

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	13
6.2 Statistical Models	13
6.2.1 Detailed Description	14
6.3 Phylo rules	14
6.3.1 Detailed Description	15
6.3.2 Macro Definition Documentation	15
6.3.2.1 DEFM_COUNTER	15
6.3.2.2 DEFM_COUNTER_LAMBDA	15
6.3.2.3 DEFM_RULE	16
6.3.2.4 DEFM_RULE_LAMBDA	16
6.3.2.5 DEFM_RULEDYN_LAMBDA	16
6.3.2.6 MAKE_DEFM_HASHER	16
6.3.3 Typedef Documentation	17
6.3.3.1 DEFMCOUNTER	17
6.3.3.2 DEFMCOUNTERS	17
6.3.3.3 DEFMMODEL	17
6.3.3.4 DEFMRULE	17
6.3.3.5 DEFMRULEDYN	17
6.3.3.6 DEFMRULES	18
6.3.3.7 DEFMRULESDYN	18
6.3.3.8 DEFMSTATSCOUNTER	18
6.3.3.9 DEFMSUPPORT	18
6.3.4 Function Documentation	18
6.3.4.1 ncol()	18
6.3.4.2 nrow()	18
6.3.4.3 operator()()	18
6.3.4.4 print()	19
6.3.4.5 rule_dyn_limit_changes()	19

6.4 DEFMArray counters	19
6.4.1 Detailed Description	21
6.4.2 Function Documentation	21
6.4.2.1 counter_absdiff()	22
6.4.2.2 counter_ctriads() [1/2]	22
6.4.2.3 counter_ctriads() [2/2]	22
6.4.2.4 counter_degree()	22
6.4.2.5 counter_density()	23
6.4.2.6 counter_diff()	23
6.4.2.7 counter_edges()	23
6.4.2.8 counter_fixed_effect()	23
6.4.2.9 counter_idegree() [1/2]	24
6.4.2.10 counter_idegree() [2/2]	24
6.4.2.11 counter_idegree15() [1/2]	24
6.4.2.12 counter_idegree15() [2/2]	24
6.4.2.13 counter_isolates() [1/2]	25
6.4.2.14 counter_isolates() [2/2]	25
6.4.2.15 counter_istar2() [1/2]	25
6.4.2.16 counter_istar2() [2/2]	25
6.4.2.17 counter_logit_intercept()	25
6.4.2.18 counter_mutual()	26
6.4.2.19 counter_nodecov()	26
6.4.2.20 counter_nodeicov()	26
6.4.2.21 counter_nodematch()	26
6.4.2.22 counter_nodeocov()	26
6.4.2.23 counter_odegree() [1/2]	27
6.4.2.24 counter_odegree() [2/2]	27
6.4.2.25 counter_odegree15() [1/2]	27
6.4.2.26 counter_odegree15() [2/2]	27
6.4.2.27 counter_ones()	27
6.4.2.28 counter_ostar2() [1/2]	28
6.4.2.29 counter_ostar2() [2/2]	28
6.4.2.30 counter_transition()	28
6.4.2.31 counter_transition_formula()	29
6.4.2.32 counter_ttriads() [1/2]	29
6.4.2.33 counter_ttriads() [2/2]	29
6.4.2.34 NETWORK_COUNTER()	29
6.4.2.35 rules_dont_become_zero()	30
6.4.2.36 rules_markov_fixed()	30
6.5 Phylo counters	30
6.5.1 Detailed Description	31
6.5.2 Function Documentation	31

6.5.2.1 counter_co_opt()	32
6.5.2.2 counter_cogain()	32
6.5.2.3 counter_gains()	32
6.5.2.4 counter_gains_from_0()	33
6.5.2.5 counter_gains_k_offspring()	33
6.5.2.6 counter_genes_changing()	33
6.5.2.7 counter_k_genes_changing()	33
6.5.2.8 counter_less_than_p_prop_genes_changing()	34
6.5.2.9 counter_longest()	34
6.5.2.10 counter_loss()	34
6.5.2.11 counter_maxfuns()	34
6.5.2.12 counter_neofun()	35
6.5.2.13 counter_neofun_a2b()	35
6.5.2.14 counter_overall_changes()	35
6.5.2.15 counter_overall_gains()	35
6.5.2.16 counter_overall_gains_from_0()	36
6.5.2.17 counter_overall_loss()	36
6.5.2.18 counter_pairwise_first_gain()	36
6.5.2.19 counter_pairwise_neofun_singlefun()	36
6.5.2.20 counter_pairwise_overall_change()	37
6.5.2.21 counter_pairwise_preserving()	37
6.5.2.22 counter_preserve_pseudogene()	37
6.5.2.23 counter_prop_genes_changing()	37
6.5.2.24 counter_subfun()	38
<b>7 Namespace Documentation</b>	<b>39</b>
7.1 barry Namespace Reference	39
7.1.1 Detailed Description	39
7.2 barry::counters Namespace Reference	39
7.2.1 Detailed Description	39
7.3 barry::counters::defm Namespace Reference	40
7.4 barry::counters::network Namespace Reference	40
7.5 barry::counters::phylo Namespace Reference	40
7.6 CHECK Namespace Reference	40
7.6.1 Detailed Description	40
7.6.2 Variable Documentation	40
7.6.2.1 BOTH	40
7.6.2.2 NONE	40
7.6.2.3 ONE	41
7.6.2.4 TWO	41
7.7 EXISTS Namespace Reference	41
7.7.1 Detailed Description	41

7.7.2 Variable Documentation	41
7.7.2.1 AS_ONE	41
7.7.2.2 AS_ZERO	42
7.7.2.3 BOTH	42
7.7.2.4 NONE	42
7.7.2.5 ONE	42
7.7.2.6 TWO	42
7.7.2.7 UNKNOWN	42
<b>8 Class Documentation</b>	<b>43</b>
8.1 BArray< Cell_Type, Data_Type > Class Template Reference	43
8.1.1 Detailed Description	45
8.1.2 Constructor & Destructor Documentation	46
8.1.2.1 BArray() [1/6]	46
8.1.2.2 BArray() [2/6]	46
8.1.2.3 BArray() [3/6]	46
8.1.2.4 BArray() [4/6]	47
8.1.2.5 BArray() [5/6]	47
8.1.2.6 BArray() [6/6]	47
8.1.2.7 ~BArray()	47
8.1.3 Member Function Documentation	47
8.1.3.1 clear()	47
8.1.3.2 col()	48
8.1.3.3 D() [1/2]	48
8.1.3.4 D() [2/2]	48
8.1.3.5 D_ptr() [1/2]	48
8.1.3.6 D_ptr() [2/2]	48
8.1.3.7 default_val()	48
8.1.3.8 flush_data()	48
8.1.3.9 get_cell()	49
8.1.3.10 get_col_vec() [1/2]	49
8.1.3.11 get_col_vec() [2/2]	49
8.1.3.12 get_entries()	49
8.1.3.13 get_row_vec() [1/2]	49
8.1.3.14 get_row_vec() [2/2]	50
8.1.3.15 insert_cell() [1/3]	50
8.1.3.16 insert_cell() [2/3]	50
8.1.3.17 insert_cell() [3/3]	50
8.1.3.18 is_dense()	50
8.1.3.19 is_empty()	51
8.1.3.20 ncol()	51
8.1.3.21 nzero()	51

8.1.3.22 <code>nrow()</code>	51
8.1.3.23 <code>operator&gt;()</code> [1/2]	51
8.1.3.24 <code>operator&gt;()</code> [2/2]	51
8.1.3.25 <code>operator*==()</code>	52
8.1.3.26 <code>operator+=()</code> [1/3]	52
8.1.3.27 <code>operator+=()</code> [2/3]	52
8.1.3.28 <code>operator+=()</code> [3/3]	52
8.1.3.29 <code>operator-=()</code> [1/3]	52
8.1.3.30 <code>operator-=()</code> [2/3]	52
8.1.3.31 <code>operator-=()</code> [3/3]	53
8.1.3.32 <code>operator/=( )</code>	53
8.1.3.33 <code>operator=()</code> [1/2]	53
8.1.3.34 <code>operator=()</code> [2/2]	53
8.1.3.35 <code>operator==( )</code>	53
8.1.3.36 <code>out_of_range()</code>	53
8.1.3.37 <code>print()</code>	54
8.1.3.38 <code>reserve()</code>	54
8.1.3.39 <code>resize()</code>	54
8.1.3.40 <code>rm_cell()</code>	54
8.1.3.41 <code>row()</code>	54
8.1.3.42 <code>set_data()</code>	54
8.1.3.43 <code>swap_cells()</code>	55
8.1.3.44 <code>swap_cols()</code>	55
8.1.3.45 <code>swap_rows()</code>	55
8.1.3.46 <code>toggle_cell()</code>	55
8.1.3.47 <code>toggle_lock()</code>	56
8.1.3.48 <code>transpose()</code>	56
8.1.3.49 <code>zero_col()</code>	56
8.1.3.50 <code>zero_row()</code>	56
8.1.4 Friends And Related Function Documentation	56
8.1.4.1 <code>BArrayCell&lt; Cell_Type, Data_Type &gt;</code>	56
8.1.4.2 <code>BArrayCell_const&lt; Cell_Type, Data_Type &gt;</code>	57
8.1.5 Member Data Documentation	57
8.1.5.1 <code>visited</code>	57
8.2 <code>BArrayCell&lt; Cell_Type, Data_Type &gt;</code> Class Template Reference	57
8.2.1 Detailed Description	57
8.2.2 Constructor & Destructor Documentation	58
8.2.2.1 <code>BArrayCell()</code>	58
8.2.2.2 <code>~BArrayCell()</code>	58
8.2.3 Member Function Documentation	58
8.2.3.1 <code>operator Cell_Type()</code>	58
8.2.3.2 <code>operator*==()</code>	58

8.2.3.3 operator+=()	59
8.2.3.4 operator-=()	59
8.2.3.5 operator/=( )	59
8.2.3.6 operator=()	59
8.2.3.7 operator==( )	59
8.3 BArrayCell_const< Cell_Type, Data_Type > Class Template Reference	60
8.3.1 Detailed Description	60
8.3.2 Constructor & Destructor Documentation	60
8.3.2.1 BArrayCell_const()	60
8.3.2.2 ~BArrayCell_const()	60
8.3.3 Member Function Documentation	61
8.3.3.1 operator Cell_Type()	61
8.3.3.2 operator"!=( )	61
8.3.3.3 operator<( )	61
8.3.3.4 operator<=( )	61
8.3.3.5 operator==( )	61
8.3.3.6 operator>( )	62
8.3.3.7 operator>=( )	62
8.4 BArrayDense< Cell_Type, Data_Type > Class Template Reference	62
8.4.1 Detailed Description	65
8.4.2 Constructor & Destructor Documentation	65
8.4.2.1 BArrayDense() [1/6]	65
8.4.2.2 BArrayDense() [2/6]	65
8.4.2.3 BArrayDense() [3/6]	66
8.4.2.4 BArrayDense() [4/6]	66
8.4.2.5 BArrayDense() [5/6]	66
8.4.2.6 BArrayDense() [6/6]	66
8.4.2.7 ~BArrayDense()	67
8.4.3 Member Function Documentation	67
8.4.3.1 clear()	67
8.4.3.2 col() [1/2]	67
8.4.3.3 col() [2/2]	67
8.4.3.4 colsum()	67
8.4.3.5 D() [1/2]	68
8.4.3.6 D() [2/2]	68
8.4.3.7 D_ptr() [1/2]	68
8.4.3.8 D_ptr() [2/2]	68
8.4.3.9 default_val()	68
8.4.3.10 get_cell()	68
8.4.3.11 get_col_vec() [1/2]	69
8.4.3.12 get_col_vec() [2/2]	69
8.4.3.13 get_data()	69



8.4.3.14	<a href="#">get_entries()</a>	69
8.4.3.15	<a href="#">get_row_vec()</a> [1/2]	69
8.4.3.16	<a href="#">get_row_vec()</a> [2/2]	70
8.4.3.17	<a href="#">insert_cell()</a> [1/2]	70
8.4.3.18	<a href="#">insert_cell()</a> [2/2]	70
8.4.3.19	<a href="#">is_dense()</a>	70
8.4.3.20	<a href="#">is_empty()</a>	70
8.4.3.21	<a href="#">ncol()</a>	71
8.4.3.22	<a href="#">nnozero()</a>	71
8.4.3.23	<a href="#">nrow()</a>	71
8.4.3.24	<a href="#">operator&gt;()</a> [1/2]	71
8.4.3.25	<a href="#">operator&gt;()</a> [2/2]	71
8.4.3.26	<a href="#">operator*=( )</a>	71
8.4.3.27	<a href="#">operator+=( )</a> [1/3]	72
8.4.3.28	<a href="#">operator+=( )</a> [2/3]	72
8.4.3.29	<a href="#">operator+=( )</a> [3/3]	72
8.4.3.30	<a href="#">operator-=( )</a> [1/3]	72
8.4.3.31	<a href="#">operator-=( )</a> [2/3]	72
8.4.3.32	<a href="#">operator-=( )</a> [3/3]	72
8.4.3.33	<a href="#">operator/=( )</a>	73
8.4.3.34	<a href="#">operator=()</a> [1/2]	73
8.4.3.35	<a href="#">operator=()</a> [2/2]	73
8.4.3.36	<a href="#">operator==( )</a>	73
8.4.3.37	<a href="#">out_of_range()</a>	73
8.4.3.38	<a href="#">print()</a>	73
8.4.3.39	<a href="#">reserve()</a>	74
8.4.3.40	<a href="#">resize()</a>	74
8.4.3.41	<a href="#">rm_cell()</a>	74
8.4.3.42	<a href="#">row()</a> [1/2]	74
8.4.3.43	<a href="#">row()</a> [2/2]	74
8.4.3.44	<a href="#">rowsum()</a>	74
8.4.3.45	<a href="#">set_data()</a>	75
8.4.3.46	<a href="#">swap_cells()</a>	75
8.4.3.47	<a href="#">swap_cols()</a>	75
8.4.3.48	<a href="#">swap_rows()</a>	75
8.4.3.49	<a href="#">toggle_cell()</a>	76
8.4.3.50	<a href="#">toggle_lock()</a>	76
8.4.3.51	<a href="#">transpose()</a>	76
8.4.3.52	<a href="#">zero_col()</a>	76
8.4.3.53	<a href="#">zero_row()</a>	76
8.4.4	<a href="#">Friends And Related Function Documentation</a>	76
8.4.4.1	<a href="#">BArrayDenseCell&lt; Cell_Type, Data_Type &gt;</a>	77

8.4.4.2 BArrayDenseCol< Cell_Type, Data_Type > . . . . .	77
8.4.4.3 BArrayDenseCol_const< Cell_Type, Data_Type > . . . . .	77
8.4.4.4 BArrayDenseRow< Cell_Type, Data_Type > . . . . .	77
8.4.4.5 BArrayDenseRow_const< Cell_Type, Data_Type > . . . . .	77
8.4.5 Member Data Documentation . . . . .	77
8.4.5.1 visited . . . . .	78
8.5 BArrayDenseCell< Cell_Type, Data_Type > Class Template Reference . . . . .	78
8.5.1 Detailed Description . . . . .	78
8.5.2 Constructor & Destructor Documentation . . . . .	79
8.5.2.1 BArrayDenseCell() . . . . .	79
8.5.2.2 ~BArrayDenseCell() . . . . .	79
8.5.3 Member Function Documentation . . . . .	79
8.5.3.1 operator Cell_Type() . . . . .	79
8.5.3.2 operator*=( ) . . . . .	79
8.5.3.3 operator+=( ) . . . . .	80
8.5.3.4 operator-=( ) . . . . .	80
8.5.3.5 operator/=( ) . . . . .	80
8.5.3.6 operator=( ) [ 1 / 2 ] . . . . .	80
8.5.3.7 operator=( ) [ 2 / 2 ] . . . . .	80
8.5.3.8 operator==( ) . . . . .	81
8.5.4 Friends And Related Function Documentation . . . . .	81
8.5.4.1 BArrayDense< Cell_Type, Data_Type > . . . . .	81
8.5.4.2 BArrayDenseCol< Cell_Type, Data_Type > . . . . .	81
8.5.4.3 BArrayDenseCol_const< Cell_Type, Data_Type > . . . . .	81
8.6 BArrayDenseCell_const< Cell_Type, Data_Type > Class Template Reference . . . . .	82
8.6.1 Detailed Description . . . . .	82
8.7 BArrayDenseCol< Cell_Type, Data_Type > Class Template Reference . . . . .	82
8.7.1 Detailed Description . . . . .	82
8.7.2 Constructor & Destructor Documentation . . . . .	82
8.7.2.1 BArrayDenseCol() . . . . .	83
8.7.3 Member Function Documentation . . . . .	83
8.7.3.1 begin() . . . . .	83
8.7.3.2 end() . . . . .	83
8.7.3.3 operator()( ) . . . . .	83
8.7.3.4 size() . . . . .	83
8.7.4 Friends And Related Function Documentation . . . . .	84
8.7.4.1 BArrayDense< Cell_Type, Data_Type > . . . . .	84
8.7.4.2 BArrayDenseCell< Cell_Type, Data_Type > . . . . .	84
8.7.4.3 BArrayDenseCell_const< Cell_Type, Data_Type > . . . . .	84
8.8 BArrayDenseCol_const< Cell_Type, Data_Type > Class Template Reference . . . . .	84
8.8.1 Detailed Description . . . . .	85
8.8.2 Constructor & Destructor Documentation . . . . .	85

8.8.2.1 BArrayDenseCol_const()	85
8.8.3 Member Function Documentation	85
8.8.3.1 begin()	85
8.8.3.2 end()	85
8.8.3.3 operator>()	86
8.8.3.4 size()	86
8.8.4 Friends And Related Function Documentation	86
8.8.4.1 BArrayDenseCell< Cell_Type, Data_Type >	86
8.8.4.2 BArrayDenseCell_const< Cell_Type, Data_Type >	86
8.9 BArrayDenseRow< Cell_Type, Data_Type > Class Template Reference	86
8.9.1 Detailed Description	87
8.9.2 Constructor & Destructor Documentation	87
8.9.2.1 BArrayDenseRow()	87
8.9.3 Member Function Documentation	87
8.9.3.1 begin()	87
8.9.3.2 end()	88
8.9.3.3 operator>()	88
8.9.3.4 size()	88
8.9.4 Friends And Related Function Documentation	88
8.9.4.1 BArrayDense< Cell_Type, Data_Type >	88
8.9.4.2 BArrayDenseCell< Cell_Type, Data_Type >	88
8.9.4.3 BArrayDenseCell_const< Cell_Type, Data_Type >	89
8.10 BArrayDenseRow_const< Cell_Type, Data_Type > Class Template Reference	89
8.10.1 Detailed Description	89
8.10.2 Constructor & Destructor Documentation	89
8.10.2.1 BArrayDenseRow_const()	90
8.10.3 Member Function Documentation	90
8.10.3.1 begin()	90
8.10.3.2 end()	90
8.10.3.3 operator>()	90
8.10.3.4 size()	90
8.10.4 Friends And Related Function Documentation	91
8.10.4.1 BArrayDenseCell< Cell_Type, Data_Type >	91
8.10.4.2 BArrayDenseCell_const< Cell_Type, Data_Type >	91
8.11 BArrayRow< Cell_Type, Data_Type > Class Template Reference	91
8.11.1 Detailed Description	91
8.11.2 Constructor & Destructor Documentation	92
8.11.2.1 BArrayRow()	92
8.11.2.2 ~BArrayRow()	92
8.11.3 Member Function Documentation	92
8.11.3.1 operator BArrayRow< Cell_Type, Data_Type >()	92
8.11.3.2 operator*=( )	92

8.11.3.3 operator+=()	92
8.11.3.4 operator-=()	93
8.11.3.5 operator/=(())	93
8.11.3.6 operator=()	93
8.11.3.7 operator==(())	93
8.12 BArrayRow_const< Cell_Type, Data_Type > Class Template Reference	93
8.12.1 Detailed Description	94
8.12.2 Constructor & Destructor Documentation	94
8.12.2.1 BArrayRow_const()	94
8.12.2.2 ~BArrayRow_const()	94
8.12.3 Member Function Documentation	94
8.12.3.1 operator BArrayRow_const< Cell_Type, Data_Type >()	94
8.12.3.2 operator!=(())	94
8.12.3.3 operator<()	95
8.12.3.4 operator<=()	95
8.12.3.5 operator==(())	95
8.12.3.6 operator>()	95
8.12.3.7 operator>=()	95
8.13 BArrayVector< Cell_Type, Data_Type > Class Template Reference	95
8.13.1 Detailed Description	96
8.13.2 Constructor & Destructor Documentation	96
8.13.2.1 BArrayVector()	96
8.13.2.2 ~BArrayVector()	97
8.13.3 Member Function Documentation	97
8.13.3.1 begin()	97
8.13.3.2 end()	97
8.13.3.3 is_col()	97
8.13.3.4 is_row()	98
8.13.3.5 operator std::vector< Cell_Type >()	98
8.13.3.6 operator*=(())	98
8.13.3.7 operator+=()	98
8.13.3.8 operator-=()	98
8.13.3.9 operator/=(())	99
8.13.3.10 operator=()	99
8.13.3.11 operator==(())	99
8.13.3.12 size()	99
8.14 BArrayVector_const< Cell_Type, Data_Type > Class Template Reference	99
8.14.1 Detailed Description	100
8.14.2 Constructor & Destructor Documentation	100
8.14.2.1 BArrayVector_const()	100
8.14.2.2 ~BArrayVector_const()	100
8.14.3 Member Function Documentation	101

