm-meat.hpp`.

#### 8.20.3.2 get\_ID()

```
const int * DEFM::get_ID ( ) const [inline]
```

Definition at line 310 of file `defm-meat.hpp`.



### 8.20.3.3 get\_m\_order()

```
size_t DEFM::get_m_order ( ) const [inline]
```

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

### 8.20.3.4 get\_model()

```
DEFMModel& DEFM::get_model ( ) [inline]
```

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

### 8.20.3.5 get\_n\_covars()

```
size_t DEFM::get_n_covars ( ) const [inline]
```

Definition at line 290 of file defm-meat.hpp.

### 8.20.3.6 get\_n\_obs()

```
size_t DEFM::get_n_obs ( ) const [inline]
```

Definition at line 285 of file defm-meat.hpp.

### 8.20.3.7 get\_n\_rows()

```
size_t DEFM::get_n_rows ( ) const [inline]
```

Definition at line 300 of file defm-meat.hpp.

### 8.20.3.8 get\_n\_y()

```
size_t DEFM::get_n_y ( ) const [inline]
```

Definition at line 280 of file defm-meat.hpp.

#### 8.20.3.9 `get_X()`

```
const double * DEFM::get_X ( ) const [inline]
```

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

#### 8.20.3.10 `get_X_names()`

```
const std::vector< std::string > & DEFM::get_X_names ( ) const [inline]
```

Definition at line 422 of file defm-meat.hpp.

#### 8.20.3.11 `get_Y()`

```
const int * DEFM::get_Y ( ) const [inline]
```

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

#### 8.20.3.12 `get_Y_names()`

```
const std::vector< std::string > & DEFM::get_Y_names ( ) const [inline]
```

Definition at line 418 of file defm-meat.hpp.

#### 8.20.3.13 `init()`

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

Definition at line 215 of file defm-meat.hpp.

#### 8.20.3.14 `is_motif()`

```
std::vector< bool > DEFM::is_motif ( ) [inline]
```

Definition at line 439 of file defm-meat.hpp.

### 8.20.3.15 logodds()

```
std::vector< double > DEFM::logodds (
    const std::vector< double > & par,
    size_t i,
    size_t j ) [inline]
```

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

### 8.20.3.16 motif\_census()

```
barry::FreqTable< int > DEFM::motif_census (
    std::vector< size_t > idx ) [inline]
```

Definition at line 321 of file defm-meat.hpp.

### 8.20.3.17 print()

```
void DEFM::print ( ) const [inline]
```

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

### 8.20.3.18 set\_names()

```
void DEFM::set_names (
    std::vector< std::string > Y_names_,
    std::vector< std::string > X_names_ ) [inline]
```

Definition at line 401 of file defm-meat.hpp.

### 8.20.3.19 simulate()

```
void DEFM::simulate (
    std::vector< double > par,
    int * y_out ) [inline]
```

Definition at line 39 of file defm-meat.hpp.

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

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

## 8.21 DEFMCOUNTERDATA Class Reference

Data class used to store arbitrary size\_t or double vectors.

```
#include <defm-types.hpp>
```

### Public Member Functions

- [DEFMCOUNTERDATA](#) ()
- [DEFMCOUNTERDATA](#) (const std::vector< size\_t > indices\_, const std::vector< double > numbers\_, const std::vector< bool > logical\_, bool is\_motif\_=true)
- size\_t [idx](#) (size\_t i) const
- double [num](#) (size\_t i) const
- bool [is\\_true](#) (size\_t i) const
- [~DEFMCOUNTERDATA](#) ()

### Public Attributes

- std::vector< size\_t > [indices](#)
- std::vector< double > [numbers](#)
- std::vector< bool > [logical](#)
- bool [is\\_motif](#)

*If false, then is a logit intercept.*

#### 8.21.1 Detailed Description

Data class used to store arbitrary size\_t or double vectors.

Definition at line 64 of file defm-types.hpp.

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

- include/barry/models/defm/[defm-types.hpp](#)

## 8.22 DEFMDATA Class Reference

Data class for [DEFM](#) arrays.

```
#include <defm-types.hpp>
```

## Public Member Functions

- [DEFMDData](#) ()  
*Vector indicating which covariates are included in the model.*
- [DEFMDData](#) ([DEFMArray](#) \*array\_, const double \*covariates\_, size\_t obs\_start\_, size\_t X\_ncol\_, size\_t X\_nrow\_)  
*Constructor.*
- double [operator\(\)](#) (size\_t i, size\_t j) const  
*Access to the row (i) column (j) data.*
- double [at](#) (size\_t i, size\_t j) const
- size\_t [ncol](#) () const
- size\_t [nrow](#) () const
- void [print](#) () const
- [~DEFMDData](#) ()

## Public Attributes

- [DEFMArray](#) \* [array](#)
- const double \* [covariates](#)  
*Vector of covariates (complete vector)*
- size\_t [obs\\_start](#)  
*Index of the observation in the data.*
- size\_t [X\\_ncol](#)  
*Number of columns in the array of covariates.*
- size\_t [X\\_nrow](#)  
*Number of rows in the array of covariates.*
- std::vector< size\_t > [covar\\_sort](#)
- std::vector< size\_t > [covar\\_used](#)  
*Value where the sorting of the covariates is stored.*

### 8.22.1 Detailed Description

Data class for [DEFM](#) arrays.

This holds information pointing to the data array, including information regarding the number of observations, the time slices of the observation, and the number of covariates in the data.

Definition at line 16 of file [defm-types.hpp](#).

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

- [include/barry/models/defm/defm-types.hpp](#)

## 8.23 DEFMRuleData Class Reference

```
#include <defm-types.hpp>
```

Inheritance diagram for DEFMRuleData:



### Public Member Functions

- double [num](#) (size\_t [i](#)) const
- size\_t [idx](#) (size\_t [i](#)) const
- bool [is\\_true](#) (size\_t [i](#)) const
- [DEFMRuleData](#) ()
- [DEFMRuleData](#) (std::vector< double > [numbers\\_](#), std::vector< size\_t > [indices\\_](#), std::vector< bool > [logical\\_](#))
- [DEFMRuleData](#) (std::vector< double > [numbers\\_](#), std::vector< size\_t > [indices\\_](#))

### Public Attributes

- std::vector< double > [numbers](#)
- std::vector< size\_t > [indices](#)
- std::vector< bool > [logical](#)
- bool [init](#) = false

### 8.23.1 Detailed Description

Definition at line 89 of file [defm-types.hpp](#).

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

- [include/barry/models/defm/defm-types.hpp](#)

## 8.24 DEFMRuleDynData Class Reference

```
#include <defm-types.hpp>
```

Inheritance diagram for DEFMRuleDynData:



Collaboration diagram for DEFMRuleDynData:



### Public Member Functions

- [DEFMRuleDynData](#) (const std::vector< double > \*counts\_, std::vector< double > numbers\_={}, std::vector< size\_t > indices\_={}, std::vector< bool > logical\_={})
- [~DEFMRuleDynData](#) ()

### Public Attributes

- const std::vector< double > \* [counts](#)

#### 8.24.1 Detailed Description

Definition at line 152 of file defm-types.hpp.

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

- include/barry/models/defm/[defm-types.hpp](#)

## 8.25 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](#) (size\_t n)
- [~Entries](#) ()
- void [resize](#) (size\_t n)

### Public Attributes

- std::vector< size\_t > [source](#)
- std::vector< size\_t > [target](#)
- std::vector< Cell\_Type > [val](#)

### 8.25.1 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 78 of file `typedefs.hpp`.

### 8.25.2 Constructor & Destructor Documentation

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

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

Definition at line 84 of file `typedefs.hpp`.



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

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

Definition at line 85 of file typedefs.hpp.

### 8.25.2.3 ~Entries()

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

Definition at line 92 of file typedefs.hpp.

## 8.25.3 Member Function Documentation

### 8.25.3.1 resize()

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

Definition at line 94 of file typedefs.hpp.

## 8.25.4 Member Data Documentation

### 8.25.4.1 source

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

Definition at line 80 of file typedefs.hpp.

### 8.25.4.2 target

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

Definition at line 81 of file typedefs.hpp.

### 8.25.4.3 val

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

Definition at line 82 of file typedefs.hpp.

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

- include/barry/typedefs.hpp

## 8.26 Flock Class Reference

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

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

### Public Member Functions

- [Flock](#) ()
- [~Flock](#) ()
- [size\\_t add\\_data](#) (std::vector< std::vector< [size\\_t](#) > > &annotations, std::vector< [size\\_t](#) > &geneid, std::vector< [int](#) > &parent, std::vector< [bool](#) > &duplication)  
*Add a tree to the flock.*
- void [set\\_seed](#) (const [size\\_t](#) &s)  
*Set the seed of the model.*
- void [init](#) ([size\\_t](#) bar\_width=[BARRY\\_PROGRESS\\_BAR\\_WIDTH](#))
- [PhyloCounters](#) \* [get\\_counters](#) ()
- [PhyloSupport](#) \* [get\\_support\\_fun](#) ()
- std::vector< std::vector< [double](#) > > \* [get\\_stats\\_support](#) ()
- std::vector< std::vector< [double](#) > > \* [get\\_stats\\_target](#) ()
- [PhyloModel](#) \* [get\\_model](#) ()
- [double likelihood\\_joint](#) (const std::vector< [double](#) > &par, [bool](#) as\_log=false, [bool](#) use\_reduced\_↵ sequence=true, [size\\_t](#) ncores=1u)  
*Returns the joint likelihood of the model.*
- [Geese](#) \* [operator\(\)](#) ([size\\_t](#) i, [bool](#) check\_bounds=true)  
*Access the i-th geese element.*

### Information about the model

- [size\\_t nfuncs](#) () const [noexcept](#)
- [size\\_t ntrees](#) () const [noexcept](#)
- std::vector< [size\\_t](#) > [nnodes](#) () const [noexcept](#)
- std::vector< [size\\_t](#) > [nleafs](#) () const [noexcept](#)
- [size\\_t nterms](#) () const
- [size\\_t support\\_size](#) () const [noexcept](#)
- std::vector< std::string > [colnames](#) () const
- [size\\_t parse\\_polytomies](#) ([bool](#) verb=true, std::vector< [size\\_t](#) > \*dist=nullptr) const [noexcept](#)  
*Check polytomies and return the largest.*
- void [print](#) () const

## Public Attributes

- `std::vector< Geese > dat`
- `size_t nfunctions = 0u`
- `bool initialized = false`
- `std::mt19937 engine`
- `PhyloModel model = PhyloModel()`

### 8.26.1 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.

### 8.26.2 Constructor & Destructor Documentation

#### 8.26.2.1 Flock()

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

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

#### 8.26.2.2 ~Flock()

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

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

### 8.26.3 Member Function Documentation

#### 8.26.3.1 add\_data()

```
size_t Flock::add_data (
    std::vector< std::vector< size_t > > & annotations,
    std::vector< size_t > & 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**

size\_t The number of tree in the model (starting from zero).

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

**8.26.3.2 colnames()**

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

Definition at line 228 of file flock-meat.hpp.

**8.26.3.3 get\_counters()**

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

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

**8.26.3.4 get\_model()**

```
PhyloModel * Flock::get_model ( ) [inline]
```

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

**8.26.3.5 get\_stats\_support()**

```
std::vector< std::vector< double > > * Flock::get_stats_support ( ) [inline]
```

Definition at line 117 of file flock-meat.hpp.

### 8.26.3.6 get\_stats\_target()

```
std::vector< std::vector< double > > * Flock::get_stats_target ( ) [inline]
```

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

### 8.26.3.7 get\_support\_fun()

```
PhyloSupport * Flock::get_support_fun ( ) [inline]
```

Definition at line 110 of file flock-meat.hpp.

### 8.26.3.8 init()

```
void Flock::init (
    size_t bar_width = BARRY_PROGRESS_BAR_WIDTH ) [inline]
```

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

### 8.26.3.9 likelihood\_joint()

```
double Flock::likelihood_joint (
    const std::vector< double > & par,
    bool as_log = false,
    bool use_reduced_sequence = true,
    size_t ncores = 1u ) [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 138 of file flock-meat.hpp.

#### 8.26.3.10 nfuncs()

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

Definition at line 171 of file flock-meat.hpp.

#### 8.26.3.11 nleafs()

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

Definition at line 199 of file flock-meat.hpp.

#### 8.26.3.12 nnodes()

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

Definition at line 185 of file flock-meat.hpp.

#### 8.26.3.13 nterms()

```
size_t Flock::nterms ( ) const [inline]
```

Definition at line 213 of file flock-meat.hpp.

#### 8.26.3.14 ntrees()

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

Definition at line 178 of file flock-meat.hpp.

#### 8.26.3.15 operator>()()

```
Geese * Flock::operator() (
    size_t 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 306 of file flock-meat.hpp.

**8.26.3.16 parse\_polytomies()**

```
size_t Flock::parse_polytomies (
    bool verb = true,
    std::vector< size_t > * dist = nullptr ) const [inline], [noexcept]
```

Check polytomies and return the largest.

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

**8.26.3.17 print()**

```
void Flock::print ( ) const [inline]
```

Definition at line 262 of file flock-meat.hpp.

**8.26.3.18 set\_seed()**

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

Set the seed of the model.

## Parameters

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

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

### 8.26.3.19 support\_size()

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

Definition at line 221 of file flock-meat.hpp.

## 8.26.4 Member Data Documentation

### 8.26.4.1 dat

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

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

### 8.26.4.2 initialized

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

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

### 8.26.4.3 model

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

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

### 8.26.4.4 nfunctions

```
size_t Flock::nfunctions = 0u
```

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



#### 8.26.4.5 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-meat.hpp

## 8.27 FreqTable< T > Class Template Reference

Frequency table of vectors.

```
#include <freqtable.hpp>
```

### Public Member Functions

- [FreqTable](#) ()
- [~FreqTable](#) ()
- [size\\_t add](#) (const std::vector< T > &x, size\_t \*h\_precomp)
- [Counts\\_type as\\_vector](#) () const
- const std::vector< double > & [get\\_data](#) () const
- const std::unordered\_map< size\_t, size\_t > & [get\\_index](#) () const
- void [clear](#) ()
- void [reserve](#) (size\_t n, size\_t k)
- void [print](#) () const
- [size\\_t size](#) () const [noexcept](#)
- *Number of unique elements in the table. (.*
- [size\\_t make\\_hash](#) (const std::vector< T > &x) const

### 8.27.1 Detailed Description

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

Frequency table of vectors.

This is mostly used in [Support](#). The main data is contained in the `data` double vector. The matrix is stored in a row-wise fashion, where the first element is the frequency with which the vector is observed.

For example, in a model with `k` terms the first `k + 1` elements of `data` would be:

- weights
- term 1
- term 2
- ...
- term k

Definition at line 22 of file freqtable.hpp.

## 8.27.2 Constructor & Destructor Documentation

### 8.27.2.1 FreqTable()

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

Definition at line 34 of file freqtable.hpp.

### 8.27.2.2 ~FreqTable()

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

Definition at line 35 of file freqtable.hpp.

## 8.27.3 Member Function Documentation

### 8.27.3.1 add()

```
template<typename T >
size_t FreqTable< T >::add (
    const std::vector< T > & x,
    size_t * h_precomp ) [inline]
```

Definition at line 59 of file freqtable.hpp.

### 8.27.3.2 as\_vector()

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

Definition at line 139 of file freqtable.hpp.

### 8.27.3.3 clear()

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

Definition at line 168 of file freqtable.hpp.

### 8.27.3.4 get\_data()

```
template<typename T = double>
const std::vector< double >& FreqTable< T >::get_data ( ) const [inline]
```

Definition at line 40 of file freqtable.hpp.

### 8.27.3.5 get\_index()

```
template<typename T = double>
const std::unordered_map<size_t,size_t>& FreqTable< T >::get_index ( ) const [inline]
```

Definition at line 41 of file freqtable.hpp.

### 8.27.3.6 make\_hash()

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

Definition at line 239 of file freqtable.hpp.

### 8.27.3.7 print()

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

Definition at line 204 of file freqtable.hpp.

### 8.27.3.8 reserve()

```
template<typename T >
void FreqTable< T >::reserve (
    size_t n,
    size_t k ) [inline]
```

Definition at line 182 of file freqtable.hpp.

### 8.27.3.9 size()

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

Number of unique elements in the table. (.

#### Returns

size\_t

Definition at line 231 of file freqtable.hpp.

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

- include/barry/freqtable.hpp

## 8.28 Geese Class Reference

Annotated Phylo [Model](#).

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

### Public Member Functions

- [~Geese](#) ()
- void [init](#) (size\_t bar\_width=[BARRY\\_PROGRESS\\_BAR\\_WIDTH](#))
- 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, size\_t ncores=1u, bool no\_update\_normalizing\_constant=false)
- double [likelihood\\_exhaust](#) (const std::vector< double > &par)
- std::vector< double > [get\\_probabilities](#) () const
- void [set\\_seed](#) (const size\_t &s)
- std::vector< std::vector< size\_t > > [simulate](#) (const std::vector< double > &par)
- std::vector< std::vector< double > > [observed\\_counts](#) ()
- void [print\\_observed\\_counts](#) ()
- void [print](#) () const

*Prints information about the GEESE.*

- void `print_nodes` () const
- void `init_node` (Node &n)
- void `update_annotations` (size\_t nodeid, std::vector< size\_t > newann)
- std::vector< std::vector< bool > > `get_states` () const

*Powerset of a gene's possible states.*

- std::vector< size\_t > `get_annotated_nodes` () const

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

- std::vector< size\_t > `get_annotations` () const

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

### Construct a new Geese object

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

#### Parameters

annotations	A vector of vectors with annotations. It should be of length $k$ (number of functions). Each vector should be of length $N$ (equal to the number of nodes, including interior). Possible values are 0, 1, and 9.
geneid	Id of the gene. It should be of length $N$ .
parent	Id of the parent gene. Also of length $N$
duplication	Logical scalar indicating the type of event (true: duplication, false: speciation.)

The ordering of the entries does not matter. Passing the nodes in post order or not makes no difference to the constructor.

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

### Information about the model

#### Parameters

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

- size\_t `nfuncs` () const `noexcept`  
*Number of functions analyzed.*
- size\_t `nnodes` () const `noexcept`  
*Number of nodes (interior + leaf)*
- size\_t `nleafs` () const `noexcept`  
*Number of leaf.*
- size\_t `nterms` () const  
*Number of terms included.*
- size\_t `support_size` () const `noexcept`  
*Number of unique sets of sufficient stats.*
- std::vector< size\_t > `nannotations` () const `noexcept`  
*Number of annotations.*
- std::vector< std::string > `colnames` () const  
*Names of the terms in the model.*

- `size_t parse_polytomies` (bool verb=true, std::vector< size\_t > \*dist=nullptr) const noexcept  
Check polytomies and return the largest.

### Geese prediction

Calculate the conditional probability

#### Parameters

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

When *res\_prob* is specified, the function will attach the member vector probabilities from the *Nodes* objects. This contains the probability that the *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< size\_t > &preorder)
- `std::vector< std::vector< double > >` `predict_exhaust_backend` (const std::vector< double > &par, const std::vector< size\_t > &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, size\_t 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_fun()` returns the computed support of the model.

- `std::mt19937 * get_engine()`
- `PhyloCounters * get_counters()`
- `PhyloModel * get_model()`
- `PhyloSupport * get_support_fun()`

### Public Attributes

- `size_t nfunctions`
- `std::map< size_t, Node > nodes`
- `barry::MapVec_type< size_t > map_to_state_id`

- `std::vector< std::vector< std::vector< size_t > > > pset_loc`  
*Locations of columns.*
- `std::vector< size_t > sequence`
- `std::vector< size_t > reduced_sequence`
- `bool initialized = false`
- `bool delete_engine = false`
- `bool delete_support = false`

## Static Public Attributes

- `static const size_t etype_default = 1ul`
- `static const size_t etype_speciation = 0ul`
- `static const size_t etype_duplication = 1ul`
- `static const size_t etype_either = 2ul`

### 8.28.1 Detailed Description

Annotated Phylo [Model](#).

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

Class representing a phylogenetic tree model with annotations.

The [Geese](#) class represents a phylogenetic tree model with annotations. It includes a total of  $N + 1$  nodes, the  $+ 1$  being the root node. The class provides methods for initializing the model, calculating the likelihood, simulating trees, and making predictions.

The class includes shared objects within a [Geese](#) object, such as `rengine`, `model`, `states`, `n_zeros`, `n_ones`, `n_dupl_events`, and `n_spec_events`. It also includes information about the type of event, such as `etype_default`, `etype_speciation`, `etype_duplication`, and `etype_either`.

The class provides constructors, a destructor, and methods for initializing the model, inheriting support, calculating the sequence, calculating the reduced sequence, calculating the likelihood, calculating the likelihood exhaustively, getting probabilities, setting the seed, simulating trees, parsing polytomies, getting observed counts, printing observed counts, printing information about the GESE, and making predictions.

See also

[Flock](#)

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

### 8.28.2 Constructor & Destructor Documentation

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

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

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

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

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

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

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

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

Definition at line 230 of file geese-meat-constructors.hpp.

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

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

Definition at line 309 of file geese-meat-constructors.hpp.

### 8.28.2.5 ~Geese()

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

Definition at line 119 of file geese-meat.hpp.

## 8.28.3 Member Function Documentation



### 8.28.3.1 calc\_reduced\_sequence()

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

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

### 8.28.3.2 calc\_sequence()

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

Definition at line 314 of file geese-meat.hpp.

### 8.28.3.3 colnames()

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

Names of the terms in the model.

Definition at line 480 of file geese-meat.hpp.

### 8.28.3.4 get\_annotated\_nodes()

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

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

Definition at line 769 of file geese-meat.hpp.

### 8.28.3.5 get\_annotations()

```
std::vector< size_t > Geese::get_annotations ( ) const [inline]
```

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

Definition at line 792 of file geese-meat.hpp.

### 8.28.3.6 get\_counters()

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

Definition at line 752 of file geese-meat.hpp.

### 8.28.3.7 get\_model()

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

Definition at line 757 of file geese-meat.hpp.

### 8.28.3.8 get\_probabilities()

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

Definition at line 406 of file geese-meat.hpp.

### 8.28.3.9 get\_rengine()

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

Definition at line 747 of file geese-meat.hpp.

### 8.28.3.10 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 765 of file geese-meat.hpp.

### 8.28.3.11 get\_support\_fun()

```
PhyloSupport * Geese::get_support_fun ( ) [inline]
```

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

### 8.28.3.12 inherit\_support()

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

Definition at line 257 of file geese-meat.hpp.

### 8.28.3.13 init()

```
void Geese::init (
    size_t bar_width = BARRY_PROGRESS_BAR_WIDTH ) [inline]
```

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

### 8.28.3.14 init\_node()

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

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

### 8.28.3.15 likelihood()

```
double Geese::likelihood (
    const std::vector< double > & par,
    bool as_log = false,
    bool use_reduced_sequence = true,
    size_t ncores = 1u,
    bool no_update_normalizing_constant = false ) [inline]
```

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

#### 8.28.3.16 likelihood\_exhaust()

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

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

#### 8.28.3.17 nannotations()

```
std::vector< size_t > Geese::nannotations ( ) const [inline], [noexcept]
```

Number of annotations.

Definition at line 471 of file geese-meat.hpp.

#### 8.28.3.18 nfuncs()

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

Number of functions analyzed.

Definition at line 427 of file geese-meat.hpp.

#### 8.28.3.19 nleafs()

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

Number of leaf.

Definition at line 441 of file geese-meat.hpp.

#### 8.28.3.20 nnodes()

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

Number of nodes (interior + leaf)

Definition at line 434 of file geese-meat.hpp.

### 8.28.3.21 nterms()

```
size_t Geese::nterms ( ) const [inline]
```

Number of terms included.

Definition at line 453 of file geese-meat.hpp.

### 8.28.3.22 observed\_counts()

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

Definition at line 522 of file geese-meat.hpp.

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

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

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

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

### 8.28.3.25 parse\_polytomies()

```
size_t Geese::parse_polytomies (
    bool verb = true,
    std::vector< size_t > * dist = nullptr ) const [inline], [noexcept]
```

Check polytomies and return the largest.

Definition at line 487 of file geese-meat.hpp.

### 8.28.3.26 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 287 of file geese-meat-predict.hpp.

### 8.28.3.27 predict\_backend()

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

< True if the array belongs to the set

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

### 8.28.3.28 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.

### 8.28.3.29 predict\_exhaust\_backend()

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

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

### 8.28.3.30 predict\_sim()

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

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

**8.28.3.31 print()**

```
void Geese::print ( ) const [inline]
```

Prints information about the GEESE.

Definition at line 656 of file geese-meat.hpp.

**8.28.3.32 print\_nodes()**

```
void Geese::print_nodes ( ) const [inline]
```

Definition at line 674 of file geese-meat.hpp.

**8.28.3.33 print\_observed\_counts()**

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

Definition at line 593 of file geese-meat.hpp.

**8.28.3.34 set\_seed()**

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

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

**8.28.3.35 simulate()**

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

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

**8.28.3.36 support\_size()**

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

Number of unique sets of sufficient stats.

Definition at line 461 of file geese-meat.hpp.

### 8.28.3.37 update\_annotations()

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

Definition at line 285 of file geese-meat.hpp.

## 8.28.4 Member Data Documentation

### 8.28.4.1 delete\_engine

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

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

### 8.28.4.2 delete\_support

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

Definition at line 143 of file geese-bones.hpp.

### 8.28.4.3 etype\_default

```
const size_t Geese::etype_default = 1ul [static]
```

Definition at line 156 of file geese-bones.hpp.

### 8.28.4.4 etype\_duplication

```
const size_t Geese::etype_duplication = 1ul [static]
```

Definition at line 158 of file geese-bones.hpp.



#### 8.28.4.5 etype\_either

```
const size_t Geese::etype_either = 2ul [static]
```

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

#### 8.28.4.6 etype\_speciation

```
const size_t Geese::etype_speciation = 0ul [static]
```

Definition at line 157 of file geese-bones.hpp.

#### 8.28.4.7 initialized

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

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

#### 8.28.4.8 map\_to\_state\_id

```
barry::MapVec_type< size_t > Geese::map_to_state_id
```

Definition at line 133 of file geese-bones.hpp.

#### 8.28.4.9 nfunctions

```
size_t Geese::nfunctions
```

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

#### 8.28.4.10 nodes

```
std::map< size_t, Node > Geese::nodes
```

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

#### 8.28.4.11 pset\_loc

```
std::vector< std::vector< std::vector< size_t > > > Geese::pset_loc
```

Locations of columns.

Definition at line 134 of file geese-bones.hpp.

#### 8.28.4.12 reduced\_sequence

```
std::vector< size_t > Geese::reduced_sequence
```

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

#### 8.28.4.13 sequence

```
std::vector< size_t > Geese::sequence
```

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

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

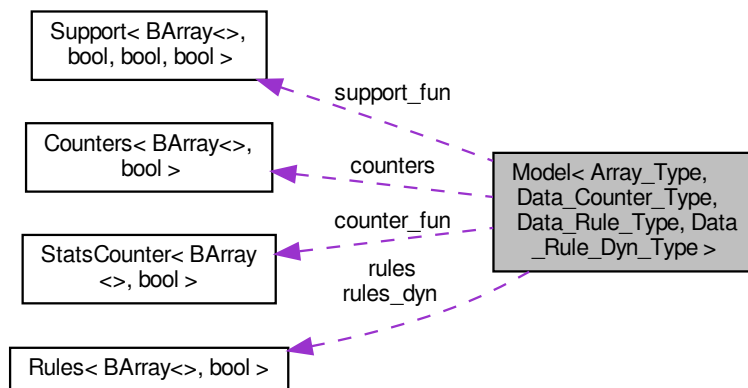
- include/barry/models/geese/[geese-bones.hpp](#)
- include/barry/models/geese/[geese-meat-constructors.hpp](#)
- include/barry/models/geese/[geese-meat-likelihood.hpp](#)
- include/barry/models/geese/[geese-meat-likelihood\\_exhaust.hpp](#)
- include/barry/models/geese/[geese-meat-predict.hpp](#)
-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

## 8.29 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>
```

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



### Public Member Functions

- void `update_normalizing_constants` (const std::vector< double > &params,)
  - Computes the normalizing constant for a given set of parameters.*
- void `set_engine` (std::mt19937 \*engine\_, bool delete\_=false)
- void `set_seed` (size\_t s)
- `Model` ()
- `Model` (size\_t 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\_)
- virtual `~Model` ()
- void `store_psets` () `noexcept`
- std::vector< double > `gen_key` (const Array\_Type &Array\_)
- size\_t `add_array` (const Array\_Type &Array\_, bool force\_new=false)
  - Adds an array to the support of not already included.*
- void `print_stats` (size\_t i) const
- virtual void `print` () const
  - Prints information about the model.*
- Array\_Type `sample` (const Array\_Type &Array\_, const std::vector< double > &params={})
- Array\_Type `sample` (const size\_t &i, const std::vector< double > &params)
- double `conditional_prob` (const Array\_Type &Array\_, const std::vector< double > &params, size\_t i, size\_t j)
  - Conditional probability ("Gibbs sampler")*

- `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\_fun ()`

#### 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\_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)`
- `void set\_counters (Counters< Array_Type, Data_Counter_Type > *counters\_)`
- `void add\_hasher (Hasher\_fun\_type< Array_Type, Data_Counter_Type > fun\_)`

#### 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\_fun\_type< Array_Type, Data_Rule_Type > count\_fun\_, Data_Rule_Type data\_)`
- `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\_fun\_type< Array_Type, Data_Rule_Dyn_Type > count\_fun\_, Data_Rule_Dyn_Type data\_)`
- `void set\_rules\_dyn (Rules< Array_Type, Data_Rule_Dyn_Type > *rules\_)`

#### Likelihood functions.

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

##### Parameters

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

- `double likelihood (const std::vector< double > &params, const size_t &i, bool as\_log=false, bool no\_update\_normconst=false)`
- `double likelihood (const std::vector< double > &params, const Array_Type &Array\_, int i=-1, bool as\_log=false, bool no\_update\_normconst=false)`
- `double likelihood (const std::vector< double > &params, const std::vector< double > &target\_, const size_t &i, bool as\_log=false, bool no\_update\_normconst=false)`
- `double likelihood (const std::vector< double > &params, const double *target\_, const size_t &i, bool as\_log=false, bool no\_update\_normconst=false)`
- `double likelihood\_total (const std::vector< double > &params, bool as\_log=false, BARRY\_NCORES\_ARG(=2), bool no\_update\_normconst=false)`

#### Extract elements by index

##### Parameters

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

- `std::vector< double > & get_normalizing_constants ()`
- `const std::vector< Array_Type > * get_pset (const size_t &i)`
- `const std::vector< double > * get_pset_stats (const size_t &i)`

### Size of the model

*Number of different supports included in the model*

*This will return the size of `stats_target`.*

#### 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.

- `size_t size () const noexcept`
- `size_t size_unique () const noexcept`
- `size_t nterms () const noexcept`
- `size_t nrules () const noexcept`
- `size_t nrules_dyn () const noexcept`
- `size_t support_size () const noexcept`
- `std::vector< std::string > colnames () const`

- `std::vector< std::vector< double > > * get_stats_target ()`  
*Raw pointers to the support and target statistics.*
- `std::vector< std::vector< double > > * get_stats_support ()`
- `std::vector< size_t > * get_arrays2support ()`
- `std::vector< std::vector< Array_Type > > * get_pset_arrays ()`
- `std::vector< std::vector< double > > * get_pset_stats ()`  
*Statistics of the support(s)*
- `std::vector< std::vector< double > > * get_pset_probs ()`

- `void set_transform_model (std::function< std::vector< double >(double *, size_t)> fun, std::vector< std::string > names)`  
*Set the transform\_model\_fun object.*
- `std::vector< double > transform_model (double *data, size_t k)`

### Protected Attributes

- `MapVec_type< double, size_t > keys2support`  
*Map of types of arrays to support sets.*
- `std::vector< std::vector< double > > params_last`  
*Vector of the previously used parameters.*
- `std::vector< double > normalizing_constants`
- `std::vector< bool > first_calc_done`
- `bool delete_counters = false`
- `bool delete_rules = false`
- `bool delete_rules_dyn = false`
- `std::function< std::vector< double >(double *, size_t k)> transform_model_fun = nullptr`

*Transformation of the model.*

- `std::vector< std::string >` [transform\\_model\\_term\\_names](#)

### Random number generation

*Random number generation*

- `std::mt19937 * rengine` = nullptr
- `bool delete\_rengine` = false

### Information about the arrays used in the model

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

*Each vector of [stats\\_support](#) has the data stored in a row-wise order, with each row starting with the weights, e.g., in a model with  $k$  terms the first  $k + 1$  elements of [stats\\_support](#) would be:*

- *weights*
- *term 1*
- *term 2*
- *...*
- *term  $k$*
- `std::vector< std::vector< double > >` [stats\\_support](#)  
*Sufficient statistics of the model (support)*
- `std::vector< size_t >` [stats\\_support\\_n\\_arrays](#)  
*Number of arrays included per support.*
- `std::vector< std::vector< double > >` [stats\\_target](#)  
*Target statistics of the model.*
- `std::vector< size_t >` [arrays2support](#)

### Container space for the powerset (and its sufficient [stats\\_target](#))

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

- `bool with\_pset` = false
- `std::vector< std::vector< Array_Type > >` [pset\\_arrays](#)  
*Arrays of the support(s)*
- `std::vector< std::vector< double > >` [pset\\_stats](#)  
*Statistics of the support(s)*
- `std::vector< std::vector< double > >` [pset\\_probs](#)  
*Probabilities of the support(s)*

### Functions to compute statistics

*Arguments are recycled to save memory and computation.*

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

## 8.29.1 Detailed Description

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

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

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

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

### Template Parameters

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

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

## 8.29.2 Constructor & Destructor Documentation

### 8.29.2.1 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 190 of file model-meat.hpp.

### 8.29.2.2 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 (
    size_t size_ ) [inline]
```

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

### 8.29.2.3 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 262 of file model-meat.hpp.

### 8.29.2.4 ~Model()

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

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

## 8.29.3 Member Function Documentation

### 8.29.3.1 add\_array()

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

Adds an array to the support of not already included.

#### Parameters

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

#### Returns

The number of the array.

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

Definition at line 495 of file model-meat.hpp.



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

```
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 371 of file model-meat.hpp.