8.14.3.1 begin()	101
8.14.3.2 end()	101
8.14.3.3 is_col()	101
8.14.3.4 is_row()	101
8.14.3.5 operator std::vector< Cell_Type >()	101
8.14.3.6 operator!=(())	102
8.14.3.7 operator<()	102
8.14.3.8 operator<=()	102
8.14.3.9 operator==(())	102
8.14.3.10 operator>()	102
8.14.3.11 operator>=()	103
8.14.3.12 size()	103
8.15 Cell< Cell_Type > Class Template Reference	103
8.15.1 Detailed Description	104
8.15.2 Constructor & Destructor Documentation	104
8.15.2.1 Cell() [1/7]	104
8.15.2.2 Cell() [2/7]	104
8.15.2.3 ~Cell()	104
8.15.2.4 Cell() [3/7]	105
8.15.2.5 Cell() [4/7]	105
8.15.2.6 Cell() [5/7]	105
8.15.2.7 Cell() [6/7]	105
8.15.2.8 Cell() [7/7]	105
8.15.3 Member Function Documentation	105
8.15.3.1 add() [1/4]	106
8.15.3.2 add() [2/4]	106
8.15.3.3 add() [3/4]	106
8.15.3.4 add() [4/4]	106
8.15.3.5 operator Cell_Type()	106
8.15.3.6 operator!=(())	106
8.15.3.7 operator=(()) [1/2]	107
8.15.3.8 operator=(()) [2/2]	107
8.15.3.9 operator==(())	107
8.15.4 Member Data Documentation	107
8.15.4.1 active	107
8.15.4.2 value	107
8.15.4.3 visited	108
8.16 Cell_const< Cell_Type > Class Template Reference	108
8.16.1 Detailed Description	108
8.17 ConstBArrayRowIter< Cell_Type, Data_Type > Class Template Reference	108
8.17.1 Detailed Description	109
8.17.2 Constructor & Destructor Documentation	109

8.17.2.1 ConstBArrayRowIter()	109
8.17.2.2 ~ConstBArrayRowIter()	109
8.17.3 Member Data Documentation	109
8.17.3.1 Array	110
8.17.3.2 current_col	110
8.17.3.3 current_row	110
8.17.3.4 iter	110
8.18 Counter< Array_Type, Data_Type > Class Template Reference	110
8.18.1 Detailed Description	111
8.18.2 Constructor & Destructor Documentation	112
8.18.2.1 Counter() [1/4]	112
8.18.2.2 Counter() [2/4]	112
8.18.2.3 Counter() [3/4]	112
8.18.2.4 Counter() [4/4]	112
8.18.2.5 ~Counter()	113
8.18.3 Member Function Documentation	113
8.18.3.1 count()	113
8.18.3.2 get_description()	113
8.18.3.3 get_hasher()	113
8.18.3.4 get_name()	113
8.18.3.5 init()	113
8.18.3.6 operator=() [1/2]	114
8.18.3.7 operator=() [2/2]	114
8.18.3.8 set_hasher()	114
8.18.4 Member Data Documentation	114
8.18.4.1 count_fun	114
8.18.4.2 data	115
8.18.4.3 desc	115
8.18.4.4 hasher_fun	115
8.18.4.5 init_fun	115
8.18.4.6 name	115
8.19 Counters< Array_Type, Data_Type > Class Template Reference	116
8.19.1 Detailed Description	116
8.19.2 Constructor & Destructor Documentation	116
8.19.2.1 Counters() [1/3]	117
8.19.2.2 ~Counters()	117
8.19.2.3 Counters() [2/3]	117
8.19.2.4 Counters() [3/3]	117
8.19.3 Member Function Documentation	117
8.19.3.1 add_counter() [1/2]	118
8.19.3.2 add_counter() [2/2]	118
8.19.3.3 add_hash()	118

8.19.3.4 gen_hash()	118
8.19.3.5 get_descriptions()	119
8.19.3.6 get_names()	119
8.19.3.7 operator=() [1/2]	119
8.19.3.8 operator=() [2/2]	119
8.19.3.9 operator[]()	120
8.19.3.10 size()	120
8.20 DEFM Class Reference	121
8.20.1 Detailed Description	122
8.20.2 Constructor & Destructor Documentation	122
8.20.2.1 DEFM()	122
8.20.3 Member Function Documentation	122
8.20.3.1 get_ID()	122
8.20.3.2 get_m_order()	122
8.20.3.3 get_model()	123
8.20.3.4 get_n_covars()	123
8.20.3.5 get_n_obs()	123
8.20.3.6 get_n_rows()	123
8.20.3.7 get_n_y()	123
8.20.3.8 get_X()	123
8.20.3.9 get_X_names()	124
8.20.3.10 get_Y()	124
8.20.3.11 get_Y_names()	124
8.20.3.12 init()	124
8.20.3.13 is_motif()	124
8.20.3.14 likelihood()	124
8.20.3.15 logodds()	125
8.20.3.16 motif_census()	125
8.20.3.17 print()	125
8.20.3.18 set_names()	125
8.20.3.19 simulate()	125
8.21 DEFMCounterData Class Reference	126
8.21.1 Detailed Description	126
8.21.2 Constructor & Destructor Documentation	126
8.21.2.1 DEFMCounterData() [1/2]	126
8.21.2.2 DEFMCounterData() [2/2]	127
8.21.2.3 ~DEFMCounterData()	127
8.21.3 Member Function Documentation	127
8.21.3.1 idx()	127
8.21.3.2 is_true()	127
8.21.3.3 num()	127
8.21.4 Member Data Documentation	128

8.21.4.1 indices	128
8.21.4.2 is_motif	128
8.21.4.3 logical	128
8.21.4.4 numbers	128
8.22 DEFMDData Class Reference	129
8.22.1 Detailed Description	130
8.22.2 Constructor & Destructor Documentation	130
8.22.2.1 DEFMDData() [1/2]	130
8.22.2.2 DEFMDData() [2/2]	130
8.22.2.3 ~DEFMDData()	130
8.22.3 Member Function Documentation	131
8.22.3.1 at()	131
8.22.4 Member Data Documentation	131
8.22.4.1 array	131
8.22.4.2 covar_sort	131
8.22.4.3 covar_used	131
8.22.4.4 covariates	131
8.22.4.5 obs_start	132
8.22.4.6 X_ncol	132
8.22.4.7 X_nrow	132
8.23 DEFMRuleData Class Reference	132
8.23.1 Detailed Description	133
8.23.2 Constructor & Destructor Documentation	133
8.23.2.1 DEFMRuleData() [1/3]	133
8.23.2.2 DEFMRuleData() [2/3]	133
8.23.2.3 DEFMRuleData() [3/3]	134
8.23.3 Member Function Documentation	134
8.23.3.1 idx()	134
8.23.3.2 is_true()	134
8.23.3.3 num()	134
8.23.4 Member Data Documentation	134
8.23.4.1 indices	134
8.23.4.2 init	135
8.23.4.3 logical	135
8.23.4.4 numbers	135
8.24 DEFMRuleDynData Class Reference	135
8.24.1 Detailed Description	136
8.24.2 Constructor & Destructor Documentation	136
8.24.2.1 DEFMRuleDynData()	136
8.24.2.2 ~DEFMRuleDynData()	137
8.24.3 Member Data Documentation	137
8.24.3.1 counts	137



8.25 Entries< Cell_Type > Class Template Reference	137
8.25.1 Detailed Description	137
8.25.2 Constructor & Destructor Documentation	138
8.25.2.1 Entries() [1/2]	138
8.25.2.2 Entries() [2/2]	138
8.25.2.3 ~Entries()	138
8.25.3 Member Function Documentation	138
8.25.3.1 resize()	138
8.25.4 Member Data Documentation	139
8.25.4.1 source	139
8.25.4.2 target	139
8.25.4.3 val	139
8.26 Flock Class Reference	139
8.26.1 Detailed Description	140
8.26.2 Constructor & Destructor Documentation	141
8.26.2.1 Flock()	141
8.26.2.2 ~Flock()	141
8.26.3 Member Function Documentation	141
8.26.3.1 add_data()	141
8.26.3.2 colnames()	142
8.26.3.3 get_counters()	142
8.26.3.4 get_model()	142
8.26.3.5 get_stats_support()	142
8.26.3.6 get_stats_target()	142
8.26.3.7 get_support_fun()	142
8.26.3.8 init()	143
8.26.3.9 likelihood_joint()	143
8.26.3.10 nfuncs()	143
8.26.3.11 nleaves()	143
8.26.3.12 nnodes()	144
8.26.3.13 nterms()	144
8.26.3.14 ntrees()	144
8.26.3.15 operator()()	144
8.26.3.16 parse_polytomies()	144
8.26.3.17 print()	145
8.26.3.18 set_seed()	145
8.26.3.19 support_size()	145
8.26.4 Member Data Documentation	145
8.26.4.1 dat	145
8.26.4.2 initialized	146
8.26.4.3 model	146
8.26.4.4 nfunctions	146

8.26.4.5 <code>rengine</code>	146
8.27 <code>FreqTable&lt; T &gt;</code> Class Template Reference	146
8.27.1 Detailed Description	147
8.27.2 Constructor & Destructor Documentation	147
8.27.2.1 <code>FreqTable()</code>	147
8.27.2.2 <code>~FreqTable()</code>	148
8.27.3 Member Function Documentation	148
8.27.3.1 <code>add()</code>	148
8.27.3.2 <code>as_vector()</code>	148
8.27.3.3 <code>clear()</code>	148
8.27.3.4 <code>get_data()</code>	148
8.27.3.5 <code>get_index()</code>	149
8.27.3.6 <code>make_hash()</code>	149
8.27.3.7 <code>print()</code>	149
8.27.3.8 <code>reserve()</code>	149
8.27.3.9 <code>size()</code>	149
8.28 Geese Class Reference	150
8.28.1 Detailed Description	153
8.28.2 Constructor & Destructor Documentation	153
8.28.2.1 <code>Geese()</code> [1/4]	153
8.28.2.2 <code>Geese()</code> [2/4]	154
8.28.2.3 <code>Geese()</code> [3/4]	154
8.28.2.4 <code>Geese()</code> [4/4]	154
8.28.2.5 <code>~Geese()</code>	154
8.28.3 Member Function Documentation	154
8.28.3.1 <code>calc_reduced_sequence()</code>	154
8.28.3.2 <code>calc_sequence()</code>	155
8.28.3.3 <code>colnames()</code>	155
8.28.3.4 <code>get_annotated_nodes()</code>	155
8.28.3.5 <code>get_counters()</code>	155
8.28.3.6 <code>get_model()</code>	155
8.28.3.7 <code>get_probabilities()</code>	155
8.28.3.8 <code>get_rengine()</code>	156
8.28.3.9 <code>get_states()</code>	156
8.28.3.10 <code>get_support_fun()</code>	156
8.28.3.11 <code>inherit_support()</code>	156
8.28.3.12 <code>init()</code>	156
8.28.3.13 <code>init_node()</code>	157
8.28.3.14 <code>likelihood()</code>	157
8.28.3.15 <code>likelihood_exhaust()</code>	157
8.28.3.16 <code>nannotations()</code>	157
8.28.3.17 <code>nfuncs()</code>	157

8.28.3.18 nleaves()	158
8.28.3.19 nnodes()	158
8.28.3.20 nterms()	158
8.28.3.21 observed_counts()	158
8.28.3.22 operator=() [1/2]	158
8.28.3.23 operator=() [2/2]	158
8.28.3.24 parse_polytomies()	159
8.28.3.25 predict()	159
8.28.3.26 predict_backend()	159
8.28.3.27 predict_exhaust()	159
8.28.3.28 predict_exhaust_backend()	160
8.28.3.29 predict_sim()	160
8.28.3.30 print()	160
8.28.3.31 print_observed_counts()	160
8.28.3.32 set_seed()	160
8.28.3.33 simulate()	161
8.28.3.34 support_size()	161
8.28.3.35 update_annotations()	161
8.28.4 Member Data Documentation	161
8.28.4.1 delete_engine	161
8.28.4.2 delete_support	161
8.28.4.3 initialized	162
8.28.4.4 map_to_nodes	162
8.28.4.5 nfunctions	162
8.28.4.6 nodes	162
8.28.4.7 pset_loc	162
8.28.4.8 reduced_sequence	162
8.28.4.9 sequence	163
8.29 Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference	163
8.29.1 Detailed Description	167
8.29.2 Constructor & Destructor Documentation	167
8.29.2.1 Model() [1/3]	168
8.29.2.2 Model() [2/3]	168
8.29.2.3 Model() [3/3]	168
8.29.2.4 ~Model()	168
8.29.3 Member Function Documentation	168
8.29.3.1 add_array()	168
8.29.3.2 add_counter() [1/2]	169
8.29.3.3 add_counter() [2/2]	169
8.29.3.4 add_hasher()	169
8.29.3.5 add_rule() [1/2]	169

8.29.3.6 add_rule() [2/2]	170
8.29.3.7 add_rule_dyn() [1/2]	170
8.29.3.8 add_rule_dyn() [2/2]	170
8.29.3.9 colnames()	170
8.29.3.10 conditional_prob()	170
8.29.3.11 gen_key()	171
8.29.3.12 get_arrays2support()	171
8.29.3.13 get_counters()	171
8.29.3.14 get_norm_const()	171
8.29.3.15 get_pset()	172
8.29.3.16 get_pset_arrays()	172
8.29.3.17 get_pset_probs()	172
8.29.3.18 get_pset_stats() [1/2]	172
8.29.3.19 get_pset_stats() [2/2]	172
8.29.3.20 get_rengine()	173
8.29.3.21 get_rules()	173
8.29.3.22 get_rules_dyn()	173
8.29.3.23 get_stats_support()	173
8.29.3.24 get_stats_target()	173
8.29.3.25 get_support_fun()	174
8.29.3.26 likelihood() [1/4]	174
8.29.3.27 likelihood() [2/4]	174
8.29.3.28 likelihood() [3/4]	174
8.29.3.29 likelihood() [4/4]	175
8.29.3.30 likelihood_total()	175
8.29.3.31 nrules()	175
8.29.3.32 nrules_dyn()	175
8.29.3.33 nterms()	175
8.29.3.34 operator=()	176
8.29.3.35 print()	176
8.29.3.36 print_stats()	176
8.29.3.37 sample() [1/2]	176
8.29.3.38 sample() [2/2]	177
8.29.3.39 set_counters()	177
8.29.3.40 set_rengine()	177
8.29.3.41 set_rules()	177
8.29.3.42 set_rules_dyn()	177
8.29.3.43 set_seed()	178
8.29.3.44 set_transform_model()	178
8.29.3.45 size()	178
8.29.3.46 size_unique()	178
8.29.3.47 store_psets()	179

8.29.3.48 support_size()	179
8.29.3.49 transform_model()	179
8.29.4 Member Data Documentation	179
8.29.4.1 arrays2support	179
8.29.4.2 counter_fun	179
8.29.4.3 counters	180
8.29.4.4 delete_counters	180
8.29.4.5 delete_engine	180
8.29.4.6 delete_rules	180
8.29.4.7 delete_rules_dyn	180
8.29.4.8 first_calc_done	181
8.29.4.9 keys2support	181
8.29.4.10 normalizing_constants	181
8.29.4.11 params_last	181
8.29.4.12 pset_arrays	182
8.29.4.13 pset_probs	182
8.29.4.14 pset_stats	182
8.29.4.15 rengine	182
8.29.4.16 rules	183
8.29.4.17 rules_dyn	183
8.29.4.18 stats_support	183
8.29.4.19 stats_support_n_arrays	183
8.29.4.20 stats_target	184
8.29.4.21 support_fun	184
8.29.4.22 transform_model_fun	184
8.29.4.23 transform_model_term_names	185
8.29.4.24 with_pset	185
8.30 NetCounterData Class Reference	185
8.30.1 Detailed Description	185
8.30.2 Constructor & Destructor Documentation	186
8.30.2.1 NetCounterData() [1/2]	186
8.30.2.2 NetCounterData() [2/2]	186
8.30.2.3 ~NetCounterData()	186
8.30.3 Member Data Documentation	186
8.30.3.1 indices	186
8.30.3.2 numbers	186
8.31 NetworkData Class Reference	187
8.31.1 Detailed Description	187
8.31.2 Constructor & Destructor Documentation	187
8.31.2.1 NetworkData() [1/3]	187
8.31.2.2 NetworkData() [2/3]	187
8.31.2.3 NetworkData() [3/3]	188

8.31.2.4 <code>~NetworkData()</code>	188
8.31.3 Member Data Documentation	188
8.31.3.1 <code>directed</code>	188
8.31.3.2 <code>vertex_attr</code>	189
8.32 Node Class Reference	189
8.32.1 Detailed Description	190
8.32.2 Constructor & Destructor Documentation	190
8.32.2.1 <code>Node()</code> [1/5]	190
8.32.2.2 <code>Node()</code> [2/5]	191
8.32.2.3 <code>Node()</code> [3/5]	191
8.32.2.4 <code>Node()</code> [4/5]	191
8.32.2.5 <code>Node()</code> [5/5]	191
8.32.2.6 <code>~Node()</code>	191
8.32.3 Member Function Documentation	191
8.32.3.1 <code>get_parent()</code>	192
8.32.3.2 <code>is_leaf()</code>	192
8.32.3.3 <code>noffspring()</code>	192
8.32.4 Member Data Documentation	192
8.32.4.1 <code>annotations</code>	192
8.32.4.2 <code>array</code>	192
8.32.4.3 <code>arrays</code>	193
8.32.4.4 <code>arrays_valid</code>	193
8.32.4.5 <code>duplication</code>	193
8.32.4.6 <code>id</code>	193
8.32.4.7 <code>narray</code>	193
8.32.4.8 <code>offspring</code>	194
8.32.4.9 <code>ord</code>	194
8.32.4.10 <code>parent</code>	194
8.32.4.11 <code>probability</code>	194
8.32.4.12 <code>subtree_prob</code>	194
8.32.4.13 <code>visited</code>	195
8.33 NodeData Class Reference	195
8.33.1 Detailed Description	195
8.33.2 Constructor & Destructor Documentation	195
8.33.2.1 <code>NodeData()</code>	195
8.33.3 Member Data Documentation	196
8.33.3.1 <code>blengths</code>	196
8.33.3.2 <code>duplication</code>	196
8.33.3.3 <code>states</code>	196
8.34 PhyloCounterData Class Reference	196
8.34.1 Detailed Description	197
8.34.2 Constructor & Destructor Documentation	197

8.34.2.1 PhyloCounterData() [1/2]	197
8.34.2.2 PhyloCounterData() [2/2]	197
8.34.3 Member Function Documentation	197
8.34.3.1 at()	197
8.34.3.2 begin()	197
8.34.3.3 empty()	198
8.34.3.4 end()	198
8.34.3.5 get_counters()	198
8.34.3.6 operator()()	198
8.34.3.7 operator[]()	198
8.34.3.8 push_back()	198
8.34.3.9 reserve()	199
8.34.3.10 shrink_to_fit()	199
8.34.3.11 size()	199
8.35 PhyloRuleDynData Class Reference	199
8.35.1 Detailed Description	200
8.35.2 Constructor & Destructor Documentation	200
8.35.2.1 PhyloRuleDynData()	200
8.35.2.2 ~PhyloRuleDynData()	200
8.35.3 Member Function Documentation	200
8.35.3.1 operator()()	200
8.35.4 Member Data Documentation	200
8.35.4.1 counts	200
8.35.4.2 duplication	201
8.35.4.3 lb	201
8.35.4.4 pos	201
8.35.4.5 ub	201
8.36 PowerSet< Array_Type, Data_Rule_Type > Class Template Reference	201
8.36.1 Detailed Description	202
8.36.2 Constructor & Destructor Documentation	203
8.36.2.1 PowerSet() [1/3]	203
8.36.2.2 PowerSet() [2/3]	203
8.36.2.3 PowerSet() [3/3]	203
8.36.2.4 ~PowerSet()	203
8.36.3 Member Function Documentation	203
8.36.3.1 add_rule() [1/2]	204
8.36.3.2 add_rule() [2/2]	204
8.36.3.3 begin()	204
8.36.3.4 calc()	204
8.36.3.5 end()	204
8.36.3.6 get_data()	205
8.36.3.7 get_data_ptr()	205

8.36.3.8 init_support()	205
8.36.3.9 operator[]()	205
8.36.3.10 reset()	205
8.36.3.11 size()	206
8.36.4 Member Data Documentation	206
8.36.4.1 coordinates_free	206
8.36.4.2 coordinates_locked	206
8.36.4.3 data	206
8.36.4.4 EmptyArray	206
8.36.4.5 M	207
8.36.4.6 N	207
8.36.4.7 n_free	207
8.36.4.8 n_locked	207
8.36.4.9 rules	207
8.36.4.10 rules_deleted	208
8.37 Progress Class Reference	208
8.37.1 Detailed Description	208
8.37.2 Constructor & Destructor Documentation	208
8.37.2.1 Progress()	208
8.37.2.2 ~Progress()	209
8.37.3 Member Function Documentation	209
8.37.3.1 end()	209
8.37.3.2 next()	209
8.38 Rule< Array_Type, Data_Type > Class Template Reference	209
8.38.1 Detailed Description	210
8.38.2 Constructor & Destructor Documentation	210
8.38.2.1 Rule() [1/2]	210
8.38.2.2 Rule() [2/2]	210
8.38.2.3 ~Rule()	211
8.38.3 Member Function Documentation	211
8.38.3.1 D()	211
8.38.3.2 get_description() [1/2]	211
8.38.3.3 get_description() [2/2]	211
8.38.3.4 get_name() [1/2]	211
8.38.3.5 get_name() [2/2]	212
8.38.3.6 operator()()	212
8.39 Rules< Array_Type, Data_Type > Class Template Reference	212
8.39.1 Detailed Description	213
8.39.2 Constructor & Destructor Documentation	213
8.39.2.1 Rules() [1/2]	213
8.39.2.2 Rules() [2/2]	213
8.39.2.3 ~Rules()	214



8.39.3 Member Function Documentation	214
8.39.3.1 add_rule() [1/2]	214
8.39.3.2 add_rule() [2/2]	214
8.39.3.3 begin()	214
8.39.3.4 end()	214
8.39.3.5 get_descriptions()	215
8.39.3.6 get_names()	215
8.39.3.7 get_seq()	215
8.39.3.8 operator()()	215
8.39.3.9 operator=()	216
8.39.3.10 size()	216
8.40 StatsCounter< Array_Type, Data_Type > Class Template Reference	216
8.40.1 Detailed Description	217
8.40.2 Constructor & Destructor Documentation	217
8.40.2.1 StatsCounter() [1/3]	217
8.40.2.2 StatsCounter() [2/3]	218
8.40.2.3 StatsCounter() [3/3]	218
8.40.2.4 ~StatsCounter()	218
8.40.3 Member Function Documentation	218
8.40.3.1 add_counter()	218
8.40.3.2 count_all()	219
8.40.3.3 count_current()	219
8.40.3.4 count_init()	219
8.40.3.5 get_counters()	219
8.40.3.6 get_descriptions()	219
8.40.3.7 get_names()	219
8.40.3.8 reset_array()	219
8.40.3.9 set_counters()	220
8.40.3.10 size()	220
8.41 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference	220
8.41.1 Detailed Description	222
8.41.2 Constructor & Destructor Documentation	222
8.41.2.1 Support() [1/3]	222
8.41.2.2 Support() [2/3]	223
8.41.2.3 Support() [3/3]	223
8.41.2.4 ~Support()	223
8.41.3 Member Function Documentation	223
8.41.3.1 add_counter()	223
8.41.3.2 add_rule() [1/2]	224
8.41.3.3 add_rule() [2/2]	224
8.41.3.4 add_rule_dyn() [1/2]	224

8.41.3.5 add_rule_dyn() [2/2]	224
8.41.3.6 calc()	224
8.41.3.7 eval_rules_dyn()	225
8.41.3.8 get_counters()	225
8.41.3.9 get_counts()	225
8.41.3.10 get_current_stats()	225
8.41.3.11 get_data()	226
8.41.3.12 get_rules()	226
8.41.3.13 get_rules_dyn()	226
8.41.3.14 init_support()	226
8.41.3.15 print()	226
8.41.3.16 reset_array() [1/2]	227
8.41.3.17 reset_array() [2/2]	227
8.41.3.18 set_counters()	227
8.41.3.19 set_rules()	227
8.41.3.20 set_rules_dyn()	227
8.41.4 Member Data Documentation	227
8.41.4.1 change_stats	228
8.41.4.2 coordiantes_n_free	228
8.41.4.3 coordiantes_n_locked	228
8.41.4.4 coordinates_free	228
8.41.4.5 coordinates_locked	228
8.41.4.6 current_stats	229
8.41.4.7 delete_counters	229
8.41.4.8 delete_rules	229
8.41.4.9 delete_rules_dyn	229
8.41.4.10 hashes	229
8.41.4.11 hashes_initialized	230
8.41.4.12 M	230
8.41.4.13 max_num_elements	230
8.41.4.14 N	230
8.41.4.15 n_counters	230
8.42 vecHasher< T > Struct Template Reference	231
8.42.1 Detailed Description	231
8.42.2 Member Function Documentation	231
8.42.2.1 operator>()	231
<b>9 File Documentation</b>	<b>233</b>
9.1 include/barry/barray-bones.hpp File Reference	233
9.2 include/barry/barray-iterator.hpp File Reference	233
9.3 include/barry/barray-meat-operators.hpp File Reference	234
9.3.1 Macro Definition Documentation	234

9.3.1.1 BARRAY_TEMPLATE	235
9.3.1.2 BARRAY_TEMPLATE_ARGS	235
9.3.1.3 BARRAY_TYPE	235
9.3.1.4 COL	235
9.3.1.5 ROW	235
9.3.2 Function Documentation	235
9.3.2.1 BARRAY_TEMPLATE() [1/6]	236
9.3.2.2 BARRAY_TEMPLATE() [2/6]	236
9.3.2.3 BARRAY_TEMPLATE() [3/6]	236
9.3.2.4 BARRAY_TEMPLATE() [4/6]	236
9.3.2.5 BARRAY_TEMPLATE() [5/6]	236
9.3.2.6 BARRAY_TEMPLATE() [6/6]	237
9.3.2.7 BARRAY_TEMPLATE_ARGS()	237
9.3.2.8 BARRAY_TYPE()	237
9.3.2.9 for()	237
9.3.2.10 operator()()	237
9.3.3 Variable Documentation	237
9.3.3.1 rhs	238
9.3.3.2 this	238
9.4 include/barray/barray-meat.hpp File Reference	238
9.4.1 Macro Definition Documentation	240
9.4.1.1 BARRAY_TEMPLATE	240
9.4.1.2 BARRAY_TEMPLATE_ARGS	240
9.4.1.3 BARRAY_TYPE	241
9.4.1.4 COL	241
9.4.1.5 ROW	241
9.4.2 Function Documentation	241
9.4.2.1 ans()	241
9.4.2.2 BARRAY_TEMPLATE() [1/24]	241
9.4.2.3 BARRAY_TEMPLATE() [2/24]	242
9.4.2.4 BARRAY_TEMPLATE() [3/24]	242
9.4.2.5 BARRAY_TEMPLATE() [4/24]	242
9.4.2.6 BARRAY_TEMPLATE() [5/24]	242
9.4.2.7 BARRAY_TEMPLATE() [6/24]	242
9.4.2.8 BARRAY_TEMPLATE() [7/24]	242
9.4.2.9 BARRAY_TEMPLATE() [8/24]	243
9.4.2.10 BARRAY_TEMPLATE() [9/24]	243
9.4.2.11 BARRAY_TEMPLATE() [10/24]	243
9.4.2.12 BARRAY_TEMPLATE() [11/24]	243
9.4.2.13 BARRAY_TEMPLATE() [12/24]	243
9.4.2.14 BARRAY_TEMPLATE() [13/24]	244
9.4.2.15 BARRAY_TEMPLATE() [14/24]	244

9.4.2.16 BARRAY_TEMPLATE() [15/24]	244
9.4.2.17 BARRAY_TEMPLATE() [16/24]	244
9.4.2.18 BARRAY_TEMPLATE() [17/24]	244
9.4.2.19 BARRAY_TEMPLATE() [18/24]	244
9.4.2.20 BARRAY_TEMPLATE() [19/24]	245
9.4.2.21 BARRAY_TEMPLATE() [20/24]	245
9.4.2.22 BARRAY_TEMPLATE() [21/24]	245
9.4.2.23 BARRAY_TEMPLATE() [22/24]	245
9.4.2.24 BARRAY_TEMPLATE() [23/24]	245
9.4.2.25 BARRAY_TEMPLATE() [24/24]	245
9.4.2.26 COL()	246
9.4.2.27 for() [1/3]	246
9.4.2.28 for() [2/3]	246
9.4.2.29 for() [3/3]	246
9.4.2.30 if() [1/17]	246
9.4.2.31 if() [2/17]	246
9.4.2.32 if() [3/17]	247
9.4.2.33 if() [4/17]	247
9.4.2.34 if() [5/17]	247
9.4.2.35 if() [6/17]	247
9.4.2.36 if() [7/17]	247
9.4.2.37 if() [8/17]	247
9.4.2.38 if() [9/17]	248
9.4.2.39 if() [10/17]	248
9.4.2.40 if() [11/17]	248
9.4.2.41 if() [12/17]	248
9.4.2.42 if() [13/17]	248
9.4.2.43 if() [14/17]	248
9.4.2.44 if() [15/17]	248
9.4.2.45 if() [16/17]	249
9.4.2.46 if() [17/17]	249
9.4.2.47 M()	249
9.4.2.48 resize() [1/2]	249
9.4.2.49 resize() [2/2]	249
9.4.2.50 return()	249
9.4.2.51 ROW() [1/2]	250
9.4.2.52 ROW() [2/2]	250
9.4.3 Variable Documentation	250
9.4.3.1 add	250
9.4.3.2 ans	250
9.4.3.3 Array_	250
9.4.3.4 check_bounds	251

9.4.3.5 <a href="#">check_exists</a>	251
9.4.3.6 <a href="#">col0</a>	251
9.4.3.7 <a href="#">const</a>	251
9.4.3.8 <a href="#">copy_data</a>	252
9.4.3.9 <a href="#">data</a>	252
9.4.3.10 <a href="#">delete_data</a>	252
9.4.3.11 <a href="#">delete_data_</a>	252
9.4.3.12 <a href="#">else</a>	252
9.4.3.13 <a href="#">false</a>	253
9.4.3.14 <a href="#">first</a>	253
9.4.3.15 <a href="#">i1</a>	253
9.4.3.16 <a href="#">j</a>	253
9.4.3.17 <a href="#">j0</a>	253
9.4.3.18 <a href="#">j1</a>	253
9.4.3.19 <a href="#">M</a>	254
9.4.3.20 <a href="#">M_</a>	254
9.4.3.21 <a href="#">N</a>	254
9.4.3.22 <a href="#">NCells</a>	254
9.4.3.23 <a href="#">report</a>	254
9.4.3.24 <a href="#">return</a>	255
9.4.3.25 <a href="#">row0</a>	255
9.4.3.26 <a href="#">search</a>	255
9.4.3.27 <a href="#">source</a>	255
9.4.3.28 <a href="#">target</a>	255
9.4.3.29 <a href="#">v</a>	255
9.4.3.30 <a href="#">value</a>	256
9.5 <a href="#">include/barry/barraycell-bones.hpp File Reference</a>	256
9.6 <a href="#">include/barry/barraycell-meat.hpp File Reference</a>	256
9.7 <a href="#">include/barry/barraydense-bones.hpp File Reference</a>	257
9.8 <a href="#">include/barry/barraydense-meat-operators.hpp File Reference</a>	257
9.8.1 <a href="#">Macro Definition Documentation</a>	258
9.8.1.1 <a href="#">BDENSE_TEMPLATE</a>	258
9.8.1.2 <a href="#">BDENSE_TEMPLATE_ARGS</a>	258
9.8.1.3 <a href="#">BDENSE_TYPE</a>	258
9.8.1.4 <a href="#">COL</a>	258
9.8.1.5 <a href="#">POS</a>	259
9.8.1.6 <a href="#">POS_N</a>	259
9.8.1.7 <a href="#">ROW</a>	259
9.8.2 <a href="#">Function Documentation</a>	259
9.8.2.1 <a href="#">BDENSE_TEMPLATE()</a> [1/4]	259
9.8.2.2 <a href="#">BDENSE_TEMPLATE()</a> [2/4]	259
9.8.2.3 <a href="#">BDENSE_TEMPLATE()</a> [3/4]	260

9.8.2.4	<a href="#">BDENSE_TEMPLATE()</a> [4/4]	260
9.8.2.5	<a href="#">BDENSE_TEMPLATE_ARGS()</a>	260
9.8.2.6	<a href="#">BDENSE_TYPE()</a>	260
9.9	<a href="#">include/barry/barraydense-meat.hpp File Reference</a>	260
9.9.1	<a href="#">Macro Definition Documentation</a>	263
9.9.1.1	<a href="#">BDENSE_TEMPLATE</a>	263
9.9.1.2	<a href="#">BDENSE_TEMPLATE_ARGS</a>	263
9.9.1.3	<a href="#">BDENSE_TYPE</a>	263
9.9.1.4	<a href="#">COL</a>	263
9.9.1.5	<a href="#">POS</a>	263
9.9.1.6	<a href="#">POS_N</a>	264
9.9.1.7	<a href="#">ROW</a>	264
9.9.1.8	<a href="#">ZERO_CELL</a>	264
9.9.2	<a href="#">Function Documentation</a>	264
9.9.2.1	<a href="#">ans()</a>	264
9.9.2.2	<a href="#">BDENSE_TEMPLATE()</a> [1/39]	264
9.9.2.3	<a href="#">BDENSE_TEMPLATE()</a> [2/39]	265
9.9.2.4	<a href="#">BDENSE_TEMPLATE()</a> [3/39]	265
9.9.2.5	<a href="#">BDENSE_TEMPLATE()</a> [4/39]	265
9.9.2.6	<a href="#">BDENSE_TEMPLATE()</a> [5/39]	265
9.9.2.7	<a href="#">BDENSE_TEMPLATE()</a> [6/39]	265
9.9.2.8	<a href="#">BDENSE_TEMPLATE()</a> [7/39]	265
9.9.2.9	<a href="#">BDENSE_TEMPLATE()</a> [8/39]	266
9.9.2.10	<a href="#">BDENSE_TEMPLATE()</a> [9/39]	266
9.9.2.11	<a href="#">BDENSE_TEMPLATE()</a> [10/39]	266
9.9.2.12	<a href="#">BDENSE_TEMPLATE()</a> [11/39]	266
9.9.2.13	<a href="#">BDENSE_TEMPLATE()</a> [12/39]	266
9.9.2.14	<a href="#">BDENSE_TEMPLATE()</a> [13/39]	267
9.9.2.15	<a href="#">BDENSE_TEMPLATE()</a> [14/39]	267
9.9.2.16	<a href="#">BDENSE_TEMPLATE()</a> [15/39]	267
9.9.2.17	<a href="#">BDENSE_TEMPLATE()</a> [16/39]	267
9.9.2.18	<a href="#">BDENSE_TEMPLATE()</a> [17/39]	267
9.9.2.19	<a href="#">BDENSE_TEMPLATE()</a> [18/39]	268
9.9.2.20	<a href="#">BDENSE_TEMPLATE()</a> [19/39]	268
9.9.2.21	<a href="#">BDENSE_TEMPLATE()</a> [20/39]	268
9.9.2.22	<a href="#">BDENSE_TEMPLATE()</a> [21/39]	268
9.9.2.23	<a href="#">BDENSE_TEMPLATE()</a> [22/39]	268
9.9.2.24	<a href="#">BDENSE_TEMPLATE()</a> [23/39]	269
9.9.2.25	<a href="#">BDENSE_TEMPLATE()</a> [24/39]	269
9.9.2.26	<a href="#">BDENSE_TEMPLATE()</a> [25/39]	269
9.9.2.27	<a href="#">BDENSE_TEMPLATE()</a> [26/39]	269
9.9.2.28	<a href="#">BDENSE_TEMPLATE()</a> [27/39]	269

9.9.2.29	BDENSE_TEMPLATE() [28/39]	270
9.9.2.30	BDENSE_TEMPLATE() [29/39]	270
9.9.2.31	BDENSE_TEMPLATE() [30/39]	270
9.9.2.32	BDENSE_TEMPLATE() [31/39]	270
9.9.2.33	BDENSE_TEMPLATE() [32/39]	270
9.9.2.34	BDENSE_TEMPLATE() [33/39]	270
9.9.2.35	BDENSE_TEMPLATE() [34/39]	271
9.9.2.36	BDENSE_TEMPLATE() [35/39]	271
9.9.2.37	BDENSE_TEMPLATE() [36/39]	271
9.9.2.38	BDENSE_TEMPLATE() [37/39]	271
9.9.2.39	BDENSE_TEMPLATE() [38/39]	271
9.9.2.40	BDENSE_TEMPLATE() [39/39]	271
9.9.2.41	for()	272
9.9.2.42	if() [1/4]	272
9.9.2.43	if() [2/4]	272
9.9.2.44	if() [3/4]	272
9.9.2.45	if() [4/4]	272
9.9.2.46	insert_cell() [1/2]	272
9.9.2.47	insert_cell() [2/2]	273
9.9.2.48	M()	273
9.9.2.49	printf_barry()	273
9.9.2.50	resize() [1/6]	273
9.9.2.51	resize() [2/6]	273
9.9.2.52	resize() [3/6]	273
9.9.2.53	resize() [4/6]	274
9.9.2.54	resize() [5/6]	274
9.9.2.55	resize() [6/6]	274
9.9.2.56	rm_cell() [1/3]	274
9.9.2.57	rm_cell() [2/3]	274
9.9.2.58	rm_cell() [3/3]	274
9.9.2.59	va_end()	275
9.9.2.60	va_start()	275
9.9.3	Variable Documentation	275
9.9.3.1	add	275
9.9.3.2	ans	275
9.9.3.3	check_bounds	275
9.9.3.4	check_exists	276
9.9.3.5	col	276
9.9.3.6	const	276
9.9.3.7	copy_data	276
9.9.3.8	data	276
9.9.3.9	delete_data	277

9.9.3.10 delete_data_ . . . . .	277
9.9.3.11 el . . . . .	277
9.9.3.12 el_colsums . . . . .	277
9.9.3.13 el_rowsums . . . . .	277
9.9.3.14 else . . . . .	278
9.9.3.15 false . . . . .	278
9.9.3.16 i1 . . . . .	278
9.9.3.17 j . . . . .	278
9.9.3.18 j0 . . . . .	278
9.9.3.19 j1 . . . . .	278
9.9.3.20 M . . . . .	279
9.9.3.21 M_ . . . . .	279
9.9.3.22 N . . . . .	279
9.9.3.23 report . . . . .	279
9.9.3.24 return . . . . .	279
9.9.3.25 source . . . . .	280
9.9.3.26 target . . . . .	280
9.9.3.27 v . . . . .	280
9.9.3.28 val0 . . . . .	280
9.9.3.29 val1 . . . . .	280
9.9.3.30 value . . . . .	280
9.10 include/barry/barraydensecell-bones.hpp File Reference . . . . .	281
9.10.1 Macro Definition Documentation . . . . .	281
9.10.1.1 POS . . . . .	281
9.11 include/barry/barraydensecell-meat.hpp File Reference . . . . .	282
9.11.1 Macro Definition Documentation . . . . .	282
9.11.1.1 POS . . . . .	282
9.12 include/barry/barraydensecol-bones.hpp File Reference . . . . .	282
9.12.1 Macro Definition Documentation . . . . .	283
9.12.1.1 POS . . . . .	283
9.12.1.2 POS_N . . . . .	283
9.12.1.3 ZERO_CELL . . . . .	283
9.13 include/barry/barraydenserow-bones.hpp File Reference . . . . .	284
9.13.1 Macro Definition Documentation . . . . .	284
9.13.1.1 POS . . . . .	284
9.13.1.2 POS_N . . . . .	285
9.13.1.3 ZERO_CELL . . . . .	285
9.14 include/barry/barrayrow-bones.hpp File Reference . . . . .	285
9.15 include/barry/barrayrow-meat.hpp File Reference . . . . .	285
9.15.1 Macro Definition Documentation . . . . .	285
9.15.1.1 BROW_TEMPLATE . . . . .	286
9.15.1.2 BROW_TEMPLATE_ARGS . . . . .	286



9.15.1.3 BROW_TYPE . . . . .	286
9.15.2 Function Documentation . . . . .	286
9.15.2.1 BROW_TEMPLATE() [1/5] . . . . .	286
9.15.2.2 BROW_TEMPLATE() [2/5] . . . . .	286
9.15.2.3 BROW_TEMPLATE() [3/5] . . . . .	287
9.15.2.4 BROW_TEMPLATE() [4/5] . . . . .	287
9.15.2.5 BROW_TEMPLATE() [5/5] . . . . .	287
9.16 include/barry/barrayvector-bones.hpp File Reference . . . . .	287
9.17 include/barry/barrayvector-meat.hpp File Reference . . . . .	288
9.18 include/barry/barry-configuration.hpp File Reference . . . . .	288
9.18.1 Macro Definition Documentation . . . . .	289
9.18.1.1 BARRY_CHECK_SUPPORT . . . . .	289
9.18.1.2 BARRY_ISFINITE . . . . .	289
9.18.1.3 BARRY_MAX_NUM_ELEMENTS . . . . .	289
9.18.1.4 BARRY_SAFE_EXP . . . . .	289
9.18.1.5 printf_barry . . . . .	289
9.18.2 Typedef Documentation . . . . .	290
9.18.2.1 Map . . . . .	290
9.19 include/barry/barry-debug.hpp File Reference . . . . .	290
9.19.1 Macro Definition Documentation . . . . .	290
9.19.1.1 BARRY_DEBUG_LEVEL . . . . .	290
9.20 include/barry/barry-macros.hpp File Reference . . . . .	291
9.20.1 Macro Definition Documentation . . . . .	291
9.20.1.1 BARRY_ONE . . . . .	291
9.20.1.2 BARRY_ONE_DENSE . . . . .	291
9.20.1.3 BARRY_UNUSED . . . . .	292
9.20.1.4 BARRY_ZERO . . . . .	292
9.20.1.5 BARRY_ZERO_DENSE . . . . .	292
9.21 include/barry/barry.hpp File Reference . . . . .	292
9.21.1 Macro Definition Documentation . . . . .	293
9.21.1.1 BARRY_HPP . . . . .	294
9.21.1.2 BARRY_VERSION . . . . .	294
9.21.1.3 BARRY_VERSION_MAYOR . . . . .	294
9.21.1.4 BARRY_VERSION_MINOR . . . . .	294
9.21.1.5 COUNTER_FUNCTION . . . . .	294
9.21.1.6 COUNTER_LAMBDA . . . . .	295
9.21.1.7 RULE_FUNCTION . . . . .	295
9.21.1.8 RULE_LAMBDA . . . . .	295
9.22 include/barry/cell-bones.hpp File Reference . . . . .	295
9.23 include/barry/cell-meat.hpp File Reference . . . . .	296
9.24 include/barry/col-bones.hpp File Reference . . . . .	296
9.25 include/barry/counters-bones.hpp File Reference . . . . .	296

9.26 include/barry/counters-meat.hpp File Reference	297
9.26.1 Macro Definition Documentation	298
9.26.1.1 COUNTER_TEMPLATE	298
9.26.1.2 COUNTER_TEMPLATE_ARGS	299
9.26.1.3 COUNTER_TYPE	299
9.26.1.4 COUNTERS_TEMPLATE	299
9.26.1.5 COUNTERS_TEMPLATE_ARGS	299
9.26.1.6 COUNTERS_TYPE	299
9.26.1.7 TMP_HASHER_CALL	299
9.26.2 Function Documentation	300
9.26.2.1 count_fun()	300
9.26.2.2 COUNTER_TEMPLATE() [1/9]	300
9.26.2.3 COUNTER_TEMPLATE() [2/9]	300
9.26.2.4 COUNTER_TEMPLATE() [3/9]	300
9.26.2.5 COUNTER_TEMPLATE() [4/9]	300
9.26.2.6 COUNTER_TEMPLATE() [5/9]	301
9.26.2.7 COUNTER_TEMPLATE() [6/9]	301
9.26.2.8 COUNTER_TEMPLATE() [7/9]	301
9.26.2.9 COUNTER_TEMPLATE() [8/9]	301
9.26.2.10 COUNTER_TEMPLATE() [9/9]	301
9.26.2.11 COUNTERS_TEMPLATE() [1/9]	302
9.26.2.12 COUNTERS_TEMPLATE() [2/9]	302
9.26.2.13 COUNTERS_TEMPLATE() [3/9]	302
9.26.2.14 COUNTERS_TEMPLATE() [4/9]	302
9.26.2.15 COUNTERS_TEMPLATE() [5/9]	302
9.26.2.16 COUNTERS_TEMPLATE() [6/9]	302
9.26.2.17 COUNTERS_TEMPLATE() [7/9]	303
9.26.2.18 COUNTERS_TEMPLATE() [8/9]	303
9.26.2.19 COUNTERS_TEMPLATE() [9/9]	303
9.26.2.20 data()	303
9.26.2.21 desc()	303
9.26.2.22 for()	303
9.26.2.23 hasher() [1/2]	304
9.26.2.24 hasher() [2/2]	304
9.26.2.25 hasher_fun() [1/2]	304
9.26.2.26 hasher_fun() [2/2]	304
9.26.2.27 if() [1/3]	304
9.26.2.28 if() [2/3]	304
9.26.2.29 if() [3/3]	305
9.26.2.30 init_fun() [1/3]	305
9.26.2.31 init_fun() [2/3]	305
9.26.2.32 init_fun() [3/3]	305