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

```
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 ) [inline]
```

Definition at line 380 of file model-meat.hpp.

### 8.29.3.4 add\_hasher()

```
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_hasher (
    Hasher_fun_type< Array_Type, Data_Counter_Type > fun_ ) [inline]
```

Definition at line 415 of file model-meat.hpp.

### 8.29.3.5 add\_rule() [1/2]

```
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 426 of file model-meat.hpp.

### 8.29.3.6 add\_rule() [2/2]

```
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 >::add_rule (
    Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_,
    Data_Rule_Type data_ )
```

**8.29.3.7 add\_rule\_dyn()** [1/2]

```
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 454 of file model-meat.hpp.

**8.29.3.8 add\_rule\_dyn()** [2/2]

```
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_ ) [inline]
```

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

**8.29.3.9 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 1154 of file model-meat.hpp.

**8.29.3.10 conditional\_prob()**

```
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 >::conditional_prob (
    const Array_Type & Array_,
    const std::vector< double > & params,
    size_t i,
    size_t j ) [inline]
```

Conditional probability ("Gibbs sampler")

Computes the conditional probability of observing  $P\{Y(i,j) = Y^C(i,j), \theta\}$ , i.e., the probability of observing the entry  $Y(i,j)$  equal to one given the rest of the array.

#### Parameters

<i>Array</i> <sub>↔</sub>	Array to check
<i>params</i>	Vector of parameters
<i>i</i>	Row entry
<i>j</i>	Column entry

#### Returns

double The conditional probability

Definition at line 1419 of file model-meat.hpp.

##### 8.29.3.11 gen\_key()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
std::vector< double > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_↔
Type >::gen_key (
    const Array_Type & Array_ ) [inline]
```

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

##### 8.29.3.12 get\_arrays2support()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
std::vector< size_t > * Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_↔
_Type >::get_arrays2support [inline]
```

Definition at line 1490 of file model-meat.hpp.

##### 8.29.3.13 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 1457 of file model-meat.hpp.

**8.29.3.14 get\_normalizing\_constants()**

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
std::vector< double > & Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_↵
_Type >::get_normalizing_constants [inline]
```

Definition at line 974 of file model-meat.hpp.

**8.29.3.15 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 size_t & i ) [inline]
```

Definition at line 981 of file model-meat.hpp.

**8.29.3.16 get\_pset\_arrays()**

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
std::vector< std::vector< Array_Type > > * Model< Array_Type, Data_Counter_Type, Data_Rule_↵
_Type, Data_Rule_Dyn_Type >::get_pset_arrays [inline]
```

Definition at line 1496 of file model-meat.hpp.

**8.29.3.17 get\_pset\_probs()**

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
std::vector< std::vector< double > > * Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_pset_probs [inline]
```

Definition at line 1506 of file model-meat.hpp.

**8.29.3.18 get\_pset\_stats() [1/2]**

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
std::vector< std::vector< double > > * Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_pset_stats [inline]
```

Statistics of the support(s)

Definition at line 1501 of file model-meat.hpp.

### 8.29.3.19 get\_pset\_stats() [2/2]

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

Definition at line 994 of file model-meat.hpp.

### 8.29.3.20 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 1452 of file model-meat.hpp.

### 8.29.3.21 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 1462 of file model-meat.hpp.

### 8.29.3.22 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 1467 of file model-meat.hpp.

### 8.29.3.23 get\_stats\_support()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
std::vector< std::vector< double > > * Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_stats_support [inline]
```

Definition at line 1484 of file model-meat.hpp.

### 8.29.3.24 get\_stats\_target()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
std::vector< std::vector< double > > * Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_stats_target [inline]
```

Raw pointers to the support and target statistics.

The support of the model is stored as a vector of vector<double>. Each element of it contains the support for an specific type of array included. It represents an array of size  $(k + 1) \times n$  unique elements, with the data stored by-row. The last element of each entry corresponds to the weights, i.e., the frequency with which such sufficient statistics are observed in the support.

Definition at line 1478 of file model-meat.hpp.

### 8.29.3.25 get\_support\_fun()

```
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_fun [inline]
```

Definition at line 1473 of file model-meat.hpp.

### 8.29.3.26 likelihood() [1/4]

```
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,
    bool no_update_normconst = false ) [inline]
```

Definition at line 676 of file model-meat.hpp.

### 8.29.3.27 likelihood() [2/4]

```
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 double * target_,
    const size_t & i,
    bool as_log = false,
    bool no_update_normconst = false ) [inline]
```

Definition at line 822 of file model-meat.hpp.

### 8.29.3.28 likelihood() [3/4]

```
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 size_t & i,
    bool as_log = false,
    bool no_update_normconst = false ) [inline]
```

Definition at line 631 of file model-meat.hpp.

### 8.29.3.29 likelihood() [4/4]

```
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 size_t & i,
    bool as_log = false,
    bool no_update_normconst = false ) [inline]
```

Definition at line 758 of file model-meat.hpp.

### 8.29.3.30 likelihood\_total()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
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,
    BARRY_NCORES_ARG(=2) ,
    bool no_update_normconst = false ) [inline]
```

Definition at line 892 of file model-meat.hpp.

### 8.29.3.31 nrules()

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

Definition at line 1125 of file model-meat.hpp.

**8.29.3.32 nrules\_dyn()**

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

Definition at line 1133 of file model-meat.hpp.

**8.29.3.33 nterms()**

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

Definition at line 1114 of file model-meat.hpp.

**8.29.3.34 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 306 of file model-meat.hpp.

**8.29.3.35 print()**

```
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 [inline],
[virtual]
```

Prints information about the model.

Definition at line 1036 of file model-meat.hpp.



### 8.29.3.36 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 (
    size_t i ) const [inline]
```

Definition at line 1006 of file model-meat.hpp.

### 8.29.3.37 sample() [1/2]

```
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 Array_Type & Array_,
    const std::vector< double > & params = {} ) [inline]
```

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

Definition at line 1250 of file model-meat.hpp.

### 8.29.3.38 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 size_t & i,
    const std::vector< double > & params ) [inline]
```

Definition at line 1170 of file model-meat.hpp.

### 8.29.3.39 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 397 of file model-meat.hpp.

**8.29.3.40 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 140 of file model-bones.hpp.

**8.29.3.41 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 436 of file model-meat.hpp.

**8.29.3.42 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 478 of file model-meat.hpp.

**8.29.3.43 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 (
    size_t s ) [inline]
```

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

**8.29.3.44 set\_transform\_model()**

```
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_transform_model (
    std::function< std::vector< double >(double *, size_t)> fun,
    std::vector< std::string > names ) [inline]
```

Set the transform\_model\_fun object.

The transform\_model function is used to transform the data

#### Parameters

<i>data</i>	
<i>target</i>	
<i>n_arrays</i>	
<i>arrays2support</i>	

Definition at line 1511 of file model-meat.hpp.

#### 8.29.3.45 size()

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

Definition at line 1097 of file model-meat.hpp.

#### 8.29.3.46 size\_unique()

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

Definition at line 1105 of file model-meat.hpp.

#### 8.29.3.47 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 356 of file model-meat.hpp.

#### 8.29.3.48 support\_size()

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

Definition at line 1141 of file model-meat.hpp.

### 8.29.3.49 transform\_model()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector<double> Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::transform_model (
    double * data,
    size_t k )
```

### 8.29.3.50 update\_normalizing\_constants()

```
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 >::update_normalizing_constants (
    const std::vector< double > & params ) [inline]
```

Computes the normalizing constant for a given set of parameters.

This function will compute the normalizing constant for a given set of parameters. It will also update the `normalizing_constants` member variable.

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

## 8.29.4 Member Data Documentation

### 8.29.4.1 arrays2support

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< size_t > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::arrays2support [protected]
```

Definition at line 65 of file `model-bones.hpp`.

### 8.29.4.2 counter\_fun

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

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

#### 8.29.4.3 counters

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Counters<Array_Type,Data_Counter_Type>* Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::counters [protected]
```

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

#### 8.29.4.4 delete\_counters

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↵
counters = false [protected]
```

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

#### 8.29.4.5 delete\_rengine

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↵
rengine = false [protected]
```

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

#### 8.29.4.6 delete\_rules

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_rules
= false [protected]
```

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

#### 8.29.4.7 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 Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↵
rules_dyn = false [protected]
```

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

#### 8.29.4.8 first\_calc\_done

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< bool > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type
>::first_calc_done [protected]
```

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

#### 8.29.4.9 keys2support

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
MapVec_type< double, size_t > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_↵
Rule_Dyn_Type >::keys2support [protected]
```

Map of types of arrays to support sets.

This is of the same length as the vector stats\_target.

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

#### 8.29.4.10 normalizing\_constants

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< double > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_↵
Type >::normalizing_constants [protected]
```

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

#### 8.29.4.11 params\_last

```
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> > Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::params_last [protected]
```

Vector of the previously used parameters.

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

#### 8.29.4.12 pset\_arrays

```
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< Array_Type > > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::pset_arrays [protected]
```

Arrays of the support(s)

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

#### 8.29.4.13 pset\_probs

```
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> > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::pset_probs [protected]
```

Probabilities of the support(s)

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

#### 8.29.4.14 pset\_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> > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::pset_stats [protected]
```

Statistics of the support(s)

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

#### 8.29.4.15 rengine

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

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

#### 8.29.4.16 rules

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Rules<Array_Type, Data_Rule_Type>* Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::rules [protected]
```

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

#### 8.29.4.17 rules\_dyn

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Rules<Array_Type, Data_Rule_Dyn_Type>* Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::rules_dyn [protected]
```

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

#### 8.29.4.18 stats\_support

```
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 > > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::stats_support [protected]
```

Sufficient statistics of the model (support)

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

#### 8.29.4.19 stats\_support\_n\_arrays

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< size_t > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::stats_support_n_arrays [protected]
```

Number of arrays included per support.

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



#### 8.29.4.20 stats\_target

```
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 > > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::stats_target [protected]
```

Target statistics of the model.

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

#### 8.29.4.21 support\_fun

```
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> Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::support_fun [protected]
```

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

#### 8.29.4.22 transform\_model\_fun

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::function<std::vector<double>double *, size_t k)> Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::transform_model_fun = nullptr [protected]
```

Transformation of the model.

When specified, this function will update the model by modifying the linear equation. For example, if the user wanted to add interaction terms, rescale, or apply other operations of the sorts, the user can do such through this function.

The function should return `void` and receive the following arguments:

- `data` Pointer to the first element of the set of sufficient statistics
- `k` `size_t` indicating the number of sufficient statistics

#### Returns

Nothing, but it will modify the model data.

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

### 8.29.4.23 transform\_model\_term\_names

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::string > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::transform_model_term_names [protected]
```

Definition at line 125 of file model-bones.hpp.

### 8.29.4.24 with\_pset

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

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

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

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

## 8.30 NetCounterData Class Reference

Data class used to store arbitrary size\_t or double vectors.

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

### Public Member Functions

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

### Public Attributes

- std::vector< size\_t > [indices](#)
- std::vector< double > [numbers](#)

#### 8.30.1 Detailed Description

Data class used to store arbitrary size\_t or double vectors.

Definition at line 56 of file network.hpp.

## 8.30.2 Constructor & Destructor Documentation

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

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

Definition at line 62 of file network.hpp.

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

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

Definition at line 63 of file network.hpp.

### 8.30.2.3 ~NetCounterData()

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

Definition at line 68 of file network.hpp.

## 8.30.3 Member Data Documentation

### 8.30.3.1 indices

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

Definition at line 59 of file network.hpp.

### 8.30.3.2 numbers

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

Definition at line 60 of file network.hpp.

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

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

## 8.31 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](#)

#### 8.31.1 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 19 of file [network.hpp](#).

#### 8.31.2 Constructor & Destructor Documentation

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

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

Definition at line 25 of file [network.hpp](#).

##### 8.31.2.2 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 33 of file network.hpp.

**8.31.2.3 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 45 of file network.hpp.

**8.31.2.4 ~NetworkData()**

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

Definition at line 51 of file network.hpp.

**8.31.3 Member Data Documentation****8.31.3.1 directed**

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

Definition at line 22 of file network.hpp.

### 8.31.3.2 vertex\_attr

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

Definition at line 23 of file network.hpp.

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

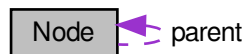
- include/barry/counters/[network.hpp](#)

## 8.32 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
- size\_t [noffspring](#) () const [noexcept](#)
- bool [is\\_leaf](#) () const [noexcept](#)

#### Construct a new Node object

- [Node](#) ()
- [Node](#) (size\_t id\_, size\_t ord\_, bool duplication\_)
- [Node](#) (size\_t id\_, size\_t ord\_, std::vector< size\_t > annotations\_, bool duplication\_)
- [Node](#) ([Node](#) &&x) [noexcept](#)
- [Node](#) (const [Node](#) &x)

## Public Attributes

- `size_t` `id`  
*Id of the node (as specified in the input)*
- `size_t` `ord`  
*Order in which the node was created.*
- `PhyloArray` `array`  
*Array of the node.*
- `std::vector< size_t >` `annotations`  
*Observed annotations (only defined for [Geese](#))*
- `bool` `duplication`
- `std::vector< PhyloArray >` `arrays` = {}  
*Arrays given all possible states.*
- `Node *` `parent` = nullptr  
*Parent node.*
- `std::vector< Node * >` `offspring` = {}  
*Offspring nodes.*
- `std::vector< size_t >` `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.*

### 8.32.1 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`.

### 8.32.2 Constructor & Destructor Documentation

#### 8.32.2.1 `Node()` [1/5]

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

Definition at line 37 of file `geese-node-bones.hpp`.

**8.32.2.2 Node()** [2/5]

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

Definition at line 57 of file geese-node-bones.hpp.

**8.32.2.3 Node()** [3/5]

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

Definition at line 63 of file geese-node-bones.hpp.

**8.32.2.4 Node()** [4/5]

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

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

**8.32.2.5 Node()** [5/5]

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

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

**8.32.2.6 ~Node()**

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

Definition at line 48 of file geese-node-bones.hpp.

**8.32.3 Member Function Documentation**



### 8.32.3.1 get\_parent()

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

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

### 8.32.3.2 is\_leaf()

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

Definition at line 112 of file geese-node-bones.hpp.

### 8.32.3.3 noffspring()

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

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

## 8.32.4 Member Data Documentation

### 8.32.4.1 annotations

```
std::vector< size_t > Node::annotations
```

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

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

### 8.32.4.2 array

```
PhyloArray Node::array
```

Array of the node.

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

### 8.32.4.3 arrays

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

Arrays given all possible states.

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

### 8.32.4.4 duplication

```
bool Node::duplication
```

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

### 8.32.4.5 id

```
size_t Node::id
```

Id of the node (as specified in the input)

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

### 8.32.4.6 narray

```
std::vector< size_t > Node::narray = {}
```

ID of the array in the model.

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

### 8.32.4.7 offspring

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

Offspring nodes.

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

#### 8.32.4.8 ord

```
size_t Node::ord
```

Order in which the node was created.

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

#### 8.32.4.9 parent

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

Parent node.

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

#### 8.32.4.10 probability

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

The probability of observing each state.

Definition at line 29 of file geese-node-bones.hpp.

#### 8.32.4.11 subtree\_prob

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

Induced subtree probabilities.

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

#### 8.32.4.12 visited

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

Definition at line 26 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](#)

## 8.33 NodeData Class Reference

Data definition for the `PhyloArray` class.

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

### Public Member Functions

- `NodeData` (const std::vector< double > &blengths\_, const std::vector< bool > &states\_, bool duplication\_  
\_=true)

### Public Attributes

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

#### 8.33.1 Detailed Description

Data definition for the `PhyloArray` class.

This holds basic information about a given node.

Definition at line 15 of file `geese-types.hpp`.

#### 8.33.2 Constructor & Destructor Documentation

##### 8.33.2.1 NodeData()

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

Definition at line 35 of file `geese-types.hpp`.

#### 8.33.3 Member Data Documentation

### 8.33.3.1 blengths

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

Branch length.

Definition at line 21 of file geese-types.hpp.

### 8.33.3.2 duplication

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

Definition at line 31 of file geese-types.hpp.

### 8.33.3.3 states

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

State of the parent node.

Definition at line 26 of file geese-types.hpp.

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

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

## 8.34 PhyloCounterData Class Reference

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

### Public Member Functions

- [PhyloCounterData](#) (std::vector< size\_t > [data\\_](#), std::vector< double > \*[counters\\_](#)=nullptr)
- [PhyloCounterData](#) ()
- size\_t [at](#) (size\_t d)
- size\_t [operator\(\)](#) (size\_t d)
- size\_t [operator\[\]](#) (size\_t d)
- void [reserve](#) (size\_t x)
- void [push\\_back](#) (size\_t x)
- void [shrink\\_to\\_fit](#) ()
- size\_t [size](#) ()
- std::vector< size\_t >::iterator [begin](#) ()
- std::vector< size\_t >::iterator [end](#) ()
- bool [empty](#) ()
- std::vector< double > \* [get\\_counters](#) ()

### 8.34.1 Detailed Description

Definition at line 45 of file geese-types.hpp.

### 8.34.2 Constructor & Destructor Documentation

#### 8.34.2.1 PhyloCounterData() [1/2]

```
PhyloCounterData::PhyloCounterData (
    std::vector< size_t > data_,
    std::vector< double > * counters_ = nullptr ) [inline]
```

Definition at line 51 of file geese-types.hpp.

#### 8.34.2.2 PhyloCounterData() [2/2]

```
PhyloCounterData::PhyloCounterData ( ) [inline]
```

Definition at line 56 of file geese-types.hpp.

### 8.34.3 Member Function Documentation

#### 8.34.3.1 at()

```
size_t PhyloCounterData::at (
    size_t d ) [inline]
```

Definition at line 58 of file geese-types.hpp.

#### 8.34.3.2 begin()

```
std::vector< size_t >::iterator PhyloCounterData::begin ( ) [inline]
```

Definition at line 66 of file geese-types.hpp.

### 8.34.3.3 empty()

```
bool PhyloCounterData::empty ( ) [inline]
```

Definition at line 69 of file geese-types.hpp.

### 8.34.3.4 end()

```
std::vector< size_t >::iterator PhyloCounterData::end ( ) [inline]
```

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

### 8.34.3.5 get\_counters()

```
std::vector< double >* PhyloCounterData::get_counters ( ) [inline]
```

Definition at line 70 of file geese-types.hpp.

### 8.34.3.6 operator()()

```
size_t PhyloCounterData::operator() (
    size_t d ) [inline]
```

Definition at line 59 of file geese-types.hpp.

### 8.34.3.7 operator[]()

```
size_t PhyloCounterData::operator[] (
    size_t d ) [inline]
```

Definition at line 60 of file geese-types.hpp.

### 8.34.3.8 push\_back()

```
void PhyloCounterData::push_back (
    size_t x ) [inline]
```

Definition at line 62 of file geese-types.hpp.

### 8.34.3.9 reserve()

```
void PhyloCounterData::reserve (
    size_t x ) [inline]
```

Definition at line 61 of file geese-types.hpp.

### 8.34.3.10 shrink\_to\_fit()

```
void PhyloCounterData::shrink_to_fit ( ) [inline]
```

Definition at line 63 of file geese-types.hpp.

### 8.34.3.11 size()

```
size_t PhyloCounterData::size ( ) [inline]
```

Definition at line 64 of file geese-types.hpp.

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

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

## 8.35 PhyloRuleDynData Class Reference

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

### Public Member Functions

- [PhyloRuleDynData](#) (const std::vector< double > \*counts\_, size\_t pos\_, size\_t lb\_, size\_t ub\_, size\_t duplication\_)
- const double [operator\(\)](#) () const
- [~PhyloRuleDynData](#) ()

### Public Attributes

- const std::vector< double > \* [counts](#)
- size\_t [pos](#)
- size\_t [lb](#)
- size\_t [ub](#)
- size\_t [duplication](#)



### 8.35.1 Detailed Description

Definition at line 74 of file geese-types.hpp.

### 8.35.2 Constructor & Destructor Documentation

#### 8.35.2.1 PhyloRuleDynData()

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

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

#### 8.35.2.2 ~PhyloRuleDynData()

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

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

### 8.35.3 Member Function Documentation

#### 8.35.3.1 operator()()

```
const double PhyloRuleDynData::operator() ( ) const [inline]
```

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

### 8.35.4 Member Data Documentation

#### 8.35.4.1 counts

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

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

#### 8.35.4.2 duplication

```
size_t PhyloRuleDynData::duplication
```

Definition at line 80 of file geese-types.hpp.

#### 8.35.4.3 lb

```
size_t PhyloRuleDynData::lb
```

Definition at line 78 of file geese-types.hpp.

#### 8.35.4.4 pos

```
size_t PhyloRuleDynData::pos
```

Definition at line 77 of file geese-types.hpp.

#### 8.35.4.5 ub

```
size_t PhyloRuleDynData::ub
```

Definition at line 79 of file geese-types.hpp.

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

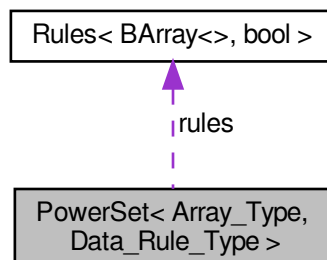
- [include/barry/models/geese/geese-types.hpp](#)

## 8.36 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](#) (size\_t N\_, size\_t M\_)

### Construct and destroy a PowerSet object

- [PowerSet](#) ()
- [PowerSet](#) (size\_t N\_, size\_t 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\\_fun\\_type](#)< Array\_Type, Data\_Rule\_Type > [count\\_fun\\_](#), Data\_Rule\_Type [data\\_](#))

### 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 size\_t &i) const

## Public Attributes

- Array\_Type [EmptyArray](#)
- std::vector< Array\_Type > [data](#)
- [Rules](#)< Array\_Type, Data\_Rule\_Type > \* [rules](#)
- size\_t [N](#)
- size\_t [M](#)
- bool [rules\\_deleted](#) = false
- std::vector< size\_t > [coordinates\\_free](#)
- std::vector< size\_t > [coordinates\\_locked](#)
- size\_t [n\\_free](#)
- size\_t [n\\_locked](#)

### 8.36.1 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 11 of file powerset-bones.hpp.

## 8.36.2 Constructor & Destructor Documentation

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

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

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

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

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

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

### 8.36.2.3 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 5 of file powerset-meat.hpp.

### 8.36.2.4 ~PowerSet()

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

Definition at line 13 of file powerset-meat.hpp.

## 8.36.3 Member Function Documentation

### 8.36.3.1 add\_rule() [1/2]

```
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 180 of file powerset-meat.hpp.

### 8.36.3.2 add\_rule() [2/2]

```
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_ ) [inline]
```

Definition at line 189 of file powerset-meat.hpp.

### 8.36.3.3 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 68 of file powerset-bones.hpp.

### 8.36.3.4 calc()

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

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

### 8.36.3.5 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 69 of file powerset-bones.hpp.

### 8.36.3.6 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 67 of file powerset-bones.hpp.

### 8.36.3.7 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 66 of file powerset-bones.hpp.

### 8.36.3.8 init\_support()

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

Definition at line 19 of file powerset-meat.hpp.

### 8.36.3.9 operator[]()

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

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

### 8.36.3.10 reset()

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

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

### 8.36.3.11 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 70 of file powerset-bones.hpp.

## 8.36.4 Member Data Documentation

### 8.36.4.1 coordinates\_free

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

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

### 8.36.4.2 coordinates\_locked

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

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

### 8.36.4.3 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 19 of file powerset-bones.hpp.

### 8.36.4.4 EmptyArray

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

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

#### 8.36.4.5 M

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

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

#### 8.36.4.6 N

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

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

#### 8.36.4.7 n\_free

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

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

#### 8.36.4.8 n\_locked

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

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

#### 8.36.4.9 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 20 of file powerset-bones.hpp.



#### 8.36.4.10 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 23 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

## 8.37 Progress Class Reference

A simple progress bar.

```
#include <progress.hpp>
```

### Public Member Functions

- [Progress](#) (int n\_, int width\_)
- [~Progress](#) ()
- void [next](#) ()
- void [end](#) ()

#### 8.37.1 Detailed Description

A simple progress bar.

Definition at line 11 of file progress.hpp.

#### 8.37.2 Constructor & Destructor Documentation

##### 8.37.2.1 Progress()

```
Progress::Progress (
    int n_,
    int width_ ) [inline]
```

Definition at line 30 of file progress.hpp.

### 8.37.2.2 ~Progress()

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

Definition at line 23 of file progress.hpp.

## 8.37.3 Member Function Documentation

### 8.37.3.1 end()

```
void Progress::end ( ) [inline]
```

Definition at line 52 of file progress.hpp.

### 8.37.3.2 next()

```
void Progress::next ( ) [inline]
```

Definition at line 41 of file progress.hpp.

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

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

## 8.38 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, size\_t i, size\_t j)
- std::string & [get\\_name](#) ()
- std::string & [get\\_description](#) ()
- std::string [get\\_name](#) () const
- std::string [get\\_description](#) () const

#### Construct a new Rule object

Construct a new [Rule](#) object

*Parameters*

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

- `Rule()`
- `Rule(Rule_fun_type< Array_Type, Data_Type > fun_, Data_Type dat_, std::string name_="", std::string desc_="")`

**8.38.1 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 <code>BArray</code> .
<i>Data_Type</i>	Any type.

Definition at line 20 of file `rules-bones.hpp`.

**8.38.2 Constructor & Destructor Documentation****8.38.2.1 Rule() [1/2]**

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

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

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

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

### 8.38.2.3 ~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.

## 8.38.3 Member Function Documentation

### 8.38.3.1 D()

```
template<typename Array_Type , typename Data_Type >
Data_Type & Rule< Array_Type, Data_Type >::D [inline]
```

Read/Write access to the data.

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

### 8.38.3.2 get\_description() [1/2]

```
template<typename Array_Type , typename Data_Type >
std::string & Rule< Array_Type, Data_Type >::get_description [inline]
```

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

### 8.38.3.3 get\_description() [2/2]

```
template<typename Array_Type , typename Data_Type >
std::string Rule< Array_Type, Data_Type >::get_description [inline]
```

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

### 8.38.3.4 get\_name() [1/2]

```
template<typename Array_Type , typename Data_Type >
std::string & Rule< Array_Type, Data_Type >::get_name [inline]
```

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

**8.38.3.5** `get_name()` [2/2]

```
template<typename Array_Type , typename Data_Type >
std::string Rule< Array_Type, Data_Type >::get_name [inline]
```

Definition at line 60 of file rules-meat.hpp.

**8.38.3.6** `operator()`

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

Definition at line 43 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

**8.39 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](#) ()
- size\_t [size](#) () const [noexcept](#)
- bool [operator\(\)](#) (const Array\_Type &a, size\_t i, size\_t j)  
*Check whether a given cell is free or locked.*
- void [get\\_seq](#) (const Array\_Type &a, std::vector< size\_t > \*free, std::vector< size\_t > \*locked=nullptr)  
*Computes the sequence of free and locked cells in an [BArray](#).*
- std::vector< std::string > [get\\_names](#) () const
- std::vector< std::string > [get\\_descriptions](#) () const
- std::vector< [Rule](#)< Array\_Type, Data\_Type > >::iterator [begin](#) ()
- std::vector< [Rule](#)< Array\_Type, Data\_Type > >::iterator [end](#) ()

**Rule adding**

*Parameters*

rule	
------	--

- void [add\\_rule](#) ([Rule](#)< Array\_Type, Data\_Type > rule)
- void [add\\_rule](#) ([Rule\\_fun\\_type](#)< Array\_Type, Data\_Type > rule\_, Data\_Type [data\\_](#), std::string [name\\_](#)="", std::string [description\\_](#)="")

**8.39.1 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 71 of file rules-bones.hpp.

**8.39.2 Constructor & Destructor Documentation****8.39.2.1 Rules() [1/2]**

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

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

**8.39.2.2 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 5 of file rules-meat.hpp.

### 8.39.2.3 ~Rules()

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

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

## 8.39.3 Member Function Documentation

### 8.39.3.1 add\_rule() [1/2]

```
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 72 of file rules-meat.hpp.

### 8.39.3.2 add\_rule() [2/2]

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

Definition at line 82 of file rules-meat.hpp.

### 8.39.3.3 begin()

```
template<typename Array_Type , typename Data_Type >
std::vector< Rule<Array_Type,Data_Type> >::iterator Rules< Array_Type, Data_Type >::begin (
) [inline]
```

Definition at line 134 of file rules-bones.hpp.

### 8.39.3.4 end()

```
template<typename Array_Type , typename Data_Type >
std::vector< Rule<Array_Type,Data_Type> >::iterator Rules< Array_Type, Data_Type >::end ( )
[inline]
```

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

### 8.39.3.5 get\_descriptions()

```
template<typename Array_Type , typename Data_Type >
std::vector< std::string > Rules< Array_Type, Data_Type >::get_descriptions [inline]
```

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

### 8.39.3.6 get\_names()

```
template<typename Array_Type , typename Data_Type >
std::vector< std::string > Rules< Array_Type, Data_Type >::get_names [inline]
```

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

### 8.39.3.7 get\_seq()

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::get_seq (
    const Array_Type & a,
    std::vector< size_t > * free,
    std::vector< size_t > * 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 117 of file rules-meat.hpp.

### 8.39.3.8 operator()()

```
template<typename Array_Type , typename Data_Type >
bool Rules< Array_Type, Data_Type >::operator() (
    const Array_Type & a,
    size_t i,
    size_t 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 101 of file rules-meat.hpp.

**8.39.3.9 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 19 of file rules-meat.hpp.

**8.39.3.10 size()**

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

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

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

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

**8.40 StatsCounter< Array\_Type, Data\_Type > Class Template Reference**

Count stats for a single Array.

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

## Public Member Functions

- [StatsCounter](#) (const Array\_Type \*Array\_)
- Creator of a [StatsCounter](#)*
- [StatsCounter](#) (const [StatsCounter](#)< Array\_Type, Data\_Type > &counter)
- Copy constructor.*
- [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 [set\\_counters](#) ([Counters](#)< Array\_Type, Data\_Type > \*counters\_)
- void [count\\_init](#) (size\_t i, size\_t 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](#) (size\_t i, size\_t j)
- std::vector< double > [count\\_all](#) ()
- [Counters](#)< Array\_Type, Data\_Type > \* [get\\_counters](#) ()
- std::vector< std::string > [get\\_names](#) () const
- std::vector< std::string > [get\\_descriptions](#) () const
- size\_t [size](#) () const

### 8.40.1 Detailed Description

```
template<typename Array_Type, typename Data_Type>
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 14 of file statscounter-bones.hpp.

### 8.40.2 Constructor & Destructor Documentation

#### 8.40.2.1 StatsCounter() [1/3]

```
template<typename Array_Type , typename Data_Type >
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 37 of file statscounter-bones.hpp.

## 8.40.2.2 StatsCounter() [2/3]

```
template<typename Array_Type , typename Data_Type >
StatsCounter< Array_Type, Data_Type >::StatsCounter (
    const StatsCounter< Array_Type, Data_Type > & counter )
```

Copy constructor.

## Parameters

<i>counter</i>	
----------------	--

## 8.40.2.3 StatsCounter() [3/3]

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

Can be created without setting the array.