9.26.2.33 name()	305
9.26.3 Variable Documentation	305
9.26.3.1 add_dims	305
9.26.3.2 count_fun_	306
9.26.3.3 counter	306
9.26.3.4 counter_	306
9.26.3.5 data_	306
9.26.3.6 desc_	307
9.26.3.7 fun	307
9.26.3.8 fun_	307
9.26.3.9 hasher_fun_	307
9.26.3.10 i	308
9.26.3.11 init_fun_	308
9.26.3.12 j	308
9.26.3.13 name_	308
9.26.3.14 noexcept	308
9.26.3.15 res	309
9.26.3.16 return	309
9.27 include/barry/counters/defm-formula.hpp File Reference	309
9.27.1 Function Documentation	309
9.27.1.1 defm_motif_parser()	310
9.28 include/barry/counters/defm.hpp File Reference	311
9.28.1 Macro Definition Documentation	313
9.28.1.1 UNI_SUB	313
9.28.2 Typedef Documentation	313
9.28.2.1 DEFMArray	313
9.29 include/barry/models/defm.hpp File Reference	314
9.30 include/barry/counters/network-css.hpp File Reference	314
9.30.1 Macro Definition Documentation	316
9.30.1.1 CSS_APPEND	316
9.30.1.2 CSS_CASE_ELSE	316
9.30.1.3 CSS_CASE_PERCEIVED	316
9.30.1.4 CSS_CASE_TRUTH	316
9.30.1.5 CSS_CHECK_SIZE	317
9.30.1.6 CSS_CHECK_SIZE_INIT	317
9.30.1.7 CSS_NET_COUNTER_LAMBDA_INIT	317
9.30.1.8 CSS_PERCEIVED_CELLS	317
9.30.1.9 CSS_SIZE	318
9.30.1.10 CSS_TRUE_CELLS	318
9.30.2 Function Documentation	318
9.30.2.1 counter_css_census01()	318
9.30.2.2 counter_css_census02()	318

9.30.2.3 counter_css_census03()	319
9.30.2.4 counter_css_census04()	319
9.30.2.5 counter_css_census05()	319
9.30.2.6 counter_css_census06()	319
9.30.2.7 counter_css_census07()	320
9.30.2.8 counter_css_census08()	320
9.30.2.9 counter_css_census09()	320
9.30.2.10 counter_css_census10()	320
9.30.2.11 counter_css_completely_false_recip_comiss()	321
9.30.2.12 counter_css_completely_false_recip_omiss()	321
9.30.2.13 counter_css_mixed_recip()	321
9.30.2.14 counter_css_partially_false_recip_commi()	321
9.30.2.15 counter_css_partially_false_recip_omiss()	322
9.31 include/barry/counters/network.hpp File Reference	322
9.31.1 Macro Definition Documentation	325
9.31.1.1 BARRY_ZERO_NETWORK	325
9.31.1.2 BARRY_ZERO_NETWORK_DENSE	326
9.31.1.3 NET_C_DATA_IDX	326
9.31.1.4 NET_C_DATA_NUM	326
9.31.1.5 NETWORK_COUNTER	326
9.31.1.6 NETWORK_COUNTER_LAMBDA	326
9.31.1.7 NETWORK_RULE	327
9.31.1.8 NETWORK_RULE_LAMBDA	327
9.31.1.9 NETWORKDENSE_COUNTER_LAMBDA	327
9.31.2 Typedef Documentation	327
9.31.2.1 NetCounter	327
9.31.2.2 NetCounters	328
9.31.2.3 NetModel	328
9.31.2.4 NetRule	328
9.31.2.5 NetRules	328
9.31.2.6 NetStatsCounter	328
9.31.2.7 NetSupport	328
9.31.2.8 Network	329
9.31.2.9 NetworkDense	329
9.31.3 Function Documentation	329
9.31.3.1 rules_zerodiag()	329
9.32 include/barry/counters/phylo.hpp File Reference	329
9.32.1 Macro Definition Documentation	332
9.32.1.1 DEFAULT_DUPLICATION	332
9.32.1.2 DUPL_DUPL	332
9.32.1.3 DUPL_EITH	332
9.32.1.4 DUPL_SPEC	332

9.32.1.5 IF_MATCHES . . . . .	333
9.32.1.6 IF_NOTMATCHES . . . . .	333
9.32.1.7 IS_DUPLICATION . . . . .	333
9.32.1.8 IS_EITHER . . . . .	333
9.32.1.9 IS_SPECIATION . . . . .	333
9.32.1.10 MAKE_DUPL_VARS . . . . .	334
9.32.1.11 PHYLO_CHECK_MISSING . . . . .	334
9.32.1.12 PHYLO_COUNTER_LAMBDA . . . . .	334
9.32.1.13 PHYLO_RULE_DYN_LAMBDA . . . . .	334
9.32.2 Typedef Documentation . . . . .	335
9.32.2.1 PhyloArray . . . . .	335
9.32.2.2 PhyloCounter . . . . .	335
9.32.2.3 PhyloCounters . . . . .	335
9.32.2.4 PhyloModel . . . . .	335
9.32.2.5 PhyloPowerSet . . . . .	335
9.32.2.6 PhyloRule . . . . .	336
9.32.2.7 PhyloRuleData . . . . .	336
9.32.2.8 PhyloRuleDyn . . . . .	336
9.32.2.9 PhyloRules . . . . .	336
9.32.2.10 PhyloRulesDyn . . . . .	336
9.32.2.11 PhyloStatsCounter . . . . .	336
9.32.2.12 PhyloSupport . . . . .	337
9.32.3 Function Documentation . . . . .	337
9.32.3.1 get_last_name() . . . . .	337
9.33 include/barry/freqtable.hpp File Reference . . . . .	337
9.34 include/barry/model-bones.hpp File Reference . . . . .	338
9.35 include/barry/model-meat.hpp File Reference . . . . .	338
9.35.1 Macro Definition Documentation . . . . .	340
9.35.1.1 MODEL_TEMPLATE . . . . .	340
9.35.1.2 MODEL_TEMPLATE_ARGS . . . . .	341
9.35.1.3 MODEL_TYPE . . . . .	341
9.35.2 Function Documentation . . . . .	341
9.35.2.1 for() . . . . .	341
9.35.2.2 if() [1/4] . . . . .	341
9.35.2.3 if() [2/4] . . . . .	341
9.35.2.4 if() [3/4] . . . . .	342
9.35.2.5 if() [4/4] . . . . .	342
9.35.2.6 insert_cell() . . . . .	342
9.35.2.7 likelihood_() . . . . .	342
9.35.2.8 MODEL_TEMPLATE() [1/33] . . . . .	342
9.35.2.9 MODEL_TEMPLATE() [2/33] . . . . .	343
9.35.2.10 MODEL_TEMPLATE() [3/33] . . . . .	343

9.35.2.11 MODEL_TEMPLATE()	[ 4/33]	343
9.35.2.12 MODEL_TEMPLATE()	[ 5/33]	343
9.35.2.13 MODEL_TEMPLATE()	[ 6/33]	343
9.35.2.14 MODEL_TEMPLATE()	[ 7/33]	344
9.35.2.15 MODEL_TEMPLATE()	[ 8/33]	344
9.35.2.16 MODEL_TEMPLATE()	[ 9/33]	344
9.35.2.17 MODEL_TEMPLATE()	[10/33]	344
9.35.2.18 MODEL_TEMPLATE()	[11/33]	344
9.35.2.19 MODEL_TEMPLATE()	[12/33]	345
9.35.2.20 MODEL_TEMPLATE()	[13/33]	345
9.35.2.21 MODEL_TEMPLATE()	[14/33]	345
9.35.2.22 MODEL_TEMPLATE()	[15/33]	345
9.35.2.23 MODEL_TEMPLATE()	[16/33]	345
9.35.2.24 MODEL_TEMPLATE()	[17/33]	346
9.35.2.25 MODEL_TEMPLATE()	[18/33]	346
9.35.2.26 MODEL_TEMPLATE()	[19/33]	346
9.35.2.27 MODEL_TEMPLATE()	[20/33]	346
9.35.2.28 MODEL_TEMPLATE()	[21/33]	346
9.35.2.29 MODEL_TEMPLATE()	[22/33]	347
9.35.2.30 MODEL_TEMPLATE()	[23/33]	347
9.35.2.31 MODEL_TEMPLATE()	[24/33]	347
9.35.2.32 MODEL_TEMPLATE()	[25/33]	347
9.35.2.33 MODEL_TEMPLATE()	[26/33]	347
9.35.2.34 MODEL_TEMPLATE()	[27/33]	347
9.35.2.35 MODEL_TEMPLATE()	[28/33]	348
9.35.2.36 MODEL_TEMPLATE()	[29/33]	348
9.35.2.37 MODEL_TEMPLATE()	[30/33]	348
9.35.2.38 MODEL_TEMPLATE()	[31/33]	348
9.35.2.39 MODEL_TEMPLATE()	[32/33]	348
9.35.2.40 MODEL_TEMPLATE()	[33/33]	348
9.35.2.41 push_back()	[ 1/2]	349
9.35.2.42 push_back()	[ 2/2]	349
9.35.2.43 return()		349
9.35.2.44 set_counters()		349
9.35.2.45 set_rules()		349
9.35.2.46 set_rules_dyn()		349
9.35.2.47 size()		349
9.35.2.48 temp_stats()		350
9.35.2.49 tmp_counts()		350
9.35.2.50 update_normalizing_constant()		350
9.35.2.51 urand()		350
9.35.3 Variable Documentation		350

9.35.3.1 a	350
9.35.3.2 count_fun_	351
9.35.3.3 counter	351
9.35.3.4 counters_	351
9.35.3.5 cumprob	351
9.35.3.6 data_	351
9.35.3.7 Data_Counter_Type	352
9.35.3.8 Data_Rule_Type	352
9.35.3.9 delete_rules	352
9.35.3.10 delete_rules_dyn	352
9.35.3.11 else	352
9.35.3.12 force_new	353
9.35.3.13 fun_	353
9.35.3.14 i	353
9.35.3.15 i_matches	353
9.35.3.16 init_fun_	353
9.35.3.17 j	354
9.35.3.18 k	354
9.35.3.19 key	354
9.35.3.20 locator	354
9.35.3.21 params	354
9.35.3.22 probs	354
9.35.3.23 pset_arrays	355
9.35.3.24 r	355
9.35.3.25 return	355
9.35.3.26 rule_fun_	355
9.35.3.27 rules	355
9.35.3.28 rules_	356
9.35.3.29 rules_dyn	356
9.35.3.30 stats	356
9.35.3.31 stats_support_n_arrays	356
9.36 include/barry/models/defm/defm-bones.hpp File Reference	356
9.37 include/barry/models/defm/defm-meat.hpp File Reference	357
9.37.1 Macro Definition Documentation	357
9.37.1.1 DEFM_LOOP_ARRAYS	357
9.37.1.2 DEFM_RANGES	358
9.37.2 Function Documentation	358
9.37.2.1 keygen_defm()	358
9.38 include/barry/models/geese.hpp File Reference	358
9.39 include/barry/models/geese/flock-bones.hpp File Reference	359
9.40 include/barry/models/geese/flock-meat.hpp File Reference	359
9.41 include/barry/models/geese/geese-bones.hpp File Reference	360

9.41.1 Macro Definition Documentation . . . . .	360
9.41.1.1 INITIALIZED . . . . .	361
9.41.2 Function Documentation . . . . .	361
9.41.2.1 keygen_full() . . . . .	361
9.41.2.2 RULE_FUNCTION() . . . . .	361
9.41.2.3 vec_diff() . . . . .	361
9.41.2.4 vector_caster() . . . . .	361
9.42 include/barry/models/geese/geese-meat-constructors.hpp File Reference . . . . .	362
9.43 include/barry/models/geese/geese-meat-likelihood.hpp File Reference . . . . .	362
9.44 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference . . . . .	363
9.45 include/barry/models/geese/geese-meat-predict.hpp File Reference . . . . .	364
9.46 include/barry/models/geese/geese-meat-predict_exhaust.hpp File Reference . . . . .	364
9.47 include/barry/models/geese/geese-meat-predict_sim.hpp File Reference . . . . .	365
9.48 include/barry/models/geese/geese-meat-simulate.hpp File Reference . . . . .	365
9.49 include/barry/models/geese/geese-meat.hpp File Reference . . . . .	366
9.50 include/barry/models/geese/geese-node-bones.hpp File Reference . . . . .	366
9.51 include/barry/powerset-bones.hpp File Reference . . . . .	367
9.52 include/barry/powerset-meat.hpp File Reference . . . . .	367
9.53 include/barry/progress.hpp File Reference . . . . .	368
9.53.1 Macro Definition Documentation . . . . .	368
9.53.1.1 BARRY_PROGRESS_BAR_WIDTH . . . . .	368
9.54 include/barry/rules-bones.hpp File Reference . . . . .	369
9.54.1 Function Documentation . . . . .	369
9.54.1.1 rule_fun_default() . . . . .	369
9.55 include/barry/rules-meat.hpp File Reference . . . . .	370
9.56 include/barry/statscounter-bones.hpp File Reference . . . . .	370
9.57 include/barry/statscounter-meat.hpp File Reference . . . . .	371
9.57.1 Macro Definition Documentation . . . . .	372
9.57.1.1 STATSCOUNTER_TEMPLATE . . . . .	372
9.57.1.2 STATSCOUNTER_TEMPLATE_ARGS . . . . .	372
9.57.1.3 STATSCOUNTER_TYPE . . . . .	372
9.57.2 Function Documentation . . . . .	372
9.57.2.1 clear() . . . . .	372
9.57.2.2 for() . . . . .	372
9.57.2.3 resize() . . . . .	373
9.57.2.4 STATSCOUNTER_TEMPLATE() [1/9] . . . . .	373
9.57.2.5 STATSCOUNTER_TEMPLATE() [2/9] . . . . .	373
9.57.2.6 STATSCOUNTER_TEMPLATE() [3/9] . . . . .	373
9.57.2.7 STATSCOUNTER_TEMPLATE() [4/9] . . . . .	373
9.57.2.8 STATSCOUNTER_TEMPLATE() [5/9] . . . . .	373
9.57.2.9 STATSCOUNTER_TEMPLATE() [6/9] . . . . .	374
9.57.2.10 STATSCOUNTER_TEMPLATE() [7/9] . . . . .	374



9.57.2.11 STATSCOUNTER_TEMPLATE() [8/9]	374
9.57.2.12 STATSCOUNTER_TEMPLATE() [9/9]	374
9.57.3 Variable Documentation	374
9.57.3.1 counter	374
9.57.3.2 counter_deleted	375
9.57.3.3 counters	375
9.57.3.4 counters_	375
9.57.3.5 current_stats	375
9.57.3.6 EmptyArray	375
9.57.3.7 f_	376
9.57.3.8 j	376
9.57.3.9 return	376
9.58 include/barry/support-bones.hpp File Reference	376
9.59 include/barry/support-meat.hpp File Reference	377
9.59.1 Macro Definition Documentation	378
9.59.1.1 BARRY_SUPPORT_MEAT_HPP	378
9.59.1.2 SUPPORT_TEMPLATE	378
9.59.1.3 SUPPORT_TEMPLATE_ARGS	379
9.59.1.4 SUPPORT_TYPE	379
9.59.2 Function Documentation	379
9.59.2.1 calc_backend_dense()	379
9.59.2.2 calc_backend_sparse()	379
9.59.2.3 for()	379
9.59.2.4 if() [1/3]	380
9.59.2.5 if() [2/3]	380
9.59.2.6 if() [3/3]	380
9.59.2.7 insert_cell() [1/2]	380
9.59.2.8 insert_cell() [2/2]	380
9.59.2.9 rm_cell()	381
9.59.2.10 SUPPORT_TEMPLATE() [1/17]	381
9.59.2.11 SUPPORT_TEMPLATE() [2/17]	381
9.59.2.12 SUPPORT_TEMPLATE() [3/17]	381
9.59.2.13 SUPPORT_TEMPLATE() [4/17]	381
9.59.2.14 SUPPORT_TEMPLATE() [5/17]	382
9.59.2.15 SUPPORT_TEMPLATE() [6/17]	382
9.59.2.16 SUPPORT_TEMPLATE() [7/17]	382
9.59.2.17 SUPPORT_TEMPLATE() [8/17]	382
9.59.2.18 SUPPORT_TEMPLATE() [9/17]	382
9.59.2.19 SUPPORT_TEMPLATE() [10/17]	382
9.59.2.20 SUPPORT_TEMPLATE() [11/17]	383
9.59.2.21 SUPPORT_TEMPLATE() [12/17]	383
9.59.2.22 SUPPORT_TEMPLATE() [13/17]	383

9.59.2.23 SUPPORT_TEMPLATE() [14/17]	383
9.59.2.24 SUPPORT_TEMPLATE() [15/17]	383
9.59.2.25 SUPPORT_TEMPLATE() [16/17]	384
9.59.2.26 SUPPORT_TEMPLATE() [17/17]	384
9.59.3 Variable Documentation	384
9.59.3.1 array_bank	384
9.59.3.2 change_stats_different	384
9.59.3.3 coord_i	384
9.59.3.4 coord_j	384
9.59.3.5 counters	385
9.59.3.6 counters_	385
9.59.3.7 delete_counters	385
9.59.3.8 delete_rules	385
9.59.3.9 delete_rules_dyn	385
9.59.3.10 else	386
9.59.3.11 f_	386
9.59.3.12 hashes	386
9.59.3.13 return	386
9.59.3.14 rules	386
9.59.3.15 rules_	387
9.59.3.16 rules_dyn	387
9.59.3.17 stats_bank	387
9.59.3.18 tmp_chng	387
9.60 include/barry/typedefs.hpp File Reference	388
9.60.1 Typedef Documentation	390
9.60.1.1 Col_type	390
9.60.1.2 Counter_fun_type	390
9.60.1.3 Counts_type	390
9.60.1.4 Hasher_fun_type	390
9.60.1.5 MapVec_type	391
9.60.1.6 Row_type	391
9.60.1.7 Rule_fun_type	391
9.60.2 Function Documentation	391
9.60.2.1 sort_array()	391
9.60.2.2 vec_equal()	392
9.60.2.3 vec_equal_approx()	392
9.60.2.4 vec_inner_prod() [1/2]	393
9.60.2.5 vec_inner_prod() [2/2]	393
9.61 README.md File Reference	393
<b>Index</b>	<b>395</b>

# Chapter 1

## Main Page

### Barry: your to-go motif accountant

This repository contains a C++ template library that essentially counts sufficient statistics on binary arrays. 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 . . . . .	13
Phylo rules . . . . .	14
DEFMArray counters . . . . .	19
Phylo counters . . . . .	30





## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

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

BArray< Cell_Type, Data_Type > . . . . .	43
BArray< bool, bool > . . . . .	43
BArray< Cell_Type, Data_Type > . . . . .	43
BArrayCell< Cell_Type, Data_Type > . . . . .	57
BArrayCell_const< Cell_Type, Data_Type > . . . . .	60
BArrayDense< Cell_Type, Data_Type > . . . . .	62
BArrayDense< bool, bool > . . . . .	62
BArrayDenseCell< Cell_Type, Data_Type > . . . . .	78
BArrayDenseCell_const< Cell_Type, Data_Type > . . . . .	82
BArrayDenseCol< Cell_Type, Data_Type > . . . . .	82
BArrayDenseCol_const< Cell_Type, Data_Type > . . . . .	84
BArrayDenseRow< Cell_Type, Data_Type > . . . . .	86
BArrayDenseRow_const< Cell_Type, Data_Type > . . . . .	89
BArrayRow< Cell_Type, Data_Type > . . . . .	91
BArrayRow_const< Cell_Type, Data_Type > . . . . .	93
BArrayVector< Cell_Type, Data_Type > . . . . .	95
BArrayVector_const< Cell_Type, Data_Type > . . . . .	99
Cell< Cell_Type > . . . . .	103
Cell< bool > . . . . .	103
Cell_const< Cell_Type > . . . . .	108
ConstBArrayRowIter< Cell_Type, Data_Type > . . . . .	108
Counter< Array_Type, Data_Type > . . . . .	110
Counters< Array_Type, Data_Type > . . . . .	116
Counters< Array_Type, Data_Type > . . . . .	116
Counters< BArray< bool, bool >, bool > . . . . .	116
Counters< BArray<>, bool > . . . . .	116
DEFMCounterData . . . . .	126
DEFMData . . . . .	129
barry::counters::defm::DEFMModel	
DEFM . . . . .	121
DEFMRuleData . . . . .	132
DEFMRuleDynData . . . . .	135
Entries< Cell_Type > . . . . .	137
Flock . . . . .	139
FreqTable< T > . . . . .	146

Geese . . . . .	150
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > . . . . .	163
NetCounterData . . . . .	185
NetworkData . . . . .	187
Node . . . . .	189
NodeData . . . . .	195
PhyloCounterData . . . . .	196
PhyloRuleDynData . . . . .	199
PowerSet< Array_Type, Data_Rule_Type > . . . . .	201
Progress . . . . .	208
Rule< Array_Type, Data_Type > . . . . .	209
Rules< Array_Type, Data_Type > . . . . .	212
Rules< BArray< bool, bool >, bool > . . . . .	212
Rules< BArray<>, bool > . . . . .	212
StatsCounter< Array_Type, Data_Type > . . . . .	216
StatsCounter< BArray<>, bool > . . . . .	216
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > . . . . .	220
Support< BArray<>, bool, bool, bool > . . . . .	220
vecHasher< T > . . . . .	231