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

## 8.40.2.4 ~StatsCounter()

```
template<typename Array_Type , typename Data_Type >
StatsCounter< Array_Type, Data_Type >::~StatsCounter ( )
```

## 8.40.3 Member Function Documentation

## 8.40.3.1 add\_counter()

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

### 8.40.3.2 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.

### 8.40.3.3 count\_current()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::count_current (
    size_t i,
    size_t j )
```

### 8.40.3.4 count\_init()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::count_init (
    size_t i,
    size_t 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.

### 8.40.3.5 get\_counters()

```
template<typename Array_Type , typename Data_Type >
Counters<Array_Type,Data_Type>* StatsCounter< Array_Type, Data_Type >::get_counters ( )
```

### 8.40.3.6 get\_descriptions()

```
template<typename Array_Type , typename Data_Type >
std::vector< std::string > StatsCounter< Array_Type, Data_Type >::get_descriptions ( ) const
```

### 8.40.3.7 get\_names()

```
template<typename Array_Type , typename Data_Type >
std::vector< std::string > StatsCounter< Array_Type, Data_Type >::get_names ( ) const
```

### 8.40.3.8 reset\_array()

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

Changes the reference array for the counting.

#### Parameters

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

#### 8.40.3.9 set\_counters()

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

#### 8.40.3.10 size()

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

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

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

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

## 8.41 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](#) (size\_t N\_, size\_t M\_)  
*Constructor specifying the dimensions of the array (empty).*
- [Support](#) ()
- [~Support](#) ()
- void [init\\_support](#) (std::vector< Array\_Type > \*array\_bank=nullptr, std::vector< double > \*stats\_↵  
bank=nullptr)
- void [calc](#) (std::vector< Array\_Type > \*array\_bank=nullptr, std::vector< double > \*stats\_bank=nullptr, size\_↵  
\_t max\_num\_elements\_=0u)  
*Computes the entire support.*

- `std::vector< double > get_counts () const`
- `std::vector< double > * get_current_stats ()`  
*List current statistics.*
- `void print () const`
- `const FreqTable< double > & 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 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_)`
- `bool eval_rules_dyn (const std::vector< double > &counts, const size_t &i, const size_t &j)`

## Public Attributes

- `size_t N`
- `size_t M`
- `bool delete_counters = true`
- `bool delete_rules = true`
- `bool delete_rules_dyn = true`
- `size_t max_num_elements = BARRY_MAX_NUM_ELEMENTS`
- `std::vector< double > current_stats`
- `std::vector< size_t > coordinates_free`
- `std::vector< size_t > coordinates_locked`
- `size_t coordiantes_n_free`
- `size_t coordiantes_n_locked`
- `std::vector< double > change_stats`
- `std::vector< size_t > hashes`
- `std::vector< bool > hashes_initialized`
- `size_t n_counters`

### 8.41.1 Detailed Description

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

Compute the support of sufficient statistics.

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

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

In the case of `rule_dyn`, the function will stablsh 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 42 of file support-bones.hpp.

### 8.41.2 Constructor & Destructor Documentation

#### 8.41.2.1 Support() [1/3]

```
template<typename Array_Type = BArray<bool, bool>, 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 89 of file support-bones.hpp.

### 8.41.2.2 Support() [2/3]

```
template<typename Array_Type = BArray<bool, bool>, 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 (
    size_t N_,
    size_t M_ ) [inline]
```

Constructor specifying the dimensions of the array (empty).

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

### 8.41.2.3 Support() [3/3]

```
template<typename Array_Type = BArray<bool, bool>, 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 105 of file support-bones.hpp.

### 8.41.2.4 ~Support()

```
template<typename Array_Type = BArray<bool, bool>, 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 112 of file support-bones.hpp.

## 8.41.3 Member Function Documentation

### 8.41.3.1 add\_counter()

```
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 416 of file support-meat.hpp.



### 8.41.3.2 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 443 of file support-meat.hpp.

### 8.41.3.3 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 453 of file support-meat.hpp.

### 8.41.3.4 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 478 of file support-meat.hpp.

### 8.41.3.5 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 488 of file support-meat.hpp.

### 8.41.3.6 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< double > * stats_bank = nullptr,
    size_t 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 382 of file support-meat.hpp.

#### 8.41.3.7 eval\_rules\_dyn()

```
template<typename Array_Type , typename Data_Counter_Type , typename Data_Rule_Type , typename
Data_Rule_Dyn_Type >
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::eval_←
rules_dyn (
    const std::vector< double > & counts,
    const size_t & i,
    const size_t & j ) [inline]
```

Definition at line 513 of file support-meat.hpp.

#### 8.41.3.8 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 counter functions.

Definition at line 593 of file support-meat.hpp.

#### 8.41.3.9 get\_counts()

```
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_counts [inline]
```

Definition at line 557 of file support-meat.hpp.

#### 8.41.3.10 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 571 of file support-meat.hpp.

#### 8.41.3.11 get\_data()

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

Definition at line 588 of file support-meat.hpp.

#### 8.41.3.12 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 598 of file support-meat.hpp.

#### 8.41.3.13 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_Dyn_Type, Data_Rule_Dyn_Type >::get_rules_dyn [inline]
```

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

Definition at line 603 of file support-meat.hpp.

#### 8.41.3.14 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< double > * stats_bank = nullptr ) [inline]
```

Definition at line 5 of file support-meat.hpp.

#### 8.41.3.15 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 576 of file support-meat.hpp.

#### 8.41.3.16 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 111 of file support-meat.hpp.

#### 8.41.3.17 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 118 of file support-meat.hpp.

### 8.41.3.18 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 426 of file support-meat.hpp.

### 8.41.3.19 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 463 of file support-meat.hpp.

### 8.41.3.20 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 498 of file support-meat.hpp.

## 8.41.4 Member Data Documentation

### 8.41.4.1 change\_stats

```
template<typename Array_Type = BArray<bool, bool>, 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 >::change_stats
```

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

#### 8.41.4.2 coordiantes\_n\_free

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::coordiantes←
_n_free
```

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

#### 8.41.4.3 coordiantes\_n\_locked

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::coordiantes←
_n_locked
```

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

#### 8.41.4.4 coordinates\_free

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< size_t > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn←
_Type >::coordinates_free
```

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

#### 8.41.4.5 coordinates\_locked

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< size_t > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn←
_Type >::coordinates_locked
```

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

#### 8.41.4.6 current\_stats

```
template<typename Array_Type = BArray<bool, bool>, 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 77 of file support-bones.hpp.

#### **8.41.4.7 delete\_counters**

```
template<typename Array_Type = BArray<bool, bool>, 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 71 of file support-bones.hpp.

#### **8.41.4.8 delete\_rules**

```
template<typename Array_Type = BArray<bool, bool>, 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 72 of file support-bones.hpp.

#### **8.41.4.9 delete\_rules\_dyn**

```
template<typename Array_Type = BArray<bool, bool>, 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 73 of file support-bones.hpp.

#### **8.41.4.10 hashes**

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< size_t > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_↵
_Type >::hashes
```

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

#### **8.41.4.11 hashes\_initialized**

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

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

**8.41.4.12 M**

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::M
```

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

**8.41.4.13 max\_num\_elements**

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::max_num←
_elements = BARRY\_MAX\_NUM\_ELEMENTS
```

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

**8.41.4.14 N**

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

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

**8.41.4.15 n\_counters**

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::n←
counters
```

Definition at line 85 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](#)

**8.42 vecHasher< T > Struct Template Reference**

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



## Public Member Functions

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

### 8.42.1 Detailed Description

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

Definition at line 105 of file typedefs.hpp.

### 8.42.2 Member Function Documentation

#### 8.42.2.1 operator()()

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

Definition at line 108 of file typedefs.hpp.

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

- `include/barry/typedefs.hpp`

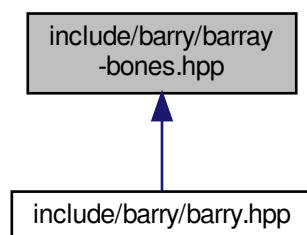


## Chapter 9

# File Documentation

### 9.1 `include/barry/barray-bones.hpp` File Reference

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



#### Classes

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

### 9.2 `include/barry/barray-iterator.hpp` File Reference

#### Classes

- class `ConstBArrayRowIter< Cell_Type, Data_Type >`

## 9.3 include/barry/barray-meat-operators.hpp File Reference

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



### Macros

- `#define BARRAY_TYPE() BArray<Cell_Type, Data_Type>`
- `#define BARRAY_TEMPLATE_ARGS() <typename Cell_Type, typename Data_Type>`
- `#define BARRAY_TEMPLATE(a, b) template BARRAY_TEMPLATE_ARGS() inline a BARRAY_TYPE()::b`
- `#define ROW(a) this->el_ij[a]`
- `#define COL(a) this->el_ji[a]`

### Functions

- `template BARRAY_TEMPLATE_ARGS () inline void checkdim_(const BARRAY_TYPE() &lhs`
- `template const BARRAY_TYPE () &rhs)`
- `BARRAY_TEMPLATE (BARRAY_TYPE()&, operator+=)(const BArray< Cell_Type`
- `for (size_t i=0u;i< nrow();++i) for(size_t j=0u`
- `j< ncol();++j) this-> operator() (i, j)+`
- `BARRAY_TEMPLATE (BARRAY_TYPE()&, operator+=)(const Cell_Type &rhs)`
- `BARRAY_TEMPLATE (BARRAY_TYPE()&, operator-=(const BArray< Cell_Type`
- `BARRAY_TEMPLATE (BARRAY_TYPE()&, operator-=(const Cell_Type &rhs)`
- `BARRAY_TEMPLATE (BARRAY_TYPE()&, operator*=(const Cell_Type &rhs)`
- `BARRAY_TEMPLATE (BARRAY_TYPE()&, operator/=(const Cell_Type &rhs)`

### Variables

- `Data_Type & rhs`
- `return * this`

#### 9.3.1 Macro Definition Documentation

### 9.3.1.1 BARRAY\_TEMPLATE

```
#define BARRAY_TEMPLATE(  
    a,  
    b )  template BARRAY_TEMPLATE_ARGS() inline a BARRAY_TYPE()::b
```

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

### 9.3.1.2 BARRAY\_TEMPLATE\_ARGS

```
#define BARRAY_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

Definition at line 9 of file barray-meat-operators.hpp.

### 9.3.1.3 BARRAY\_TYPE

```
#define BARRAY_TYPE( ) BArray<Cell_Type, Data_Type>
```

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

### 9.3.1.4 COL

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

Definition at line 15 of file barray-meat-operators.hpp.

### 9.3.1.5 ROW

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

Definition at line 14 of file barray-meat-operators.hpp.

## 9.3.2 Function Documentation

### 9.3.2.1 BARRAY\_TEMPLATE() [1/6]

```
BARRAY_TEMPLATE (
    BARRAY_TYPE() & ,
    operator* ) const &
```

Definition at line 88 of file barray-meat-operators.hpp.

### 9.3.2.2 BARRAY\_TEMPLATE() [2/6]

```
BARRAY_TEMPLATE (
    BARRAY_TYPE() & ,
    operator+ ) const
```

### 9.3.2.3 BARRAY\_TEMPLATE() [3/6]

```
BARRAY_TEMPLATE (
    BARRAY_TYPE() & ,
    operator+ ) const &
```

Definition at line 46 of file barray-meat-operators.hpp.

### 9.3.2.4 BARRAY\_TEMPLATE() [4/6]

```
BARRAY_TEMPLATE (
    BARRAY_TYPE() & ,
    operator- ) const
```

### 9.3.2.5 BARRAY\_TEMPLATE() [5/6]

```
BARRAY_TEMPLATE (
    BARRAY_TYPE() & ,
    operator- ) const &
```

Definition at line 75 of file barray-meat-operators.hpp.

### 9.3.2.6 BARRAY\_TEMPLATE() [6/6]

```
BARRAY_TEMPLATE (
    BARRAY_TYPE() & ,
    operator/ ) const &
```

Definition at line 105 of file barray-meat-operators.hpp.

### 9.3.2.7 BARRAY\_TEMPLATE\_ARGS()

```
template BARRAY_TEMPLATE_ARGS ( ) const &
```

### 9.3.2.8 BARRAY\_TYPE()

```
template const BARRAY_TYPE ( ) &
```

Definition at line 20 of file barray-meat-operators.hpp.

### 9.3.2.9 for()

```
for ( ) [pure virtual]
```

Definition at line 66 of file barray-meat-operators.hpp.

### 9.3.2.10 operator>()

```
j< ncol(); ++j) this-> operator() (
    i ,
    j )
```

## 9.3.3 Variable Documentation

### 9.3.3.1 rhs

Data\_Type & rhs

#### Initial value:

```
{  
    checkdim_(*this, rhs)
```

Definition at line 33 of file barray-meat-operators.hpp.

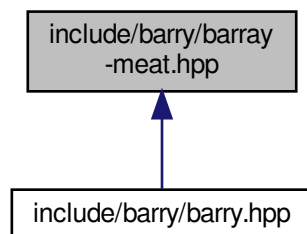
### 9.3.3.2 this

```
return * this
```

Definition at line 43 of file barray-meat-operators.hpp.

## 9.4 include/barry/barray-meat.hpp File Reference

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]

### 9.4.1 Macro Definition Documentation



#### 9.4.1.1 COL

```
#define COL(  
    a ) this->el\_ji[a]
```

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

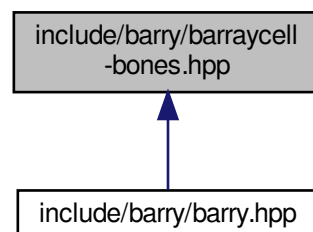
#### 9.4.1.2 ROW

```
#define ROW(  
    a ) this->el\_ij[a]
```

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

## 9.5 include/barry/barraycell-bones.hpp File Reference

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

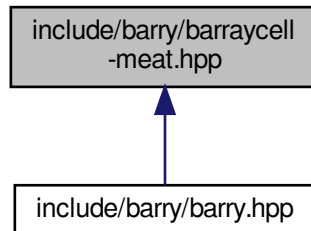


### Classes

- class [BArrayCell](#)< [Cell\\_Type](#), [Data\\_Type](#) >
- class [BArrayCell\\_const](#)< [Cell\\_Type](#), [Data\\_Type](#) >

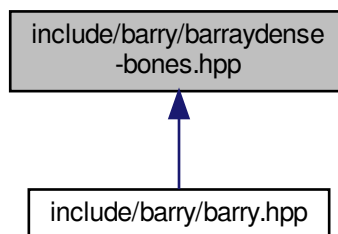
## 9.6 include/barry/barraycell-meat.hpp File Reference

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



## 9.7 include/barry/barraydense-bones.hpp File Reference

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

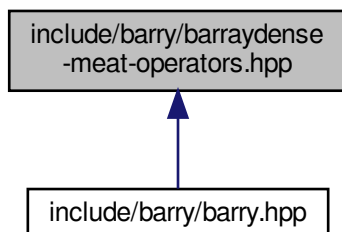


### Classes

- class [BArrayDense< Cell\\_Type, Data\\_Type >](#)  
*Baseline class for binary arrays.*

## 9.8 include/barry/barraydense-meat-operators.hpp File Reference

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



### Macros

- `#define BDENSE_TYPE() BArrayDense<Cell_Type, Data_Type>`
- `#define BDENSE_TEMPLATE_ARGS() <typename Cell_Type, typename Data_Type>`
- `#define BDENSE_TEMPLATE(a, b) template BDENSE_TEMPLATE_ARGS() inline a BDENSE_TYPE():b`
- `#define ROW(a) this->el_ij[a]`
- `#define COL(a) this->el_ji[a]`
- `#define POS(a, b) (b)*N + (a)`
- `#define POS_N(a, b, c) (b)*(c) + (a)`

### Functions

- `template BDENSE_TEMPLATE_ARGS () inline void checkdim_(const BDENSE_TYPE() &lhs`
- `template const BDENSE_TYPE () &rhs)`
- `BDENSE_TEMPLATE (BDENSE_TYPE()&, operator+=)(const BDENSE_TYPE() &rhs)`
- `BDENSE_TEMPLATE (BDENSE_TYPE()&, operator-=)(const BDENSE_TYPE() &rhs)`
- `BDENSE_TEMPLATE (BDENSE_TYPE()&, operator*=)(const Cell_Type &rhs)`
- `BDENSE_TEMPLATE (BDENSE_TYPE()&, operator/=)(const Cell_Type &rhs)`

### 9.8.1 Macro Definition Documentation

#### 9.8.1.1 BDENSE\_TEMPLATE

```

#define BDENSE_TEMPLATE(
    a,
    b ) template BDENSE_TEMPLATE_ARGS() inline a BDENSE_TYPE():b
  
```

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

### 9.8.1.2 BDENSE\_TEMPLATE\_ARGS

```
#define BDENSE_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

Definition at line 9 of file `barraydense-meat-operators.hpp`.

### 9.8.1.3 BDENSE\_TYPE

```
#define BDENSE_TYPE( ) BArrayDense<Cell_Type, Data_Type>
```

Definition at line 7 of file `barraydense-meat-operators.hpp`.

### 9.8.1.4 COL

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

Definition at line 15 of file `barraydense-meat-operators.hpp`.

### 9.8.1.5 POS

```
#define POS(  
    a,  
    b ) (b)*N + (a)
```

Definition at line 16 of file `barraydense-meat-operators.hpp`.

### 9.8.1.6 POS\_N

```
#define POS_N(  
    a,  
    b,  
    c ) (b)*(c) + (a)
```

Definition at line 17 of file `barraydense-meat-operators.hpp`.

### 9.8.1.7 ROW

```
#define ROW(  
    a ) this->el\_ij[a]
```

Definition at line 14 of file barraydense-meat-operators.hpp.

## 9.8.2 Function Documentation

### 9.8.2.1 BDENSE\_TEMPLATE() [1/4]

```
BDENSE_TEMPLATE (  
    BDENSE\_TYPE() & ,  
    operator* ) const &
```

Definition at line 90 of file barraydense-meat-operators.hpp.

### 9.8.2.2 BDENSE\_TEMPLATE() [2/4]

```
BDENSE_TEMPLATE (  
    BDENSE\_TYPE() & ,  
    operator+ ) const &
```

Definition at line 34 of file barraydense-meat-operators.hpp.

### 9.8.2.3 BDENSE\_TEMPLATE() [3/4]

```
BDENSE_TEMPLATE (  
    BDENSE\_TYPE() & ,  
    operator- ) const &
```

Definition at line 61 of file barraydense-meat-operators.hpp.

### 9.8.2.4 BDENSE\_TEMPLATE() [4/4]

```
BDENSE_TEMPLATE (  
    BDENSE\_TYPE() & ,  
    operator/ ) const &
```

Definition at line 101 of file barraydense-meat-operators.hpp.

### 9.8.2.5 BDENSE\_TEMPLATE\_ARGS()

```
template BDENSE_TEMPLATE_ARGS ( ) const &
```

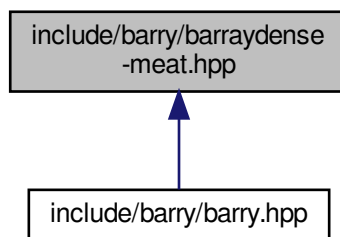
### 9.8.2.6 BDENSE\_TYPE()

```
template const BDENSE_TYPE ( ) &
```

Definition at line 22 of file `barrydense-meat-operators.hpp`.

## 9.9 include/barry/barrydense-meat.hpp File Reference

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]`
- `#define POS(a, b) (b)*N + (a)`
- `#define POS_N(a, b, c) (b)*(c) + (a)`
- `#define ZERO_CELL static_cast<Cell_Type>(0.0)`

### 9.9.1 Macro Definition Documentation

#### 9.9.1.1 COL

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

Definition at line 24 of file `barrydense-meat.hpp`.

### 9.9.1.2 POS

```
#define POS(  
    a,  
    b ) (b)*N + (a)
```

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

### 9.9.1.3 POS\_N

```
#define POS_N(  
    a,  
    b,  
    c ) (b)*(c) + (a)
```

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

### 9.9.1.4 ROW

```
#define ROW(  
    a ) this->el\_ij[a]
```

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

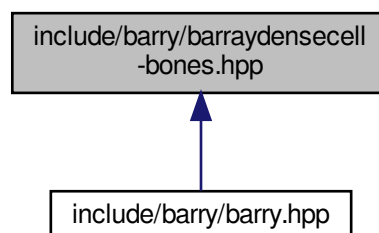
### 9.9.1.5 ZERO\_CELL

```
#define ZERO_CELL static_cast<Cell_Type>(0.0)
```

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

## 9.10 include/barry/barraydensecell-bones.hpp File Reference

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



## Classes

- class `BArrayDenseCell< Cell_Type, Data_Type >`

## Macros

- `#define POS(a, b) (a) + (b) * N`

### 9.10.1 Macro Definition Documentation

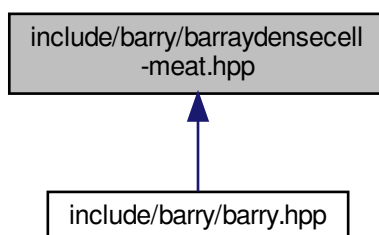
#### 9.10.1.1 POS

```
#define POS(  
    a,  
    b ) (a) + (b) * N
```

Definition at line 6 of file `barraydensecell-bones.hpp`.

## 9.11 `include/barry/barraydensecell-meat.hpp` File Reference

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



## Macros

- `#define POS(a, b) (a) + (b) * dat->N`

### 9.11.1 Macro Definition Documentation



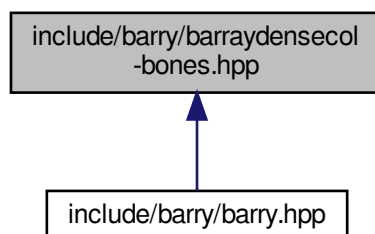
### 9.11.1.1 POS

```
#define POS(  
    a,  
    b ) (a) + (b) * dat->N
```

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

## 9.12 include/barry/barraydensecol-bones.hpp File Reference

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



### Classes

- class [BArrayDenseCol< Cell\\_Type, Data\\_Type >](#)
- class [BArrayDenseCol\\_const< Cell\\_Type, Data\\_Type >](#)

### Macros

- #define [POS\(a, b\)](#) (b)\*N + (a)
- #define [POS\\_N\(a, b, c\)](#) (b)\*(c) + (a)
- #define [ZERO\\_CELL](#) static\_cast<Cell\_Type>(0.0)

### 9.12.1 Macro Definition Documentation

#### 9.12.1.1 POS

```
#define POS(  
    a,  
    b ) (b)*N + (a)
```

Definition at line 4 of file barraydensecol-bones.hpp.

### 9.12.1.2 POS\_N

```
#define POS_N(  
    a,  
    b,  
    c ) (b)*(c) + (a)
```

Definition at line 5 of file `barraydensecol-bones.hpp`.

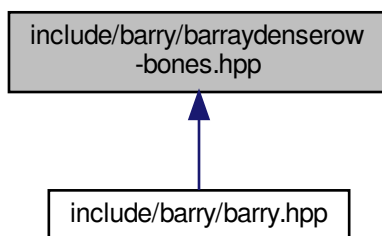
### 9.12.1.3 ZERO\_CELL

```
#define ZERO_CELL static_cast<Cell_Type>(0.0)
```

Definition at line 6 of file `barraydensecol-bones.hpp`.

## 9.13 `include/barry/barraydenserow-bones.hpp` File Reference

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



## Classes

- class `BArrayDenseRow< Cell_Type, Data_Type >`
- class `BArrayDenseRow_const< Cell_Type, Data_Type >`

## Macros

- `#define POS(a, b) (b) * N + (a)`
- `#define POS_N(a, b, c) (b)*(c) + (a)`
- `#define ZERO_CELL static_cast< Cell_Type >(0.0)`

### 9.13.1 Macro Definition Documentation

#### 9.13.1.1 POS

```
#define POS(  
    a,  
    b ) (b) * N + (a)
```

Definition at line 4 of file barraydenserow-bones.hpp.

#### 9.13.1.2 POS\_N

```
#define POS_N(  
    a,  
    b,  
    c ) (b)*(c) + (a)
```

Definition at line 5 of file barraydenserow-bones.hpp.

#### 9.13.1.3 ZERO\_CELL

```
#define ZERO_CELL static_cast< Cell_Type >(0.0)
```

Definition at line 6 of file barraydenserow-bones.hpp.

## 9.14 include/barry/barrayrow-bones.hpp File Reference

### Classes

- class [BArrayRow< Cell\\_Type, Data\\_Type >](#)
- class [BArrayRow\\_const< Cell\\_Type, Data\\_Type >](#)

## 9.15 include/barry/barrayrow-meat.hpp File Reference

### Macros

- #define [BROW\\_TYPE\(\)](#) [BArrayRow<Cell\\_Type, Data\\_Type>](#)
- #define [BROW\\_TEMPLATE\\_ARGS\(\)](#) <typename Cell\_Type, typename Data\_Type>
- #define [BROW\\_TEMPLATE\(a, b\)](#) template [BROW\\_TEMPLATE\\_ARGS\(\)](#) inline a [BROW\\_TYPE\(\)](#)::b

## Functions

- [BROW\\_TEMPLATE](#) (void, operator=)(const [BROW\\_TYPE](#)() &val)
- [BROW\\_TEMPLATE](#) (void, operator+=)(const [BROW\\_TYPE](#)() &val)
- [BROW\\_TEMPLATE](#) (void, operator-=)(const [BROW\\_TYPE](#)() &val)
- [BROW\\_TEMPLATE](#) (void, operator\*=)(const [BROW\\_TYPE](#)() &val)
- [BROW\\_TEMPLATE](#) (void, operator/=(const [BROW\\_TYPE](#)() &val)

### 9.15.1 Macro Definition Documentation

#### 9.15.1.1 BROW\_TEMPLATE

```
#define BROW_TEMPLATE(  
    a,  
    b )  template BROW\_TEMPLATE\_ARGS() inline a BROW\_TYPE()::b
```

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

#### 9.15.1.2 BROW\_TEMPLATE\_ARGS

```
#define BROW_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

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

#### 9.15.1.3 BROW\_TYPE

```
#define BROW_TYPE( ) BArrayRow<Cell_Type, Data_Type>
```

Definition at line 4 of file barrayrow-meat.hpp.

### 9.15.2 Function Documentation

#### 9.15.2.1 BROW\_TEMPLATE() [1/5]

```
BROW_TEMPLATE (  
    void ,  
    operator* ) const &
```

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

**9.15.2.2 BROW\_TEMPLATE()** [2/5]

```
BROW_TEMPLATE (
    void ,
    operator+ ) const &
```

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

**9.15.2.3 BROW\_TEMPLATE()** [3/5]

```
BROW_TEMPLATE (
    void ,
    operator- ) const &
```

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

**9.15.2.4 BROW\_TEMPLATE()** [4/5]

```
BROW_TEMPLATE (
    void ,
    operator/ ) const &
```

Definition at line 55 of file barrayrow-meat.hpp.

**9.15.2.5 BROW\_TEMPLATE()** [5/5]

```
BROW_TEMPLATE (
    void ,
    operator ) const &
```

Definition at line 11 of file barrayrow-meat.hpp.

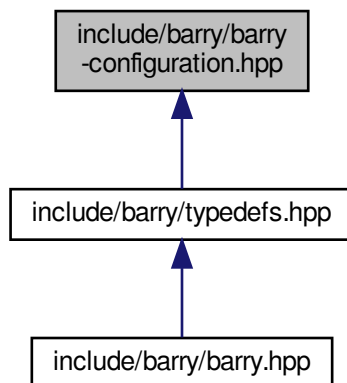
**9.16 include/barry/barrayvector-bones.hpp File Reference****Classes**

- class [BArrayVector< Cell\\_Type, Data\\_Type >](#)  
*Row or column of a [BArray](#)*
- class [BArrayVector\\_const< Cell\\_Type, Data\\_Type >](#)

## 9.17 include/barry/barrayvector-meat.hpp File Reference

## 9.18 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`.
  - `BARRY_DEBUG_LEVEL`, when defined, will make things verbose.
- 
- `#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< size_t >(std::numeric_limits< size_t >::max() / 2u)`
  - `template<typename Ta, typename Tb >`  
`using Map = std::map< Ta, Tb >`

## 9.18.1 Macro Definition Documentation

### 9.18.1.1 BARRY\_CHECK\_SUPPORT

```
#define BARRY_CHECK_SUPPORT(  
    x,  
    maxs )
```

Definition at line 47 of file barry-configuration.hpp.

### 9.18.1.2 BARRY\_ISFINITE

```
#define BARRY_ISFINITE(  
    a )
```

Definition at line 40 of file barry-configuration.hpp.

### 9.18.1.3 BARRY\_MAX\_NUM\_ELEMENTS

```
#define BARRY_MAX_NUM_ELEMENTS static_cast< size_t >(std::numeric_limits< size_t >::max()  
/2u)
```

Definition at line 55 of file barry-configuration.hpp.

### 9.18.1.4 BARRY\_SAFE\_EXP

```
#define BARRY_SAFE_EXP -100.0
```

Definition at line 33 of file barry-configuration.hpp.

### 9.18.1.5 printf\_barry

```
#define printf_barry printf
```

Definition at line 51 of file barry-configuration.hpp.

## 9.18.2 Typedef Documentation

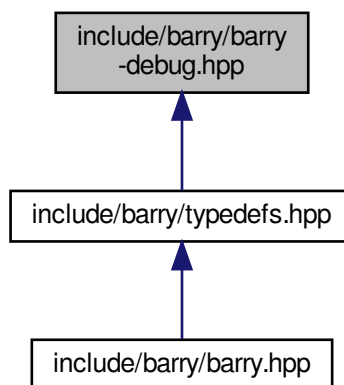
### 9.18.2.1 Map

```
template<typename Ta , typename Tb >  
using Map = std::map<Ta,Tb>
```

Definition at line 27 of file barry-configuration.hpp.

## 9.19 include/barry/barry-debug.hpp File Reference

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



### Macros

- #define BARRY\_DEBUG\_LEVEL 0

### 9.19.1 Macro Definition Documentation

#### 9.19.1.1 BARRY\_DEBUG\_LEVEL

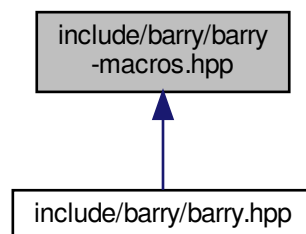
```
#define BARRY_DEBUG_LEVEL 0
```

Definition at line 5 of file barry-debug.hpp.



## 9.20 include/barry/barry-macros.hpp File Reference

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



### Macros

- `#define BARRY_ZERO Cell<Cell_Type>(0.0)`
- `#define BARRY_ZERO_DENSE static_cast<Cell_Type>(0.0)`
- `#define BARRY_ONE Cell<Cell_Type>(1.0)`
- `#define BARRY_ONE_DENSE static_cast<Cell_Type>(1.0)`
- `#define BARRY_UNUSED(expr) do { (void)(expr); } while (0);`
- `#define BARRY_NCORES_ARG(default) size_t ncores default`

### 9.20.1 Macro Definition Documentation

#### 9.20.1.1 BARRY\_NCORES\_ARG

```
#define BARRY_NCORES_ARG(  
    default ) size_t ncores default
```

Definition at line 15 of file barry-macros.hpp.

#### 9.20.1.2 BARRY\_ONE

```
#define BARRY_ONE Cell<Cell_Type>(1.0)
```

Definition at line 7 of file barry-macros.hpp.

### 9.20.1.3 BARRY\_ONE\_DENSE

```
#define BARRY_ONE_DENSE static_cast<Cell_Type>(1.0)
```

Definition at line 8 of file barry-macros.hpp.

### 9.20.1.4 BARRY\_UNUSED

```
#define BARRY_UNUSED(  
    expr ) do { (void)(expr); } while (0);
```

Definition at line 10 of file barry-macros.hpp.

### 9.20.1.5 BARRY\_ZERO

```
#define BARRY_ZERO Cell<Cell_Type>(0.0)
```

Definition at line 4 of file barry-macros.hpp.

### 9.20.1.6 BARRY\_ZERO\_DENSE

```
#define BARRY_ZERO_DENSE static_cast<Cell_Type>(0.0)
```

Definition at line 5 of file barry-macros.hpp.

## 9.21 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 <float>  
#include <string>  
#include <cstdint>  
#include <memory>  
#include <regex>
```

```

#include <iterator>
#include "typedefs.hpp"
#include "barry-macros.hpp"
#include "freqtable.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 "barraydense-bones.hpp"
#include "barraydensecell-bones.hpp"
#include "barraydenserow-bones.hpp"
#include "barraydensecol-bones.hpp"
#include "barraydense-meat.hpp"
#include "barraydensecell-meat.hpp"
#include "barraydense-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 dependency graph for barry.hpp:



## Namespaces

- [barry](#)  
*barry: Your go-to motif accountant*
- [barry::counters](#)  
*Tree class and Treeliterator class.*
- [barry::counters::network](#)

## Macros

- `#define BARRY_HPP`
- `#define BARRY_VERSION_MAYOR 0`
- `#define BARRY_VERSION_MINOR 1`
- `#define BARRY_VERSION BARRY_VERSION_MAYOR ## . ## BARRY_VERSION_MINOR`
- `#define COUNTER_FUNCTION(a)`
- `#define COUNTER_LAMBDA(a)`
- `#define RULE_FUNCTION(a)`
- `#define RULE_LAMBDA(a)`

## 9.21.1 Macro Definition Documentation

### 9.21.1.1 BARRY\_HPP

```
#define BARRY_HPP
```

Definition at line 29 of file barry.hpp.

### 9.21.1.2 BARRY\_VERSION

```
#define BARRY_VERSION BARRY_VERSION_MAYOR ## . ## BARRY_VERSION_MINOR
```

Definition at line 33 of file barry.hpp.

### 9.21.1.3 BARRY\_VERSION\_MAYOR

```
#define BARRY_VERSION_MAYOR 0
```

Definition at line 31 of file barry.hpp.

### 9.21.1.4 BARRY\_VERSION\_MINOR

```
#define BARRY_VERSION_MINOR 1
```

Definition at line 32 of file barry.hpp.

### 9.21.1.5 COUNTER\_FUNCTION

```
#define COUNTER_FUNCTION(  
    a )
```

**Value:**

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \  
inline double (a) (const Array_Type & Array, size_t i, size_t j, Data_Type & data) \  
{
```

Definition at line 92 of file barry.hpp.

### 9.21.1.6 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, size_t i, size_t j, Data_Type & data)
```

Definition at line 95 of file barry.hpp.

### 9.21.1.7 RULE\_FUNCTION

```
#define RULE_FUNCTION(  
    a )
```

**Value:**

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \  
inline bool (a) (const Array_Type & Array, size_t i, size_t j, Data_Type & data) \  
{
```

Definition at line 99 of file barry.hpp.

### 9.21.1.8 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, size_t i, size_t j, Data_Type & data)
```

Definition at line 102 of file barry.hpp.

## 9.22 include/barry/cell-bones.hpp File Reference

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

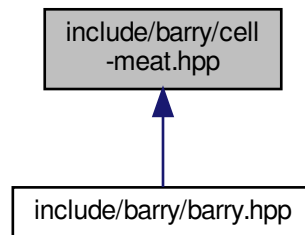


## Classes

- class [Cell< Cell\\_Type >](#)  
*Entries in [BArray](#). For now, it only has two members:*

### 9.23 include/barry/cell-meat.hpp File Reference

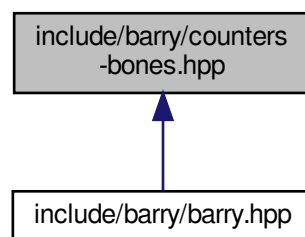
This graph shows which files directly or indirectly include this file:



### 9.24 include/barry/col-bones.hpp File Reference

### 9.25 include/barry/counters-bones.hpp File Reference

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.*

## 9.26 include/barry/counters-meat.hpp File Reference

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



### Macros

- `#define COUNTER_TYPE() Counter<Array_Type,Data_Type>`
- `#define COUNTER_TEMPLATE_ARGS() <typename Array_Type, typename Data_Type>`
- `#define COUNTER_TEMPLATE(a, b) template COUNTER_TEMPLATE_ARGS() inline a COUNTER_TYPE()↔`  
::b
- `#define TMP_HASHER_CALL Hasher_fun_type<Array_Type,Data_Type>`
- `#define COUNTERS_TYPE() Counters<Array_Type,Data_Type>`
- `#define COUNTERS_TEMPLATE_ARGS() <typename Array_Type, typename Data_Type>`
- `#define COUNTERS_TEMPLATE(a, b) template COUNTERS_TEMPLATE_ARGS() inline a COUNTERS_TYPE()↔`  
::b

### Functions

- `COUNTER_TEMPLATE (, Counter)(const Counter< Array_Type`
- `Data_Type init_fun (counter_.init_fun)`
- `Data_Type hasher_fun (counter_.hasher_fun)`
- `Data_Type &&counter_.init_fun (std::move(counter_.init_fun))`
- `Data_Type &&counter_.hasher_fun (std::move(counter_.hasher_fun))`
- `Data_Type &&counter_.data (std::move(counter_.data))`
- `Data_Type &&counter_.name (std::move(counter_.name))`
- `Data_Type &&counter_.desc (std::move(counter_.desc))`
- *Move constructor.*
- `COUNTER_TEMPLATE (COUNTER_TYPE(), operator=)(const Counter< Array_Type`
- `COUNTER_TEMPLATE (COUNTER_TYPE() &, operator=)(Counter< Array_Type`
- `COUNTER_TEMPLATE (double, count)(Array_Type &Array`
- *< Move assignment*
- `return count_fun (Array, i, j, data)`
- `COUNTER_TEMPLATE (double, init)(Array_Type &Array`
- `return init_fun (Array, i, j, data)`
- `COUNTER_TEMPLATE (std::string, get_name)() const`
- `COUNTER_TEMPLATE (std::string, get_description)() const`
- `COUNTER_TEMPLATE (void, set_hasher)(Hasher_fun_type< Array_Type`

- [COUNTER\\_TEMPLATE](#) ([TMP\\_HASHER\\_CALL](#), [get\\_hasher](#)())
- [COUNTERS\\_TEMPLATE](#) ([, Counters](#)())
- [COUNTERS\\_TEMPLATE](#) ([COUNTER\\_TYPE](#)() &, [operator\[\]](#))([size\\_t idx](#))
- [Data\\_Type hasher](#) ([counter\\_.hasher](#))
- [Data\\_Type &&counters\\_ hasher](#) ([std::move\(counters\\_.hasher\)](#))
- [COUNTERS\\_TEMPLATE](#) ([COUNTERS\\_TYPE](#)(), [operator=](#))(const [Counters](#)< [Array\\_Type](#)
- [COUNTERS\\_TEMPLATE](#) ([COUNTERS\\_TYPE](#)() &, [operator=](#))([Counters](#)< [Array\\_Type](#)
- [COUNTERS\\_TEMPLATE](#) ([void, add\\_counter](#))([Counter](#)< [Array\\_Type](#)
- [COUNTERS\\_TEMPLATE](#) ([std::vector< std::string >, get\\_names](#)()) const
- [COUNTERS\\_TEMPLATE](#) ([std::vector< std::string >, get\\_descriptions](#)()) const
- [COUNTERS\\_TEMPLATE](#) ([std::vector< double >, gen\\_hash](#))(const [Array\\_Type](#) &[array](#)
- [for](#) ([auto &c:data](#))
- [if](#) ([add\\_dims](#))
- [if](#) ([hasher](#))
- [if](#) ([res.size\(\)==0u](#)) [res.push\\_back\(0.0\)](#)
- [COUNTERS\\_TEMPLATE](#) ([void, add\\_hash](#))([Hasher\\_fun\\_type](#)< [Array\\_Type](#)

## Variables

- [Data\\_Type & counter\\_](#)
- [Data\\_Type &&counter\\_ noexcept](#)
- [size\\_t i](#)
- [size\\_t size\\_t j](#)
- [Data\\_Type fun](#)
- [Data\\_Type counter](#)
- [return](#)
- [Data\\_Type count\\_fun\\_](#)
- [Data\\_Type Counter\\_fun\\_type< Array\\_Type, Data\\_Type > init\\_fun\\_](#)
- [Data\\_Type Counter\\_fun\\_type< Array\\_Type, Data\\_Type > Hasher\\_fun\\_type< Array\\_Type, Data\\_Type > hasher\\_fun\\_](#)
- [Data\\_Type Counter\\_fun\\_type< Array\\_Type, Data\\_Type > Hasher\\_fun\\_type< Array\\_Type, Data\\_Type > Data\\_Type data\\_](#)
- [Data\\_Type Counter\\_fun\\_type< Array\\_Type, Data\\_Type > Hasher\\_fun\\_type< Array\\_Type, Data\\_Type > Data\\_Type std::string name\\_](#)
- [Data\\_Type Counter\\_fun\\_type< Array\\_Type, Data\\_Type > Hasher\\_fun\\_type< Array\\_Type, Data\\_Type > Data\\_Type std::string std::string desc\\_](#)
- [bool add\\_dims](#)
- [return res](#)
- [Data\\_Type fun\\_](#)

## 9.26.1 Macro Definition Documentation

### 9.26.1.1 COUNTER\_TEMPLATE

```
#define COUNTER_TEMPLATE(
    a,
    b )  template COUNTER_TEMPLATE_ARGS() inline a COUNTER_TYPE()::b
```

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



### 9.26.1.2 COUNTER\_TEMPLATE\_ARGS

```
#define COUNTER_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

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

### 9.26.1.3 COUNTER\_TYPE

```
#define COUNTER_TYPE( ) Counter<Array_Type,Data_Type>
```

Definition at line 4 of file counters-meat.hpp.

### 9.26.1.4 COUNTERS\_TEMPLATE

```
#define COUNTERS_TEMPLATE(  
    a,  
    b ) template COUNTERS_TEMPLATE_ARGS() inline a COUNTERS_TYPE()::b
```

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

### 9.26.1.5 COUNTERS\_TEMPLATE\_ARGS

```
#define COUNTERS_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

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

### 9.26.1.6 COUNTERS\_TYPE

```
#define COUNTERS_TYPE( ) Counters<Array_Type,Data_Type>
```

Definition at line 125 of file counters-meat.hpp.

### 9.26.1.7 TMP\_HASHER\_CALL

```
#define TMP_HASHER_CALL Hasher_fun_type<Array_Type,Data_Type>
```

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

## 9.26.2 Function Documentation

### 9.26.2.1 count\_fun()

```
return count_fun (
    Array ,
    i ,
    j ,
    data )
```

### 9.26.2.2 COUNTER\_TEMPLATE() [1/9]

```
COUNTER_TEMPLATE (
    Counter ) const
```

### 9.26.2.3 COUNTER\_TEMPLATE() [2/9]

```
COUNTER_TEMPLATE (
    COUNTER_TYPE() & ,
    operator )
```

### 9.26.2.4 COUNTER\_TEMPLATE() [3/9]

```
COUNTER_TEMPLATE (
    COUNTER_TYPE() ,
    operator ) const
```

### 9.26.2.5 COUNTER\_TEMPLATE() [4/9]

```
COUNTER_TEMPLATE (
    double ,
    count ) &
```

< Move assignment

**9.26.2.6 COUNTER\_TEMPLATE()** [5/9]

```
COUNTER_TEMPLATE (
    double ,
    init ) &
```

**9.26.2.7 COUNTER\_TEMPLATE()** [6/9]

```
COUNTER_TEMPLATE (
    std::string ,
    get_description ) const
```

Definition at line 107 of file counters-meat.hpp.

**9.26.2.8 COUNTER\_TEMPLATE()** [7/9]

```
COUNTER_TEMPLATE (
    std::string ,
    get_name ) const
```

Definition at line 103 of file counters-meat.hpp.

**9.26.2.9 COUNTER\_TEMPLATE()** [8/9]

```
COUNTER_TEMPLATE (
    TMP_HASHER_CALL ,
    get_hasher )
```

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

**9.26.2.10 COUNTER\_TEMPLATE()** [9/9]

```
COUNTER_TEMPLATE (
    void ,
    set_hasher )
```

**9.26.2.11 COUNTERS\_TEMPLATE()** [1/9]

```
COUNTERS_TEMPLATE (
    Counters )
```

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

**9.26.2.12 COUNTERS\_TEMPLATE()** [2/9]

```
COUNTERS_TEMPLATE (
    COUNTER_TYPE() & ,
    operator [ ] )
```

Definition at line 134 of file counters-meat.hpp.

**9.26.2.13 COUNTERS\_TEMPLATE()** [3/9]

```
COUNTERS_TEMPLATE (
    COUNTERS_TYPE() & ,
    operator )
```

**9.26.2.14 COUNTERS\_TEMPLATE()** [4/9]

```
COUNTERS_TEMPLATE (
    COUNTERS_TYPE() ,
    operator ) const
```

**9.26.2.15 COUNTERS\_TEMPLATE()** [5/9]

```
COUNTERS_TEMPLATE (
    std::vector< double > ,
    gen_hash ) const &
```

**9.26.2.16 COUNTERS\_TEMPLATE()** [6/9]

```
COUNTERS_TEMPLATE (
    std::vector< std::string > ,
    get_descriptions ) const
```

Definition at line 213 of file counters-meat.hpp.

**9.26.2.17 COUNTERS\_TEMPLATE() [7/9]**

```
COUNTERS_TEMPLATE (
    std::vector< std::string > ,
    get_names ) const
```

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

**9.26.2.18 COUNTERS\_TEMPLATE() [8/9]**

```
COUNTERS_TEMPLATE (
    void ,
    add_counter )
```

**9.26.2.19 COUNTERS\_TEMPLATE() [9/9]**

```
COUNTERS_TEMPLATE (
    void ,
    add_hash )
```

**9.26.2.20 data()**

```
Data_Type&& counter_ data (
    std::move(counter_.data) )
```

**9.26.2.21 desc()**

```
Data_Type&& counter_ desc (
    std::move(counter_.desc) )
```

Move constructor.

Definition at line 32 of file counters-meat.hpp.

**9.26.2.22 for()**

```
for (
    auto &c:data )
```

Definition at line 233 of file counters-meat.hpp.

**9.26.2.23 hasher()** [1/2]

```
Data_Type hasher (  
    counter_.  hasher )
```

Definition at line 141 of file counters-meat.hpp.

**9.26.2.24 hasher()** [2/2]

```
Data_Type&& counters_ hasher (  
    std::move(counters_.hasher) )
```

Definition at line 144 of file counters-meat.hpp.

**9.26.2.25 hasher\_fun()** [1/2]

```
Data_Type hasher_fun (  
    counter_.  hasher_fun )
```

Definition at line 13 of file counters-meat.hpp.

**9.26.2.26 hasher\_fun()** [2/2]

```
Data_Type&& counter_ hasher_fun (  
    std::move(counter_.hasher_fun) )
```

**9.26.2.27 if()** [1/3]

```
if (  
    add_dims )
```

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

**9.26.2.28 if()** [2/3]

```
if (  
    hasher )
```

Definition at line 255 of file counters-meat.hpp.

**9.26.2.29 if()** [3/3]

```
if (
    res.  size() == 0u )
```

**9.26.2.30 init\_fun()** [1/3]

```
return init_fun (
    Array ,
    i ,
    j ,
    data )
```

**9.26.2.31 init\_fun()** [2/3]

```
Data_Type init_fun (
    counter_.  init_fun )
```

**9.26.2.32 init\_fun()** [3/3]

```
Data_Type&& counter_ init_fun (
    std::move(counter_.init_fun) )
```

**9.26.2.33 name()**

```
Data_Type&& counter_ name (
    std::move(counter_.name) )
```

**9.26.3 Variable Documentation****9.26.3.1 add\_dims**

```
bool add_dims
```

**Initial value:**

```
{
    std::vector<double> res
```

Definition at line 227 of file counters-meat.hpp.

### 9.26.3.2 count\_fun\_

Data\_Type count\_fun\_

Definition at line 179 of file counters-meat.hpp.

### 9.26.3.3 counter

Data\_Type counter

#### Initial value:

```
{
    data.push_back(counter)
```

Definition at line 170 of file counters-meat.hpp.

### 9.26.3.4 counter\_

Data\_Type & counter\_

#### Initial value:

```
{
    if (this != &counter_) {
        this->count_fun = counter_.count_fun;
        this->init_fun = counter_.init_fun;
        this->hasher_fun = counter_.hasher_fun;

        this->data = counter_.data;
        this->name = counter_.name;
        this->desc = counter_.desc;
    }
    return *this
```

Definition at line 12 of file counters-meat.hpp.

### 9.26.3.5 data\_

Data\_Type Counter\_fun\_type<Array\_Type,Data\_Type> Hasher\_fun\_type<Array\_Type,Data\_Type> Data↔  
\_Type data\_

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



### 9.26.3.6 desc\_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> Hasher_fun_type<Array_Type,Data_Type> Data←  
_Type std::string std::string desc_
```

#### Initial value:

```
{  
    data.emplace_back(Counter<Array_Type,Data_Type>(  
        count_fun_,  
        init_fun_,  
        hasher_fun_,  
        data_,  
        name_,  
        desc_  
    ))  
}
```

Definition at line 184 of file counters-meat.hpp.

### 9.26.3.7 fun

```
Data_Type fun
```

#### Initial value:

```
{  
    hasher_fun = fun
```

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

### 9.26.3.8 fun\_

```
Data_Type fun_
```

#### Initial value:

```
{  
    hasher = fun_
```

Definition at line 270 of file counters-meat.hpp.

### 9.26.3.9 hasher\_fun\_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> Hasher_fun_type<Array_Type,Data_Type> hasher←  
_fun_
```

Definition at line 181 of file counters-meat.hpp.

**9.26.3.10 i**

```
size_t i
```

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

**9.26.3.11 init\_fun\_**

```
Data_Type Counter_fun_type<Array_Type,Data_Type> init_fun_
```

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

**9.26.3.12 j**

```
size_t j
```

**Initial value:**

```
{
    if (count_fun == nullptr)
        return 0.0
```

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

**9.26.3.13 name\_**

```
Data_Type Counter_fun_type<Array_Type,Data_Type> Hasher_fun_type<Array_Type,Data_Type> Data↔
_Type std::string name_
```

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

**9.26.3.14 noexcept**

```
Data_Type &&counters_ noexcept
```

**Initial value:**

```
{
    if (this != &counter_)
    {
        this->data = std::move(counter_.data);

        this->count_fun = std::move(counter_.count_fun);
        this->init_fun = std::move(counter_.init_fun);
        this->hasher_fun = std::move(counter_.hasher_fun);

        this->name = std::move(counter_.name);
        this->desc = std::move(counter_.desc);
    }
    return *this
```

Definition at line 26 of file counters-meat.hpp.

**9.26.3.15 res**

```
return res
```

Definition at line 265 of file counters-meat.hpp.

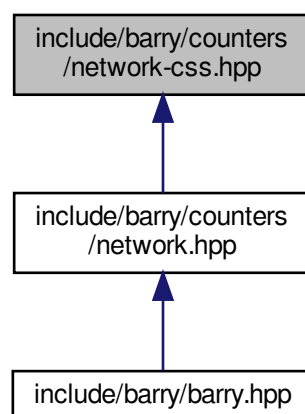
**9.26.3.16 return**

```
return
```

Definition at line 175 of file counters-meat.hpp.

**9.27 include/barry/counters/network-css.hpp File Reference**

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

**Macros**

- #define `CSS_SIZE()`
- #define `CSS_MATCH_TYPE()`
- #define `CSS_CASE_TRUTH()` if ((i < n) && (j < n))
- #define `CSS_TRUE_CELLS()`
- #define `CSS_CASE_PERCEIVED()` else if (((i >= s) && (i < e)) & ((j >= s) && (j < e)))
- #define `CSS_PERCEIVED_CELLS()`
- #define `CSS_CASE_ELSE()`
- #define `CSS_CHECK_SIZE_INIT()`
- #define `CSS_CHECK_SIZE()`
- #define `CSS_APPEND(name)`
- #define `CSS_NET_COUNTER_LAMBDA_INIT()`

## Functions

- `template<typename Tnet = Network>`  
`void counter_css_partially_false_recip_commi (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_, size_t counter_type=0u)`  
*Counts errors of commission.*
- `template<typename Tnet = Network>`  
`void counter_css_partially_false_recip_omiss (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_, size_t counter_type=0u)`  
*Counts errors of omission.*
- `template<typename Tnet = Network>`  
`void counter_css_completely_false_recip_comiss (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_, size_t counter_type=0u)`  
*Counts completely false reciprocity (comission)*
- `template<typename Tnet = Network>`  
`void counter_css_completely_false_recip_omiss (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_, size_t counter_type=0u)`  
*Counts completely false reciprocity (omission)*
- `template<typename Tnet = Network>`  
`void counter_css_mixed_recip (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_, size_t counter_type=0u)`  
*Counts mixed reciprocity errors.*
- `template<typename Tnet = Network>`  
`void counter_css_census01 (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_, size_t counter_type=0u)`
- `template<typename Tnet = Network>`  
`void counter_css_census02 (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_, size_t counter_type=0u)`
- `template<typename Tnet = Network>`  
`void counter_css_census03 (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_, size_t counter_type=0u)`
- `template<typename Tnet = Network>`  
`void counter_css_census04 (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_, size_t counter_type=0u)`
- `template<typename Tnet = Network>`  
`void counter_css_census05 (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_, size_t counter_type=0u)`
- `template<typename Tnet = Network>`  
`void counter_css_census06 (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_, size_t counter_type=0u)`
- `template<typename Tnet = Network>`  
`void counter_css_census07 (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_, size_t counter_type=0u)`
- `template<typename Tnet = Network>`  
`void counter_css_census08 (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_, size_t counter_type=0u)`
- `template<typename Tnet = Network>`  
`void counter_css_census09 (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_, size_t counter_type=0u)`
- `template<typename Tnet = Network>`  
`void counter_css_census10 (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_, size_t counter_type=0u)`

### 9.27.1 Macro Definition Documentation

### 9.27.1.1 CSS\_APPEND

```
#define CSS_APPEND(
    name )
```

**Value:**

```
std::string name_ = (name);\
for (size_t i = 0u; i < end_.size(); ++i) { \
    std::string tmpname = name_ + " (" + std::to_string(i) + ")" + \
    ((counter_type == 1u) ? " (only perceiver)" : ((counter_type == 2u)? " (only alters)": ""));\
    counters->add_counter(tmp_count, tmp_init, nullptr, \
        NetCounterData({netsize, i == 0u ? netsize : end_[i-1], end_[i], counter_type, i}, {}),\
        tmpname);}
```

Definition at line 81 of file network-css.hpp.

### 9.27.1.2 CSS\_CASE\_ELSE

```
#define CSS_CASE_ELSE( )
```

Definition at line 66 of file network-css.hpp.

### 9.27.1.3 CSS\_CASE\_PERCEIVED

```
#define CSS_CASE_PERCEIVED( ) else if (((i >= s) && (i < e)) & ((j >= s) && (j < e)))
```

Definition at line 48 of file network-css.hpp.

### 9.27.1.4 CSS\_CASE\_TRUTH

```
#define CSS_CASE_TRUTH( ) if ((i < n) && (j < n))
```

Definition at line 32 of file network-css.hpp.

### 9.27.1.5 CSS\_CHECK\_SIZE

```
#define CSS_CHECK_SIZE( )
```

**Value:**

```
for (size_t i = 0u; i < end_.size(); ++i) {\
    if (i == 0u) continue; \
    else if (end_[i] < end_[i-1u]) \
        throw std::logic_error("Endpoints should be specified in order.");}
```

Definition at line 76 of file network-css.hpp.

### 9.27.1.6 CSS\_CHECK\_SIZE\_INIT

```
#define CSS_CHECK_SIZE_INIT( )
```

#### Value:

```
/* The indices fall within the network */ \
if ((data.indices.at(0) > Array.ncol()) \
| (data.indices.at(2) > Array.ncol())) \
    throw std::range_error("The network does not match the prescribed size.");
```

Definition at line 70 of file network-css.hpp.

### 9.27.1.7 CSS\_MATCH\_TYPE

```
#define CSS_MATCH_TYPE( )
```

#### Value:

```
if (ctype != 0u) { /* all counts */ \
    if (ctype == 1u) { /* Only if perceiver */ \
        if ((i_ != ego_id) && (j_ != ego_id)) return 0.0; \
    } else if (ctype == 2u) { /* Only if not perceiver */ \
        if ((i_ == ego_id) || (j_ == ego_id)) return 0.0; \
    } \
};
```

Definition at line 22 of file network-css.hpp.

### 9.27.1.8 CSS\_NET\_COUNTER\_LAMBDA\_INIT

```
#define CSS_NET_COUNTER_LAMBDA_INIT( )
```

#### Value:

```
NETWORK_COUNTER_LAMBDA(tmp_init) {\
    CSS_CHECK_SIZE_INIT() \
    return 0.0; \
};
```

Definition at line 89 of file network-css.hpp.

### 9.27.1.9 CSS\_PERCEIVED\_CELLS

```
#define CSS_PERCEIVED_CELLS( )
```

#### Value:

```
size_t i_ = i - s; \
size_t j_ = j - s; \
CSS_MATCH_TYPE() \
double tji = static_cast<double>(Array(j - s, i - s, false)); \
double pji = static_cast<double>(Array(j, i, false)); \
double tij = static_cast<double>(Array(i - s, j - s, false));
```

Definition at line 55 of file network-css.hpp.

### 9.27.1.10 CSS\_SIZE

```
#define CSS_SIZE( )
```

**Value:**

```
size_t n      = data.indices[0u]; \
size_t s      = data.indices[1u]; \
size_t e      = data.indices[2u]; \
size_t ctype  = data.indices[3u]; \
size_t ego_id = data.indices[4u]; \
if (ctype > 2) \
    throw std::range_error("Counter type should be 0, 1, or 2.");
```

Definition at line 8 of file network-css.hpp.

### 9.27.1.11 CSS\_TRUE\_CELLS

```
#define CSS_TRUE_CELLS( )
```

**Value:**

```
size_t i_ = i; \
size_t j_ = j; \
CSS_MATCH_TYPE() \
double tji = static_cast<double>(Array(j, i, false)); \
double pij = static_cast<double>(Array(i + s, j + s, false)); \
double pji = static_cast<double>(Array(j + s, i + s, false));
```

Definition at line 39 of file network-css.hpp.

## 9.27.2 Function Documentation

### 9.27.2.1 counter\_css\_census01()

```
template<typename Tnet = Network>
void counter_css_census01 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_,
    size_t counter_type = 0u ) [inline]
```

Definition at line 324 of file network-css.hpp.

### 9.27.2.2 counter\_css\_census02()

```
template<typename Tnet = Network>
void counter_css_census02 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_,
    size_t counter_type = 0u ) [inline]
```

Definition at line 389 of file network-css.hpp.

### 9.27.2.3 counter\_css\_census03()

```
template<typename Tnet = Network>
void counter_css_census03 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_,
    size_t counter_type = 0u ) [inline]
```

Definition at line 429 of file network-css.hpp.

### 9.27.2.4 counter\_css\_census04()

```
template<typename Tnet = Network>
void counter_css_census04 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_,
    size_t counter_type = 0u ) [inline]
```

Definition at line 469 of file network-css.hpp.

### 9.27.2.5 counter\_css\_census05()

```
template<typename Tnet = Network>
void counter_css_census05 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_,
    size_t counter_type = 0u ) [inline]
```

Definition at line 509 of file network-css.hpp.

### 9.27.2.6 counter\_css\_census06()

```
template<typename Tnet = Network>
void counter_css_census06 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_,
    size_t counter_type = 0u ) [inline]
```

Definition at line 549 of file network-css.hpp.



### 9.27.2.7 counter\_css\_census07()

```
template<typename Tnet = Network>
void counter_css_census07 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_,
    size_t counter_type = 0u ) [inline]
```

Definition at line 589 of file network-css.hpp.

### 9.27.2.8 counter\_css\_census08()

```
template<typename Tnet = Network>
void counter_css_census08 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_,
    size_t counter_type = 0u ) [inline]
```

Definition at line 629 of file network-css.hpp.

### 9.27.2.9 counter\_css\_census09()

```
template<typename Tnet = Network>
void counter_css_census09 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_,
    size_t counter_type = 0u ) [inline]
```

Definition at line 669 of file network-css.hpp.

### 9.27.2.10 counter\_css\_census10()

```
template<typename Tnet = Network>
void counter_css_census10 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_,
    size_t counter_type = 0u ) [inline]
```

Definition at line 709 of file network-css.hpp.

**9.27.2.11 counter\_css\_completely\_false\_recip\_comiss()**

```
template<typename Tnet = Network>
void counter_css_completely_false_recip_comiss (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_,
    size_t counter_type = 0u ) [inline]
```

Counts completely false reciprocity (comission)

Definition at line 200 of file network-css.hpp.

**9.27.2.12 counter\_css\_completely\_false\_recip\_omiss()**

```
template<typename Tnet = Network>
void counter_css_completely_false_recip_omiss (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_,
    size_t counter_type = 0u ) [inline]
```

Counts completely false reciprocity (omission)

Definition at line 241 of file network-css.hpp.

**9.27.2.13 counter\_css\_mixed\_recip()**

```
template<typename Tnet = Network>
void counter_css_mixed_recip (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_,
    size_t counter_type = 0u ) [inline]
```

Counts mixed reciprocity errors.

Definition at line 282 of file network-css.hpp.

**9.27.2.14 counter\_css\_partially\_false\_recip\_commi()**

```
template<typename Tnet = Network>
void counter_css_partially_false_recip_commi (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_,
    size_t counter_type = 0u ) [inline]
```

Counts errors of commission.

## Parameters

<i>netsize</i>	Size of the reference (true) network
<i>end_</i>	Vector indicating one past the ending index of each network. (see details)
<i>counter_type</i>	Size_t indicating the type of counter to use. Possible values are: 0: Count all, 1: Only count if perceiver is involved, and 2: Only count if perceiver is not involved.

The *end\_* parameter should be of length *N* of *networks* - 1. It is assumed that the first network ends at *netsize*.

Definition at line 107 of file network-css.hpp.

## 9.27.2.15 counter\_css\_partially\_false\_recip\_omiss()

```
template<typename Tnet = Network>
void counter_css_partially_false_recip_omiss (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_,
    size_t counter_type = 0u ) [inline]
```

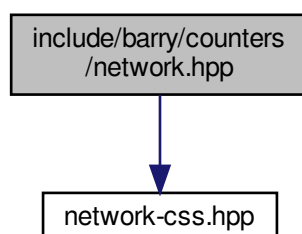
Counts errors of omission.

Definition at line 155 of file network-css.hpp.