## 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	43
<a href="#">BArrayCell&lt; Cell_Type, Data_Type &gt;</a>	57
<a href="#">BArrayCell_const&lt; Cell_Type, Data_Type &gt;</a>	60
<a href="#">BArrayDense&lt; Cell_Type, Data_Type &gt;</a>	
Baseline class for binary arrays	62
<a href="#">BArrayDenseCell&lt; Cell_Type, Data_Type &gt;</a>	78
<a href="#">BArrayDenseCell_const&lt; Cell_Type, Data_Type &gt;</a>	82
<a href="#">BArrayDenseCol&lt; Cell_Type, Data_Type &gt;</a>	82
<a href="#">BArrayDenseCol_const&lt; Cell_Type, Data_Type &gt;</a>	84
<a href="#">BArrayDenseRow&lt; Cell_Type, Data_Type &gt;</a>	86
<a href="#">BArrayDenseRow_const&lt; Cell_Type, Data_Type &gt;</a>	89
<a href="#">BArrayRow&lt; Cell_Type, Data_Type &gt;</a>	91
<a href="#">BArrayRow_const&lt; Cell_Type, Data_Type &gt;</a>	93
<a href="#">BArrayVector&lt; Cell_Type, Data_Type &gt;</a>	
Row or column of a <a href="#">BArray</a>	95
<a href="#">BArrayVector_const&lt; Cell_Type, Data_Type &gt;</a>	99
<a href="#">Cell&lt; Cell_Type &gt;</a>	
Entries in <a href="#">BArray</a> . For now, it only has two members:	103
<a href="#">Cell_const&lt; Cell_Type &gt;</a>	108
<a href="#">ConstBArrayRowIter&lt; Cell_Type, Data_Type &gt;</a>	108
<a href="#">Counter&lt; Array_Type, Data_Type &gt;</a>	
A counter function based on change statistics	110
<a href="#">Counters&lt; Array_Type, Data_Type &gt;</a>	
Vector of counters	116
<a href="#">DEFM</a>	121
<a href="#">DEFMCounterData</a>	
Data class used to store arbitrary size_t or double vectors	126
<a href="#">DEFMData</a>	129
<a href="#">DEFMRuleData</a>	132
<a href="#">DEFMRuleDynData</a>	135
<a href="#">Entries&lt; Cell_Type &gt;</a>	
A wrapper class to store source, target, val from a <a href="#">BArray</a> object	137
<a href="#">Flock</a>	
A <a href="#">Flock</a> is a group of <a href="#">Geese</a>	139

<a href="#">FreqTable&lt; T &gt;</a>	
Frequency table of vectors	146
<a href="#">Geese</a>	
Annotated Phylo <a href="#">Model</a>	150
<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:	163
<a href="#">NetCounterData</a>	
Data class used to store arbitrary size_t or double vectors	185
<a href="#">NetworkData</a>	
Data class for Networks	187
<a href="#">Node</a>	
A single node for the model	189
<a href="#">NodeData</a>	
Data definition for the <a href="#">PhyloArray</a> class	195
<a href="#">PhyloCounterData</a>	196
<a href="#">PhyloRuleDynData</a>	199
<a href="#">PowerSet&lt; Array_Type, Data_Rule_Type &gt;</a>	
Powerset of a binary array	201
<a href="#">Progress</a>	
A simple progress bar	208
<a href="#">Rule&lt; Array_Type, Data_Type &gt;</a>	
Rule for determining if a cell should be included in a sequence	209
<a href="#">Rules&lt; Array_Type, Data_Type &gt;</a>	
Vector of objects of class <a href="#">Rule</a>	212
<a href="#">StatsCounter&lt; Array_Type, Data_Type &gt;</a>	
Count stats for a single Array	216
<a href="#">Support&lt; Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type &gt;</a>	
Compute the support of sufficient statistics	220
<a href="#">vecHasher&lt; T &gt;</a>	231

## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

include/barry/barray-bones.hpp	233
include/barry/barray-iterator.hpp	233
include/barry/barray-meat-operators.hpp	234
include/barry/barray-meat.hpp	238
include/barry/barraycell-bones.hpp	256
include/barry/barraycell-meat.hpp	256
include/barry/barraydense-bones.hpp	257
include/barry/barraydense-meat-operators.hpp	257
include/barry/barraydense-meat.hpp	260
include/barry/barraydensecell-bones.hpp	281
include/barry/barraydensecell-meat.hpp	282
include/barry/barraydensecol-bones.hpp	282
include/barry/barraydenserow-bones.hpp	284
include/barry/barrayrow-bones.hpp	285
include/barry/barrayrow-meat.hpp	285
include/barry/barrayvector-bones.hpp	287
include/barry/barrayvector-meat.hpp	288
include/barry/barray-configuration.hpp	288
include/barry/barray-debug.hpp	290
include/barry/barray-macros.hpp	291
include/barry/barray.hpp	292
include/barry/cell-bones.hpp	295
include/barry/cell-meat.hpp	296
include/barry/col-bones.hpp	296
include/barry/counters-bones.hpp	296
include/barry/counters-meat.hpp	297
include/barry/freqtable.hpp	337
include/barry/model-bones.hpp	338
include/barry/model-meat.hpp	338
include/barry/powerset-bones.hpp	367
include/barry/powerset-meat.hpp	367
include/barry/progress.hpp	368
include/barry/rules-bones.hpp	369
include/barry/rules-meat.hpp	370
include/barry/statscounter-bones.hpp	370

include/barry/statscounter-meat.hpp	371
include/barry/support-bones.hpp	376
include/barry/support-meat.hpp	377
include/barry/typedefs.hpp	388
include/barry/counters/defm-formula.hpp	309
include/barry/counters/defm.hpp	311
include/barry/counters/network-css.hpp	314
include/barry/counters/network.hpp	322
include/barry/counters/phylo.hpp	329
include/barry/models/defm.hpp	314
include/barry/models/geese.hpp	358
include/barry/models/defm/defm-bones.hpp	356
include/barry/models/defm/defm-meat.hpp	357
include/barry/models/geese/flock-bones.hpp	359
include/barry/models/geese/flock-meat.hpp	359
include/barry/models/geese/geese-bones.hpp	360
include/barry/models/geese/geese-meat-constructors.hpp	362
include/barry/models/geese/geese-meat-likelihood.hpp	362
include/barry/models/geese/geese-meat-likelihood_exhaust.hpp	363
include/barry/models/geese/geese-meat-predict.hpp	364
include/barry/models/geese/geese-meat-predict_exhaust.hpp	364
include/barry/models/geese/geese-meat-predict_sim.hpp	365
include/barry/models/geese/geese-meat-simulate.hpp	365
include/barry/models/geese/geese-meat.hpp	366
include/barry/models/geese/geese-node-bones.hpp	366

## Chapter 6

# Module Documentation

### 6.1 Counting

#### Classes

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

#### 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.

### 6.2 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 Phylo rules

[Rules](#) for phylogenetic modeling.

## Classes

- class [DEFMRuleDynData](#)
- class [PhyloRuleDynData](#)

## Macros

- `#define MAKE\_DEFM\_HASHER(hasher, a, cov)`
- `#define DEFM\_RULEDYN\_LAMBDA(a)`

## Functions

- void [rule\\_dyn\\_limit\\_changes](#) ([PhyloSupport](#) \*support, size\_t pos, size\_t lb, size\_t ub, size\_t duplication=[DEFAULT\\_DUPLICATION](#))  
*Overall functional gains.*
- double [DEFMData::operator](#)() (size\_t i, size\_t j) const  
*Access to the row (i) column (j) data.*
- size\_t [DEFMData::ncol](#) () const
- size\_t [DEFMData::nrow](#) () const
- void [DEFMData::print](#) () const

### Convenient typedefs for network objects.

- typedef [Counter](#)< [DEFMArray](#), [DEFMCounterData](#) > [DEFMCounter](#)
- typedef [Counters](#)< [DEFMArray](#), [DEFMCounterData](#) > [DEFMCounters](#)
- typedef [Support](#)< [DEFMArray](#), [DEFMCounterData](#), [DEFMRuleData](#), [DEFMRuleDynData](#) > [DEFMSupport](#)
- typedef [StatsCounter](#)< [DEFMArray](#), [DEFMCounterData](#) > [DEFMStatsCounter](#)
- typedef [Model](#)< [DEFMArray](#), [DEFMCounterData](#), [DEFMRuleData](#), [DEFMRuleDynData](#) > [DEFMModel](#)
- typedef [Rule](#)< [DEFMArray](#), [DEFMRuleData](#) > [DEFMRule](#)
- typedef [Rules](#)< [DEFMArray](#), [DEFMRuleData](#) > [DEFMRules](#)
- typedef [Rule](#)< [DEFMArray](#), [DEFMRuleDynData](#) > [DEFMRuleDyn](#)
- typedef [Rules](#)< [DEFMArray](#), [DEFMRuleDynData](#) > [DEFMRulesDyn](#)



## Macros for defining counters

- `#define DEFM_COUNTER(a) inline double (a) (const DEFMArr & 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 DEFMArr & Array, size_t i, size_t j, bool & data)`
- `#define DEFM_RULE_LAMBDA(a)`

### 6.3.1 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> ).
--------------	--

### 6.3.2 Macro Definition Documentation

#### 6.3.2.1 DEFM\_COUNTER

```
#define DEFM_COUNTER(
    a ) inline double (a) (const DEFMArr & Array, size_t i, size_t j, DEFMCOUNTERData
& data)
```

Function for definition of a network counter function

Definition at line 214 of file `defm.hpp`.

#### 6.3.2.2 DEFM\_COUNTER\_LAMBDA

```
#define DEFM_COUNTER_LAMBDA(
    a )
```

#### Value:

```
Counter_fun_type<DEFMArr, DEFMCOUNTERData> a = \
[] (const DEFMArr & Array, size_t i, size_t j, DEFMCOUNTERData & data) -> double
```

Lambda function for definition of a network counter function

Definition at line 218 of file `defm.hpp`.

### 6.3.2.3 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 228 of file defm.hpp.

### 6.3.2.4 DEFM\_RULE\_LAMBDA

```
#define DEFM_RULE_LAMBDA(  
    a )
```

**Value:**

```
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 232 of file defm.hpp.

### 6.3.2.5 DEFM\_RULEDYN\_LAMBDA

```
#define DEFM_RULEDYN_LAMBDA(  
    a )
```

**Value:**

```
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 238 of file defm.hpp.

### 6.3.2.6 MAKE\_DEFM\_HASHER

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

**Value:**

```
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; \  
};
```

Definition at line 197 of file defm.hpp.

### 6.3.3 Typedef Documentation

#### 6.3.3.1 DEFMCOUNTER

```
typedef Counter<DEFMArray, DEFMCOUNTERData > DEFMCOUNTER
```

Definition at line 154 of file defm.hpp.

#### 6.3.3.2 DEFMCOUNTERS

```
typedef Counters<DEFMArray, DEFMCOUNTERData> DEFMCOUNTERS
```

Definition at line 155 of file defm.hpp.

#### 6.3.3.3 DEFMMODEL

```
typedef Model<DEFMArray, DEFMCOUNTERData,DEFMRULEData,DEFMRULEDynData> DEFMMODEL
```

Definition at line 158 of file defm.hpp.

#### 6.3.3.4 DEFMRULE

```
typedef Rule<DEFMArray, DEFMRULEData> DEFMRULE
```

Definition at line 161 of file defm.hpp.

#### 6.3.3.5 DEFMRULEDYN

```
typedef Rule<DEFMArray, DEFMRULEDynData> DEFMRULEDYN
```

Definition at line 163 of file defm.hpp.

#### 6.3.3.6 DEFMRules

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

Definition at line 162 of file defm.hpp.

#### 6.3.3.7 DEFMRulesDyn

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

Definition at line 164 of file defm.hpp.

#### 6.3.3.8 DEFMStatsCounter

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

Definition at line 157 of file defm.hpp.

#### 6.3.3.9 DEFMSupport

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

Definition at line 156 of file defm.hpp.

### 6.3.4 Function Documentation

#### 6.3.4.1 ncol()

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

Definition at line 175 of file defm.hpp.

#### 6.3.4.2 nrow()

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

Definition at line 179 of file defm.hpp.

#### 6.3.4.3 operator()(i)

```
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 170 of file defm.hpp.

**6.3.4.4 print()**

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

Definition at line 183 of file defm.hpp.

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

```
void rule_dyn_limit_changes (
    PhyloSupport * support,
    size_t pos,
    size_t lb,
    size_t ub,
    size_t duplication = DEFAULT_DUPLICATION ) [inline]
```

Overall functional gains.

## Parameters

<i>support</i>	<a href="#">PhyloSupport</a> 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 2182 of file phylo.hpp.

**6.4 DEFMArray counters**[Counters](#) for network models.

## 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)
 

*Prevalence of ones.*
- 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.*
- 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\\_iddegree](#) (NetCounters< Tnet > \*counters, std::vector< size\_t > d)  
*Counts number of vertices with a given in-degree.*
- template<> void [counter\\_iddegree](#) (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.*

## Returns true if the cell is free

### Parameters

<i>rules</i>	A pointer to a DEFMRules object ( <a href="#">Rules</a> <DEFMArray, bool>).
--------------	---

- 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.*

## 6.4.1 Detailed Description

[Counters](#) for network models.

### Parameters

<i>counters</i>	A pointer to a DEFMCounters object ( <a href="#">Counters</a> <DEFMArray, DEFMCounterData>).
<i>counters</i>	A pointer to a NetCounters object ( <a href="#">Counters</a> <Network, NetCounterData>).

## 6.4.2 Function Documentation

#### 6.4.2.1 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.

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

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

Definition at line 665 of file network.hpp.

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

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

Definition at line 610 of file network.hpp.

#### 6.4.2.4 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.



#### 6.4.2.5 counter\_density()

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

Definition at line 729 of file network.hpp.

#### 6.4.2.6 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.

#### 6.4.2.7 counter\_edges()

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

Number of edges.

Definition at line 152 of file network.hpp.

#### 6.4.2.8 counter\_fixed\_effect()

```
void counter_fixed_effect (
    DEFMArrays * 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 of a vector of counters
<i>covar_index</i>	If >= than 0, then the interaction

Definition at line 785 of file defm.hpp.

#### 6.4.2.9 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.

#### 6.4.2.10 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.

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

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

Definition at line 785 of file network.hpp.

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

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

Definition at line 757 of file network.hpp.

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

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

Definition at line 215 of file network.hpp.

**6.4.2.14 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.

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

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

Definition at line 338 of file network.hpp.

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

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

Definition at line 312 of file network.hpp.

**6.4.2.17 counter\_logit\_intercept()**

```
void counter_logit_intercept (
    DEFMArrays * 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 326 of file defm.hpp.

#### 6.4.2.18 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.

#### 6.4.2.19 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.

#### 6.4.2.20 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.

#### 6.4.2.21 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.

#### 6.4.2.22 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.

**6.4.2.23 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.

**6.4.2.24 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.

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

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

Definition at line 862 of file network.hpp.

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

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

Definition at line 834 of file network.hpp.

**6.4.2.27 counter\_ones()**

```
void counter_ones (
    DEFMArrays * 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 256 of file defm.hpp.

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

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

Definition at line 404 of file network.hpp.

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

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

Definition at line 376 of file network.hpp.

**6.4.2.30 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 445 of file defm.hpp.

**6.4.2.31 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 754 of file defm.hpp.

**6.4.2.32 counter\_ttriads() [1/2]**

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

Definition at line 531 of file network.hpp.

**6.4.2.33 counter\_ttriads() [2/2]**

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

Definition at line 441 of file network.hpp.

**6.4.2.34 NETWORK\_COUNTER()**

```
NETWORK_COUNTER (
    init_single_attr )
```

Definition at line 997 of file network.hpp.

#### 6.4.2.35 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 853 of file defm.hpp.

#### 6.4.2.36 rules\_markov\_fixed()

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

Number of edges.

Definition at line 828 of file defm.hpp.

## 6.5 Phylo counters

[Counters](#) for phylogenetic modeling.

### Functions

- void [counter\\_overall\\_gains](#) ([PhyloCounters](#) \*counters, size\_t duplication=[DEFAULT\\_DUPLICATION](#))  
*Overall functional gains.*
- void [counter\\_gains](#) ([PhyloCounters](#) \*counters, std::vector< size\_t > nfun, size\_t duplication=[DEFAULT\\_DUPLICATION](#))  
*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=[DEFAULT\\_DUPLICATION](#))  
*k genes gain function nfun*
- void [counter\\_genes\\_changing](#) ([PhyloCounters](#) \*counters, size\_t duplication=[DEFAULT\\_DUPLICATION](#))  
*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=[DEFAULT\\_DUPLICATION](#))  
*Keeps track of how many pairs of genes preserve pseudostate.*
- void [counter\\_prop\\_genes\\_changing](#) ([PhyloCounters](#) \*counters, size\_t duplication=[DEFAULT\\_DUPLICATION](#))  
*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=[DEFAULT\\_DUPLICATION](#))  
*Overall functional loss.*



- void `counter_maxfun` (`PhyloCounters *counters`, `size_t lb`, `size_t ub`, `size_t duplication=DEFAULT_DUPLICATION`)  
*Cap the number of functions per gene.*
- void `counter_loss` (`PhyloCounters *counters`, `std::vector< size_t > nfun`, `size_t duplication=DEFAULT_DUPLICATION`)  
*Total count of losses for an specific function.*
- void `counter_overall_changes` (`PhyloCounters *counters`, `size_t duplication=DEFAULT_DUPLICATION`)  
*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=DEFAULT_DUPLICATION`)  
*Total count of Sub-functionalization events.*
- void `counter_cogain` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=DEFAULT_DUPLICATION`)  
*Co-evolution (joint gain or loss)*
- void `counter_longest` (`PhyloCounters *counters`, `size_t duplication=DEFAULT_DUPLICATION`)  
*Longest branch mutates (either by gain or by loss)*
- void `counter_neofun` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=DEFAULT_DUPLICATION`)  
*Total number of neofunctionalization events.*
- void `counter_pairwise_neofun_singlefun` (`PhyloCounters *counters`, `size_t nfunA`, `size_t duplication=DEFAULT_DUPLICATION`)  
*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=DEFAULT_DUPLICATION`)  
*Total number of neofunctionalization events.*
- void `counter_co_opt` (`PhyloCounters *counters`, `size_t nfunA`, `size_t nfunB`, `size_t duplication=DEFAULT_DUPLICATION`)  
*Function co-opting.*
- void `counter_k_genes_changing` (`PhyloCounters *counters`, `size_t k`, `size_t duplication=DEFAULT_DUPLICATION`)  
*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=DEFAULT_DUPLICATION`)  
*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=DEFAULT_DUPLICATION`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `counter_overall_gains_from_0` (`PhyloCounters *counters`, `size_t duplication=DEFAULT_DUPLICATION`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `counter_pairwise_overall_change` (`PhyloCounters *counters`, `size_t duplication=DEFAULT_DUPLICATION`)  
*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=DEFAULT_DUPLICATION`)  
*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=DEFAULT_DUPLICATION`)  
*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 = DEFAULT_DUPLICATION ) [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 1393 of file phylo.hpp.

### 6.5.2.2 counter\_cogain()

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

Co-evolution (joint gain or loss)

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

Definition at line 888 of file phylo.hpp.

### 6.5.2.3 counter\_gains()

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

Functional gains for a specific function (nfun).

Definition at line 193 of file phylo.hpp.

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

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

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

Needs to specify function a.

Definition at line 1727 of file phylo.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 = DEFAULT_DUPLICATION ) [inline]
```

k genes gain function nfun

Definition at line 253 of file phylo.hpp.

#### 6.5.2.6 counter\_genes\_changing()

```
void counter_genes_changing (
    PhyloCounters * counters,
    size_t duplication = DEFAULT_DUPLICATION ) [inline]
```

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

Definition at line 325 of file phylo.hpp.

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

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

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

Definition at line 1491 of file phylo.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 = DEFAULT_DUPLICATION ) [inline]
```

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

< How many genes diverge the parent

Definition at line 1611 of file phylo.hpp.

### 6.5.2.9 counter\_longest()

```
void counter_longest (
    PhyloCounters * counters,
    size_t duplication = DEFAULT_DUPLICATION ) [inline]
```

Longest branch mutates (either by gain or by loss)

Definition at line 945 of file phylo.hpp.

### 6.5.2.10 counter\_loss()

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

Total count of losses for an specific function.

Definition at line 688 of file phylo.hpp.

### 6.5.2.11 counter\_maxfuns()

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

Cap the number of functions per gene.

Definition at line 626 of file phylo.hpp.

#### 6.5.2.12 counter\_neofun()

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

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 1115 of file phylo.hpp.

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

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

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 1260 of file phylo.hpp.

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

```
void counter_overall_changes (
    PhyloCounters * counters,
    size_t duplication = DEFAULT_DUPLICATION ) [inline]
```

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

Definition at line 740 of file phylo.hpp.

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

```
void counter_overall_gains (
    PhyloCounters * counters,
    size_t duplication = DEFAULT_DUPLICATION ) [inline]
```

Overall functional gains.

Total number of gains (irrespective of the function).

Definition at line 155 of file phylo.hpp.

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

```
void counter_overall_gains_from_0 (
    PhyloCounters * counters,
    size_t duplication = DEFAULT_DUPLICATION ) [inline]
```

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

Needs to specify function a.

Definition at line 1793 of file phylo.hpp.

#### 6.5.2.17 counter\_overall\_loss()

```
void counter_overall_loss (
    PhyloCounters * counters,
    size_t duplication = DEFAULT_DUPLICATION ) [inline]
```

Overall functional loss.

Definition at line 578 of file phylo.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 = DEFAULT_DUPLICATION ) [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 2045 of file phylo.hpp.

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

```
void counter_pairwise_neofun_singlefun (
    PhyloCounters * counters,
    size_t nfunA,
    size_t duplication = DEFAULT_DUPLICATION ) [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 1196 of file phylo.hpp.

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

```
void counter_pairwise_overall_change (
    PhyloCounters * counters,
    size_t duplication = DEFAULT_DUPLICATION ) [inline]
```

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

Needs to specify function a.

Definition at line 1841 of file phylo.hpp.

**6.5.2.21 counter\_pairwise\_preserving()**

```
void counter_pairwise_preserving (
    PhyloCounters * counters,
    size_t nfunA,
    size_t nfunB,
    size_t duplication = DEFAULT_DUPLICATION ) [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 1906 of file phylo.hpp.