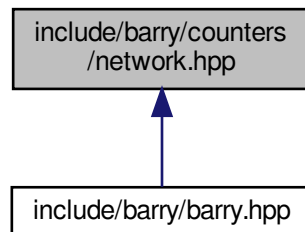
## 9.28 include/barry/counters/network.hpp File Reference

```
#include "network-css.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 size\_t 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\(a\)](#)
- #define [NETWORKDENSE\\_COUNTER\\_LAMBDA\(a\)](#)

### Macros for defining rules

- #define [NETWORK\\_RULE\(a\)](#)
- #define [NETWORK\\_RULE\\_LAMBDA\(a\)](#)

## Functions

- `template<typename Tnet = Network>`  
`void counter_edges (NetCounters< Tnet > *counters)`  
*Number of edges.*
- `template<typename Tnet = Network>`  
`void counter_isolates (NetCounters< Tnet > *counters)`  
*Number of isolated vertices.*
- `template<> void counter_isolates (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter_mutual (NetCounters< Tnet > *counters)`  
*Number of mutual ties.*
- `template<typename Tnet = Network>`  
`void counter_istar2 (NetCounters< Tnet > *counters)`
- `template<> void counter_istar2 (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter_ostar2 (NetCounters< Tnet > *counters)`
- `template<> void counter_ostar2 (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter_ttriads (NetCounters< Tnet > *counters)`
- `template<> void counter_ttriads (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter_ctriads (NetCounters< Tnet > *counters)`
- `template<> void counter_ctriads (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter_density (NetCounters< Tnet > *counters)`
- `template<typename Tnet = Network>`  
`void counter_idegree15 (NetCounters< Tnet > *counters)`
- `template<> void counter_idegree15 (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter_odegree15 (NetCounters< Tnet > *counters)`
- `template<> void counter_odegree15 (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter_absdiff (NetCounters< Tnet > *counters, size_t attr_id, double alpha=1.0)`  
*Sum of absolute attribute difference between ego and alter.*
- `template<typename Tnet = Network>`  
`void counter_diff (NetCounters< Tnet > *counters, size_t 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)`
- `template<typename Tnet = Network>`  
`void counter_nodeicov (NetCounters< Tnet > *counters, size_t attr_id)`
- `template<typename Tnet = Network>`  
`void counter_nodeocov (NetCounters< Tnet > *counters, size_t attr_id)`
- `template<typename Tnet = Network>`  
`void counter_nodecov (NetCounters< Tnet > *counters, size_t attr_id)`
- `template<typename Tnet = Network>`  
`void counter_nodematch (NetCounters< Tnet > *counters, size_t attr_id)`
- `template<typename Tnet = Network>`  
`void counter_idegree (NetCounters< Tnet > *counters, std::vector< size_t > d)`  
*Counts number of vertices with a given in-degree.*
- `template<> void counter_idegree (NetCounters< NetworkDense > *counters, std::vector< size_t > d)`
- `template<typename Tnet = Network>`  
`void counter_odegree (NetCounters< Tnet > *counters, std::vector< size_t > d)`  
*Counts number of vertices with a given out-degree.*
- `template<> void counter_odegree (NetCounters< NetworkDense > *counters, std::vector< size_t > d)`

- `template<typename Tnet = Network>`  
`void counter_degree (NetCounters< Tnet > *counters, std::vector< size_t > d)`  
*Counts number of vertices with a given out-degree.*

### Rules for network models

#### Parameters

rules	A pointer to a NetRules object ( <i>Rules&lt;Network, bool&gt;</i> ).
-------	---

- `template<typename Tnet = Network>`  
`void rules_zerodiag (NetRules< Tnet > *rules)`  
*Number of edges.*

### Convenient typedefs for network objects.

- `#define BARRY_ZERO_NETWORK 0.0`
- `#define BARRY_ZERO_NETWORK_DENSE 0`
- `typedef BArray< double, NetworkData > Network`
- `typedef BArrayDense< int, NetworkData > NetworkDense`
- `template<typename Tnet = Network>`  
`using NetCounter = Counter< Tnet, NetCounterData >`
- `template<typename Tnet = Network>`  
`using NetCounters = Counters< Tnet, NetCounterData >`
- `template<typename Tnet = Network>`  
`using NetSupport = Support< Tnet, NetCounterData >`
- `template<typename Tnet = Network>`  
`using NetStatsCounter = StatsCounter< Tnet, NetCounterData >`
- `template<typename Tnet >`  
`using NetModel = Model< Tnet, NetCounterData >`
- `template<typename Tnet = Network>`  
`using NetRule = Rule< Tnet, bool >`
- `template<typename Tnet = Network>`  
`using NetRules = Rules< Tnet, bool >`

## 9.28.1 Macro Definition Documentation

### 9.28.1.1 BARRY\_ZERO\_NETWORK

```
#define BARRY_ZERO_NETWORK 0.0
```

Definition at line 85 of file network.hpp.

### 9.28.1.2 BARRY\_ZERO\_NETWORK\_DENSE

```
#define BARRY_ZERO_NETWORK_DENSE 0
```

Definition at line 86 of file network.hpp.

### 9.28.1.3 NET\_C\_DATA\_IDX

```
#define NET_C_DATA_IDX(  
    i ) (data.indices[i])
```

Definition at line 74 of file network.hpp.

### 9.28.1.4 NET\_C\_DATA\_NUM

```
#define NET_C_DATA_NUM(  
    i ) (data.numbers[i])
```

Definition at line 75 of file network.hpp.

### 9.28.1.5 NETWORK\_COUNTER

```
#define NETWORK_COUNTER(  
    a )
```

#### Value:

```
template<typename Tnet = Network>\  
inline double (a) (const Tnet & Array, size_t i, size_t j, NetCounterData & data)
```

Function for definition of a network counter function

Definition at line 114 of file network.hpp.

### 9.28.1.6 NETWORK\_COUNTER\_LAMBDA

```
#define NETWORK_COUNTER_LAMBDA(  
    a )
```

#### Value:

```
Counter_fun_type<Tnet, NetCounterData> a = \  
    [] (const Tnet & Array, size_t i, size_t j, NetCounterData & data)
```

Lambda function for definition of a network counter function

Definition at line 119 of file network.hpp.

### 9.28.1.7 NETWORK\_RULE

```
#define NETWORK_RULE(  
    a )
```

**Value:**

```
template<typename Tnet = Network>\  
inline bool (a) (const Tnet & Array, size_t i, size_t j, bool & data)
```

Function for definition of a network counter function

Definition at line 133 of file network.hpp.

### 9.28.1.8 NETWORK\_RULE\_LAMBDA

```
#define NETWORK_RULE_LAMBDA(  
    a )
```

**Value:**

```
Rule_fun_type<Tnet, bool> a = \  
[] (const Tnet & Array, size_t i, size_t j, bool & data)
```

Lambda function for definition of a network counter function

Definition at line 138 of file network.hpp.

### 9.28.1.9 NETWORKDENSE\_COUNTER\_LAMBDA

```
#define NETWORKDENSE_COUNTER_LAMBDA(  
    a )
```

**Value:**

```
Counter_fun_type<NetworkDense, NetCounterData> a = \  
[] (const NetworkDense & Array, size_t i, size_t j, NetCounterData & data)
```

Definition at line 123 of file network.hpp.

## 9.28.2 Typedef Documentation

### 9.28.2.1 NetCounter

```
template<typename Tnet = Network>  
using NetCounter = Counter<Tnet, NetCounterData >
```

Definition at line 89 of file network.hpp.



### 9.28.2.2 NetCounters

```
template<typename Tnet = Network>  
using NetCounters = Counters<Tnet, NetCounterData>
```

Definition at line 92 of file network.hpp.

### 9.28.2.3 NetModel

```
template<typename Tnet >  
using NetModel = Model<Tnet, NetCounterData>
```

Definition at line 101 of file network.hpp.

### 9.28.2.4 NetRule

```
template<typename Tnet = Network>  
using NetRule = Rule<Tnet, bool>
```

Definition at line 104 of file network.hpp.

### 9.28.2.5 NetRules

```
template<typename Tnet = Network>  
using NetRules = Rules<Tnet, bool>
```

Definition at line 107 of file network.hpp.

### 9.28.2.6 NetStatsCounter

```
template<typename Tnet = Network>  
using NetStatsCounter = StatsCounter<Tnet, NetCounterData>
```

Definition at line 98 of file network.hpp.

### 9.28.2.7 NetSupport

```
template<typename Tnet = Network>  
using NetSupport = Support<Tnet, NetCounterData >
```

Definition at line 95 of file network.hpp.

### 9.28.2.8 Network

```
typedef BArray<double, NetworkData> Network
```

Definition at line 82 of file network.hpp.

### 9.28.2.9 NetworkDense

```
typedef BArrayDense<int, NetworkData> NetworkDense
```

Definition at line 83 of file network.hpp.

## 9.28.3 Function Documentation

### 9.28.3.1 rules\_zerodiag()

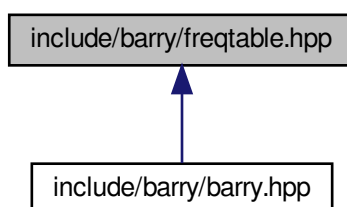
```
template<typename Tnet = Network>  
void rules_zerodiag (  
    NetRules< Tnet > * rules ) [inline]
```

Number of edges.

Definition at line 1381 of file network.hpp.

## 9.29 include/barry/freqtable.hpp File Reference

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

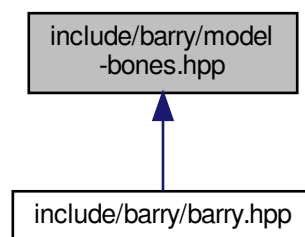


## Classes

- class [FreqTable< T >](#)  
*Frequency table of vectors.*

## 9.30 include/barry/model-bones.hpp File Reference

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

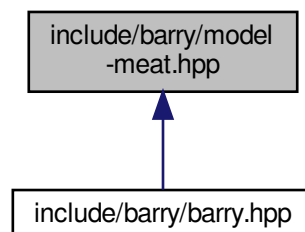


## Classes

- 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:*

## 9.31 include/barry/model-meat.hpp File Reference

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



## Functions

- double [update\\_normalizing\\_constant](#) (const std::vector< double > &params, const double \*support, size\_t k, size\_t n)
- double [likelihood\\_](#) (const double \*stats\_target, const std::vector< double > &params, const double normalizing\_constant, size\_t n\_params, bool log\_=false)

### 9.31.1 Function Documentation

#### 9.31.1.1 likelihood\_()

```
double likelihood_ (
    const double * stats_target,
    const std::vector< double > & params,
    const double normalizing_constant,
    size_t n_params,
    bool log_ = false ) [inline]
```

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

#### 9.31.1.2 update\_normalizing\_constant()

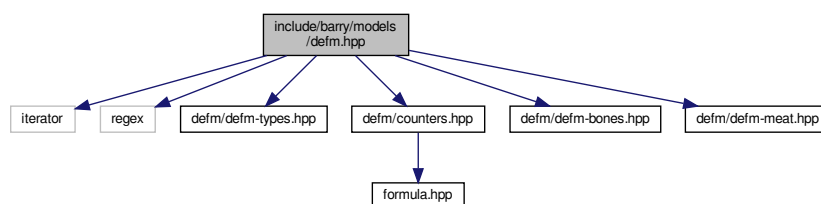
```
double update_normalizing_constant (
    const std::vector< double > & params,
    const double * support,
    size_t k,
    size_t n ) [inline]
```

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

## 9.32 include/barry/models/defm.hpp File Reference

```
#include <iterator>
#include <regex>
#include "defm/defm-types.hpp"
#include "defm/counters.hpp"
#include "defm/defm-bones.hpp"
#include "defm/defm-meat.hpp"
```

Include dependency graph for defm.hpp:



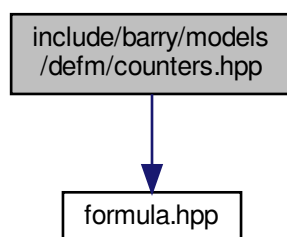
## Namespaces

- [defm](#)

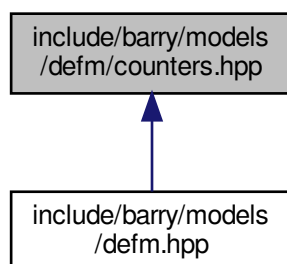
## 9.33 include/barry/models/defm/counters.hpp File Reference

```
#include "formula.hpp"
```

Include dependency graph for counters.hpp:



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



## Macros

- #define [MAKE\\_DEFM\\_HASHER](#)(hasher, a, cov)
- #define [DEFM RULEDYN\\_LAMBDA](#)(a)
- #define [UNI\\_SUB](#)(a)

### Macros for defining counters

- #define `DEFM_COUNTER(a)` inline double (a) (const `DEFMArray` & Array, size\_t i, size\_t j, `DEFMCounterData` & data)
- #define `DEFM_COUNTER_LAMBDA(a)`

### Macros for defining rules

- #define `DEFM_RULE(a)` inline bool (a) (const `DEFMArray` & Array, size\_t i, size\_t j, bool & data)
- #define `DEFM_RULE_LAMBDA(a)`

## Functions

- void `counter_ones` (`DEFMCounters` \*counters, int covar\_index=-1, std::string vname="", const std::vector< std::string > \*x\_names=nullptr)  
*Prevalence of ones.*
- void `counter_logit_intercept` (`DEFMCounters` \*counters, size\_t n\_y, std::vector< size\_t > which={}, int covar\_index=-1, std::string vname="", const std::vector< std::string > \*x\_names=nullptr, const std::vector< std::string > \*y\_names=nullptr)
- void `counter_transition` (`DEFMCounters` \*counters, std::vector< size\_t > coords, std::vector< bool > signs, size\_t m\_order, size\_t n\_y, int covar\_index=-1, std::string vname="", const std::vector< std::string > \*x\_names=nullptr, const std::vector< std::string > \*y\_names=nullptr)  
*Prevalence of ones.*
- void `counter_transition_formula` (`DEFMCounters` \*counters, std::string formula, size\_t m\_order, size\_t n\_y, int covar\_index=-1, std::string vname="", const std::vector< std::string > \*x\_names=nullptr, const std::vector< std::string > \*y\_names=nullptr)  
*Prevalence of ones.*
- void `counter_fixed_effect` (`DEFMCounters` \*counters, int covar\_index, double k, std::string vname="", const std::vector< std::string > \*x\_names=nullptr)  
*Prevalence of ones.*

### Returns true if the cell is free

#### Parameters

rules	A pointer to a <code>DEFMRules</code> object ( <code>Rules&lt;DEFMArray, bool&gt;</code> ).
-------	---

- void `rules_markov_fixed` (`DEFMRules` \*rules, size\_t markov\_order)  
*Number of edges.*
- void `rules_dont_become_zero` (`DEFMSupport` \*support, std::vector< size\_t > ids)  
*Blocks switching a one to zero.*

## 9.33.1 Macro Definition Documentation

### 9.33.1.1 DEFM\_COUNTER

```
#define DEFM_COUNTER(  
    a ) inline double (a) (const DEFMArray & Array, size_t i, size_t j, DEFMCounterData  
& data)
```

Function for definition of a network counter function

Definition at line 39 of file counters.hpp.

#### 9.33.1.2 DEFM\_COUNTER\_LAMBDA

```
#define DEFM_COUNTER_LAMBDA(  
    a )
```

**Value:**

```
barry::Counter_fun_type<DEFMArray, DEFMCounterData> a = \  
    [](const DEFMArray & Array, size_t i, size_t j, DEFMCounterData & data) -> double
```

Lambda function for definition of a network counter function

Definition at line 43 of file counters.hpp.

#### 9.33.1.3 DEFM\_RULE

```
#define DEFM_RULE(  
    a ) inline bool (a) (const DEFMArray & Array, size_t i, size_t j, bool & data)
```

Function for definition of a network counter function

Definition at line 53 of file counters.hpp.

#### 9.33.1.4 DEFM\_RULE\_LAMBDA

```
#define DEFM_RULE_LAMBDA(  
    a )
```

**Value:**

```
barry::Rule_fun_type<DEFMArray, DEFMRuleData> a = \  
    [](const DEFMArray & Array, size_t i, size_t j, DEFMRuleData & data) -> bool
```

Lambda function for definition of a network counter function

Definition at line 57 of file counters.hpp.

### 9.33.1.5 DEFM\_RULEDYN\_LAMBDA

```
#define DEFM_RULEDYN_LAMBDA(  
    a )
```

**Value:**

```
barry::Rule_fun_type<DEFMArray, DEFMRuleDynData> a = \  
[] (const DEFMArray & Array, size_t i, size_t j, DEFMRuleDynData & data) -> bool
```

Lambda function for definition of a network counter function

Definition at line 63 of file counters.hpp.

### 9.33.1.6 UNI\_SUB

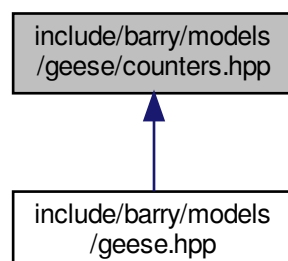
```
#define UNI_SUB(  
    a )
```

**Value:**

```
(\  
    ((a) == 0) ? u8"\u2080" : (\  
    ((a) == 1) ? u8"\u2081" : (\  
    ((a) == 2) ? u8"\u2082" : (\  
    ((a) == 3) ? u8"\u2083" : (\  
    ((a) == 4) ? u8"\u2084" : (\  
    ((a) == 5) ? u8"\u2085" : (\  
    ((a) == 6) ? u8"\u2086" : (\  
    ((a) == 7) ? u8"\u2087" : (\  
    ((a) == 8) ? u8"\u2088" : \  
    u8"\u2089" )))))))\  
)
```

## 9.34 include/barry/models/geese/counters.hpp File Reference

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





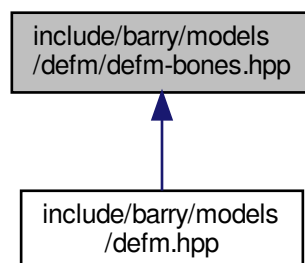
## Functions

- void `rule_dyn_limit_changes` (`PhyloSupport` \*support, `size_t` pos, `size_t` lb, `size_t` ub, `size_t` duplication=`Geese::etype_default`)  
*Overall functional gains.*
- `#define MAKE_DUPL_VARS()`
- `#define IS_EITHER()` (`DATA_AT == Geese::etype_either`)
- `#define IS_DUPLICATION()` (`((DATA_AT == Geese::etype_duplication) & (DPL))`)
- `#define IS_SPECIATION()` (`((DATA_AT == Geese::etype_speciation) & (!DPL))`)
- `#define IF_MATCHES()`
- `#define IF_NOTMATCHES()`
- `#define PHYLO_COUNTER_LAMBDA(a)`  
*Extension of a simple counter.*
- `#define PHYLO_RULE_DYN_LAMBDA(a)`
- `#define PHYLO_CHECK_MISSING()`
- `std::string get_last_name` (`size_t` d)
- void `counter_overall_gains` (`PhyloCounters` \*counters, `size_t` duplication=`Geese::etype_default`)  
*Overall functional gains.*
- void `counter_gains` (`PhyloCounters` \*counters, `std::vector< size_t >` nfun, `size_t` duplication=`Geese::etype_default`)  
*Functional gains for a specific function (nfun).*
- void `counter_gains_k_offspring` (`PhyloCounters` \*counters, `std::vector< size_t >` nfun, `size_t` k=1u, `size_t` duplication=`Geese::etype_default`)  
*k genes gain function nfun*
- void `counter_genes_changing` (`PhyloCounters` \*counters, `size_t` duplication=`Geese::etype_default`)  
*Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)*
- void `counter_preserve_pseudogene` (`PhyloCounters` \*counters, `size_t` nfunA, `size_t` nfunB, `size_t` duplication=`Geese::etype_default`)  
*Keeps track of how many pairs of genes preserve pseudostate.*
- void `counter_prop_genes_changing` (`PhyloCounters` \*counters, `size_t` duplication=`Geese::etype_default`)  
*Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)*
- void `counter_overall_loss` (`PhyloCounters` \*counters, `size_t` duplication=`Geese::etype_default`)  
*Overall functional loss.*
- void `counter_maxfuns` (`PhyloCounters` \*counters, `size_t` lb, `size_t` ub, `size_t` duplication=`Geese::etype_default`)  
*Cap the number of functions per gene.*
- void `counter_loss` (`PhyloCounters` \*counters, `std::vector< size_t >` nfun, `size_t` duplication=`Geese::etype_default`)  
*Total count of losses for an specific function.*
- void `counter_overall_changes` (`PhyloCounters` \*counters, `size_t` duplication=`Geese::etype_default`)  
*Total number of changes. Use this statistic to account for "preservation".*
- void `counter_subfun` (`PhyloCounters` \*counters, `size_t` nfunA, `size_t` nfunB, `size_t` duplication=`Geese::etype_default`)  
*Total count of Sub-functionalization events.*
- void `counter_cogain` (`PhyloCounters` \*counters, `size_t` nfunA, `size_t` nfunB, `size_t` duplication=`Geese::etype_default`)  
*Co-evolution (joint gain or loss)*
- void `counter_longest` (`PhyloCounters` \*counters, `size_t` duplication=`Geese::etype_default`)  
*Longest branch mutates (either by gain or by loss)*
- void `counter_neofun` (`PhyloCounters` \*counters, `size_t` nfunA, `size_t` nfunB, `size_t` duplication=`Geese::etype_default`)  
*Total number of neofunctionalization events.*
- void `counter_pairwise_neofun_singlefun` (`PhyloCounters` \*counters, `size_t` nfunA, `size_t` duplication=`Geese::etype_default`)  
*Total number of neofunctionalization events  $\sum_u \sum_{\{w < u\}} [x(u,a)*(1 - x(w,a)) + (1 - x(u,a)) * x(w,a)]$  change  
stat:  $\delta\{x(u,a): 0 \rightarrow 1\} = 1 - 2 * x(w,a)$*
- void `counter_neofun_a2b` (`PhyloCounters` \*counters, `size_t` nfunA, `size_t` nfunB, `size_t` duplication=`Geese::etype_default`)  
*Total number of neofunctionalization events.*

- void `counter_co_opt` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=Geese::etype_default`)  
*Function co-opting.*
- void `counter_k_genes_changing` (`PhyloCounters *counters`, `size_t k`, `size_t duplication=Geese::etype_default`)  
*Indicator function. Equals to one if  $k$  genes changed and zero otherwise.*
- void `counter_less_than_p_prop_genes_changing` (`PhyloCounters *counters`, `double p`, `size_t duplication=Geese::etype_default`)  
*Indicator function. Equals to one if  $k$  genes changed and zero otherwise.*
- void `counter_gains_from_0` (`PhyloCounters *counters`, `std::vector< size_t > nfun`, `size_t duplication=Geese::etype_default`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `counter_overall_gains_from_0` (`PhyloCounters *counters`, `size_t duplication=Geese::etype_default`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `counter_pairwise_overall_change` (`PhyloCounters *counters`, `size_t duplication=Geese::etype_default`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `counter_pairwise_preserving` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=Geese::etype_default`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `counter_pairwise_first_gain` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=Geese::etype_default`)  
*Used when all the functions are in 0 (like the root node prob.)*

## 9.35 include/barry/models/defm/defm-bones.hpp File Reference

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

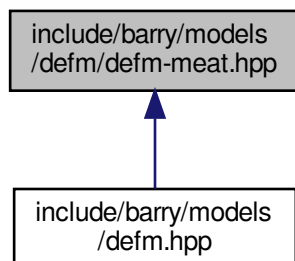


## Classes

- class `DEFM`

## 9.36 include/barry/models/defm/defm-meat.hpp File Reference

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



### Macros

- `#define` [DEFM\\_RANGES](#)(a)
- `#define` [DEFM\\_LOOP\\_ARRAYS](#)(a) `for` (size\_t a = 0u; a < (nobs\_i - M\_order); ++a)

### Functions

- `std::vector< double >` [keygen\\_defm](#) (const [DEFMArray](#) &Array\_, [DEFMCounterData](#) \*data)

### 9.36.1 Macro Definition Documentation

#### 9.36.1.1 DEFM\_LOOP\_ARRAYS

```
#define DEFM_LOOP_ARRAYS(
    a )    for (size_t a = 0u; a < (nobs_i - M_order); ++a)
```

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

#### 9.36.1.2 DEFM\_RANGES

```
#define DEFM_RANGES(
    a )
```

##### Value:

```
size_t start_i = start_end[a * 2u];\
size_t end_i   = start_end[a * 2u + 1u];\
size_t nobs_i  = end_i - start_i + 1u;
```

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

## 9.36.2 Function Documentation

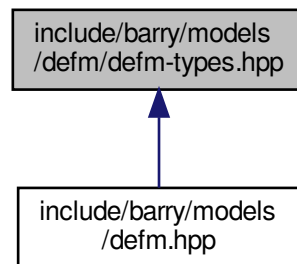
### 9.36.2.1 keygen\_defm()

```
std::vector< double > keygen_defm (
    const DEFMArray & Array_,
    DEFMCounterData * data ) [inline]
```

Definition at line 4 of file defm-meat.hpp.

## 9.37 include/barry/models/defm/defm-types.hpp File Reference

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



### Classes

- class [DEFMData](#)  
*Data class for **DEFM** arrays.*
- class [DEFMCounterData](#)  
*Data class used to store arbitrary size\_t or double vectors.*
- class [DEFMRuleData](#)
- class [DEFMRuleDynData](#)

### Typedefs

- typedef `barry::BArrayDense< int, DEFMData >` [DEFMArray](#)

#### Convenient typedefs for network objects.

- typedef `barry::Counter< DEFMArray, DEFMCounterData >` [DEFMCounter](#)
- typedef `barry::Counters< DEFMArray, DEFMCounterData >` [DEFMCounters](#)
- typedef `barry::Support< DEFMArray, DEFMCounterData, DEFMRuleData, DEFMRuleDynData >` [DEFMSupport](#)
- typedef `barry::StatsCounter< DEFMArray, DEFMCounterData >` [DEFMStatsCounter](#)
- typedef `barry::Model< DEFMArray, DEFMCounterData, DEFMRuleData, DEFMRuleDynData >` [DEFMModel](#)
- typedef `barry::Rule< DEFMArray, DEFMRuleData >` [DEFMRule](#)
- typedef `barry::Rules< DEFMArray, DEFMRuleData >` [DEFMRules](#)
- typedef `barry::Rule< DEFMArray, DEFMRuleDynData >` [DEFMRuleDyn](#)
- typedef `barry::Rules< DEFMArray, DEFMRuleDynData >` [DEFMRulesDyn](#)

### 9.37.1 Typedef Documentation

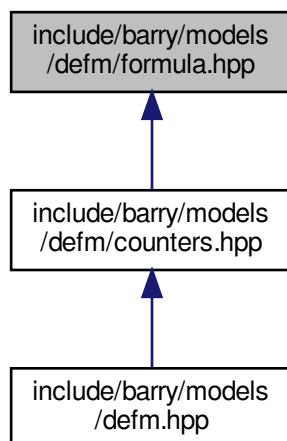
#### 9.37.1.1 DEFMArray

```
typedef barry::BArrayDense<int, DEFMData> DEFMArray
```

Definition at line 5 of file defm-types.hpp.

## 9.38 include/barry/models/defm/formula.hpp File Reference

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



### Functions

- void `defm_motif_parser` (std::string formula, std::vector< size\_t > &locations, std::vector< bool > &signs, size\_t m\_order, size\_t y\_ncol)  
*Parses a motif formula.*

#### 9.38.1 Function Documentation

### 9.38.1.1 defm\_motif\_parser()

```
void defm_motif_parser (
    std::string formula,
    std::vector< size_t > & locations,
    std::vector< bool > & signs,
    size_t m_order,
    size_t y_ncol ) [inline]
```

Parses a motif formula.

This function will take the formula and generate the corresponding input for `defm::counter_transition()`. Formulas can be specified in the following ways:

- Intercept effect: {...} No transition, only including the current state.
- Transition effect: {...} > {...} Includes current and previous states.

The general notation is `[0]y[column id]_[row id]`. A preceeding zero means that the value of the cell is considered to be zero. The column id goes between 0 and the number of columns in the array - 1 (so it is indexed from 0,) and the row id goes from 0 to `m_order`.

#### Intercept effects

Intercept effects only involve a single set of curly brackets. Using the 'greater-than' symbol (i.e., '<') is only for transition effects. When specifying intercept effects, users can skip the `row_id`, e.g., `y0_0` is equivalent to `y0`. If the passed `row_id` is different from the Markov order, i.e., `row_id != m_order`, then the function returns with an error.

Examples:

- `"{y0, 0y1}"` is equivalent to set a motif with the first element equal to one and the second to zero.

#### Transition effects

Transition effects can be specified using two sets of curly brackets and an greater-than symbol, i.e., `{...} > {...}`. The first set of brackets, which we call LHS, can only hold `row_id` that are less than `m_order`.

#### Parameters

<i>formula</i>	
<i>locations</i>	
<i>signs</i>	
<i>m_order</i>	
<i>y_ncol</i>	

Definition at line 46 of file `formula.hpp`.

## 9.39 include/barry/models/geese.hpp File Reference

```
#include "geese/geese-types.hpp"
#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-meat.hpp"
#include "geese/counters.hpp"
```

Include dependency graph for geese.hpp:

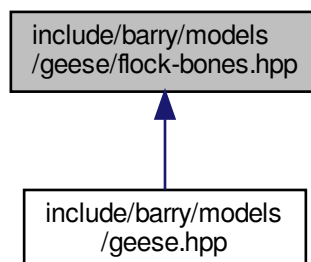


### Namespaces

- [geese](#)

## 9.40 include/barry/models/geese/flock-bones.hpp File Reference

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

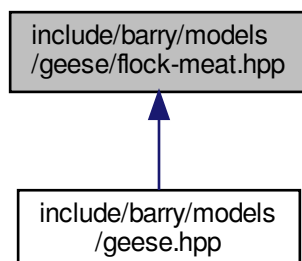


### Classes

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

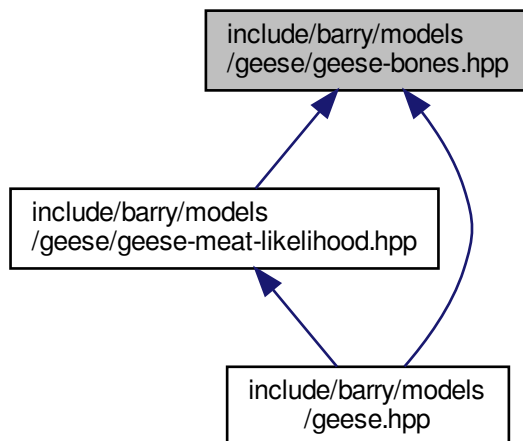
## 9.41 include/barry/models/geese/flock-meat.hpp File Reference

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



## 9.42 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 PhyloArray &array, const PhyloCounterData *d)`
- `bool vec_diff (const std::vector< size_t > &s, const std::vector< size_t > &a)`

### 9.42.1 Macro Definition Documentation

#### 9.42.1.1 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.

### 9.42.2 Function Documentation

#### 9.42.2.1 keygen\_full()

```
std::vector< double > keygen_full (
    const PhyloArray & array,
    const PhyloCounterData * d ) [inline]
```

Definition at line 36 of file geese-bones.hpp.

#### 9.42.2.2 RULE\_FUNCTION()

```
RULE_FUNCTION (
    rule_empty_free )
```

Definition at line 26 of file geese-bones.hpp.

### 9.42.2.3 `vec_diff()`

```
bool vec_diff (
    const std::vector< size_t > & s,
    const std::vector< size_t > & a ) [inline]
```

Definition at line 60 of file geese-bones.hpp.

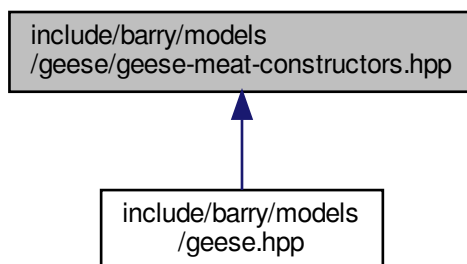
### 9.42.2.4 `vector_caster()`

```
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.

## 9.43 `include/barry/models/geese/geese-meat-constructors.hpp` File Reference

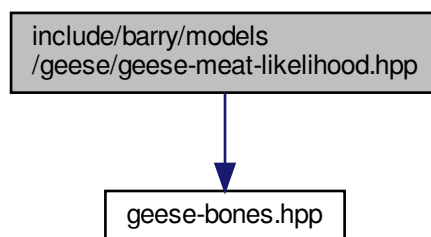
This graph shows which files directly or indirectly include this file:



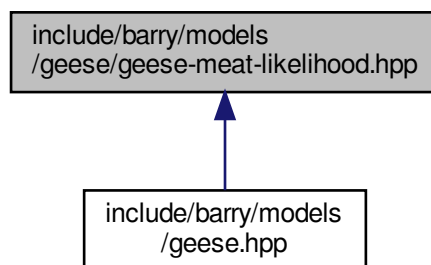
## 9.44 include/barry/models/geese/geese-meat-likelihood.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat-likelihood.hpp:

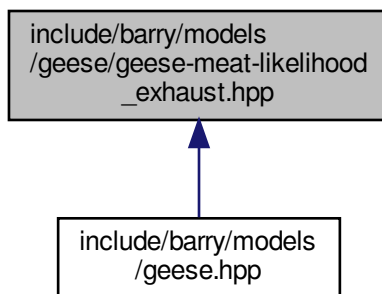


This graph shows which files directly or indirectly include this file:



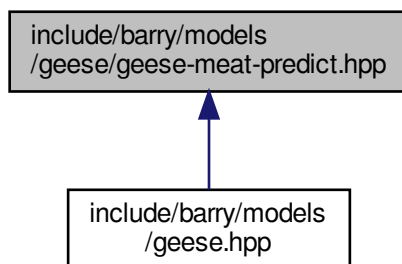
## 9.45 include/barry/models/geese/geese-meat-likelihood\_exhaust.hpp File Reference

This graph shows which files directly or indirectly include this file:



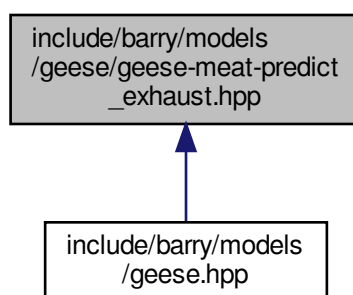
## 9.46 include/barry/models/geese/geese-meat-predict.hpp File Reference

This graph shows which files directly or indirectly include this file:



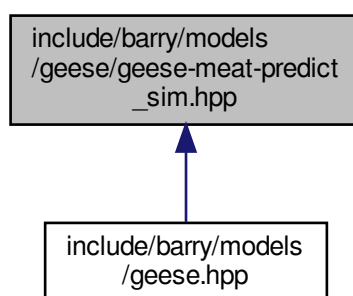
## 9.47 include/barry/models/geese/geese-meat-predict\_exhaust.hpp File Reference

This graph shows which files directly or indirectly include this file:



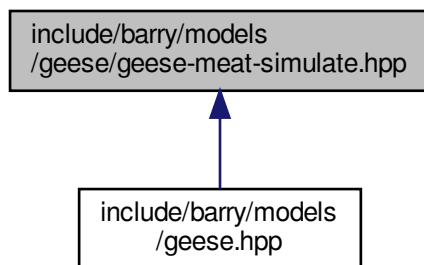
## 9.48 include/barry/models/geese/geese-meat-predict\_sim.hpp File Reference

This graph shows which files directly or indirectly include this file:



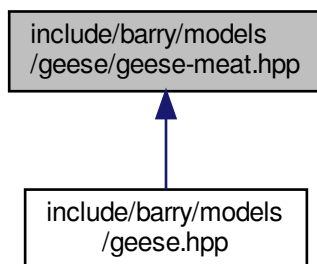
## 9.49 include/barry/models/geese/geese-meat-simulate.hpp File Reference

This graph shows which files directly or indirectly include this file:



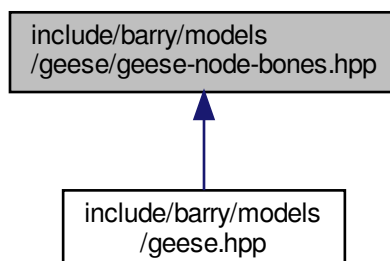
## 9.50 include/barry/models/geese/geese-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



## 9.51 include/barry/models/geese/geese-node-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:

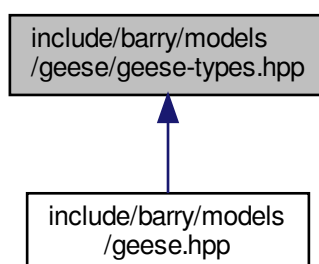


### Classes

- class [Node](#)  
*A single node for the model.*

## 9.52 include/barry/models/geese/geese-types.hpp File Reference

This graph shows which files directly or indirectly include this file:



### Classes

- class [NodeData](#)  
*Data definition for the `PhyloArray` class.*
- class [PhyloCounterData](#)
- class [PhyloRuleDynData](#)

## Macros

- `#define POS(a, b) (b)*N + (a)`

## Typedefs

Convenient typedefs for Node objects.

- `typedef std::vector< std::pair< size_t, size_t > > PhyloRuleData`
- `typedef barry::BArrayDense< size_t, NodeData > PhyloArray`
- `typedef barry::Counter< PhyloArray, PhyloCounterData > PhyloCounter`
- `typedef barry::Counters< PhyloArray, PhyloCounterData > PhyloCounters`
- `typedef barry::Rule< PhyloArray, PhyloRuleData > PhyloRule`
- `typedef barry::Rules< PhyloArray, PhyloRuleData > PhyloRules`
- `typedef barry::Rule< PhyloArray, PhyloRuleDynData > PhyloRuleDyn`
- `typedef barry::Rules< PhyloArray, PhyloRuleDynData > PhyloRulesDyn`
- `typedef barry::Support< PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport`
- `typedef barry::StatsCounter< PhyloArray, PhyloCounterData > PhyloStatsCounter`
- `typedef barry::Model< PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel`
- `typedef barry::PowerSet< PhyloArray, PhyloRuleData > PhyloPowerSet`

## 9.52.1 Macro Definition Documentation

### 9.52.1.1 POS

```
#define POS(
    a,
    b ) (b)*N + (a)
```

Definition at line 4 of file geese-types.hpp.