**6.5.2.22 counter\_preserve\_pseudogene()**

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

Keeps track of how many pairs of genes preserve pseudostate.

Definition at line 394 of file phylo.hpp.

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

```
void counter_prop_genes_changing (
    PhyloCounters * counters,
    size_t duplication = DEFAULT_DUPLICATION ) [inline]
```

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

Definition at line 476 of file phylo.hpp.

#### 6.5.2.24 counter\_subfun()

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

Total count of Sub-functionalization events.

It requires to specify data = {funA, funB}

Definition at line 799 of file phylo.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

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

#### 7.2.1 Detailed Description

Tree class and Treeliterator class.

### 7.3 `barry::counters::defm` Namespace Reference

### 7.4 `barry::counters::network` Namespace Reference

### 7.5 `barry::counters::phylo` Namespace Reference

### 7.6 `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

#### 7.6.1 Detailed Description

Integer constants used to specify which cell should be check.

#### 7.6.2 Variable Documentation

##### 7.6.2.1 `BOTH`

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

Definition at line 27 of file typedefs.hpp.

##### 7.6.2.2 `NONE`

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

Definition at line 28 of file typedefs.hpp.

### 7.6.2.3 ONE

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

Definition at line 29 of file typedefs.hpp.

### 7.6.2.4 TWO

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

Definition at line 30 of file typedefs.hpp.

## 7.7 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`

### 7.7.1 Detailed Description

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

### 7.7.2 Variable Documentation

#### 7.7.2.1 AS\_ONE

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

Definition at line 45 of file typedefs.hpp.

#### 7.7.2.2 AS\_ZERO

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

Definition at line 44 of file typedefs.hpp.

#### 7.7.2.3 BOTH

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

Definition at line 38 of file typedefs.hpp.

#### 7.7.2.4 NONE

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

Definition at line 39 of file typedefs.hpp.

#### 7.7.2.5 ONE

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

Definition at line 40 of file typedefs.hpp.

#### 7.7.2.6 TWO

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

Definition at line 41 of file typedefs.hpp.

#### 7.7.2.7 UNKNOWN

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

Definition at line 43 of file typedefs.hpp.

## 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
- 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.

- `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

<code>i,j</code>	<i>Coordinates</i>
<code>check_bounds</code>	<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

<code>i,j</code>	<i>Row,column</i>
<code>check_bounds</code>	<i>When true and out of range, the function throws an error.</i>
<code>check_exists</code>	<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`
- `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

<i>Cell_Type</i>	Type of cell (any type).
<i>Data_Type</i>	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 240 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 reserve()

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

#### 8.1.3.39 resize()

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

#### 8.1.3.40 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 )
```

#### 8.1.3.41 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
```

#### 8.1.3.42 set\_data()

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

Set the data object.



## Parameters

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

**8.1.3.43 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 )
```

**8.1.3.44 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 )
```

**8.1.3.45 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 )
```

**8.1.3.46 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 )
```

#### 8.1.3.47 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 )
```

#### 8.1.3.48 transpose()

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

#### 8.1.3.49 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 )
```

#### 8.1.3.50 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 barraycell-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 barraycell-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 barraycell-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 `barraycell-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 `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`

## 8.4 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 j) 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	Coordinates
check_bounds	If <code>false</code> avoids checking bounds.

- `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

<code>i,j</code>	<i>Row,column</i>
<code>check_bounds</code>	<i>When <code>true</code> and out of range, the function throws an error.</i>
<code>check_exists</code>	<i>Wither check if the cell exists (before trying to delete/add), or, in the case of <code>swap_cells</code>, 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 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

- `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 = bool, typename Data_Type = bool>
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 )
```

Edgelist with data.

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

```
template<typename Cell_Type = bool, typename Data_Type = bool>
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 )
```

Edgelist with no data (simpler)

**8.4.2.5 BArrayDense()** [5/6]

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

Copy constructor.

**8.4.2.6 BArrayDense()** [6/6]

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

Move operator.

### 8.4.2.7 ~BArrayDense()

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

## 8.4.3 Member Function Documentation

### 8.4.3.1 clear()

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

### 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 490 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 476 of file `barraydense-meat.hpp`.

### 8.4.3.4 colsum()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
const Cell_Type BArrayDense< Cell_Type, Data_Type >::colsum (
    size_t i ) const
```

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

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

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

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

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

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

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

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

#### 8.4.3.9 default\_val()

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

#### 8.4.3.10 get\_cell()

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

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

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

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

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

**8.4.3.13 get\_data()**

```
template<typename Cell_Type = bool, typename Data_Type = bool>
const std::vector< Cell_Type >& BArrayDense< Cell_Type, Data_Type >::get_data ( ) const
```

**8.4.3.14 get\_entries()**

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

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

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

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

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

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

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

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

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

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



**8.4.3.21 ncol()**

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

**8.4.3.22 nnozero()**

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

**8.4.3.23 nrow()**

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

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

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

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

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

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

**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 = bool, typename Data_Type = bool>
BArrayDense<Cell_Type,Data_Type>& BArrayDense< Cell_Type, Data_Type >::operator-=( (
    const std::pair< size_t, size_t > & coords )
```

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

Move assignment.

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

```
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 > & Array_ )
```

Assignment constructor.

**8.4.3.36 operator==( )**

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

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

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

**8.4.3.38 print( )**

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

**8.4.3.39 reserve()**

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

**8.4.3.40 resize()**

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

**8.4.3.41 rm\_cell()**

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

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

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

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

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

**8.4.3.44 rowsum()**

```
template<typename Cell_Type = bool, typename Data_Type = bool>
const Cell_Type BArrayDense< Cell_Type, Data_Type >::rowsum (
    size_t i ) const
```

**8.4.3.45 set\_data()**

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

**8.4.3.46 swap\_cells()**

```
template<typename Cell_Type = bool, typename Data_Type = bool>
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 )
```

**8.4.3.47 swap\_cols()**

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

**8.4.3.48 swap\_rows()**

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

#### 8.4.3.49 toggle\_cell()

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

#### 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 = bool, typename Data_Type = bool>
void BArrayDense< Cell_Type, Data_Type >::transpose ( )
```

#### 8.4.3.52 zero\_col()

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

#### 8.4.3.53 zero\_row()

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

### 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 72 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 52 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 34 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 43 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 62 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 24 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 77 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 bararraydenserow-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 bararraydenserow-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 bararraydenserow-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 bararraydenserow-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 bararraydenserow-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.

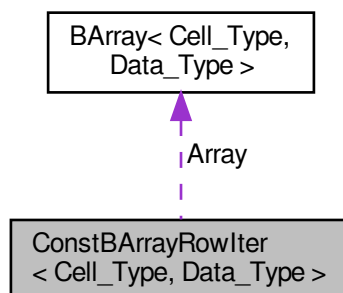
The documentation for this class was generated from the following file:

- [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\_) noexcept  
*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.

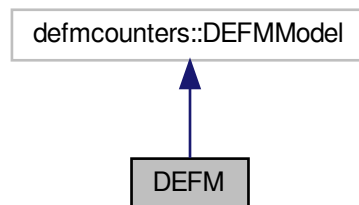
The documentation for this class was generated from the following file:

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

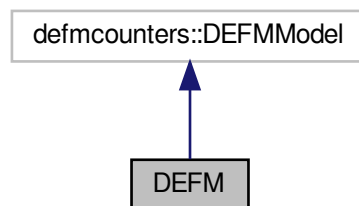
## 8.20 DEFM Class Reference

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

Inheritance diagram for DEFM:



Collaboration diagram for DEFM:



### Public Member Functions

- `DEFM` (`const` int \*id, `const` int \*y, `const` double \*x, size\_t id\_length, size\_t y\_ncol, size\_t x\_ncol, size\_t m\_order)
- `defmcounters::DEFMModel` & `get_model` ()
- void `init` ()
- double `likelihood` (std::vector< double > &par, bool as\_log=`false`)
- 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 ()`

### 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 (
    const int * id,
    const int * y,
    const double * x,
    size_t id_length,
    size_t y_ncol,
    size_t x_ncol,
    size_t m_order ) [inline]
```

Definition at line 104 of file `defm-meat.hpp`.

### 8.20.3 Member Function Documentation

#### 8.20.3.1 get\_ID()

```
const int * DEFM::get_ID ( ) const [inline]
```

Definition at line 259 of file `defm-meat.hpp`.

#### 8.20.3.2 get\_m\_order()

```
size_t DEFM::get_m_order ( ) const [inline]
```

Definition at line 244 of file `defm-meat.hpp`.

### 8.20.3.3 get\_model()

```
defmcounters::DEFMModel& DEFM::get_model ( ) [inline]
```

Definition at line 48 of file defm-bones.hpp.

### 8.20.3.4 get\_n\_covars()

```
size_t DEFM::get_n_covars ( ) const [inline]
```

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

### 8.20.3.5 get\_n\_obs()

```
size_t DEFM::get_n_obs ( ) const [inline]
```

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

### 8.20.3.6 get\_n\_rows()

```
size_t DEFM::get_n_rows ( ) const [inline]
```

Definition at line 249 of file defm-meat.hpp.

### 8.20.3.7 get\_n\_y()

```
size_t DEFM::get_n_y ( ) const [inline]
```

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

### 8.20.3.8 get\_X()

```
const double * DEFM::get_X ( ) const [inline]
```

Definition at line 264 of file defm-meat.hpp.

### 8.20.3.9 get\_X\_names()

```
const std::vector< std::string > & DEFM::get_X_names ( ) const [inline]
```

Definition at line 371 of file defm-meat.hpp.

### 8.20.3.10 get\_Y()

```
const int * DEFM::get_Y ( ) const [inline]
```

Definition at line 254 of file defm-meat.hpp.

### 8.20.3.11 get\_Y\_names()

```
const std::vector< std::string > & DEFM::get_Y_names ( ) const [inline]
```

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

### 8.20.3.12 init()

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

Definition at line 188 of file defm-meat.hpp.

### 8.20.3.13 is\_motif()

```
std::vector< bool > DEFM::is_motif ( ) [inline]
```

Definition at line 388 of file defm-meat.hpp.

### 8.20.3.14 likelihood()

```
double DEFM::likelihood (
    std::vector< double > & par,
    bool as_log = false )
```

### 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 308 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 270 of file defm-meat.hpp.

### 8.20.3.17 print()

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

Definition at line 375 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 350 of file defm-meat.hpp.

### 8.20.3.19 simulate()

```
void DEFM::simulate (
    std::vector< double > par,
    int * y_out ) [inline]
```

Definition at line 38 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.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 75 of file defm.hpp.

#### 8.21.2 Constructor & Destructor Documentation

##### 8.21.2.1 DEFMCOUNTERDATA() [1/2]

```
DEFMCOUNTERDATA::DEFMCOUNTERDATA ( ) [inline]
```

Definition at line 83 of file defm.hpp.



### 8.21.2.2 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 84 of file defm.hpp.

### 8.21.2.3 ~DEFMCOUNTERDATA()

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

Definition at line 96 of file defm.hpp.

## 8.21.3 Member Function Documentation

### 8.21.3.1 idx()

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

Definition at line 92 of file defm.hpp.

### 8.21.3.2 is\_true()

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

Definition at line 94 of file defm.hpp.

### 8.21.3.3 num()

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

Definition at line 93 of file defm.hpp.

## 8.21.4 Member Data Documentation

### 8.21.4.1 indices

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

Definition at line 78 of file defm.hpp.

### 8.21.4.2 is\_motif

```
bool DEFMCOUNTERData::is_motif
```

If false, then is a logit intercept.

Definition at line 81 of file defm.hpp.

### 8.21.4.3 logical

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

Definition at line 80 of file defm.hpp.

### 8.21.4.4 numbers

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

Definition at line 79 of file defm.hpp.

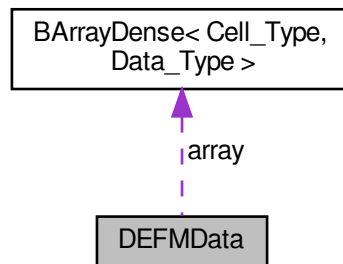
The documentation for this class was generated from the following file:

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

## 8.22 DEFMDData Class Reference

```
#include <defm.hpp>
```

Collaboration diagram for DEFMDData:



### 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

Definition at line 27 of file defm.hpp.

### 8.22.2 Constructor & Destructor Documentation

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

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

Vector indicating which covariates are included in the model.

Definition at line 38 of file defm.hpp.

#### 8.22.2.2 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_start_↔</i>	Location of the current observation in the covariates vector
<i>X_ncol_</i>	Number of columns (covariates.)

Definition at line 47 of file defm.hpp.

#### 8.22.2.3 ~DEFMDData()

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

Definition at line 69 of file defm.hpp.

## 8.22.3 Member Function Documentation

### 8.22.3.1 at()

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

## 8.22.4 Member Data Documentation

### 8.22.4.1 array

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

Definition at line 30 of file defm.hpp.

### 8.22.4.2 covar\_sort

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

Definition at line 35 of file defm.hpp.

### 8.22.4.3 covar\_used

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

Value where the sorting of the covariates is stored.

Definition at line 36 of file defm.hpp.

### 8.22.4.4 covariates

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

Vector of covariates (complete vector)

Definition at line 31 of file defm.hpp.

#### 8.22.4.5 obs\_start

```
size_t DEFMDData::obs_start
```

Index of the observation in the data.

Definition at line 32 of file defm.hpp.

#### 8.22.4.6 X\_ncol

```
size_t DEFMDData::X_ncol
```

Number of columns in the array of covariates.

Definition at line 33 of file defm.hpp.

#### 8.22.4.7 X\_nrow

```
size_t DEFMDData::X_nrow
```

Number of rows in the array of covariates.

Definition at line 34 of file defm.hpp.

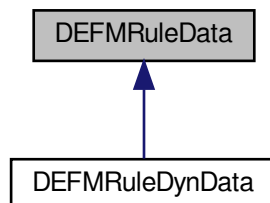
The documentation for this class was generated from the following file:

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

## 8.23 DEFMRuleData Class Reference

```
#include <defm.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 100 of file `defm.hpp`.

### 8.23.2 Constructor & Destructor Documentation

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

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

Definition at line 113 of file `defm.hpp`.

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

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

Definition at line 115 of file `defm.hpp`.

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

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

Definition at line 121 of file defm.hpp.

## 8.23.3 Member Function Documentation

### 8.23.3.1 idx()

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

Definition at line 110 of file defm.hpp.

### 8.23.3.2 is\_true()

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

Definition at line 111 of file defm.hpp.

### 8.23.3.3 num()

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

Definition at line 109 of file defm.hpp.

## 8.23.4 Member Data Documentation

### 8.23.4.1 indices

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

Definition at line 104 of file defm.hpp.



#### 8.23.4.2 init

```
bool DEFMRuleData::init = false
```

Definition at line 107 of file defm.hpp.

#### 8.23.4.3 logical

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

Definition at line 105 of file defm.hpp.

#### 8.23.4.4 numbers

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

Definition at line 103 of file defm.hpp.

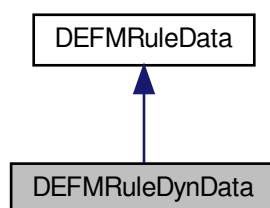
The documentation for this class was generated from the following file:

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

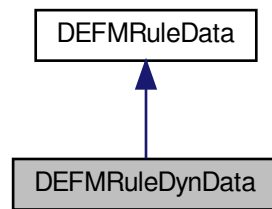
## 8.24 DEFMRuleDynData Class Reference

```
#include <defm.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 135 of file defm.hpp.

### 8.24.2 Constructor & Destructor Documentation

#### 8.24.2.1 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 139 of file defm.hpp.

### 8.24.2.2 ~DEFMRuleDynData()

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

Definition at line 146 of file defm.hpp.

## 8.24.3 Member Data Documentation

### 8.24.3.1 counts

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

Definition at line 137 of file defm.hpp.

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

- include/barry/counters/[defm.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::PhyloCounters](#) \* [get\\_counters](#) ()
- [phylocounters::PhyloSupport](#) \* [get\\_support\\_fun](#) ()
- std::vector< std::vector< double > > \* [get\\_stats\\_support](#) ()
- std::vector< std::vector< double > > \* [get\\_stats\\_target](#) ()
- [phylocounters::PhyloModel](#) \* [get\\_model](#) ()
- double [likelihood\\_joint](#) (const std::vector< double > &par, bool as\_log=false, bool use\_reduced\_sequence=true)  
*Returns the joint likelihood of the model.*
- [Geese](#) \* [operator\(\)](#) (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](#)
- [phylocounters::PhyloModel](#) [model](#) = [phylocounters::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 224 of file flock-meat.hpp.

### 8.26.3.3 get\_counters()

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

Definition at line 100 of file flock-meat.hpp.

### 8.26.3.4 get\_model()

```
phylocounters::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()

```
phylocounters::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 ) [inline]
```

Returns the joint likelihood of the model.

#### Parameters

<i>par</i>	Vector of model parameters.
<i>as_log</i>	When <i>true</i> it will return the value as log.
<i>use_reduced_sequence</i>	When <i>true</i> (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 167 of file flock-meat.hpp.

### 8.26.3.11 nleafs()

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

Definition at line 195 of file flock-meat.hpp.

### 8.26.3.12 nnodes()

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

Definition at line 181 of file flock-meat.hpp.

### 8.26.3.13 nterms()

```
size_t Flock::nterms ( ) const [inline]
```

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

### 8.26.3.14 ntrees()

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

Definition at line 174 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 302 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 231 of file flock-meat.hpp.

#### 8.26.3.17 print()

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

Definition at line 258 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

s	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 217 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

```
phylocounters::PhyloModel Flock::model = phylocounters::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)
- 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 [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.*

### 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 <i>true</i> 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 <i>nleafs</i> times.
only_annotated	When <i>true</i> , it will make the predictions only on the induced sub-tree with annotated leafs.
use_reduced_sequence	Passed to the <i>likelihood</i> method.
preorder	For the tree traversal.

When *res\_prob* is specified, the function will attach the member vector probabilities from the *Nodes* objects. This contains the probability that the *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::PhyloCounters * get_counters()`
- `phylocounters::PhyloModel * get_model()`
- `phylocounters::PhyloSupport * get_support_fun()`

**Public Attributes**

- `size_t nfunctions`
- `std::map< size_t, Node > nodes`
- `barry::MapVec_type< size_t > map_to_nodes`
- `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`

**8.28.1 Detailed Description**

Annotated Phylo [Model](#).

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

Definition at line 82 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 216 of file geese-meat-constructors.hpp.

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

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

Definition at line 295 of file geese-meat-constructors.hpp.

### 8.28.2.5 ~Geese()

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

Definition at line 144 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 383 of file geese-meat.hpp.

### 8.28.3.2 calc\_sequence()

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

Definition at line 339 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 505 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 721 of file geese-meat.hpp.

### 8.28.3.5 get\_counters()

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

Definition at line 704 of file geese-meat.hpp.

### 8.28.3.6 get\_model()

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

Definition at line 709 of file geese-meat.hpp.

### 8.28.3.7 get\_probabilities()

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

Definition at line 431 of file geese-meat.hpp.

### 8.28.3.8 get\_engine()

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

Definition at line 699 of file geese-meat.hpp.

### 8.28.3.9 get\_states()

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

Powerset of a gene's possible states.

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

#### Returns

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

Definition at line 717 of file geese-meat.hpp.

### 8.28.3.10 get\_support\_fun()

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

Definition at line 713 of file geese-meat.hpp.

### 8.28.3.11 inherit\_support()

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

Definition at line 282 of file geese-meat.hpp.

### 8.28.3.12 init()

```
void Geese::init (
    size_t bar_width = BARRY_PROGRESS_BAR_WIDTH ) [inline]
```

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

### 8.28.3.13 init\_node()

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

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

### 8.28.3.14 likelihood()

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

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

### 8.28.3.15 likelihood\_exhaust()

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

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

### 8.28.3.16 nannotations()

```
std::vector< size_t > Geese::nannotations ( ) const [inline], [noexcept]
```

Number of annotations.

Definition at line 496 of file geese-meat.hpp.

### 8.28.3.17 nfuncs()

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

Number of functions analyzed.

Definition at line 452 of file geese-meat.hpp.

### 8.28.3.18 nleafs()

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

Number of leaf.

Definition at line 466 of file geese-meat.hpp.

### 8.28.3.19 nnodes()

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

Number of nodes (interior + leaf)

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

### 8.28.3.20 nterms()

```
size_t Geese::nterms ( ) const [inline]
```

Number of terms included.

Definition at line 478 of file geese-meat.hpp.

### 8.28.3.21 observed\_counts()

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

Definition at line 547 of file geese-meat.hpp.

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

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

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

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



### 8.28.3.24 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 512 of file geese-meat.hpp.

### 8.28.3.25 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 272 of file geese-meat-predict.hpp.

### 8.28.3.26 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.

### 8.28.3.27 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.

### 8.28.3.28 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.

### 8.28.3.29 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.

### 8.28.3.30 print()

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

Prints information about the GEESE.

Definition at line 681 of file geese-meat.hpp.

### 8.28.3.31 print\_observed\_counts()

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

Definition at line 618 of file geese-meat.hpp.

### 8.28.3.32 set\_seed()

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

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

### 8.28.3.33 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.

### 8.28.3.34 support\_size()

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

Number of unique sets of sufficient stats.

Definition at line 486 of file geese-meat.hpp.

### 8.28.3.35 update\_annotations()

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

Definition at line 310 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 120 of file geese-bones.hpp.

### 8.28.4.2 delete\_support

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

Definition at line 121 of file geese-bones.hpp.

#### 8.28.4.3 initialized

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

Definition at line 119 of file geese-bones.hpp.

#### 8.28.4.4 map\_to\_nodes

```
barry::MapVec_type< size_t > Geese::map_to_nodes
```

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

#### 8.28.4.5 nfunctions

```
size_t Geese::nfunctions
```

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

#### 8.28.4.6 nodes

```
std::map< size_t, Node > Geese::nodes
```

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

#### 8.28.4.7 pset\_loc

```
std::vector< std::vector< std::vector< size_t > > > Geese::pset_loc
```

Locations of columns.

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

#### 8.28.4.8 reduced\_sequence

```
std::vector< size_t > Geese::reduced_sequence
```

Definition at line 116 of file geese-bones.hpp.

#### 8.28.4.9 sequence

```
std::vector< size_t > Geese::sequence
```

Definition at line 115 of file geese-bones.hpp.

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

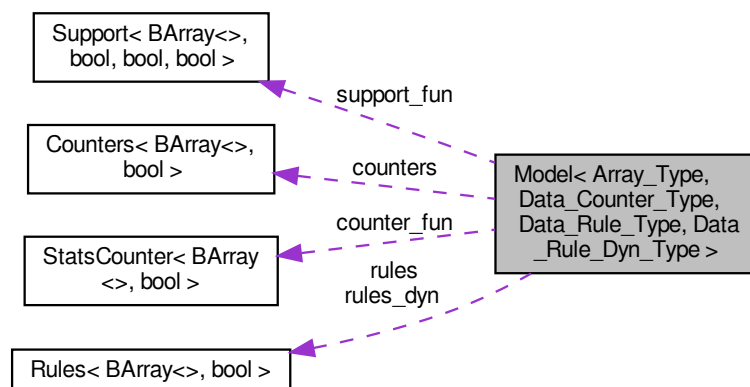
- include/barry/models/geese/geese-bones.hpp
- include/barry/models/geese/geese-meat-constructors.hpp
- include/barry/models/geese/geese-meat-likelihood.hpp
- include/barry/models/geese/geese-meat-likelihood\_exhaust.hpp
- include/barry/models/geese/geese-meat-predict.hpp
- include/barry/models/geese/geese-meat-predict\_exhaust.hpp
- 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 `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*

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

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

**Extract elements by index***Parameters*

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

- double `get_norm_const` (`const` `std::vector< double >` &params, `const` `size_t` &i, `bool` as\_log=`false`)
- `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\_freq 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 <code>BArray</code> 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 132 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 166 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 204 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 159 of file model-bones.hpp.

**8.29.3 Member Function Documentation****8.29.3.1 add\_array()**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↔
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
```

```
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 )
```

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.