## 9.52.2 Typedef Documentation

### 9.52.2.1 PhyloArray

```
typedef barry::BArrayDense<size_t, NodeData> PhyloArray
```

Definition at line 104 of file geese-types.hpp.



### 9.52.2.2 PhyloCounter

```
typedef barry::Counter<PhyloArray, PhyloCounterData > PhyloCounter
```

Definition at line 105 of file geese-types.hpp.

### 9.52.2.3 PhyloCounters

```
typedef barry::Counters< PhyloArray, PhyloCounterData> PhyloCounters
```

Definition at line 106 of file geese-types.hpp.

### 9.52.2.4 PhyloModel

```
typedef barry::Model<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel
```

Definition at line 116 of file geese-types.hpp.

### 9.52.2.5 PhyloPowerSet

```
typedef barry::PowerSet<PhyloArray, PhyloRuleData> PhyloPowerSet
```

Definition at line 117 of file geese-types.hpp.

### 9.52.2.6 PhyloRule

```
typedef barry::Rule<PhyloArray, PhyloRuleData> PhyloRule
```

Definition at line 108 of file geese-types.hpp.

### 9.52.2.7 PhyloRuleData

```
typedef std::vector< std::pair< size_t, size_t > > PhyloRuleData
```

Definition at line 101 of file geese-types.hpp.

### 9.52.2.8 PhyloRuleDyn

```
typedef barry::Rule<PhyloArray,PhyloRuleDynData> PhyloRuleDyn
```

Definition at line 111 of file geese-types.hpp.

### 9.52.2.9 PhyloRules

```
typedef barry::Rules<PhyloArray,PhyloRuleData> PhyloRules
```

Definition at line 109 of file geese-types.hpp.

### 9.52.2.10 PhyloRulesDyn

```
typedef barry::Rules<PhyloArray,PhyloRuleDynData> PhyloRulesDyn
```

Definition at line 112 of file geese-types.hpp.

### 9.52.2.11 PhyloStatsCounter

```
typedef barry::StatsCounter<PhyloArray, PhyloCounterData> PhyloStatsCounter
```

Definition at line 115 of file geese-types.hpp.

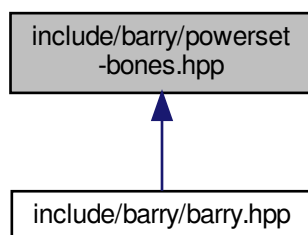
### 9.52.2.12 PhyloSupport

```
typedef barry::Support<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport
```

Definition at line 114 of file geese-types.hpp.

## 9.53 include/barry/powerset-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:

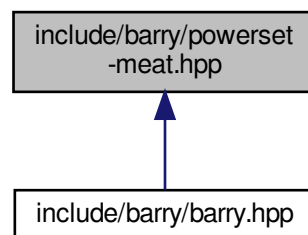


## Classes

- class [PowerSet< Array\\_Type, Data\\_Rule\\_Type >](#)  
*Powerset of a binary array.*

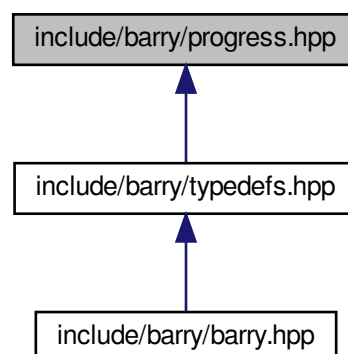
## 9.54 include/barry/powerset-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



## 9.55 include/barry/progress.hpp File Reference

This graph shows which files directly or indirectly include this file:



## Classes

- class [Progress](#)  
*A simple progress bar.*

## Macros

- `#define BARRY_PROGRESS_BAR_WIDTH 80`

### 9.55.1 Macro Definition Documentation

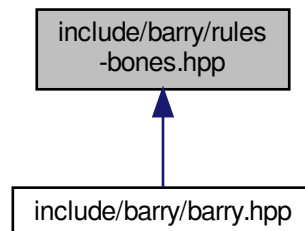
#### 9.55.1.1 BARRY\_PROGRESS\_BAR\_WIDTH

```
#define BARRY_PROGRESS_BAR_WIDTH 80
```

Definition at line 5 of file progress.hpp.

## 9.56 include/barry/rules-bones.hpp File Reference

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, size_t i, size_t j, Data_Type *dat)`

## 9.56.1 Function Documentation

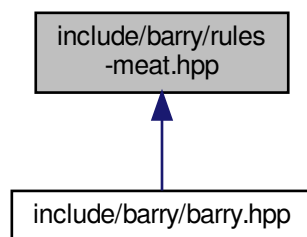
### 9.56.1.1 rule\_fun\_default()

```
template<typename Array_Type , typename Data_Type >
bool rule_fun_default (
    const Array_Type * array,
    size_t i,
    size_t j,
    Data_Type * dat )
```

Definition at line 5 of file rules-bones.hpp.

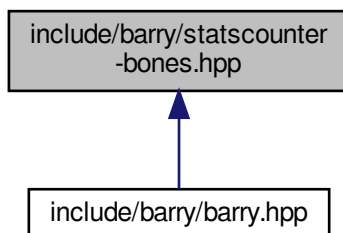
## 9.57 include/barry/rules-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



## 9.58 include/barry/statscounter-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:

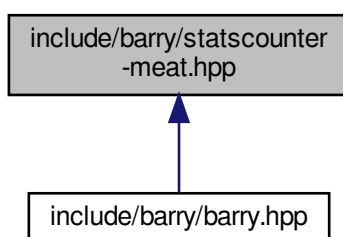


## Classes

- class [StatsCounter](#)< [Array\\_Type](#), [Data\\_Type](#) >  
Count stats for a single Array.

## 9.59 include/barry/statscounter-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



## Macros

- #define [STATSCOUNTER\\_TYPE](#)() [StatsCounter](#)<[Array\\_Type](#),[Data\\_Type](#)>
- #define [STATSCOUNTER\\_TEMPLATE\\_ARGS](#)() <typename [Array\\_Type](#), typename [Data\\_Type](#)>
- #define [STATSCOUNTER\\_TEMPLATE](#)(a, b) template [STATSCOUNTER\\_TEMPLATE\\_ARGS](#)() inline a [STATSCOUNTER\\_TYPE](#)()::b

## Functions

- [STATSCOUNTER\\_TEMPLATE](#) (, [StatsCounter](#))(const [StatsCounter](#)< [Array\\_Type](#)
- [EmptyArray](#) clear ()
- [STATSCOUNTER\\_TEMPLATE](#) (,~[StatsCounter](#)())()
- [STATSCOUNTER\\_TEMPLATE](#) (void, reset\_array)(const [Array\\_Type](#) \*Array\_)
- [STATSCOUNTER\\_TEMPLATE](#) (void, add\_counter)([Counter](#)< [Array\\_Type](#)
- [STATSCOUNTER\\_TEMPLATE](#) (void, set\_counters)([Counters](#)< [Array\\_Type](#)
- [STATSCOUNTER\\_TEMPLATE](#) (void, count\_init)(size\_t i
- [current\\_stats](#) resize ([counters](#)->size(), 0.0)
- for (size\_t n=0u;n< [counters](#)->size();++n) [current\\_stats](#)[n]
- [STATSCOUNTER\\_TEMPLATE](#) (void, count\_current)(size\_t i
- [STATSCOUNTER\\_TEMPLATE](#) (std::vector< std::string >, get\_names)() const
- [STATSCOUNTER\\_TEMPLATE](#) (std::vector< std::string >, get\_descriptions)() const

## Variables

- Data\_Type & [counter](#)
- [EmptyArray](#) = \*Array
- [current\\_stats](#) = counter.current\_stats
- [counters](#) = new [Counters](#)<Array\_Type,Data\_Type>((\*counter.counters))
- [counter\\_deleted](#) = false
- Data\_Type [f\\_](#)
- [return](#)
- Data\_Type \* [counters\\_](#)
- size\_t [j](#)

## 9.59.1 Macro Definition Documentation

### 9.59.1.1 STATSCOUNTER\_TEMPLATE

```
#define STATSCOUNTER_TEMPLATE(
    a,
    b )  template STATSCOUNTER\_TEMPLATE\_ARGS( ) inline a STATSCOUNTER\_TYPE( )::b
```

Definition at line 8 of file statscounter-meat.hpp.

### 9.59.1.2 STATSCOUNTER\_TEMPLATE\_ARGS

```
template STATSCOUNTER\_TEMPLATE\_ARGS( ) <typename Array_Type, typename Data_Type>
```

Definition at line 6 of file statscounter-meat.hpp.

### 9.59.1.3 STATSCOUNTER\_TYPE

```
template Data_Type * STATSCOUNTER\_TYPE( ) StatsCounter<Array_Type,Data_Type>
```

Definition at line 4 of file statscounter-meat.hpp.

## 9.59.2 Function Documentation

### 9.59.2.1 clear()

```
EmptyArray clear ( )
```

**9.59.2.2 for()**

```
for (
    size_t n = 0; n < counters->size(); ++n )
```

**9.59.2.3 resize()**

```
current_stats resize (
    counters-> size(),
    0. 0 )
```

**9.59.2.4 STATSCOUNTER\_TEMPLATE() [1/9]**

```
STATSCOUNTER_TEMPLATE (
    StatsCounter ) const
```

**9.59.2.5 STATSCOUNTER\_TEMPLATE() [2/9]**

```
STATSCOUNTER_TEMPLATE (
    ~ StatsCounter )
```

Definition at line 27 of file statscounter-meat.hpp.

**9.59.2.6 STATSCOUNTER\_TEMPLATE() [3/9]**

```
STATSCOUNTER_TEMPLATE (
    std::vector< std::string > ,
    get_descriptions ) const
```

Definition at line 256 of file statscounter-meat.hpp.

**9.59.2.7 STATSCOUNTER\_TEMPLATE() [4/9]**

```
STATSCOUNTER_TEMPLATE (
    std::vector< std::string > ,
    get_names ) const
```

Definition at line 251 of file statscounter-meat.hpp.



#### 9.59.2.8 STATSCOUNTER\_TEMPLATE() [5/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    add_counter )
```

#### 9.59.2.9 STATSCOUNTER\_TEMPLATE() [6/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    count_current )
```

#### 9.59.2.10 STATSCOUNTER\_TEMPLATE() [7/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    count_init )
```

#### 9.59.2.11 STATSCOUNTER\_TEMPLATE() [8/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    reset_array ) const
```

Definition at line 34 of file statscounter-meat.hpp.

#### 9.59.2.12 STATSCOUNTER\_TEMPLATE() [9/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    set_counters )
```

### 9.59.3 Variable Documentation

### 9.59.3.1 counter

```
Data_Type& counter
```

#### Initial value:

```
{  
    Array      = counter.Array
```

Definition at line 12 of file statscounter-meat.hpp.

### 9.59.3.2 counter\_deleted

```
counter_deleted = false
```

Definition at line 23 of file statscounter-meat.hpp.

### 9.59.3.3 counters

```
counters = new Counters<Array_Type,Data_Type> ((*counter.counters))
```

Definition at line 22 of file statscounter-meat.hpp.

### 9.59.3.4 counters\_

```
Data_Type* counters_
```

#### Initial value:

```
{  
  
    if (!counter_deleted)  
        delete counters
```

Definition at line 53 of file statscounter-meat.hpp.

### 9.59.3.5 current\_stats

```
current_stats = counter.current_stats
```

Definition at line 19 of file statscounter-meat.hpp.

### 9.59.3.6 EmptyArray

```
EmptyArray = *Array
```

Definition at line 17 of file statscounter-meat.hpp.

### 9.59.3.7 f\_

```
Data_Type f_
```

**Initial value:**

```
{  
    counters->add_counter(f_)
```

Definition at line 44 of file statscounter-meat.hpp.

### 9.59.3.8 j

```
size_t j
```

**Initial value:**

```
{  
  
    if (counters->size() == 0u)  
        throw std::logic_error("No counters added: Cannot count without knowing what to count!")
```

Definition at line 66 of file statscounter-meat.hpp.

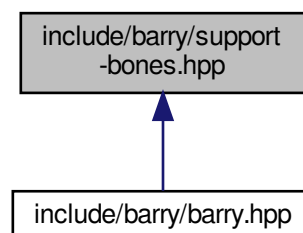
### 9.59.3.9 return

```
return
```

Definition at line 49 of file statscounter-meat.hpp.

## 9.60 include/barry/support-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:

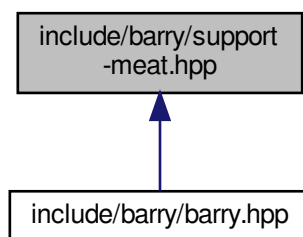


## Classes

- class [Support](#)< [Array\\_Type](#), [Data\\_Counter\\_Type](#), [Data\\_Rule\\_Type](#), [Data\\_Rule\\_Dyn\\_Type](#) >  
*Compute the support of sufficient statistics.*

## 9.61 include/barry/support-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



## Macros

- `#define` [BARRY\\_SUPPORT\\_MEAT\\_HPP](#) 1

### 9.61.1 Macro Definition Documentation

#### 9.61.1.1 BARRY\_SUPPORT\_MEAT\_HPP

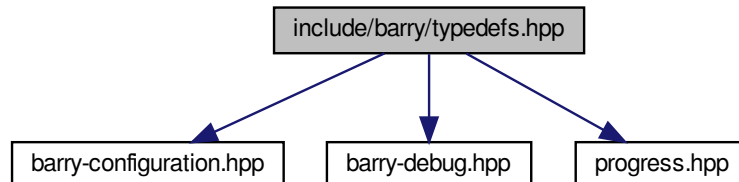
```
#define BARRY_SUPPORT_MEAT_HPP 1
```

Definition at line 2 of file support-meat.hpp.

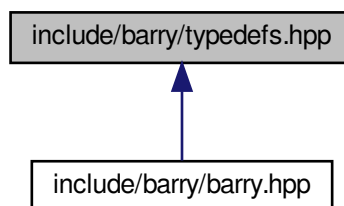
## 9.62 include/barry/typedefs.hpp File Reference

```
#include "barry-configuration.hpp"  
#include "barry-debug.hpp"  
#include "progress.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 std::vector< std::pair< std::vector< double >, size\_t > > [Counts\\_type](#)
- template<typename Cell\_Type >  
using [Row\\_type](#) = [Map](#)< size\_t, [Cell](#)< Cell\_Type > >
- template<typename Cell\_Type >  
using [Col\\_type](#) = [Map](#)< size\_t, [Cell](#)< Cell\_Type > \* >
- template<typename Ta = double, typename Tb = size\_t>  
using [MapVec\\_type](#) = std::unordered\_map< std::vector< Ta >, Tb, [vecHasher](#)< Ta > >
- template<typename Array\_Type , typename Data\_Type >  
using [Hasher\\_fun\\_type](#) = std::function< std::vector< double >(const Array\_Type &, Data\_Type \*)>  
*Hasher function used by the counter.*
- template<typename Array\_Type , typename Data\_Type >  
using [Counter\\_fun\\_type](#) = std::function< double(const Array\_Type &, size\_t, size\_t, Data\_Type &)>  
*Counter and rule functions.*
- template<typename Array\_Type , typename Data\_Type >  
using [Rule\\_fun\\_type](#) = std::function< bool(const Array\_Type &, size\_t, size\_t, Data\_Type &)>

## Functions

- std::vector< size\_t > [sort\\_array](#) (const double \*v, size\_t start, size\_t ncols, size\_t nrows)  
*Ascending sorting an array.*
- template<typename T >  
T [vec\\_inner\\_prod](#) (const T \*a, const T \*b, size\_t n)
- template<> double [vec\\_inner\\_prod](#) (const double \*a, const double \*b, size\_t n)
- 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-100)

## 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

## 9.62.1 Typedef Documentation

### 9.62.1.1 Col\_type

```
template<typename Cell_Type >
using Col_type = Map< size_t, Cell<Cell_Type>* >
```

Definition at line 70 of file typedefs.hpp.

### 9.62.1.2 Counter\_fun\_type

```
template<typename Array_Type , typename Data_Type >
using Counter_fun_type = std::function<double(const Array_Type &, size_t, size_t, Data_Type
&)>
```

[Counter](#) and rule functions.

#### Parameters

<i>Array_Type</i>	a <a href="#">BArray</a>
<i>unit, size_t</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 187 of file typedefs.hpp.

### 9.62.1.3 Counts\_type

```
typedef std::vector< std::pair< std::vector<double>, size_t > > Counts_type
```

Definition at line 51 of file typedefs.hpp.

### 9.62.1.4 Hasher\_fun\_type

```
template<typename Array_Type , typename Data_Type >
using Hasher_fun_type = std::function<std::vector<double>(const Array_Type &, Data_Type *)>
```

Hasher function used by the counter.

Used to characterize the support of the array.

## Template Parameters

<i>Array_Type</i>	
-------------------	--

Definition at line 200 of file typedefs.hpp.

### 9.62.1.5 MapVec\_type

```
template<typename Ta = double, typename Tb = size_t>
using MapVec_type = std::unordered_map< std::vector< Ta >, Tb, vecHasher<Ta> >
```

Definition at line 128 of file typedefs.hpp.

### 9.62.1.6 Row\_type

```
template<typename Cell_Type >
using Row_type = Map< size_t, Cell<Cell_Type> >
```

Definition at line 67 of file typedefs.hpp.

### 9.62.1.7 Rule\_fun\_type

```
template<typename Array_Type , typename Data_Type >
using Rule_fun_type = std::function<bool(const Array_Type &, size_t, size_t, Data_Type &)>
```

Definition at line 190 of file typedefs.hpp.

## 9.62.2 Function Documentation

### 9.62.2.1 sort\_array()

```
std::vector< size_t > sort_array (
    const double * v,
    size_t start,
    size_t ncols,
    size_t nrows ) [inline]
```

Ascending sorting an array.

It will sort an array solving ties using the next column. Data is stored column-wise.



## Template Parameters

<i>T</i>	
----------	--

## Parameters

<i>v</i>	
<i>nrows</i>	

## Returns

std::vector<size\_t> The sorting index.

Definition at line 141 of file typedefs.hpp.

### 9.62.2.2 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 210 of file typedefs.hpp.

### 9.62.2.3 vec\_equal\_approx()

```
template<typename T >
bool vec_equal_approx (
    const std::vector< T > & a,
    const std::vector< T > & b,
    double eps = 1e-100 ) [inline]
```

Definition at line 235 of file typedefs.hpp.

#### 9.62.2.4 `vec_inner_prod()` [1/2]

```
template<>
double vec_inner_prod (
    const double * a,
    const double * b,
    size_t n ) [inline]
```

Definition at line 286 of file typedefs.hpp.

#### 9.62.2.5 `vec_inner_prod()` [2/2]

```
template<typename T >
T vec_inner_prod (
    const T * a,
    const T * b,
    size_t n ) [inline]
```

Definition at line 263 of file typedefs.hpp.

## 9.63 README.md File Reference

# Index

- ~BArray
  - BArray< Cell\_Type, Data\_Type >, [63](#)
- ~BArrayCell
  - BArrayCell< Cell\_Type, Data\_Type >, [74](#)
- ~BArrayCell\_const
  - BArrayCell\_const< Cell\_Type, Data\_Type >, [77](#)
- ~BArrayDense
  - BArrayDense< Cell\_Type, Data\_Type >, [83](#)
- ~BArrayDenseCell
  - BArrayDenseCell< Cell\_Type, Data\_Type >, [97](#)
- ~BArrayRow
  - BArrayRow< Cell\_Type, Data\_Type >, [110](#)
- ~BArrayRow\_const
  - BArrayRow\_const< Cell\_Type, Data\_Type >, [112](#)
- ~BArrayVector
  - BArrayVector< Cell\_Type, Data\_Type >, [115](#)
- ~BArrayVector\_const
  - BArrayVector\_const< Cell\_Type, Data\_Type >, [118](#)
- ~Cell
  - Cell< Cell\_Type >, [122](#)
- ~ConstBArrayRowIter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [127](#)
- ~Counter
  - Counter< Array\_Type, Data\_Type >, [130](#)
- ~Counters
  - Counters< Array\_Type, Data\_Type >, [135](#)
- ~DEFMCounterData
  - Phylo rules, [43](#)
- ~DEFMData
  - Phylo rules, [43](#)
- ~DEFMRuleDynData
  - Phylo rules, [43](#)
- ~Entries
  - Entries< Cell\_Type >, [149](#)
- ~Flock
  - Flock, [151](#)
- ~FreqTable
  - FreqTable< T >, [158](#)
- ~Geese
  - Geese, [164](#)
- ~Model
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [180](#)
- ~NetCounterData
  - NetCounterData, [199](#)
- ~NetworkData
  - NetworkData, [201](#)
- ~Node
  - Node, [204](#)
- ~PhyloRuleDynData
  - PhyloRuleDynData, [213](#)
- ~PowerSet
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [216](#)
- ~Progress
  - Progress, [221](#)
- ~Rule
  - Rule< Array\_Type, Data\_Type >, [223](#)
- ~Rules
  - Rules< Array\_Type, Data\_Type >, [226](#)
- ~StatsCounter
  - StatsCounter< Array\_Type, Data\_Type >, [231](#)
- ~Support
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [236](#)
- active
  - Cell< Cell\_Type >, [125](#)
- add
  - Cell< Cell\_Type >, [123](#), [124](#)
  - FreqTable< T >, [158](#)
- add\_array
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [180](#)
- add\_counter
  - Counters< Array\_Type, Data\_Type >, [135](#), [136](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [180](#), [181](#)
  - StatsCounter< Array\_Type, Data\_Type >, [231](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [236](#)
- add\_data
  - Flock, [151](#)
- add\_dims
  - counters-meat.hpp, [283](#)
- add\_hash
  - Counters< Array\_Type, Data\_Type >, [136](#)
- add\_hasher
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [181](#)
- add\_rule

- Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
181
- PowerSet< Array\_Type, Data\_Rule\_Type >, 216,  
217
- Rules< Array\_Type, Data\_Type >, 227
- Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
236, 237
- add\_rule\_dyn
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
181, 182
  - Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
237
- annotations
  - Node, 205
- Array
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >,  
127
- array
  - Node, 205
  - Phylo rules, 44
- arrays
  - Node, 205
- arrays2support
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
192
- AS\_ONE
  - EXISTS, 57
- as\_vector
  - FreqTable< T >, 158
- AS\_ZERO
  - EXISTS, 57
- at
  - Phylo rules, 39
  - PhyloCounterData, 210
- BArray
  - BArray< Cell\_Type, Data\_Type >, 62, 63
- BArray< Cell\_Type, Data\_Type >, 59
  - ~BArray, 63
  - BArray, 62, 63
  - BArrayCell< Cell\_Type, Data\_Type >, 73
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 73
  - clear, 63
  - col, 63
  - D, 64
  - D\_ptr, 64
  - default\_val, 64
  - flush\_data, 64
  - get\_cell, 64
  - get\_col\_vec, 65
  - get\_entries, 65
  - get\_row\_vec, 65
  - insert\_cell, 66
  - is\_dense, 66
  - is\_empty, 66
  - ncol, 67
  - nnozero, 67
  - nrow, 67
  - operator\*=: 67
  - operator(), 67
  - operator+=, 68
  - operator-=, 68
  - operator/=: 69
  - operator=: 69
  - operator==, 69
  - out\_of\_range, 69
  - print, 69
  - print\_n, 70
  - reserve, 70
  - resize, 70
  - rm\_cell, 70
  - row, 70
  - set\_data, 71
  - swap\_cells, 71
  - swap\_cols, 71
  - swap\_rows, 71
  - toggle\_cell, 72
  - toggle\_lock, 72
  - transpose, 72
  - visited, 73
  - zero\_col, 72
  - zero\_row, 72
- barray-meat-operators.hpp
  - BARRAY\_TEMPLATE, 248–250
  - BARRAY\_TEMPLATE\_ARGS, 249, 251
  - BARRAY\_TYPE, 249, 251
  - COL, 249
  - for, 251
  - operator(), 251
  - rhs, 251
  - ROW, 249
  - this, 252
- barray-meat.hpp
  - COL, 252
  - ROW, 253
- BARRAY\_TEMPLATE
  - barray-meat-operators.hpp, 248–250
- BARRAY\_TEMPLATE\_ARGS
  - barray-meat-operators.hpp, 249, 251
- BARRAY\_TYPE
  - barray-meat-operators.hpp, 249, 251
- BArrayCell
  - BArrayCell< Cell\_Type, Data\_Type >, 74
- BArrayCell< Cell\_Type, Data\_Type >, 74
  - ~BArrayCell, 74
  - BArray< Cell\_Type, Data\_Type >, 73
  - BArrayCell, 74
  - operator Cell\_Type, 75
  - operator\*=: 75
  - operator+=, 75
  - operator-=, 75
  - operator/=: 75

- operator=, 75
- operator==, 76
- BArrayCell\_const
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 77
- BArrayCell\_const< Cell\_Type, Data\_Type >, 76
  - ~BArrayCell\_const, 77
  - BArray< Cell\_Type, Data\_Type >, 73
  - BArrayCell\_const, 77
  - operator Cell\_Type, 77
  - operator!=, 77
  - operator<, 77
  - operator<=, 78
  - operator>, 78
  - operator>=, 78
  - operator==, 78
- BArrayDense
  - BArrayDense< Cell\_Type, Data\_Type >, 82, 83
- BArrayDense< Cell\_Type, Data\_Type >, 79
  - ~BArrayDense, 83
  - BArrayDense, 82, 83
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 95, 99
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 95, 102
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 95
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 95, 106
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 95
  - clear, 83
  - col, 84
  - colsum, 84
  - D, 84
  - D\_ptr, 85
  - default\_val, 85
  - get\_cell, 85
  - get\_col\_vec, 85, 86
  - get\_data, 86
  - get\_entries, 86
  - get\_row\_vec, 86, 87
  - insert\_cell, 87
  - is\_dense, 87
  - is\_empty, 88
  - ncol, 88
  - nnozero, 88
  - nrow, 88
  - operator\*=, 89
  - operator(), 88, 89
  - operator+=, 89
  - operator-=, 90
  - operator/=, 90
  - operator=, 90, 91
  - operator==, 91
  - out\_of\_range, 91
  - print, 91
  - reserve, 91
  - resize, 92
  - rm\_cell, 92
  - row, 92
  - rowsum, 92
  - set\_data, 93
  - swap\_cells, 93
  - swap\_cols, 93
  - swap\_rows, 93
  - toggle\_cell, 94
  - toggle\_lock, 94
  - transpose, 94
  - visited, 96
  - zero\_col, 94
  - zero\_row, 94
- barraydense-meat-operators.hpp
  - BDENSE\_TEMPLATE, 255, 257
  - BDENSE\_TEMPLATE\_ARGS, 255, 257
  - BDENSE\_TYPE, 256, 258
  - COL, 256
  - POS, 256
  - POS\_N, 256
  - ROW, 256
- barraydense-meat.hpp
  - COL, 258
  - POS, 258
  - POS\_N, 259
  - ROW, 259
  - ZERO\_CELL, 259
- BArrayDenseCell
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 97
- BArrayDenseCell< Cell\_Type, Data\_Type >, 96
  - ~BArrayDenseCell, 97
  - BArrayDense< Cell\_Type, Data\_Type >, 95, 99
  - BArrayDenseCell, 97
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 99, 102
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 99, 104
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 106
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 109
  - operator Cell\_Type, 97
  - operator\*=, 97
  - operator+=, 98
  - operator-=, 98
  - operator/=, 98
  - operator=, 98
  - operator==, 99
- barraydensecell-bones.hpp
  - POS, 260
- barraydensecell-meat.hpp
  - POS, 260
- BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 100
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 102
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 104
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 106
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 109

- BArrayDenseCol
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 100
- BArrayDenseCol< Cell\_Type, Data\_Type >, 100
  - BArrayDense< Cell\_Type, Data\_Type >, 95, 102
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 99, 102
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 102
  - BArrayDenseCol, 100
  - begin, 101
  - end, 101
  - operator(), 101
  - size, 101
- barraydensecol-bones.hpp
  - POS, 261
  - POS\_N, 261
  - ZERO\_CELL, 262
- BArrayDenseCol\_const
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 103
- BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 102
  - BArrayDense< Cell\_Type, Data\_Type >, 95
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 99, 104
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 104
  - BArrayDenseCol\_const, 103
  - begin, 103
  - end, 103
  - operator(), 103
  - size, 104
- BArrayDenseRow
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 105
- BArrayDenseRow< Cell\_Type, Data\_Type >, 104
  - BArrayDense< Cell\_Type, Data\_Type >, 95, 106
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 106
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 106
  - BArrayDenseRow, 105
  - begin, 105
  - end, 105
  - operator(), 106
  - size, 106
- barraydenserow-bones.hpp
  - POS, 263
  - POS\_N, 263
  - ZERO\_CELL, 263
- BArrayDenseRow\_const
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 107
- BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 107
  - BArrayDense< Cell\_Type, Data\_Type >, 95
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 109
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 109
  - BArrayDenseRow\_const, 107
  - begin, 108
  - end, 108
  - operator(), 108
  - size, 108
- BArrayRow
  - BArrayRow< Cell\_Type, Data\_Type >, 110
- BArrayRow< Cell\_Type, Data\_Type >, 109
  - ~BArrayRow, 110
  - BArrayRow, 110
  - operator BArrayRow< Cell\_Type, Data\_Type >, 110
  - operator\*=: 110
  - operator+=, 110
  - operator-=, 110
  - operator/=: 111
  - operator=, 111
  - operator==, 111
- barrayrow-meat.hpp
  - BROW\_TEMPLATE, 264, 265
  - BROW\_TEMPLATE\_ARGS, 264
  - BROW\_TYPE, 264
- BArrayRow\_const
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 112
- BArrayRow\_const< Cell\_Type, Data\_Type >, 111
  - ~BArrayRow\_const, 112
  - BArrayRow\_const, 112
  - operator BArrayRow\_const< Cell\_Type, Data\_Type >, 112
  - operator!=, 112
  - operator<, 112
  - operator<=: 113
  - operator>, 113
  - operator>=: 113
  - operator==, 113
- BArrayVector
  - BArrayVector< Cell\_Type, Data\_Type >, 114
- BArrayVector< Cell\_Type, Data\_Type >, 113
  - ~BArrayVector, 115
  - BArrayVector, 114
  - begin, 115
  - end, 115
  - is\_col, 115
  - is\_row, 115
  - operator std::vector< Cell\_Type >, 116
  - operator\*=: 116
  - operator+=, 116
  - operator-=, 116
  - operator/=: 116
  - operator=, 117
  - operator==, 117
  - size, 117
- BArrayVector\_const
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 118
- BArrayVector\_const< Cell\_Type, Data\_Type >, 117
  - ~BArrayVector\_const, 118
  - BArrayVector\_const, 118
  - begin, 119
  - end, 119