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

```
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_counter (
    Counter< Array_Type, Data_Counter_Type > & counter )
```

##### 8.29.3.3 add\_counter() [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_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 )
```

##### 8.29.3.4 add\_hasher()

```
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_hasher (
    Hasher_fun_type< Array_Type, Data_Counter_Type > fun_ )
```

##### 8.29.3.5 add\_rule() [1/2]

```
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< Array_Type, Data_Rule_Type > & rule )
```

**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 = 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_dyn
(
    Rule< Array_Type, Data_Rule_Dyn_Type > & rule )
```

**8.29.3.8 add\_rule\_dyn() [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_dyn
(
    Rule_fun_type< Array_Type, Data_Rule_Dyn_Type > count_fun_,
    Data_Rule_Dyn_Type data_ )
```

**8.29.3.9 colnames()**

```
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 >::colnames ( ) const
```

**8.29.3.10 conditional\_prob()**

```
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 >::conditional_prob (
    const Array_Type & Array_,
    const std::vector< double > & params,
    size_t i,
    size_t j )
```

Conditional probability ("Gibbs sampler")

Computes the conditional probability of observing  $P\{Y(i,j) = Y^C, \theta\}$ , i.e., the probability of observing the entry  $Y(i,j)$  equal to one given the rest of the array.

#### Parameters

<i>Array</i> ↔ —	Array to check
<i>params</i>	Vector of parameters
<i>i</i>	Row entry
<i>j</i>	Column entry

#### Returns

double The conditional probability

##### 8.29.3.11 gen\_key()

```
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 >::gen_key (
    const Array_Type & Array_ )
```

##### 8.29.3.12 get\_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 >::get_arrays2support ( )
```

##### 8.29.3.13 get\_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 >::get_counters ( )
```

##### 8.29.3.14 get\_norm\_const()

```
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 >::get_norm_↔
const (
    const std::vector< double > & params,
    const size_t & i,
    bool as_log = false )
```

**8.29.3.15 get\_pset()**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
const std::vector< Array_Type >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_pset (
    const size_t & i )
```

**8.29.3.16 get\_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 >::get_pset_arrays ( )
```

**8.29.3.17 get\_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 >::get_pset_probs ( )
```

**8.29.3.18 get\_pset\_stats() [1/2]**

```
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 >::get_pset_stats ( )
```

Statistics of the support(s)

**8.29.3.19 get\_pset\_stats() [2/2]**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
const std::vector< double >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_pset_stats (
    const size_t & i )
```

### 8.29.3.20 get\_engine()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
const std::mt19937* Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_engine ( ) const
```

### 8.29.3.21 get\_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 >::get_rules ( )
```

### 8.29.3.22 get\_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 >::get_rules_dyn ( )
```

### 8.29.3.23 get\_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 >::get_stats_support ( )
```

### 8.29.3.24 get\_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 >::get_stats_target ( )
```

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) x 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.

**8.29.3.25 get\_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 >::get_support_fun ( )
```

**8.29.3.26 likelihood() [1/4]**

```
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
(
    const std::vector< double > & params,
    const Array_Type & Array_,
    int i = -1,
    bool as_log = false )
```

**8.29.3.27 likelihood() [2/4]**

```
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
(
    const std::vector< double > & params,
    const double * target_,
    const size_t & i,
    bool as_log = false )
```

**8.29.3.28 likelihood() [3/4]**

```
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
(
    const std::vector< double > & params,
    const size_t & i,
    bool as_log = false )
```



**8.29.3.29 likelihood()** [4/4]

```
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
(
    const std::vector< double > & params,
    const std::vector< double > & target_,
    const size_t & i,
    bool as_log = false )
```

**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 )
```

**8.29.3.31 nrules()**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::nrules ( )
const [noexcept]
```

**8.29.3.32 nrules\_dyn()**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::nrules_dyn
( ) const [noexcept]
```

**8.29.3.33 nterms()**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::nterms ( )
const [noexcept]
```

**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 248 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 971 of file model-meat.hpp.

**8.29.3.36 print\_stats()**

```
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 >::print_stats (
    size_t i ) const
```

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

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

### 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 1098 of file model-meat.hpp.

### 8.29.3.39 set\_counters()

```
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_counters
(
    Counters< Array_Type, Data_Counter_Type > * counters_ )
```

### 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 * rengine_,
    bool delete_ = false ) [inline]
```

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

### 8.29.3.41 set\_rules()

```
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_rules (
    Rules< Array_Type, Data_Rule_Type > * rules_ )
```

### 8.29.3.42 set\_rules\_dyn()

```
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_rules_dyn
(
    Rules< Array_Type, Data_Rule_Dyn_Type > * rules_ )
```

**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 139 of file model-bones.hpp.

**8.29.3.44 set\_transform\_model()**

```
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_transform_model (
    std::function< std::vector< double >(double *, size_t)> fun,
    std::vector< std::string > names )
```

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>	

**8.29.3.45 size()**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::size ( )
const [noexcept]
```

**8.29.3.46 size\_unique()**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::size_unique
( ) const [noexcept]
```

### 8.29.3.47 store\_psets()

```
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 >::store_psets (
) [noexcept]
```

### 8.29.3.48 support\_size()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::support_size ( ) const [noexcept]
```

### 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 )
```

## 8.29.4 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 [DEFMArray 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.*
- `phylocounters::PhyloArray array`  
*Array of the node.*
- `std::vector< size_t > annotations`  
*Observed annotations (only defined for *Geese*)*
- `bool duplication`
- `std::vector< phylocounters::PhyloArray > arrays = {}`  
*Arrays given all possible states.*
- `std::vector< bool > arrays_valid = {}`  
*Whether the arrays are valid according to the rules of the model.*
- `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 39 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 59 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 65 of file geese-node-bones.hpp.

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

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

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

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

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

Definition at line 88 of file geese-node-bones.hpp.

### 8.32.2.6 ~Node()

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

Definition at line 50 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 104 of file geese-node-bones.hpp.

#### 8.32.3.2 `is_leaf()`

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

Definition at line 116 of file geese-node-bones.hpp.

#### 8.32.3.3 `noffspring()`

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

Definition at line 110 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`

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

Array of the node.

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

### 8.32.4.3 arrays

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

Arrays given all possible states.

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

### 8.32.4.4 arrays\_valid

```
std::vector< bool > Node::arrays_valid = {}
```

Whether the arrays are valid according to the rules of the model.

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

### 8.32.4.5 duplication

```
bool Node::duplication
```

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

### 8.32.4.6 id

```
size_t Node::id
```

Id of the node (as specified in the input)

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

### 8.32.4.7 narray

```
std::vector< size_t > Node::narray = {}
```

ID of the array in the model.

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

#### 8.32.4.8 offspring

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

Offspring nodes.

Definition at line 26 of file geese-node-bones.hpp.

#### 8.32.4.9 ord

```
size_t Node::ord
```

Order in which the node was created.

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

#### 8.32.4.10 parent

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

Parent node.

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

#### 8.32.4.11 probability

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

The probability of observing each state.

Definition at line 31 of file geese-node-bones.hpp.

#### 8.32.4.12 subtree\_prob

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

Induced subtree probabilities.

Definition at line 30 of file geese-node-bones.hpp.

### 8.32.4.13 visited

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

Definition at line 28 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 <phylo.hpp>
```

### Public Member Functions

- `NodeData` (`const` `std::vector< double >` &blengths\_, `const` `std::vector< bool >` &states\_, `bool` duplication\_  
\_ = true)

### Public Attributes

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

### 8.33.1 Detailed Description

Data definition for the `PhyloArray` class.

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

This holds basic information about a given node.

Definition at line 38 of file phylo.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 58 of file phylo.hpp.

### 8.33.3 Member Data Documentation

#### 8.33.3.1 blengths

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

Branch length.

Definition at line 44 of file phylo.hpp.

#### 8.33.3.2 duplication

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

Definition at line 54 of file phylo.hpp.

#### 8.33.3.3 states

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

State of the parent node.

Definition at line 49 of file phylo.hpp.

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

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

## 8.34 PhyloCounterData Class Reference

```
#include <phylo.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 69 of file phylo.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 75 of file phylo.hpp.

#### 8.34.2.2 PhyloCounterData() [2/2]

```
PhyloCounterData::PhyloCounterData ( ) [inline]
```

Definition at line 80 of file phylo.hpp.

### 8.34.3 Member Function Documentation

#### 8.34.3.1 at()

```
size_t PhyloCounterData::at (
    size_t d ) [inline]
```

Definition at line 82 of file phylo.hpp.

#### 8.34.3.2 begin()

```
std::vector< size_t >::iterator PhyloCounterData::begin ( ) [inline]
```

Definition at line 90 of file phylo.hpp.

#### 8.34.3.3 empty()

```
bool PhyloCounterData::empty ( ) [inline]
```

Definition at line 93 of file phylo.hpp.

#### 8.34.3.4 end()

```
std::vector< size_t >::iterator PhyloCounterData::end ( ) [inline]
```

Definition at line 91 of file phylo.hpp.

#### 8.34.3.5 get\_counters()

```
std::vector< double >* PhyloCounterData::get_counters ( ) [inline]
```

Definition at line 94 of file phylo.hpp.

#### 8.34.3.6 operator()()

```
size_t PhyloCounterData::operator() (
    size_t d ) [inline]
```

Definition at line 83 of file phylo.hpp.

#### 8.34.3.7 operator[]()

```
size_t PhyloCounterData::operator[] (
    size_t d ) [inline]
```

Definition at line 84 of file phylo.hpp.

#### 8.34.3.8 push\_back()

```
void PhyloCounterData::push_back (
    size_t x ) [inline]
```

Definition at line 86 of file phylo.hpp.



**8.34.3.9 reserve()**

```
void PhyloCounterData::reserve (
    size_t x ) [inline]
```

Definition at line 85 of file phylo.hpp.

**8.34.3.10 shrink\_to\_fit()**

```
void PhyloCounterData::shrink_to_fit ( ) [inline]
```

Definition at line 87 of file phylo.hpp.

**8.34.3.11 size()**

```
size_t PhyloCounterData::size ( ) [inline]
```

Definition at line 88 of file phylo.hpp.

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

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

**8.35 PhyloRuleDynData Class Reference**

```
#include <phylo.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 2147 of file phylo.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 2155 of file phylo.hpp.

#### 8.35.2.2 ~PhyloRuleDynData()

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

Definition at line 2169 of file phylo.hpp.

### 8.35.3 Member Function Documentation

#### 8.35.3.1 operator()()

```
const double PhyloRuleDynData::operator() ( ) const [inline]
```

Definition at line 2164 of file phylo.hpp.

### 8.35.4 Member Data Documentation

#### 8.35.4.1 counts

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

Definition at line 2149 of file phylo.hpp.

#### 8.35.4.2 duplication

```
size_t PhyloRuleDynData::duplication
```

Definition at line 2153 of file phylo.hpp.

#### 8.35.4.3 lb

```
size_t PhyloRuleDynData::lb
```

Definition at line 2151 of file phylo.hpp.

#### 8.35.4.4 pos

```
size_t PhyloRuleDynData::pos
```

Definition at line 2150 of file phylo.hpp.

#### 8.35.4.5 ub

```
size_t PhyloRuleDynData::ub
```

Definition at line 2152 of file phylo.hpp.

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

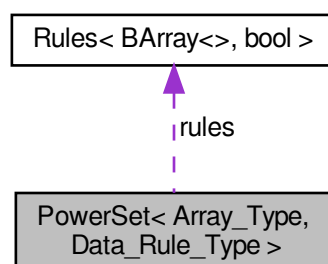
- include/barry/counters/[phylo.hpp](#)

## 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 173 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 182 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 144 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 160 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**

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

- [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 <a href="#">BArray</a> .
<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 179 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> .
<code>_</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

<code>Array↔</code>	<i>New array over which the support will be computed.</i>
<code>—</code>	

- `void reset\_array ()`
- `void reset\_array (const Array_Type &Array\_)`

### Manage counters

#### Parameters

<code>f_</code>	<i>A counter to be added.</i>
<code>counters↔</code>	<i>A vector of counters to be added.</i>
<code>—</code>	

- `void add\_counter (Counter< Array_Type, Data_Counter_Type > f\_)`
- `void set\_counters (Counters< Array_Type, Data_Counter_Type > *counters\_)`

### Manage rules

#### Parameters

<code>f_</code>	<i>A rule to be added.</i>
<code>counters↔</code>	<i>A vector of rules to be added.</i>
<code>—</code>	

- `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 statistics.

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

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

Definition at line 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 87 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 96 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 103 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 110 of file support-bones.hpp.

## 8.41.3 Member Function Documentation

### 8.41.3.1 add\_counter()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_counter
(
    Counter< Array_Type, Data_Counter_Type > f_ )
```

**8.41.3.2 add\_rule() [1/2]**

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
    Rule< Array_Type, Data_Rule_Type > * f_ )
```

**8.41.3.3 add\_rule() [2/2]**

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
    Rule< Array_Type, Data_Rule_Type > f_ )
```

**8.41.3.4 add\_rule\_dyn() [1/2]**

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
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_ )
```

**8.41.3.5 add\_rule\_dyn() [2/2]**

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
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_ )
```

**8.41.3.6 calc()**

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

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.

#### 8.41.3.7 eval\_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 >::eval_↵
rules_dyn (
    const std::vector< double > & counts,
    const size_t & i,
    const size_t & j )
```

#### 8.41.3.8 get\_counters()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Counters<Array_Type, Data_Counter_Type>* Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_counters ( )
```

Vector of counter functions.

#### 8.41.3.9 get\_counts()

```
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 >::get_counts ( ) const
```

#### 8.41.3.10 get\_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 >::get_current_stats ( )
```

List current statistics.

**8.41.3.11 get\_data()**

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
const FreqTable< double >& Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_↵
Rule_Dyn_Type >::get_data ( ) const
```

**8.41.3.12 get\_rules()**

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Rules<Array_Type,Data_Rule_Type>* Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_rules ( )
```

Vector of static rules (cells to iterate).

**8.41.3.13 get\_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>
Rules<Array_Type,Data_Rule_Dyn_Type>* Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_rules_dyn ( )
```

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

**8.41.3.14 init\_support()**

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

**8.41.3.15 print()**

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::print ( )
const
```

#### 8.41.3.16 reset\_array() [1/2]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::reset_array
( )
```

#### 8.41.3.17 reset\_array() [2/2]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::reset_array
(
    const Array_Type & Array_ )
```

#### 8.41.3.18 set\_counters()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_↵
counters (
    Counters< Array_Type, Data_Counter_Type > * counters_ )
```

#### 8.41.3.19 set\_rules()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_rules (
    Rules< Array_Type, Data_Rule_Type > * rules_ )
```

#### 8.41.3.20 set\_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>
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_ )
```

### 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 80 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 78 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 79 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 76 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 77 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 75 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 69 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 70 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 71 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 81 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 82 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 68 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 72 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 68 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 83 of file support-bones.hpp.

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

- [include/barry/support-bones.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 = el[POS(i, j)]`
- `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

```
template 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

```
template Data_Type 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 ( ) = el[POS(i, j)] [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`.

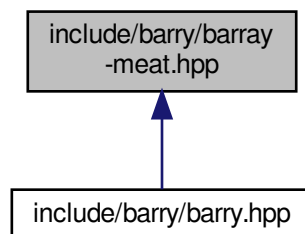
### 9.3.3.2 this

```
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 `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

- [BARRAY\\_TEMPLATE](#) (, [BArray](#))(size\_t N\_
- [el\\_ij](#) [resize](#) (N)
- [el\\_ji](#) [resize](#) (M)
- [for](#) (size\_t i=0u;i< [source.size](#)();++i)
- Data\_Type bool [M](#) ([Array\\_.M](#))
- [BARRAY\\_TEMPLATE](#) ([BARRAY\\_TYPE](#)() &, operator=)(const [BArray](#)< Cell\_Type
- [BARRAY\\_TEMPLATE](#) (, [BArray](#))([BARRAY\\_TYPE](#)() &&x) [noexcept](#)
- [BARRAY\\_TEMPLATE](#) ([BARRAY\\_TYPE](#)() &, operator=)([BARRAY\\_TYPE](#)() &&x) [noexcept](#)
- [BARRAY\\_TEMPLATE](#) (bool, operator=)(const [BARRAY\\_TYPE](#)() &[Array\\_](#))
- [BARRAY\\_TEMPLATE](#) (,~[BArray](#)())()
- [BARRAY\\_TEMPLATE](#) (void, set\_data)(Data\_Type \*[data\\_](#)
- [BARRAY\\_TEMPLATE](#) (Data\_Type \*, D\_ptr)()
- [BARRAY\\_TEMPLATE](#) (Data\_Type &, D)()
- [BARRAY\\_TEMPLATE](#) (void, out\_of\_range)(size\_t i
- [BARRAY\\_TEMPLATE](#) (Cell\_Type, get\_cell)(size\_t i
- [if](#) ([ROW](#)(i).size()==0u) [return](#)(Cell\_Type) 0.0
- [if](#) ([search](#) !=[ROW](#)(i).end()) [return](#) [search](#) -> [second.value](#)
- [return](#) (Cell\_Type) 0.0
- [BARRAY\\_TEMPLATE](#) (std::vector< Cell\_Type >, get\_row\_vec)(size\_t i
- std::vector< Cell\_Type > [ans](#) ([ncol](#)(),(Cell\_Type) [false](#))
- [for](#) (const auto &iter :[row](#)(i, [false](#))) [ans](#)[iter.first]
- [BARRAY\\_TEMPLATE](#) (void, get\_row\_vec)(std
- [BARRAY\\_TEMPLATE](#) ([BARRAY\\_TYPE](#)() &, operator=)(const std
- [BARRAY\\_TEMPLATE](#) (void, [insert\\_cell](#))(size\_t i
- [if](#) ([check\\_exists](#))
- [COL](#) (j).emplace(i
- & [ROW](#) (i)[j])
- [BARRAY\\_TEMPLATE](#) (void, swap\_cells)(size\_t i0
- [if](#) ([report](#) !=nullptr)(\*[report](#))
- [if](#) ([check0](#) &[check1](#))
- [else if](#) (![check0](#) &[check1](#))
- [else if](#) ([check0](#) &![check1](#))
- [BARRAY\\_TEMPLATE](#) (void, toggle\_cell)(size\_t i
- [BARRAY\\_TEMPLATE](#) (void, swap\_rows)(size\_t i0
- [if](#) ([ROW](#)(i0).size()==0u) [move0](#)
- [if](#) ([ROW](#)(i1).size()==0u) [move1](#)
- [if](#) (![move0](#) &&![move1](#)) [return](#)
- [ROW](#) (i0).swap([ROW](#)(i1))
- [BARRAY\\_TEMPLATE](#) (void, swap\_cols)(size\_t j0
- [if](#) ([COL](#)(j0).size()==0u) [check0](#)
- [if](#) ([COL](#)(j1).size()==0u) [check1](#)
- [if](#) ([check0](#) &&[check1](#))
- [else if](#) ([check0](#) &&![check1](#))
- [else if](#) (![check0](#) &&[check1](#))
- [BARRAY\\_TEMPLATE](#) (void, zero\_row)(size\_t i
- [for](#) (auto row=[row0.begin](#)();row !=[row0.end](#)();++row) [rm\\_cell](#)(i
- [BARRAY\\_TEMPLATE](#) (void, zero\_col)(size\_t j
- [if](#) ([COL](#)(j).size()==0u) [return](#)
- [BARRAY\\_TEMPLATE](#) (void, transpose)()
- [BARRAY\\_TEMPLATE](#) (void, [clear](#))(bool hard)
- [BARRAY\\_TEMPLATE](#) (void, [resize](#))(size\_t N\_
- [if](#) ([M](#)< [M](#)) [for](#)(size\_t j = N\_

## Variables

- size\_t `M_`
- size\_t `const std::vector< size_t > & source`
- size\_t `const std::vector< size_t > const std::vector< size_t > & target`
- size\_t `const std::vector< size_t > const std::vector< size_t > const std::vector< Cell_Type > & value`
- size\_t `const std::vector< size_t > const std::vector< size_t > const std::vector< Cell_Type > bool add`
- `if(source.size() !=value.size()) throw std N = N_`
- `M = M_`
- `return`
- `Data_Type & Array_`
- `Data_Type bool copy_data`
- `bool delete_data_`
- `data = data_`
- `delete_data = delete_data_`
- size\_t `j const`
- size\_t `j`
- `auto search = ROW(i).find(j)`
- `return ans`
- size\_t `const Cell< Cell_Type > & v`
- size\_t `const Cell< Cell_Type > bool check_bounds`
- size\_t `const Cell< Cell_Type > bool bool check_exists`
- `else`
- `NCells`
- size\_t `j0`
- size\_t `size_t i1`
- size\_t `size_t size_t j1`
- size\_t `size_t size_t bool int int * report`
- `auto row0 = ROW(i)`
- `row first`
- `row false`
- `auto col0 = COL(j)`

## 9.4.1 Macro Definition Documentation

### 9.4.1.1 BARRAY\_TEMPLATE

```
#define BARRAY_TEMPLATE(  
    a,  
    b )  template BARRAY_TEMPLATE_ARGS() inline a BARRAY_TYPE()::b
```

Definition at line 17 of file `barray-meat.hpp`.

### 9.4.1.2 BARRAY\_TEMPLATE\_ARGS

```
#define BARRAY_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

Definition at line 15 of file `barray-meat.hpp`.