- is\_col, [119](#)
- is\_row, [119](#)
- operator std::vector< Cell\_Type >, [119](#)
- operator!=, [119](#)
- operator<, [120](#)
- operator<=, [120](#)
- operator>, [120](#)
- operator>=, [120](#)
- operator==, [120](#)
- size, [121](#)
- barry, [55](#)
- barry-configuration.hpp
  - BARRY\_CHECK\_SUPPORT, [267](#)
  - BARRY\_ISFINITE, [267](#)
  - BARRY\_MAX\_NUM\_ELEMENTS, [267](#)
  - BARRY\_SAFE\_EXP, [267](#)
  - Map, [268](#)
  - printf\_barry, [267](#)
- barry-debug.hpp
  - BARRY\_DEBUG\_LEVEL, [268](#)
- barry-macros.hpp
  - BARRY\_NCORES\_ARG, [269](#)
  - BARRY\_ONE, [269](#)
  - BARRY\_ONE\_DENSE, [269](#)
  - BARRY\_UNUSED, [270](#)
  - BARRY\_ZERO, [270](#)
  - BARRY\_ZERO\_DENSE, [270](#)
- barry.hpp
  - BARRY\_HPP, [272](#)
  - BARRY\_VERSION, [272](#)
  - BARRY\_VERSION\_MAYOR, [272](#)
  - BARRY\_VERSION\_MINOR, [272](#)
  - COUNTER\_FUNCTION, [272](#)
  - COUNTER\_LAMBDA, [272](#)
  - RULE\_FUNCTION, [273](#)
  - RULE\_LAMBDA, [273](#)
- barry::counters, [55](#)
- barry::counters::network, [56](#)
- BARRY\_CHECK\_SUPPORT
  - barry-configuration.hpp, [267](#)
- BARRY\_DEBUG\_LEVEL
  - barry-debug.hpp, [268](#)
- BARRY\_HPP
  - barry.hpp, [272](#)
- BARRY\_ISFINITE
  - barry-configuration.hpp, [267](#)
- BARRY\_MAX\_NUM\_ELEMENTS
  - barry-configuration.hpp, [267](#)
- BARRY\_NCORES\_ARG
  - barry-macros.hpp, [269](#)
- BARRY\_ONE
  - barry-macros.hpp, [269](#)
- BARRY\_ONE\_DENSE
  - barry-macros.hpp, [269](#)
- BARRY\_PROGRESS\_BAR\_WIDTH
  - progress.hpp, [328](#)
- BARRY\_SAFE\_EXP
  - barry-configuration.hpp, [267](#)
- BARRY\_SUPPORT\_MEAT\_HPP
  - support-meat.hpp, [336](#)
- BARRY\_UNUSED
  - barry-macros.hpp, [270](#)
- BARRY\_VERSION
  - barry.hpp, [272](#)
- BARRY\_VERSION\_MAYOR
  - barry.hpp, [272](#)
- BARRY\_VERSION\_MINOR
  - barry.hpp, [272](#)
- BARRY\_ZERO
  - barry-macros.hpp, [270](#)
- BARRY\_ZERO\_DENSE
  - barry-macros.hpp, [270](#)
- BARRY\_ZERO\_NETWORK
  - network.hpp, [298](#)
- BARRY\_ZERO\_NETWORK\_DENSE
  - network.hpp, [298](#)
- BDENSE\_TEMPLATE
  - barraydense-meat-operators.hpp, [255](#), [257](#)
- BDENSE\_TEMPLATE\_ARGS
  - barraydense-meat-operators.hpp, [255](#), [257](#)
- BDENSE\_TYPE
  - barraydense-meat-operators.hpp, [256](#), [258](#)
- begin
  - BArrayDenseCol< Cell\_Type, Data\_Type >, [101](#)
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, [103](#)
  - BArrayDenseRow< Cell\_Type, Data\_Type >, [105](#)
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, [108](#)
  - BArrayVector< Cell\_Type, Data\_Type >, [115](#)
  - BArrayVector\_const< Cell\_Type, Data\_Type >, [119](#)
  - PhyloCounterData, [210](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [217](#)
  - Rules< Array\_Type, Data\_Type >, [227](#)
- blengths
  - NodeData, [208](#)
- BOTH
  - CHECK, [56](#)
  - EXISTS, [57](#)
- BROW\_TEMPLATE
  - barrayrow-meat.hpp, [264](#), [265](#)
- BROW\_TEMPLATE\_ARGS
  - barrayrow-meat.hpp, [264](#)
- BROW\_TYPE
  - barrayrow-meat.hpp, [264](#)
- calc
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [217](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [237](#)
- calc\_reduced\_sequence
  - Geese, [164](#)
- calc\_sequence
  - Geese, [165](#)
- Cell

- Cell< Cell\_Type >, 122, 123
- Cell< Cell\_Type >, 121
  - ~Cell, 122
  - active, 125
  - add, 123, 124
  - Cell, 122, 123
  - operator Cell\_Type, 124
  - operator!=, 124
  - operator=, 124, 125
  - operator==, 125
  - value, 125
  - visited, 125
- Cell\_const< Cell\_Type >, 126
- change\_stats
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 241
- CHECK, 56
  - BOTH, 56
  - NONE, 56
  - ONE, 56
  - TWO, 56
- clear
  - BArray< Cell\_Type, Data\_Type >, 63
  - BArrayDense< Cell\_Type, Data\_Type >, 83
  - FreqTable< T >, 158
  - statscounter-meat.hpp, 331
- COL
  - barray-meat-operators.hpp, 249
  - barray-meat.hpp, 252
  - barraydense-meat-operators.hpp, 256
  - barraydense-meat.hpp, 258
- col
  - BArray< Cell\_Type, Data\_Type >, 63
  - BArrayDense< Cell\_Type, Data\_Type >, 84
- Col\_type
  - typedefs.hpp, 339
- colnames
  - Flock, 152
  - Geese, 165
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 182
- colsum
  - BArrayDense< Cell\_Type, Data\_Type >, 84
- conditional\_prob
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 182
- ConstBArrayRowIter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, 127
- ConstBArrayRowIter< Cell\_Type, Data\_Type >, 126
  - ~ConstBArrayRowIter, 127
  - Array, 127
  - ConstBArrayRowIter, 127
  - current\_col, 128
  - current\_row, 128
  - iter, 128
- coordinantes\_n\_free
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 241
- coordinantes\_n\_locked
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 242
- coordinates\_free
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 219
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 242
- coordinates\_locked
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 219
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 242
- count
  - Counter< Array\_Type, Data\_Type >, 131
- count\_all
  - StatsCounter< Array\_Type, Data\_Type >, 231
- count\_current
  - StatsCounter< Array\_Type, Data\_Type >, 232
- count\_fun
  - Counter< Array\_Type, Data\_Type >, 132
  - counters-meat.hpp, 278
- count\_fun\_
  - counters-meat.hpp, 283
- count\_init
  - StatsCounter< Array\_Type, Data\_Type >, 232
- Counter
  - Counter< Array\_Type, Data\_Type >, 130
- counter
  - counters-meat.hpp, 284
  - statscounter-meat.hpp, 333
- Counter< Array\_Type, Data\_Type >, 128
  - ~Counter, 130
  - count, 131
  - count\_fun, 132
  - Counter, 130
  - data, 132
  - desc, 133
  - get\_description, 131
  - get\_hasher, 131
  - get\_name, 131
  - hasher\_fun, 133
  - init, 131
  - init\_fun, 133
  - name, 133
  - operator=, 131, 132
  - set\_hasher, 132
- counter\_
  - counters-meat.hpp, 284
- counter\_absdiff
  - Network counters, 26
- counter\_co\_opt



- Counting, [17](#)
- Phylo counters, [48](#)
- counter\_cogain
  - Counting, [18](#)
  - Phylo counters, [49](#)
- counter\_css\_census01
  - network-css.hpp, [291](#)
- counter\_css\_census02
  - network-css.hpp, [291](#)
- counter\_css\_census03
  - network-css.hpp, [291](#)
- counter\_css\_census04
  - network-css.hpp, [292](#)
- counter\_css\_census05
  - network-css.hpp, [292](#)
- counter\_css\_census06
  - network-css.hpp, [292](#)
- counter\_css\_census07
  - network-css.hpp, [292](#)
- counter\_css\_census08
  - network-css.hpp, [293](#)
- counter\_css\_census09
  - network-css.hpp, [293](#)
- counter\_css\_census10
  - network-css.hpp, [293](#)
- counter\_css\_completely\_false\_recip\_comiss
  - network-css.hpp, [293](#)
- counter\_css\_completely\_false\_recip\_omiss
  - network-css.hpp, [294](#)
- counter\_css\_mixed\_recip
  - network-css.hpp, [294](#)
- counter\_css\_partially\_false\_recip\_commi
  - network-css.hpp, [294](#)
- counter\_css\_partially\_false\_recip\_omiss
  - network-css.hpp, [295](#)
- counter\_ctriads
  - Network counters, [27](#)
- counter\_degree
  - Network counters, [27](#)
- counter\_deleted
  - statscounter-meat.hpp, [334](#)
- counter\_density
  - Network counters, [27](#)
- counter\_diff
  - Network counters, [28](#)
- counter\_edges
  - Network counters, [28](#)
- counter\_fixed\_effect
  - Network counters, [28](#)
- counter\_fun
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
[192](#)
- Counter\_fun\_type
  - typedefs.hpp, [339](#)
- COUNTER\_FUNCTION
  - barry.hpp, [272](#)
- counter\_gains
  - Counting, [18](#)
  - Phylo counters, [49](#)
- counter\_gains\_from\_0
  - Counting, [18](#)
  - Phylo counters, [49](#)
- counter\_gains\_k\_offspring
  - Counting, [19](#)
  - Phylo counters, [49](#)
- counter\_genes\_changing
  - Counting, [19](#)
  - Phylo counters, [50](#)
- counter\_idegree
  - Network counters, [29](#)
- counter\_idegree15
  - Network counters, [29](#)
- counter\_isolates
  - Network counters, [29](#), [30](#)
- counter\_istar2
  - Network counters, [30](#)
- counter\_k\_genes\_changing
  - Counting, [19](#)
  - Phylo counters, [50](#)
- COUNTER\_LAMBDA
  - barry.hpp, [272](#)
- counter\_less\_than\_p\_prop\_genes\_changing
  - Counting, [19](#)
  - Phylo counters, [50](#)
- counter\_logit\_intercept
  - Network counters, [30](#)
- counter\_longest
  - Counting, [20](#)
  - Phylo counters, [50](#)
- counter\_loss
  - Counting, [20](#)
  - Phylo counters, [51](#)
- counter\_maxfun
  - Counting, [20](#)
  - Phylo counters, [51](#)
- counter\_mutual
  - Network counters, [30](#)
- counter\_neofun
  - Counting, [20](#)
  - Phylo counters, [51](#)
- counter\_neofun\_a2b
  - Counting, [21](#)
  - Phylo counters, [51](#)
- counter\_nodecov
  - Network counters, [31](#)
- counter\_nodeicov
  - Network counters, [31](#)
- counter\_nodematch
  - Network counters, [31](#)
- counter\_nodecov
  - Network counters, [31](#)
- counter\_odegree
  - Network counters, [31](#), [32](#)
- counter\_odegree15
  - Network counters, [32](#)

- counter\_ones
  - Network counters, [32](#)
- counter\_ostar2
  - Network counters, [33](#)
- counter\_overall\_changes
  - Counting, [21](#)
  - Phylo counters, [52](#)
- counter\_overall\_gains
  - Counting, [21](#)
  - Phylo counters, [52](#)
- counter\_overall\_gains\_from\_0
  - Counting, [21](#)
  - Phylo counters, [52](#)
- counter\_overall\_loss
  - Counting, [22](#)
  - Phylo counters, [52](#)
- counter\_pairwise\_first\_gain
  - Counting, [22](#)
  - Phylo counters, [53](#)
- counter\_pairwise\_neofun\_singlefun
  - Counting, [22](#)
  - Phylo counters, [53](#)
- counter\_pairwise\_overall\_change
  - Counting, [22](#)
  - Phylo counters, [53](#)
- counter\_pairwise\_preserving
  - Counting, [23](#)
  - Phylo counters, [53](#)
- counter\_preserve\_pseudogene
  - Counting, [23](#)
  - Phylo counters, [54](#)
- counter\_prop\_genes\_changing
  - Counting, [23](#)
  - Phylo counters, [54](#)
- counter\_subfun
  - Counting, [23](#)
  - Phylo counters, [54](#)
- COUNTER\_TEMPLATE
  - counters-meat.hpp, [276, 278, 279](#)
- COUNTER\_TEMPLATE\_ARGS
  - counters-meat.hpp, [276](#)
- counter\_transition
  - Network counters, [33](#)
- counter\_transition\_formula
  - Network counters, [34](#)
- counter\_ttriads
  - Network counters, [34](#)
- COUNTER\_TYPE
  - counters-meat.hpp, [277](#)
- Counters
  - Counters< Array\_Type, Data\_Type >, [134, 135](#)
- counters
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [192](#)
  - statscounter-meat.hpp, [334](#)
- Counters< Array\_Type, Data\_Type >, [134](#)
- ~Counters, [135](#)
- add\_counter, [135, 136](#)
- add\_hash, [136](#)
- Counters, [134, 135](#)
- gen\_hash, [136](#)
- get\_descriptions, [136](#)
- get\_names, [137](#)
- operator=, [137](#)
- operator[], [138](#)
- size, [138](#)
- counters-meat.hpp
  - add\_dims, [283](#)
  - count\_fun, [278](#)
  - count\_fun\_, [283](#)
  - counter, [284](#)
  - counter\_, [284](#)
  - COUNTER\_TEMPLATE, [276, 278, 279](#)
  - COUNTER\_TEMPLATE\_ARGS, [276](#)
  - COUNTER\_TYPE, [277](#)
  - COUNTERS\_TEMPLATE, [277, 279–281](#)
  - COUNTERS\_TEMPLATE\_ARGS, [277](#)
  - COUNTERS\_TYPE, [277](#)
  - data, [281](#)
  - data\_, [284](#)
  - desc, [281](#)
  - desc\_, [284](#)
  - for, [281](#)
  - fun, [285](#)
  - fun\_, [285](#)
  - hasher, [281, 282](#)
  - hasher\_fun, [282](#)
  - hasher\_fun\_, [285](#)
  - i, [285](#)
  - if, [282](#)
  - init\_fun, [283](#)
  - init\_fun\_, [286](#)
  - j, [286](#)
  - name, [283](#)
  - name\_, [286](#)
  - noexcept, [286](#)
  - res, [286](#)
  - return, [287](#)
  - TMP\_HASHER\_CALL, [277](#)
- counters.hpp
  - DEFM\_COUNTER, [306](#)
  - DEFM\_COUNTER\_LAMBDA, [307](#)
  - DEFM\_RULE, [307](#)
  - DEFM\_RULE\_LAMBDA, [307](#)
  - DEFM\_RULEDYN\_LAMBDA, [307](#)
  - UNI\_SUB, [308](#)
- counters\_
  - statscounter-meat.hpp, [334](#)
- COUNTERS\_TEMPLATE
  - counters-meat.hpp, [277, 279–281](#)
- COUNTERS\_TEMPLATE\_ARGS
  - counters-meat.hpp, [277](#)
- COUNTERS\_TYPE
  - counters-meat.hpp, [277](#)
- Counting, [13](#)

- counter\_co\_opt, [17](#)
- counter\_cogain, [18](#)
- counter\_gains, [18](#)
- counter\_gains\_from\_0, [18](#)
- counter\_gains\_k\_offspring, [19](#)
- counter\_genes\_changing, [19](#)
- counter\_k\_genes\_changing, [19](#)
- counter\_less\_than\_p\_prop\_genes\_changing, [19](#)
- counter\_longest, [20](#)
- counter\_loss, [20](#)
- counter\_maxfun, [20](#)
- counter\_neofun, [20](#)
- counter\_neofun\_a2b, [21](#)
- counter\_overall\_changes, [21](#)
- counter\_overall\_gains, [21](#)
- counter\_overall\_gains\_from\_0, [21](#)
- counter\_overall\_loss, [22](#)
- counter\_pairwise\_first\_gain, [22](#)
- counter\_pairwise\_neofun\_singlefun, [22](#)
- counter\_pairwise\_overall\_change, [22](#)
- counter\_pairwise\_preserving, [23](#)
- counter\_preserve\_pseudogene, [23](#)
- counter\_prop\_genes\_changing, [23](#)
- counter\_subfun, [23](#)
- get\_last\_name, [24](#)
- IF\_MATCHES, [15](#)
- IF\_NOTMATCHES, [15](#)
- IS\_DUPLICATION, [15](#)
- IS\_EITHER, [15](#)
- IS\_SPECIATION, [16](#)
- MAKE\_DEFM\_HASHER, [16](#)
- MAKE\_DUPL\_VARS, [16](#)
- PHYLO\_CHECK\_MISSING, [16](#)
- PHYLO\_COUNTER\_LAMBDA, [17](#)
- PHYLO\_RULE\_DYN\_LAMBDA, [17](#)
- counts
  - Phylo rules, [44](#)
  - PhyloRuleDynData, [213](#)
- Counts\_type
  - typedefs.hpp, [339](#)
- covar\_sort
  - Phylo rules, [44](#)
- covar\_used
  - Phylo rules, [44](#)
- covariates
  - Phylo rules, [44](#)
- CSS\_APPEND
  - network-css.hpp, [288](#)
- CSS\_CASE\_ELSE
  - network-css.hpp, [289](#)
- CSS\_CASE\_PERCEIVED
  - network-css.hpp, [289](#)
- CSS\_CASE\_TRUTH
  - network-css.hpp, [289](#)
- CSS\_CHECK\_SIZE
  - network-css.hpp, [289](#)
- CSS\_CHECK\_SIZE\_INIT
  - network-css.hpp, [289](#)
- CSS\_MATCH\_TYPE
  - network-css.hpp, [290](#)
- CSS\_NET\_COUNTER\_LAMBDA\_INIT
  - network-css.hpp, [290](#)
- CSS\_PERCEIVED\_CELLS
  - network-css.hpp, [290](#)
- CSS\_SIZE
  - network-css.hpp, [290](#)
- CSS\_TRUE\_CELLS
  - network-css.hpp, [291](#)
- current\_col
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [128](#)
- current\_row
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [128](#)
- current\_stats
  - statscounter-meat.hpp, [334](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [242](#)
- D
  - BArray< Cell\_Type, Data\_Type >, [64](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [84](#)
  - Rule< Array\_Type, Data\_Type >, [224](#)
- D\_ptr
  - BArray< Cell\_Type, Data\_Type >, [64](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [85](#)
- dat
  - Flock, [156](#)
- data
  - Counter< Array\_Type, Data\_Type >, [132](#)
  - counters-meat.hpp, [281](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [219](#)
- data\_
  - counters-meat.hpp, [284](#)
- default\_val
  - BArray< Cell\_Type, Data\_Type >, [64](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [85](#)
- DEFM, [139](#)
  - DEFM, [140](#)
  - get\_column\_major, [140](#)
  - get\_ID, [140](#)
  - get\_m\_order, [140](#)
  - get\_model, [141](#)
  - get\_n\_covars, [141](#)
  - get\_n\_obs, [141](#)
  - get\_n\_rows, [141](#)
  - get\_n\_y, [141](#)
  - get\_X, [141](#)
  - get\_X\_names, [142](#)
  - get\_Y, [142](#)
  - get\_Y\_names, [142](#)
  - init, [142](#)
  - is\_motif, [142](#)
  - logodds, [142](#)
  - motif\_census, [143](#)
  - print, [143](#)

- set\_names, [143](#)
- simulate, [143](#)
- defm, [57](#)
- defm-meat.hpp
  - DEFM\_LOOP\_ARRAYS, [311](#)
  - DEFM\_RANGES, [311](#)
  - keygen\_defm, [312](#)
- defm-types.hpp
  - DEFMArray, [313](#)
- DEFM\_COUNTER
  - counters.hpp, [306](#)
- DEFM\_COUNTER\_LAMBDA
  - counters.hpp, [307](#)
- DEFM\_LOOP\_ARRAYS
  - defm-meat.hpp, [311](#)
- defm\_motif\_parser
  - formula.hpp, [313](#)
- DEFM\_RANGES
  - defm-meat.hpp, [311](#)
- DEFM\_RULE
  - counters.hpp, [307](#)
- DEFM\_RULE\_LAMBDA
  - counters.hpp, [307](#)
- DEFM\_RULEDYN\_LAMBDA
  - counters.hpp, [307](#)
- DEFMArray
  - defm-types.hpp, [313](#)
- DEFMCounter
  - Phylo rules, [37](#)
- DEFMCounterData, [144](#)
  - Phylo rules, [39](#)
- DEFMCounters
  - Phylo rules, [37](#)
- DEFMData, [144](#)
  - Phylo rules, [39](#)
- DEFMModel
  - Phylo rules, [37](#)
- DEFMRule
  - Phylo rules, [38](#)
- DEFMRuleData, [146](#)
  - Phylo rules, [40](#)
- DEFMRuleDyn
  - Phylo rules, [38](#)
- DEFMRuleDynData, [147](#)
  - Phylo rules, [40](#)
- DEFMRules
  - Phylo rules, [38](#)
- DEFMRulesDyn
  - Phylo rules, [38](#)
- DEFMStatsCounter
  - Phylo rules, [38](#)
- DEFMSupport
  - Phylo rules, [38](#)
- delete\_counters
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [193](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [242](#)
- delete\_engine
  - Geese, [172](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [193](#)
- delete\_rules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [193](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [243](#)
- delete\_rules\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [193](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [243](#)
- delete\_support
  - Geese, [172](#)
- desc
  - Counter< Array\_Type, Data\_Type >, [133](#)
  - counters-meat.hpp, [281](#)
- desc\_
  - counters-meat.hpp, [284](#)
- directed
  - NetworkData, [201](#)
- duplication
  - Node, [206](#)
  - NodeData, [209](#)
  - PhyloRuleDynData, [213](#)
- empty
  - PhyloCounterData, [210](#)
- EmptyArray
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [219](#)
  - statscounter-meat.hpp, [334](#)
- end
  - BArrayDenseCol< Cell\_Type, Data\_Type >, [101](#)
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, [103](#)
  - BArrayDenseRow< Cell\_Type, Data\_Type >, [105](#)
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, [108](#)
  - BArrayVector< Cell\_Type, Data\_Type >, [115](#)
  - BArrayVector\_const< Cell\_Type, Data\_Type >, [119](#)
  - PhyloCounterData, [211](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [217](#)
  - Progress, [222](#)
  - Rules< Array\_Type, Data\_Type >, [227](#)
- Entries
  - Entries< Cell\_Type >, [148](#)
- Entries< Cell\_Type >, [148](#)
  - ~Entries, [149](#)
  - Entries, [148](#)

- resize, [149](#)
- source, [149](#)
- target, [149](#)
- val, [149](#)
- etype\_default
  - Geese, [172](#)
- etype\_duplication
  - Geese, [172](#)
- etype\_either
  - Geese, [172](#)
- etype\_speciation
  - Geese, [173](#)
- eval\_rules\_dyn
  - Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [238](#)
- EXISTS, [57](#)
  - AS\_ONE, [57](#)
  - AS\_ZERO, [57](#)
  - BOTH, [57](#)
  - NONE, [58](#)
  - ONE, [58](#)
  - TWO, [58](#)
  - UNKNOWN, [58](#)
- f\_
  - statscounter-meat.hpp, [335](#)
- first\_calc\_done
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [193](#)
- Flock, [150](#)
  - ~Flock, [151](#)
  - add\_data, [151](#)
  - colnames, [152](#)
  - dat, [156](#)
  - Flock, [151](#)
  - get\_counters, [152](#)
  - get\_model, [152](#)
  - get\_stats\_support, [152](#)
  - get\_stats\_target, [152](#)
  - get\_support\_fun, [153](#)
  - init, [153](#)
  - initialized, [156](#)
  - likelihood\_joint, [153](#)
  - model, [156](#)
  - nfunctions, [156](#)
  - nfuncs, [153](#)
  - nleaves, [154](#)
  - nnodes, [154](#)
  - nterms, [154](#)
  - ntrees, [154](#)
  - operator(), [154](#)
  - parse\_polytomies, [155](#)
  - print, [155](#)
  - rengine, [156](#)
  - set\_seed, [155](#)
  - support\_size, [155](#)
- flush\_data
  - BArray< Cell\_Type, Data\_Type >, [64](#)
- for
  - barray-meat-operators.hpp, [251](#)
  - counters-meat.hpp, [281](#)
  - statscounter-meat.hpp, [331](#)
- formula.hpp
  - defm\_motif\_parser, [313](#)
- FreqTable
  - FreqTable< T >, [158](#)
- FreqTable< T >, [157](#)
  - ~FreqTable, [158](#)
  - add, [158](#)
  - as\_vector, [158](#)
  - clear, [158](#)
  - FreqTable, [158](#)
  - get\_data, [159](#)
  - get\_index, [159](#)
  - make\_hash, [159](#)
  - print, [159](#)
  - reserve, [159](#)
  - size, [160](#)
- fun
  - counters-meat.hpp, [285](#)
- fun\_
  - counters-meat.hpp, [285](#)
- Geese, [160](#)
  - ~Geese, [164](#)
  - calc\_reduced\_sequence, [164](#)
  - calc\_sequence, [165](#)
  - colnames, [165](#)
  - delete\_engine, [172](#)
  - delete\_support, [172](#)
  - etype\_default, [172](#)
  - etype\_duplication, [172](#)
  - etype\_either, [172](#)
  - etype\_speciation, [173](#)
  - Geese, [163](#), [164](#)
  - get\_annotated\_nodes, [165](#)
  - get\_annotations, [165](#)
  - get\_counters, [165](#)
  - get\_model, [166](#)
  - get\_probabilities, [166](#)
  - get\_rengine, [166](#)
  - get\_states, [166](#)
  - get\_support\_fun, [166](#)
  - inherit\_support, [167](#)
  - init, [167](#)
  - init\_node, [167](#)
  - initialized, [173](#)
  - likelihood, [167](#)
  - likelihood\_exhaust, [167](#)
  - map\_to\_state\_id, [173](#)
  - nannotations, [168](#)
  - nfunctions, [173](#)
  - nfuncs, [168](#)
  - nleaves, [168](#)
  - nnodes, [168](#)
  - nodes, [173](#)

- nterms, 168
- observed\_counts, 169
- operator=, 169
- parse\_polytomies, 169
- predict, 169
- predict\_backend, 170
- predict\_exhaust, 170
- predict\_exhaust\_backend, 170
- predict\_sim, 170
- print, 170
- print\_nodes, 171
- print\_observed\_counts, 171
- pset\_loc, 173
- reduced\_sequence, 174
- sequence, 174
- set\_seed, 171
- simulate, 171
- support\_size, 171
- update\_annotations, 171
- geese, 58
- geese-bones.hpp
  - INITIALIZED, 317
  - keygen\_full, 317
  - RULE\_FUNCTION, 317
  - vec\_diff, 317
  - vector\_caster, 318
- geese-types.hpp
  - PhyloArray, 324
  - PhyloCounter, 324
  - PhyloCounters, 325
  - PhyloModel, 325
  - PhyloPowerSet, 325
  - PhyloRule, 325
  - PhyloRuleData, 325
  - PhyloRuleDyn, 325
  - PhyloRules, 326
  - PhyloRulesDyn, 326
  - PhyloStatsCounter, 326
  - PhyloSupport, 326
  - POS, 324
- gen\_hash
  - Counters< Array\_Type, Data\_Type >, 136
- gen\_key
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 183
- get\_annotated\_nodes
  - Geese, 165
- get\_annotations
  - Geese, 165
- get\_arrays2support
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 183
- get\_cell
  - BArray< Cell\_Type, Data\_Type >, 64
  - BArrayDense< Cell\_Type, Data\_Type >, 85
- get\_col\_vec
  - BArray< Cell\_Type, Data\_Type >, 65
  - BArrayDense< Cell\_Type, Data\_Type >, 85, 86
- get\_column\_major
  - DEFM, 140
- get\_counters
  - Flock, 152
  - Geese, 165
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 183
  - PhyloCounterData, 211
  - StatsCounter< Array\_Type, Data\_Type >, 232
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 238
- get\_counts
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 238
- get\_current\_stats
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 238
- get\_data
  - BArrayDense< Cell\_Type, Data\_Type >, 86
  - FreqTable< T >, 159
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 217
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 239
- get\_data\_ptr
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 218
- get\_description
  - Counter< Array\_Type, Data\_Type >, 131
  - Rule< Array\_Type, Data\_Type >, 224
- get\_descriptions
  - Counters< Array\_Type, Data\_Type >, 136
  - Rules< Array\_Type, Data\_Type >, 227
  - StatsCounter< Array\_Type, Data\_Type >, 232
- get\_entries
  - BArray< Cell\_Type, Data\_Type >, 65
  - BArrayDense< Cell\_Type, Data\_Type >, 86
- get\_hasher
  - Counter< Array\_Type, Data\_Type >, 131
- get\_ID
  - DEFM, 140
- get\_index
  - FreqTable< T >, 159
- get\_last\_name
  - Counting, 24
- get\_m\_order
  - DEFM, 140
- get\_model
  - DEFM, 141
  - Flock, 152
  - Geese, 166
- get\_n\_covars
  - DEFM, 141

- get\_n\_obs
  - DEFM, [141](#)
- get\_n\_rows
  - DEFM, [141](#)
- get\_n\_y
  - DEFM, [141](#)
- get\_name
  - Counter< Array\_Type, Data\_Type >, [131](#)
  - Rule< Array\_Type, Data\_Type >, [224](#)
- get\_names
  - Counters< Array\_Type, Data\_Type >, [137](#)
  - Rules< Array\_Type, Data\_Type >, [228](#)
  - StatsCounter< Array\_Type, Data\_Type >, [232](#)
- get\_normalizing\_constants
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [183](#)
- get\_parent
  - Node, [204](#)
- get\_probabilities
  - Geese, [166](#)
- get\_pset
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [184](#)
- get\_pset\_arrays
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [184](#)
- get\_pset\_probs
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [184](#)
- get\_pset\_stats
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [184](#)
- get\_rengine
  - Geese, [166](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [185](#)
- get\_row\_vec
  - BArray< Cell\_Type, Data\_Type >, [65](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [86](#), [87](#)
- get\_rules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [185](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [239](#)
- get\_rules\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [185](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [239](#)
- get\_seq
  - Rules< Array\_Type, Data\_Type >, [228](#)
- get\_states
  - Geese, [166](#)
- get\_stats\_support
  - Flock, [152](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [185](#)
- get\_stats\_target
  - Flock, [152](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [185](#)
- get\_support\_fun
  - Flock, [153](#)
  - Geese, [166](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [186](#)
- get\_X
  - DEFM, [141](#)
- get\_X\_names
  - DEFM, [142](#)
- get\_Y
  - DEFM, [142](#)
- get\_Y\_names
  - DEFM, [142](#)
- hasher
  - counters-meat.hpp, [281](#), [282](#)
- hasher\_fun
  - Counter< Array\_Type, Data\_Type >, [133](#)
  - counters-meat.hpp, [282](#)
- hasher\_fun\_
  - counters-meat.hpp, [285](#)
- Hasher\_fun\_type
  - typedefs.hpp, [339](#)
- hashes
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [243](#)
- hashes\_initialized
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [243](#)
- i
  - counters-meat.hpp, [285](#)
- id
  - Node, [206](#)
- idx
  - Phylo rules, [40](#), [41](#)
- if
  - counters-meat.hpp, [282](#)
- IF\_MATCHES
  - Counting, [15](#)
- IF\_NOTMATCHES



- Counting, 15
- include/barry/barray-bones.hpp, 247
- include/barry/barray-iterator.hpp, 247
- include/barry/barray-meat-operators.hpp, 248
- include/barry/barray-meat.hpp, 252
- include/barry/barraycell-bones.hpp, 253
- include/barry/barraycell-meat.hpp, 254
- include/barry/barraydense-bones.hpp, 254
- include/barry/barraydense-meat-operators.hpp, 255
- include/barry/barraydense-meat.hpp, 258
- include/barry/barraydensecell-bones.hpp, 259
- include/barry/barraydensecell-meat.hpp, 260
- include/barry/barraydensecol-bones.hpp, 261
- include/barry/barraydenserow-bones.hpp, 262
- include/barry/barrayrow-bones.hpp, 263
- include/barry/barrayrow-meat.hpp, 263
- include/barry/barrayvector-bones.hpp, 265
- include/barry/barrayvector-meat.hpp, 266
- include/barry/barry-configuration.hpp, 266
- include/barry/barry-debug.hpp, 268
- include/barry/barry-macros.hpp, 269
- include/barry/barry.hpp, 270
- include/barry/cell-bones.hpp, 273
- include/barry/cell-meat.hpp, 274
- include/barry/col-bones.hpp, 274
- include/barry/counters-bones.hpp, 274
- include/barry/counters-meat.hpp, 275
- include/barry/counters/network-css.hpp, 287
- include/barry/counters/network.hpp, 295
- include/barry/freqtable.hpp, 302
- include/barry/model-bones.hpp, 303
- include/barry/model-meat.hpp, 303
- include/barry/models/defm.hpp, 304
- include/barry/models/defm/counters.hpp, 305
- include/barry/models/defm/defm-bones.hpp, 310
- include/barry/models/defm/defm-meat.hpp, 311
- include/barry/models/defm/defm-types.hpp, 312
- include/barry/models/defm/formula.hpp, 313
- include/barry/models/geese.hpp, 315
- include/barry/models/geese/counters.hpp, 308
- include/barry/models/geese/flock-bones.hpp, 315
- include/barry/models/geese/flock-meat.hpp, 316
- include/barry/models/geese/geese-bones.hpp, 316
- include/barry/models/geese/geese-meat-constructors.hpp, 318
- include/barry/models/geese/geese-meat-likelihood.hpp, 319
- include/barry/models/geese/geese-meat-likelihood\_exhaust.hpp, 320
- include/barry/models/geese/geese-meat-predict.hpp, 320
- include/barry/models/geese/geese-meat-predict\_exhaust.hpp, 321
- include/barry/models/geese/geese-meat-predict\_sim.hpp, 321
- include/barry/models/geese/geese-meat-simulate.hpp, 322
- include/barry/models/geese/geese-meat.hpp, 322
- include/barry/models/geese/geese-node-bones.hpp, 323
- include/barry/models/geese/geese-types.hpp, 323
- include/barry/powerset-bones.hpp, 326
- include/barry/powerset-meat.hpp, 327
- include/barry/progress.hpp, 327
- include/barry/rules-bones.hpp, 328
- include/barry/rules-meat.hpp, 329
- include/barry/statscounter-bones.hpp, 329
- include/barry/statscounter-meat.hpp, 330
- include/barry/support-bones.hpp, 335
- include/barry/support-meat.hpp, 336
- include/barry/typedefs.hpp, 337
- indices
  - NetCounterData, 199
  - Phylo rules, 44, 45
- inherit\_support
  - Geese, 167
- init
  - Counter< Array\_Type, Data\_Type >, 131
  - DEFM, 142
  - Flock, 153
  - Geese, 167
  - Phylo rules, 45
- init\_fun
  - Counter< Array\_Type, Data\_Type >, 133
  - counters-meat.hpp, 283
- init\_fun\_
  - counters-meat.hpp, 286
- init\_node
  - Geese, 167
- init\_support
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 218
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 239
- INITIALIZED
  - geese-bones.hpp, 317
- initialized
  - Flock, 156
  - Geese, 173
- insert\_cell
  - BArray< Cell\_Type, Data\_Type >, 66
  - BArrayDense< Cell\_Type, Data\_Type >, 87
- is\_col
  - BArrayVector< Cell\_Type, Data\_Type >, 115
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 119
- is\_dense
  - BArray< Cell\_Type, Data\_Type >, 66
  - BArrayDense< Cell\_Type, Data\_Type >, 87
- IS\_DUPLICATION
  - Counting, 15
- IS\_EITHER
  - Counting, 15
- is\_empty
  - BArray< Cell\_Type, Data\_Type >, 66
  - BArrayDense< Cell\_Type, Data\_Type >, 88



- is\_leaf
  - Node, [205](#)
- is\_motif
  - DEFM, [142](#)
  - Phylo rules, [45](#)
- is\_row
  - BArrayVector< Cell\_Type, Data\_Type >, [115](#)
  - BArrayVector\_const< Cell\_Type, Data\_Type >, [119](#)
- IS\_SPECIATION
  - Counting, [16](#)
- is\_true
  - Phylo rules, [41](#)
- iter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [128](#)
- j
  - counters-meat.hpp, [286](#)
  - statscounter-meat.hpp, [335](#)
- keygen\_defm
  - defm-meat.hpp, [312](#)
- keygen\_full
  - geese-bones.hpp, [317](#)
- keys2support
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [194](#)
- lb
  - PhyloRuleDynData, [214](#)
- likelihood
  - Geese, [167](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [186](#), [187](#)
- likelihood\_
  - model-meat.hpp, [304](#)
- likelihood\_exhaust
  - Geese, [167](#)
- likelihood\_joint
  - Flock, [153](#)
- likelihood\_total
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [187](#)
- logical
  - Phylo rules, [45](#)
- logodds
  - DEFM, [142](#)
- M
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [219](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [243](#)
- MAKE\_DEFM\_HASHER
  - Counting, [16](#)
- MAKE\_DUPL\_VARS
  - Counting, [16](#)
- make\_hash
  - FreqTable< T >, [159](#)
- Map
  - barry-configuration.hpp, [268](#)
- map\_to\_state\_id
  - Geese, [173](#)
- MapVec\_type
  - typedefs.hpp, [340](#)
- max\_num\_elements
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [244](#)
- Model
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [179](#)
- model
  - Flock, [156](#)
- Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [175](#)
  - ~Model, [180](#)
  - add\_array, [180](#)
  - add\_counter, [180](#), [181](#)
  - add\_hasher, [181](#)
  - add\_rule, [181](#)
  - add\_rule\_dyn, [181](#), [182](#)
  - arrays2support, [192](#)
  - colnames, [182](#)
  - conditional\_prob, [182](#)
  - counter\_fun, [192](#)
  - counters, [192](#)
  - delete\_counters, [193](#)
  - delete\_engine, [193](#)
  - delete\_rules, [193](#)
  - delete\_rules\_dyn, [193](#)
  - first\_calc\_done, [193](#)
  - gen\_key, [183](#)
  - get\_arrays2support, [183](#)
  - get\_counters, [183](#)
  - get\_normalizing\_constants, [183](#)
  - get\_pset, [184](#)
  - get\_pset\_arrays, [184](#)
  - get\_pset\_probs, [184](#)
  - get\_pset\_stats, [184](#)
  - get\_engine, [185](#)
  - get\_rules, [185](#)
  - get\_rules\_dyn, [185](#)
  - get\_stats\_support, [185](#)
  - get\_stats\_target, [185](#)
  - get\_support\_fun, [186](#)
  - keys2support, [194](#)
  - likelihood, [186](#), [187](#)
  - likelihood\_total, [187](#)
  - Model, [179](#)
  - normalizing\_constants, [194](#)
  - nrules, [187](#)

- nrules\_dyn, 187
- nterms, 188
- operator=, 188
- params\_last, 194
- print, 188
- print\_stats, 188
- pset\_arrays, 194
- pset\_probs, 195
- pset\_stats, 195
- rengine, 195
- rules, 195
- rules\_dyn, 196
- sample, 189
- set\_counters, 189
- set\_rengine, 189
- set\_rules, 190
- set\_rules\_dyn, 190
- set\_seed, 190
- set\_transform\_model, 190
- size, 191
- size\_unique, 191
- stats\_support, 196
- stats\_support\_n\_arrays, 196
- stats\_target, 196
- store\_psets, 191
- support\_fun, 197
- support\_size, 191
- transform\_model, 191
- transform\_model\_fun, 197
- transform\_model\_term\_names, 197
- update\_normalizing\_constants, 192
- with\_pset, 198
- model-meat.hpp
  - likelihood\_, 304
  - update\_normalizing\_constant, 304
- motif\_census
  - DEFM, 143
- N
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 220
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 244
- n\_counters
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 244
- n\_free
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 220
- n\_locked
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 220
- name
  - Counter< Array\_Type, Data\_Type >, 133
  - counters-meat.hpp, 283
- name\_
  - counters-meat.hpp, 286
- nannotations
  - Geese, 168
- narray
  - Node, 206
- ncol
  - BArray< Cell\_Type, Data\_Type >, 67
  - BArrayDense< Cell\_Type, Data\_Type >, 88
  - Phylo rules, 41
- NET\_C\_DATA\_IDX
  - network.hpp, 299
- NET\_C\_DATA\_NUM
  - network.hpp, 299
- NetCounter
  - network.hpp, 300
- NetCounterData, 198
  - ~NetCounterData, 199
  - indices, 199
  - NetCounterData, 199
  - numbers, 199
- NetCounters
  - network.hpp, 300
- NetModel
  - network.hpp, 301
- NetRule
  - network.hpp, 301
- NetRules
  - network.hpp, 301
- NetStatsCounter
  - network.hpp, 301
- NetSupport
  - network.hpp, 301
- Network
  - network.hpp, 301
- Network counters, 24
  - counter\_absdiff, 26
  - counter\_ctriads, 27
  - counter\_degree, 27
  - counter\_density, 27
  - counter\_diff, 28
  - counter\_edges, 28
  - counter\_fixed\_effect, 28
  - counter\_idegree, 29
  - counter\_idegree15, 29
  - counter\_isolates, 29, 30
  - counter\_istar2, 30
  - counter\_logit\_intercept, 30
  - counter\_mutual, 30
  - counter\_nodecov, 31
  - counter\_nodeicov, 31
  - counter\_nodematch, 31
  - counter\_nodeocov, 31
  - counter\_odegree, 31, 32
  - counter\_odegree15, 32
  - counter\_ones, 32
  - counter\_ostar2, 33
  - counter\_transition, 33
  - counter\_transition\_formula, 34
  - counter\_ttriads, 34
  - NETWORK\_COUNTER, 34
  - rules\_dont\_become\_zero, 34
  - rules\_markov\_fixed, 35

- network-css.hpp
  - counter\_css\_census01, [291](#)
  - counter\_css\_census02, [291](#)
  - counter\_css\_census03, [291](#)
  - counter\_css\_census04, [292](#)
  - counter\_css\_census05, [292](#)
  - counter\_css\_census06, [292](#)
  - counter\_css\_census07, [292](#)
  - counter\_css\_census08, [293](#)
  - counter\_css\_census09, [293](#)
  - counter\_css\_census10, [293](#)
  - counter\_css\_completely\_false\_recip\_comiss, [293](#)
  - counter\_css\_completely\_false\_recip\_omiss, [294](#)
  - counter\_css\_mixed\_recip, [294](#)
  - counter\_css\_partially\_false\_recip\_commi, [294](#)
  - counter\_css\_partially\_false\_recip\_omiss, [295](#)
  - CSS\_APPEND, [288](#)
  - CSS\_CASE\_ELSE, [289](#)
  - CSS\_CASE\_PERCEIVED, [289](#)
  - CSS\_CASE\_TRUTH, [289](#)
  - CSS\_CHECK\_SIZE, [289](#)
  - CSS\_CHECK\_SIZE\_INIT, [289](#)
  - CSS\_MATCH\_TYPE, [290](#)
  - CSS\_NET\_COUNTER\_LAMBDA\_INIT, [290](#)
  - CSS\_PERCEIVED\_CELLS, [290](#)
  - CSS\_SIZE, [290](#)
  - CSS\_TRUE\_CELLS, [291](#)
- network.hpp
  - BARRY\_ZERO\_NETWORK, [298](#)
  - BARRY\_ZERO\_NETWORK\_DENSE, [298](#)
  - NET\_C\_DATA\_IDX, [299](#)
  - NET\_C\_DATA\_NUM, [299](#)
  - NetCounter, [300](#)
  - NetCounters, [300](#)
  - NetModel, [301](#)
  - NetRule, [301](#)
  - NetRules, [301](#)
  - NetStatsCounter, [301](#)
  - NetSupport, [301](#)
  - Network, [301](#)
  - NETWORK\_COUNTER, [299](#)
  - NETWORK\_COUNTER\_LAMBDA, [299](#)
  - NETWORK\_RULE, [299](#)
  - NETWORK\_RULE\_LAMBDA, [300](#)
  - NetworkDense, [302](#)
  - NETWORKDENSE\_COUNTER\_LAMBDA, [300](#)
  - rules\_zerodiag, [302](#)
- NETWORK\_COUNTER
  - Network counters, [34](#)
  - network.hpp, [299](#)
- NETWORK\_COUNTER\_LAMBDA
  - network.hpp, [299](#)
- NETWORK\_RULE
  - network.hpp, [299](#)
- NETWORK\_RULE\_LAMBDA
  - network.hpp, [300](#)
- NetworkData, [200](#)
  - ~NetworkData, [201](#)
  - directed, [201](#)
  - NetworkData, [200](#), [201](#)
  - vertex\_attr, [201](#)
- NetworkDense
  - network.hpp, [302](#)
- NETWORKDENSE\_COUNTER\_LAMBDA
  - network.hpp, [300](#)
- next
  - Progress, [222](#)
- nfunctions
  - Flock, [156](#)
  - Geese, [173](#)
- nfuncs
  - Flock, [153](#)
  - Geese, [168](#)
- nleaves
  - Flock, [154](#)
  - Geese, [168](#)
- nnodes
  - Flock, [154](#)
  - Geese, [168](#)
- nnozero
  - BArray< Cell\_Type, Data\_Type >, [67](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [88](#)
- Node, [202](#)
  - ~Node, [204](#)
  - annotations, [205](#)
  - array, [205](#)
  - arrays, [205](#)
  - duplication, [206](#)
  - get\_parent, [204](#)
  - id, [206](#)
  - is\_leaf, [205](#)
  - narray, [206](#)
  - Node, [203](#), [204](#)
  - noffspring, [205](#)
  - offspring, [206](#)
  - ord, [206](#)
  - parent, [207](#)
  - probability, [207](#)
  - subtree\_prob, [207](#)
  - visited, [207](#)
- NodeData, [208](#)
  - blengths, [208](#)
  - duplication, [209](#)
  - NodeData, [208](#)
  - states, [209](#)
- nodes
  - Geese, [173](#)
- noexcept
  - counters-meat.hpp, [286](#)
- noffspring
  - Node, [205](#)
- NONE
  - CHECK, [56](#)
  - EXISTS, [58](#)
- normalizing\_constants
  - Model<     Array\_Type,     Data\_Counter\_Type,

- Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 194
- nrow
  - BArray< Cell\_Type, Data\_Type >, 67
  - BArrayDense< Cell\_Type, Data\_Type >, 88
  - Phylo rules, 41
- nrules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 187
- nrules\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 187
- nterms
  - Flock, 154
  - Geese, 168
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 188
- ntrees
  - Flock, 154
- num
  - Phylo rules, 41, 42
- numbers
  - NetCounterData, 199
  - Phylo rules, 45, 46
- obs\_start
  - Phylo rules, 46
- observed\_counts
  - Geese, 169
- offspring
  - Node, 206
- ONE
  - CHECK, 56
  - EXISTS, 58
- operator BArrayRow< Cell\_Type, Data\_Type >
  - BArrayRow< Cell\_Type, Data\_Type >, 110
- operator BArrayRow\_const< Cell\_Type, Data\_Type >
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 112
- operator Cell\_Type
  - BArrayCell< Cell\_Type, Data\_Type >, 75
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 77
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 97
  - Cell< Cell\_Type >, 124
- operator std::vector< Cell\_Type >
  - BArrayVector< Cell\_Type, Data\_Type >, 116
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 119
- operator!=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 77
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 112
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 119
  - Cell< Cell\_Type >, 124
- operator<
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 77
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 112
- BArrayVector\_const< Cell\_Type, Data\_Type >, 120
- operator<=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 78
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 113
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 120
- operator>
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 78
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 113
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 120
- operator>=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 78
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 113
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 120
- operator\*=
  - BArray< Cell\_Type, Data\_Type >, 67
  - BArrayCell< Cell\_Type, Data\_Type >, 75
  - BArrayDense< Cell\_Type, Data\_Type >, 89
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 97
  - BArrayRow< Cell\_Type, Data\_Type >, 110
  - BArrayVector< Cell\_Type, Data\_Type >, 116
- operator()
  - BArray< Cell\_Type, Data\_Type >, 67
  - barray-meat-operators.hpp, 251
  - BArrayDense< Cell\_Type, Data\_Type >, 88, 89
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 101
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 103
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 106
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 108
  - Flock, 154
  - Phylo rules, 42
  - PhyloCounterData, 211
  - PhyloRuleDynData, 213
  - Rule< Array\_Type, Data\_Type >, 225
  - Rules< Array\_Type, Data\_Type >, 228
  - vecHasher< T >, 245
- operator+=
  - BArray< Cell\_Type, Data\_Type >, 68
  - BArrayCell< Cell\_Type, Data\_Type >, 75
  - BArrayDense< Cell\_Type, Data\_Type >, 89
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 98
  - BArrayRow< Cell\_Type, Data\_Type >, 110
  - BArrayVector< Cell\_Type, Data\_Type >, 116
- operator-=
  - BArray< Cell\_Type, Data\_Type >, 68
  - BArrayCell< Cell\_Type, Data\_Type >, 75
  - BArrayDense< Cell\_Type, Data\_Type >, 90
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 98
  - BArrayRow< Cell\_Type, Data\_Type >, 110
  - BArrayVector< Cell\_Type, Data\_Type >, 116
- operator/=
  - BArray< Cell\_Type, Data\_Type >, 69
  - BArrayCell< Cell\_Type, Data\_Type >, 75

- BArrayDense< Cell\_Type, Data\_Type >, 90
- BArrayDenseCell< Cell\_Type, Data\_Type >, 98
- BArrayRow< Cell\_Type, Data\_Type >, 111
- BArrayVector< Cell\_Type, Data\_Type >, 116
- operator=
  - BArray< Cell\_Type, Data\_Type >, 69
  - BArrayCell< Cell\_Type, Data\_Type >, 75
  - BArrayDense< Cell\_Type, Data\_Type >, 90, 91
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 98
  - BArrayRow< Cell\_Type, Data\_Type >, 111
  - BArrayVector< Cell\_Type, Data\_Type >, 117
  - Cell< Cell\_Type >, 124, 125
  - Counter< Array\_Type, Data\_Type >, 131, 132
  - Counters< Array\_Type, Data\_Type >, 137
  - Geese, 169
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 188
  - Rules< Array\_Type, Data\_Type >, 229
- operator==
  - BArray< Cell\_Type, Data\_Type >, 69
  - BArrayCell< Cell\_Type, Data\_Type >, 76
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 78
  - BArrayDense< Cell\_Type, Data\_Type >, 91
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 99
  - BArrayRow< Cell\_Type, Data\_Type >, 111
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 113
  - BArrayVector< Cell\_Type, Data\_Type >, 117
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 120
  - Cell< Cell\_Type >, 125
- operator[]
  - Counters< Array\_Type, Data\_Type >, 138
  - PhyloCounterData, 211
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 218
- ord
  - Node, 206
- out\_of\_range
  - BArray< Cell\_Type, Data\_Type >, 69
  - BArrayDense< Cell\_Type, Data\_Type >, 91
- params\_last
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 194
- parent
  - Node, 207
- parse\_polytomies
  - Flock, 155
  - Geese, 169
- Phylo counters, 47
  - counter\_co\_opt, 48
  - counter\_cogain, 49
  - counter\_gains, 49
  - counter\_gains\_from\_0, 49
  - counter\_gains\_k\_offspring, 49
  - counter\_genes\_changing, 50
  - counter\_k\_genes\_changing, 50
  - counter\_less\_than\_p\_prop\_genes\_changing, 50
  - counter\_longest, 50
  - counter\_loss, 51
  - counter\_maxfuns, 51
  - counter\_neofun, 51
  - counter\_neofun\_a2b, 51
  - counter\_overall\_changes, 52
  - counter\_overall\_gains, 52
  - counter\_overall\_gains\_from\_0, 52
  - counter\_overall\_loss, 52
  - counter\_pairwise\_first\_gain, 53
  - counter\_pairwise\_neofun\_singlefun, 53
  - counter\_pairwise\_overall\_change, 53
  - counter\_pairwise\_preserving, 53
  - counter\_preserve\_pseudogene, 54
  - counter\_prop\_genes\_changing, 54
  - counter\_subfun, 54
- Phylo rules, 35
  - ~DEFMCounterData, 43
  - ~DEFMData, 43
  - ~DEFMRuleDynData, 43
- array, 44
- at, 39
- counts, 44
- covar\_sort, 44
- covar\_used, 44
- covariates, 44
- DEFMCounter, 37
- DEFMCounterData, 39
- DEFMCounters, 37
- DEFMData, 39
- DEFMModel, 37
- DEFMRule, 38
- DEFMRuleData, 40
- DEFMRuleDyn, 38
- DEFMRuleDynData, 40
- DEFMRules, 38
- DEFMRulesDyn, 38
- DEFMStatsCounter, 38
- DEFMSupport, 38
- idx, 40, 41
- indices, 44, 45
- init, 45
- is\_motif, 45
- is\_true, 41
- logical, 45
- ncol, 41
- nrow, 41
- num, 41, 42
- numbers, 45, 46
- obs\_start, 46
- operator(), 42
- print, 42
- rule\_dyn\_limit\_changes, 42
- X\_ncol, 46
- X\_nrow, 46
- PHYLO\_CHECK\_MISSING
  - Counting, 16
- PHYLO\_COUNTER\_LAMBDA

- Counting, [17](#)
- PHYLO\_RULE\_DYN\_LAMBDA
  - Counting, [17](#)
- PhyloArray
  - geese-types.hpp, [324](#)
- PhyloCounter
  - geese-types.hpp, [324](#)
- PhyloCounterData, [209](#)
  - at, [210](#)
  - begin, [210](#)
  - empty, [210](#)
  - end, [211](#)
  - get\_counters, [211](#)
  - operator(), [211](#)
  - operator[], [211](#)
  - PhyloCounterData, [210](#)
  - push\_back, [211](#)
  - reserve, [211](#)
  - shrink\_to\_fit, [212](#)
  - size, [212](#)
- PhyloCounters
  - geese-types.hpp, [325](#)
- PhyloModel
  - geese-types.hpp, [325](#)
- PhyloPowerSet
  - geese-types.hpp, [325](#)
- PhyloRule
  - geese-types.hpp, [325](#)
- PhyloRuleData
  - geese-types.hpp, [325](#)
- PhyloRuleDyn
  - geese-types.hpp, [325](#)
- PhyloRuleDynData, [212](#)
  - ~PhyloRuleDynData, [213](#)
  - counts, [213](#)
  - duplication, [213](#)
  - lb, [214](#)
  - operator(), [213](#)
  - PhyloRuleDynData, [213](#)
  - pos, [214](#)
  - ub, [214](#)
- PhyloRules
  - geese-types.hpp, [326](#)
- PhyloRulesDyn
  - geese-types.hpp, [326](#)
- PhyloStatsCounter
  - geese-types.hpp, [326](#)
- PhyloSupport
  - geese-types.hpp, [326](#)
- POS
  - barraydense-meat-operators.hpp, [256](#)
  - barraydense-meat.hpp, [258](#)
  - barraydensecell-bones.hpp, [260](#)
  - barraydensecell-meat.hpp, [260](#)
  - barraydensecol-bones.hpp, [261](#)
  - barraydenserow-bones.hpp, [263](#)
  - geese-types.hpp, [324](#)
- pos
  - PhyloRuleDynData, [214](#)
- POS\_N
  - barraydense-meat-operators.hpp, [256](#)
  - barraydense-meat.hpp, [259](#)
  - barraydensecol-bones.hpp, [261](#)
  - barraydenserow-bones.hpp, [263](#)
- PowerSet
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [216](#)
- PowerSet< Array\_Type, Data\_Rule\_Type >, [214](#)
  - ~PowerSet, [216](#)
  - add\_rule, [216](#), [217](#)
  - begin, [217](#)
  - calc, [217](#)
  - coordinates\_free, [219](#)
  - coordinates\_locked, [219](#)
  - data, [219](#)
  - EmptyArray, [219](#)
  - end, [217](#)
  - get\_data, [217](#)
  - get\_data\_ptr, [218](#)
  - init\_support, [218](#)
  - M, [219](#)
  - N, [220](#)
  - n\_free, [220](#)
  - n\_locked, [220](#)
  - operator[], [218](#)
  - PowerSet, [216](#)
  - reset, [218](#)
  - rules, [220](#)
  - rules\_deleted, [220](#)
  - size, [218](#)
- predict
  - Geese, [169](#)
- predict\_backend
  - Geese, [170](#)
- predict\_exhaust
  - Geese, [170](#)
- predict\_exhaust\_backend
  - Geese, [170](#)
- predict\_sim
  - Geese, [170](#)
- print
  - BArray< Cell\_Type, Data\_Type >, [69](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [91](#)
  - DEFM, [143](#)
  - Flock, [155](#)
  - FreqTable< T >, [159](#)
  - Geese, [170](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [188](#)
  - Phylo rules, [42](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [240](#)
- print\_n
  - BArray< Cell\_Type, Data\_Type >, [70](#)
- print\_nodes

- Geese, [171](#)
- print\_observed\_counts
  - Geese, [171](#)
- print\_stats
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [188](#)
- printf\_barry
  - barry-configuration.hpp, [267](#)
- probability
  - Node, [207](#)
- Progress, [221](#)
  - ~Progress, [221](#)
  - end, [222](#)
  - next, [222](#)
  - Progress, [221](#)
- progress.hpp
  - BARRY\_PROGRESS\_BAR\_WIDTH, [328](#)
- pset\_arrays
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [194](#)
- pset\_loc
  - Geese, [173](#)
- pset\_probs
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [195](#)
- pset\_stats
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [195](#)
- push\_back
  - PhyloCounterData, [211](#)
- README.md, [342](#)
- reduced\_sequence
  - Geese, [174](#)
- rengine
  - Flock, [156](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [195](#)
- res
  - counters-meat.hpp, [286](#)
- reserve
  - BArray< Cell\_Type, Data\_Type >, [70](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [91](#)
  - FreqTable< T >, [159](#)
  - PhyloCounterData, [211](#)
- reset
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [218](#)
- reset\_array
  - StatsCounter< Array\_Type, Data\_Type >, [232](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [240](#)
- resize
  - BArray< Cell\_Type, Data\_Type >, [70](#)
- BArrayDense< Cell\_Type, Data\_Type >, [92](#)
- Entries< Cell\_Type >, [149](#)
- statscounter-meat.hpp, [332](#)
- return
  - counters-meat.hpp, [287](#)
  - statscounter-meat.hpp, [335](#)
- rhs
  - barray-meat-operators.hpp, [251](#)
- rm\_cell
  - BArray< Cell\_Type, Data\_Type >, [70](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [92](#)
- ROW
  - barray-meat-operators.hpp, [249](#)
  - barray-meat.hpp, [253](#)
  - barraydense-meat-operators.hpp, [256](#)
  - barraydense-meat.hpp, [259](#)
- row
  - BArray< Cell\_Type, Data\_Type >, [70](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [92](#)
- Row\_type
  - typedefs.hpp, [340](#)
- rowsum
  - BArrayDense< Cell\_Type, Data\_Type >, [92](#)
- Rule
  - Rule< Array\_Type, Data\_Type >, [223](#)
- Rule< Array\_Type, Data\_Type >, [222](#)
  - ~Rule, [223](#)
  - D, [224](#)
  - get\_description, [224](#)
  - get\_name, [224](#)
  - operator(), [225](#)
  - Rule, [223](#)
- rule\_dyn\_limit\_changes
  - Phylo rules, [42](#)
- rule\_fun\_default
  - rules-bones.hpp, [329](#)
- Rule\_fun\_type
  - typedefs.hpp, [340](#)
- RULE\_FUNCTION
  - barry.hpp, [273](#)
  - geese-bones.hpp, [317](#)
- RULE\_LAMBDA
  - barry.hpp, [273](#)
- Rules
  - Rules< Array\_Type, Data\_Type >, [226](#)
- rules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [195](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [220](#)
  - Rules< Array\_Type, Data\_Type >, [225](#)
  - ~Rules, [226](#)
  - add\_rule, [227](#)
  - begin, [227](#)
  - end, [227](#)
  - get\_descriptions, [227](#)
  - get\_names, [228](#)
  - get\_seq, [228](#)



- operator(), 228
- operator=, 229
- Rules, 226
- size, 229
- rules-bones.hpp
  - rule\_fun\_default, 329
- rules\_deleted
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 220
- rules\_dont\_become\_zero
  - Network counters, 34
- rules\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 196
- rules\_markov\_fixed
  - Network counters, 35
- rules\_zerodiag
  - network.hpp, 302
- sample
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 189
- sequence
  - Geese, 174
- set\_counters
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 189
  - StatsCounter< Array\_Type, Data\_Type >, 233
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 240
- set\_data
  - BArray< Cell\_Type, Data\_Type >, 71
  - BArrayDense< Cell\_Type, Data\_Type >, 93
- set\_hasher
  - Counter< Array\_Type, Data\_Type >, 132
- set\_names
  - DEFM, 143
- set\_rengine
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 189
- set\_rules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 190
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 241
- set\_rules\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 190
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 241
- set\_seed
  - Flock, 155
  - Geese, 171
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 190
  - set\_transform\_model
    - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 190
  - shrink\_to\_fit
    - PhyloCounterData, 212
  - simulate
    - DEFM, 143
    - Geese, 171
  - size
    - BArrayDenseCol< Cell\_Type, Data\_Type >, 101
    - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 104
    - BArrayDenseRow< Cell\_Type, Data\_Type >, 106
    - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 108
    - BArrayVector< Cell\_Type, Data\_Type >, 117
    - BArrayVector\_const< Cell\_Type, Data\_Type >, 121
    - Counters< Array\_Type, Data\_Type >, 138
    - FreqTable< T >, 160
    - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 191
    - PhyloCounterData, 212
    - PowerSet< Array\_Type, Data\_Rule\_Type >, 218
    - Rules< Array\_Type, Data\_Type >, 229
    - StatsCounter< Array\_Type, Data\_Type >, 233
  - size\_unique
    - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 191
  - sort\_array
    - typedefs.hpp, 340
  - source
    - Entries< Cell\_Type >, 149
  - states
    - NodeData, 209
  - Statistical Models, 24
  - stats\_support
    - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 196
  - stats\_support\_n\_arrays
    - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 196
  - stats\_target
    - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 196
  - StatsCounter
    - StatsCounter< Array\_Type, Data\_Type >, 230,



- 231
- StatsCounter< Array\_Type, Data\_Type >, 229
  - ~StatsCounter, 231
  - add\_counter, 231
  - count\_all, 231
  - count\_current, 232
  - count\_init, 232
  - get\_counters, 232
  - get\_descriptions, 232
  - get\_names, 232
  - reset\_array, 232
  - set\_counters, 233
  - size, 233
  - StatsCounter, 230, 231
- statscounter-meat.hpp
  - clear, 331
  - counter, 333
  - counter\_deleted, 334
  - counters, 334
  - counters\_, 334
  - current\_stats, 334
  - EmptyArray, 334
  - f\_, 335
  - for, 331
  - j, 335
  - resize, 332
  - return, 335
  - STATSCOUNTER\_TEMPLATE, 331–333
  - STATSCOUNTER\_TEMPLATE\_ARGS, 331
  - STATSCOUNTER\_TYPE, 331
- STATSCOUNTER\_TEMPLATE
  - statscounter-meat.hpp, 331–333
- STATSCOUNTER\_TEMPLATE\_ARGS
  - statscounter-meat.hpp, 331
- STATSCOUNTER\_TYPE
  - statscounter-meat.hpp, 331
- store\_psets
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 191
- subtree\_prob
  - Node, 207
- Support
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 235, 236
- Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 233
  - ~Support, 236
  - add\_counter, 236
  - add\_rule, 236, 237
  - add\_rule\_dyn, 237
  - calc, 237
  - change\_stats, 241
  - coordiantes\_n\_free, 241
  - coordiantes\_n\_locked, 242
  - coordinates\_free, 242
  - coordinates\_locked, 242
  - current\_stats, 242
  - delete\_counters, 242
  - delete\_rules, 243
  - delete\_rules\_dyn, 243
  - eval\_rules\_dyn, 238
  - get\_counters, 238
  - get\_counts, 238
  - get\_current\_stats, 238
  - get\_data, 239
  - get\_rules, 239
  - get\_rules\_dyn, 239
  - hashes, 243
  - hashes\_initialized, 243
  - init\_support, 239
  - M, 243
  - max\_num\_elements, 244
  - N, 244
  - n\_counters, 244
  - print, 240
  - reset\_array, 240
  - set\_counters, 240
  - set\_rules, 241
  - set\_rules\_dyn, 241
  - Support, 235, 236
- support-meat.hpp
  - BARRY\_SUPPORT\_MEAT\_HPP, 336
- support\_fun
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 197
- support\_size
  - Flock, 155
  - Geese, 171
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 191
- swap\_cells
  - BArray< Cell\_Type, Data\_Type >, 71
  - BArrayDense< Cell\_Type, Data\_Type >, 93
- swap\_cols
  - BArray< Cell\_Type, Data\_Type >, 71
  - BArrayDense< Cell\_Type, Data\_Type >, 93
- swap\_rows
  - BArray< Cell\_Type, Data\_Type >, 71
  - BArrayDense< Cell\_Type, Data\_Type >, 93
- target
  - Entries< Cell\_Type >, 149
- this
  - barray-meat-operators.hpp, 252
- TMP\_HASHER\_CALL
  - counters-meat.hpp, 277
- toggle\_cell
  - BArray< Cell\_Type, Data\_Type >, 72
  - BArrayDense< Cell\_Type, Data\_Type >, 94
- toggle\_lock
  - BArray< Cell\_Type, Data\_Type >, 72
  - BArrayDense< Cell\_Type, Data\_Type >, 94
- transform\_model

- Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
[191](#)
- transform\_model\_fun
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
[197](#)
- transform\_model\_term\_names
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
[197](#)
- transpose
  - BArray< Cell\_Type, Data\_Type > , [72](#)
  - BArrayDense< Cell\_Type, Data\_Type > , [94](#)
- TWO
  - CHECK, [56](#)
  - EXISTS, [58](#)
- typedefs.hpp
  - Col\_type, [339](#)
  - Counter\_fun\_type, [339](#)
  - Counts\_type, [339](#)
  - Hasher\_fun\_type, [339](#)
  - MapVec\_type, [340](#)
  - Row\_type, [340](#)
  - Rule\_fun\_type, [340](#)
  - sort\_array, [340](#)
  - vec\_equal, [341](#)
  - vec\_equal\_approx, [341](#)
  - vec\_inner\_prod, [341](#), [342](#)
- ub
  - PhyloRuleDynData, [214](#)
- UNKNOWN
  - EXISTS, [58](#)
- UNI\_SUB
  - counters.hpp, [308](#)
- update\_annotations
  - Geese, [171](#)
- update\_normalizing\_constant
  - model-meat.hpp, [304](#)
- update\_normalizing\_constants
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
[192](#)
- val
  - Entries< Cell\_Type > , [149](#)
- value
  - Cell< Cell\_Type > , [125](#)
- vec\_diff
  - geese-bones.hpp, [317](#)
- vec\_equal
  - typedefs.hpp, [341](#)
- vec\_equal\_approx
  - typedefs.hpp, [341](#)
- vec\_inner\_prod
  - typedefs.hpp, [341](#), [342](#)
- vecHasher< T > , [244](#)
  - operator(), [245](#)
- vector\_caster
  - geese-bones.hpp, [318](#)
- vertex\_attr
  - NetworkData, [201](#)
- visited
  - BArray< Cell\_Type, Data\_Type > , [73](#)
  - BArrayDense< Cell\_Type, Data\_Type > , [96](#)
  - Cell< Cell\_Type > , [125](#)
  - Node, [207](#)
- with\_pset
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type > ,  
[198](#)
- X\_ncol
  - Phylo rules, [46](#)
- X\_nrow
  - Phylo rules, [46](#)
- ZERO\_CELL
  - barraydense-meat.hpp, [259](#)
  - barraydensecol-bones.hpp, [262](#)
  - barraydenserow-bones.hpp, [263](#)
- zero\_col
  - BArray< Cell\_Type, Data\_Type > , [72](#)
  - BArrayDense< Cell\_Type, Data\_Type > , [94](#)
- zero\_row
  - BArray< Cell\_Type, Data\_Type > , [72](#)
  - BArrayDense< Cell\_Type, Data\_Type > , [94](#)