### 9.4.1.3 BARRAY\_TYPE

```
#define BARRAY_TYPE( ) BArray<Cell_Type, Data_Type>
```

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

### 9.4.1.4 COL

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

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

### 9.4.1.5 ROW

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

Definition at line 20 of file barray-meat.hpp.

## 9.4.2 Function Documentation

### 9.4.2.1 ans()

```
std::vector< Cell_Type > ans (  
    ncol() ,  
    (Cell_Type) false )
```

### 9.4.2.2 BARRAY\_TEMPLATE() [1/24]

```
BARRAY_TEMPLATE (  
    BArray ) && [noexcept]
```

Definition at line 230 of file barray-meat.hpp.

#### 9.4.2.3 BARRAY\_TEMPLATE() [2/24]

```
BARRAY_TEMPLATE (
    BArray )
```

#### 9.4.2.4 BARRAY\_TEMPLATE() [3/24]

```
BARRAY_TEMPLATE (
    ~ BArray )
```

Definition at line 339 of file barray-meat.hpp.

#### 9.4.2.5 BARRAY\_TEMPLATE() [4/24]

```
BARRAY_TEMPLATE (
    BARRAY_TYPE() & ,
    operator- ) const
```

Definition at line 597 of file barray-meat.hpp.

#### 9.4.2.6 BARRAY\_TEMPLATE() [5/24]

```
BARRAY_TEMPLATE (
    BARRAY_TYPE() & ,
    operator ) && [noexcept]
```

Definition at line 272 of file barray-meat.hpp.

#### 9.4.2.7 BARRAY\_TEMPLATE() [6/24]

```
BARRAY_TEMPLATE (
    BARRAY_TYPE() & ,
    operator ) const
```

#### 9.4.2.8 BARRAY\_TEMPLATE() [7/24]

```
BARRAY_TEMPLATE (
    bool ,
    operator == ) const &
```

Definition at line 321 of file barray-meat.hpp.

#### 9.4.2.9 BARRAY\_TEMPLATE() [8/24]

```
BARRAY_TEMPLATE (
    Cell_Type ,
    get_cell )
```

#### 9.4.2.10 BARRAY\_TEMPLATE() [9/24]

```
BARRAY_TEMPLATE (
    Data_Type & ,
    D )
```

Definition at line 372 of file barray-meat.hpp.

#### 9.4.2.11 BARRAY\_TEMPLATE() [10/24]

```
BARRAY_TEMPLATE (
    Data_Type * ,
    D_ptr )
```

Definition at line 361 of file barray-meat.hpp.

#### 9.4.2.12 BARRAY\_TEMPLATE() [11/24]

```
BARRAY_TEMPLATE (
    std::vector< Cell_Type > ,
    get_row_vec )
```

#### 9.4.2.13 BARRAY\_TEMPLATE() [12/24]

```
BARRAY_TEMPLATE (
    void ,
    clear )
```

Definition at line 1130 of file barray-meat.hpp.

**9.4.2.14 BARRAY\_TEMPLATE()** [13/24]

```
BARRAY_TEMPLATE (
    void ,
    get_row_vec )
```

Definition at line 452 of file barray-meat.hpp.

**9.4.2.15 BARRAY\_TEMPLATE()** [14/24]

```
BARRAY_TEMPLATE (
    void ,
    insert_cell )
```

**9.4.2.16 BARRAY\_TEMPLATE()** [15/24]

```
BARRAY_TEMPLATE (
    void ,
    out_of_range )
```

**9.4.2.17 BARRAY\_TEMPLATE()** [16/24]

```
BARRAY_TEMPLATE (
    void ,
    resize )
```

**9.4.2.18 BARRAY\_TEMPLATE()** [17/24]

```
BARRAY_TEMPLATE (
    void ,
    set_data )
```

**9.4.2.19 BARRAY\_TEMPLATE()** [18/24]

```
BARRAY_TEMPLATE (
    void ,
    swap_cells )
```

**9.4.2.20 BARRAY\_TEMPLATE()** [19/24]

```
BARRAY_TEMPLATE (
    void ,
    swap_cols )
```

**9.4.2.21 BARRAY\_TEMPLATE()** [20/24]

```
BARRAY_TEMPLATE (
    void ,
    swap_rows )
```

**9.4.2.22 BARRAY\_TEMPLATE()** [21/24]

```
BARRAY_TEMPLATE (
    void ,
    toggle_cell )
```

**9.4.2.23 BARRAY\_TEMPLATE()** [22/24]

```
BARRAY_TEMPLATE (
    void ,
    transpose )
```

Definition at line 1069 of file barray-meat.hpp.

**9.4.2.24 BARRAY\_TEMPLATE()** [23/24]

```
BARRAY_TEMPLATE (
    void ,
    zero_col )
```

**9.4.2.25 BARRAY\_TEMPLATE()** [24/24]

```
BARRAY_TEMPLATE (
    void ,
    zero_row )
```

#### 9.4.2.26 COL()

```
COL (
    j )
```

#### 9.4.2.27 for() [1/3]

```
for (
    auto row = row0.begin(); row != row0.end(); ++row )
```

#### 9.4.2.28 for() [2/3]

```
for (
    const auto &iter : rowi, false )
```

#### 9.4.2.29 for() [3/3]

```
for ( )
```

Definition at line 51 of file barray-meat.hpp.

#### 9.4.2.30 if() [1/17]

```
else if (
    !check0 && check1 )
```

Definition at line 1008 of file barray-meat.hpp.

#### 9.4.2.31 if() [2/17]

```
else if (
    !check0 & check1 )
```

Definition at line 856 of file barray-meat.hpp.



**9.4.2.32 if() [3/17]**

```
if (
    !move0 &&! move1 )
```

**9.4.2.33 if() [4/17]**

```
else if (
    check0 &! check1 )
```

Definition at line 864 of file barray-meat.hpp.

**9.4.2.34 if() [5/17]**

```
else if (
    check0 &&! check1 )
```

Definition at line 999 of file barray-meat.hpp.

**9.4.2.35 if() [6/17]**

```
if (
    check0 && check1 )
```

Definition at line 972 of file barray-meat.hpp.

**9.4.2.36 if() [7/17]**

```
if (
    check0 & check1 )
```

Definition at line 838 of file barray-meat.hpp.

**9.4.2.37 if() [8/17]**

```
else if (
    check_exists == CHECK::BOTH )
```

Definition at line 679 of file barray-meat.hpp.

**9.4.2.38 if()** [9/17]

```
if (
    COL(j).size() == 0u )
```

**9.4.2.39 if()** [10/17]

```
if (
    COL(j0).size() == 0u )
```

**9.4.2.40 if()** [11/17]

```
if (
    COL(j1).size() == 0u )
```

**9.4.2.41 if()** [12/17]

```
else if ( ) = N_
```

Definition at line 86 of file barray-meat.hpp.

**9.4.2.42 if()** [13/17]

```
if (
    report ! == nullptr )
```

**9.4.2.43 if()** [14/17]

```
if (
    ROW(i).size() == 0u )
```

**9.4.2.44 if()** [15/17]

```
if (
    ROW(i0).size() == 0u )
```

**9.4.2.45 if() [16/17]**

```
if (
    ROW(i1).size() == 0u )
```

**9.4.2.46 if() [17/17]**

```
if (
    search ! = ROW(i).end() ) -> second.value
```

**9.4.2.47 M()**

```
Data_Type bool M (
    Array_ M )
```

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

**9.4.2.48 resize() [1/2]**

```
el_ji resize (
    M )
```

**9.4.2.49 resize() [2/2]**

```
el_ij resize (
    N )
```

**9.4.2.50 return()**

```
return (
    Cell_Type )
```

**9.4.2.51 ROW() [1/2]**

```
& ROW (
    i )
```

**9.4.2.52 ROW() [2/2]**

```
ROW (
    i0 )
```

**9.4.3 Variable Documentation****9.4.3.1 add**

```
size_t const std::vector< size_t > const std::vector< size_t > bool add
```

**Initial value:**

```
{
    if (source.size() != target.size())
        throw std::length_error("-source- and -target- don't match on length.")
```

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

**9.4.3.2 ans**

```
return ans
```

Definition at line 449 of file barray-meat.hpp.

**9.4.3.3 Array\_**

```
Data_Type & Array_
```

Definition at line 134 of file barray-meat.hpp.

#### 9.4.3.4 check\_bounds

bool check\_bounds

##### Initial value:

```
{  
    if (check_bounds) {  
        out_of_range(i0,0u);  
        out_of_range(i1,0u);  
    }  
  
    bool move0=true, move1=true
```

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

#### 9.4.3.5 check\_exists

size\_t bool int check\_exists

##### Initial value:

```
{  
    if (check_bounds)  
        out_of_range(i,j)
```

Definition at line 673 of file barray-meat.hpp.

#### 9.4.3.6 col0

auto col0 = COL(j)

Definition at line 1061 of file barray-meat.hpp.

#### 9.4.3.7 const

size\_t bool check\_bounds const

##### Initial value:

```
{  
    if (i >= N)  
        throw std::range_error("The row is out of range.")
```

Definition at line 402 of file barray-meat.hpp.

#### 9.4.3.8 copy\_data

```
Data_Type bool copy_data
```

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

#### 9.4.3.9 data

```
data = data_
```

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

#### 9.4.3.10 delete\_data

```
delete_data = delete_data_
```

Definition at line 355 of file barray-meat.hpp.

#### 9.4.3.11 delete\_data\_

```
bool delete_data_
```

##### Initial value:

```
{  
    if ((data != nullptr) && delete_data)  
        delete data
```

Definition at line 348 of file barray-meat.hpp.

#### 9.4.3.12 else

```
else (  
    void )
```

##### Initial value:

```
{  
    ROW(i).insert(std::pair< size_t, Cell<Cell_Type>>(j, v))
```

Definition at line 703 of file barray-meat.hpp.

#### 9.4.3.13 false

`row false`

Definition at line 1042 of file barray-meat.hpp.

#### 9.4.3.14 first

`row first`

Definition at line 1042 of file barray-meat.hpp.

#### 9.4.3.15 i1

`size_t i1`

Definition at line 776 of file barray-meat.hpp.

#### 9.4.3.16 j

`size_t j`

##### Initial value:

```
{  
    if (init_fun == nullptr)  
        return 0.0
```

Definition at line 414 of file barray-meat.hpp.

#### 9.4.3.17 j0

`size_t j0`

Definition at line 775 of file barray-meat.hpp.

#### 9.4.3.18 j1

`size_t j1`

Definition at line 776 of file barray-meat.hpp.

#### 9.4.3.19 M

M = M\_

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

#### 9.4.3.20 M\_

size\_t M\_

##### Initial value:

```
{  
  
    if (N_ < N)  
        for (size_t i = N_; i < N; ++i)  
            zero_row(i, false)
```

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

#### 9.4.3.21 N

```
if (source.size() != target.size()) throw std::if (source.size() != value.size()) throw std::N =  
N_
```

Definition at line 43 of file barray-meat.hpp.

#### 9.4.3.22 NCells

NCells

Definition at line 707 of file barray-meat.hpp.

#### 9.4.3.23 report

size\_t size\_t size\_t bool int int\* report

Definition at line 779 of file barray-meat.hpp.



#### 9.4.3.24 return

```
return
```

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

#### 9.4.3.25 row0

```
auto row0 = ROW(i)
```

Definition at line 1040 of file barray-meat.hpp.

#### 9.4.3.26 search

```
auto search = ROW(i).find(j)
```

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

#### 9.4.3.27 source

```
size_t const std::vector< size_t > & source
```

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

#### 9.4.3.28 target

```
size_t const std::vector< size_t > const std::vector< size_t > & target
```

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

#### 9.4.3.29 v

```
size_t Cell_Type v
```

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

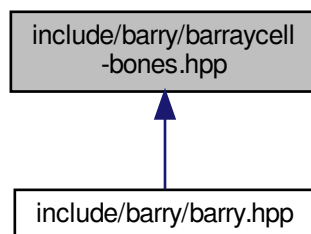
#### 9.4.3.30 value

```
size_t const std::vector< size_t > const std::vector< size_t > const std::vector< Cell_Type  
>& value
```

Definition at line 33 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:

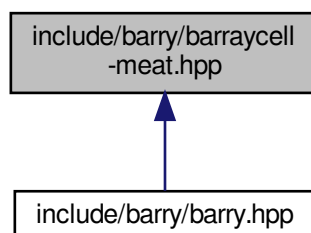


### Classes

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

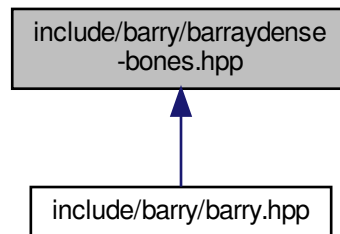
## 9.6 include/barry/barraycell-meat.hpp File Reference

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:

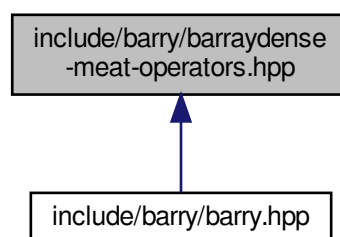


### Classes

- 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

```
template 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

```
template Data_Type 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 barrydense-meat-operators.hpp.

### 9.8.2.4 BDENSE\_TEMPLATE() [4/4]

```
BDENSE_TEMPLATE (
    BDENSE_TYPE() & ,
    operator/ ) const &
```

Definition at line 101 of file barrydense-meat-operators.hpp.

### 9.8.2.5 BDENSE\_TEMPLATE\_ARGS()

```
template BDENSE_TEMPLATE_ARGS ( ) const &
```

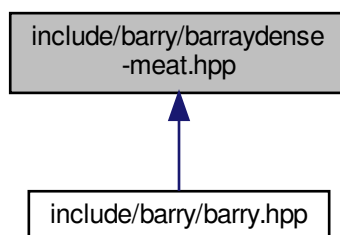
### 9.8.2.6 BDENSE\_TYPE()

```
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 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)`
- `#define ZERO_CELL static_cast<Cell_Type>(0.0)`

## Functions

- `BDENSE_TEMPLATE (, BArrayDense)(size_t N_`
- `el resize (N *M, ZERO_CELL)`
- `el_rowsums resize (N, ZERO_CELL)`
- `el_colsums resize (M, ZERO_CELL)`
- `for (size_t i=0u;i< source.size();++i)`
- `BDENSE_TEMPLATE (, BArrayDense)(const BDENSE_TYPE() &Array_`
- `bool M (Array_.M)`
- `BDENSE_TEMPLATE (BDENSE_TYPE() &, operator=)(const BDENSE_TYPE() &Array_)`
- `BDENSE_TEMPLATE (, BArrayDense)(BDENSE_TYPE() &&x) noexcept`
- `BDENSE_TEMPLATE (BDENSE_TYPE() &, operator=)(BDENSE_TYPE() &&x) noexcept`
- `BDENSE_TEMPLATE (bool, operator==)(const BDENSE_TYPE() &Array_)`
- `BDENSE_TEMPLATE (, ~BArrayDense)()`
- `BDENSE_TEMPLATE (void, set_data)(Data_Type *data_`
- `BDENSE_TEMPLATE (Data_Type *, D_ptr)()`
- `BDENSE_TEMPLATE (const Data_Type *, D_ptr)() const`
- `BDENSE_TEMPLATE (Data_Type &, D)()`
- `BDENSE_TEMPLATE (const Data_Type &, D)() const`
- `BDENSE_TEMPLATE (void, out_of_range)(size_t i`
- `BDENSE_TEMPLATE (Cell_Type, get_cell)(size_t i`
- `BDENSE_TEMPLATE (std::vector< Cell_Type >, get_row_vec)(size_t i`
- `std::vector< Cell_Type > ans (ncol(), static_cast< Cell_Type >(false))`
- `BDENSE_TEMPLATE (void, get_row_vec)(std`
- `BDENSE_TEMPLATE (Entries< Cell_Type >, get_entries)() const`
- `BDENSE_TEMPLATE (bool, is_empty)(size_t i`
- `BDENSE_TEMPLATE (size_t, nrow)() const noexcept`
- `BDENSE_TEMPLATE (size_t, ncol)() const noexcept`
- `BDENSE_TEMPLATE (size_t, nnonzero)() const noexcept`
- `BDENSE_TEMPLATE (Cell< Cell_Type >, default_val)() const`
- `BDENSE_TEMPLATE (BDENSE_TYPE() &, operator+=)(const std`
- `BDENSE_TEMPLATE (BDENSE_TYPE() &, operator-=)(const std`
- `BDENSE_TEMPLATE (void, insert_cell)(size_t i`
- `if (el[POS(i, j)]==BARRY_ZERO_DENSE)`
- `BDENSE_TEMPLATE (void, swap_cells)(size_t i0`
- `if ((i0==i1) &&(j0==j1)) return`
- `rm_cell (i0, j0, false, false)`
- `rm_cell (i1, j1, false, false)`
- `insert_cell (i0, j0, val1, false, false)`
- `insert_cell (i1, j1, val0, false, false)`
- `BDENSE_TEMPLATE (void, toggle_cell)(size_t i`
- `else rm_cell (i, j, false, false)`
- `BDENSE_TEMPLATE (void, swap_rows)(size_t i0`

- `BDENSE_TEMPLATE` (void, swap\_cols)(size\_t j0
- `BDENSE_TEMPLATE` (void, zero\_row)(size\_t i
- `if` (el\_rowsums[i]==ZERO\_CELL) `return`
- `BDENSE_TEMPLATE` (void, zero\_col)(size\_t j
- `if` (el\_colsums[j]==ZERO\_CELL) `return`
- `BDENSE_TEMPLATE` (void, transpose)()
- `BDENSE_TEMPLATE` (void, clear)(bool hard)
- `BDENSE_TEMPLATE` (void, resize)(size\_t N\_
- `el` `resize` (N\_\*M\_, ZERO\_CELL)
- `el_rowsums` `resize` (N\_, ZERO\_CELL)
- `el_colsums` `resize` (M\_, ZERO\_CELL)
- `BDENSE_TEMPLATE` (void, reserve)()
- `BDENSE_TEMPLATE` (void, print)(const char \*fmt
- `va_start` (args, fmt)
- `printf_barry` (fmt, args)
- `va_end` (args)
- `BDENSE_TEMPLATE` (const std::vector< Cell\_Type > &, get\_data)() `const`
- `BDENSE_TEMPLATE` (const Cell\_Type, rowsum)(size\_t i) `const`
- `BDENSE_TEMPLATE` (const Cell\_Type, colsum)(size\_t j) `const`

## Variables

- size\_t `M_`
- size\_t `const` std::vector< size\_t > & `source`
- size\_t `const` std::vector< size\_t > `const` std::vector< size\_t > & `target`
- size\_t `const` std::vector< size\_t > `const` std::vector< size\_t > `const` std::vector< Cell\_Type > & `value`
- size\_t `const` std::vector< size\_t > `const` std::vector< size\_t > `const` std::vector< Cell\_Type > `bool` `add`
- `if`(source.size() !=value.size()) `throw` std `N` = `N_`
- `M` = `M_`
- `return`
- `bool` `copy_data`
- `bool` `delete_data_`
- `data` = `data_`
- `delete_data` = `delete_data_`
- size\_t `j` `const`
- size\_t `j`
- `return` `el` [`POS`(i, j)] == `ZERO_CELL`
- `return` `ans`
- size\_t `const` `Cell`< `Cell_Type` > & `v`
- size\_t `const` `Cell`< `Cell_Type` > `bool` `check_bounds`
- size\_t `const` `Cell`< `Cell_Type` > `bool` `bool` `check_exists`
- `else`
- `el_rowsums` [`i`] = (`v.value` - `old`)
- `el_colsums` [`j`] = (`v.value` - `old`)
- size\_t `j0`
- size\_t `size_t` `i1`
- size\_t `size_t` `size_t` `j1`
- size\_t `size_t` `size_t` `bool` `int` `int` \* `report`
- `Cell_Type` `val0` = `el`[`POS`(i0,j0)]
- `Cell_Type` `val1` = `el`[`POS`(i1,j1)]
- `false`
- `col`



## 9.9.1 Macro Definition Documentation

### 9.9.1.1 BDENSE\_TEMPLATE

```
#define BDENSE_TEMPLATE(  
    a,  
    b )  template BDENSE_TEMPLATE_ARGS() inline a BDENSE_TYPE()::b
```

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

### 9.9.1.2 BDENSE\_TEMPLATE\_ARGS

```
#define BDENSE_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

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

### 9.9.1.3 BDENSE\_TYPE

```
#define BDENSE_TYPE( ) BArrayDense<Cell_Type, Data_Type>
```

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

### 9.9.1.4 COL

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

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

### 9.9.1.5 POS

```
#define POS(  
    a,  
    b )  (b)*N + (a)
```

Definition at line 32 of file barraydense-meat.hpp.

### 9.9.1.6 POS\_N

```
#define POS_N(  
    a,  
    b,  
    c ) (b)*(c) + (a)
```

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

### 9.9.1.7 ROW

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

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

### 9.9.1.8 ZERO\_CELL

```
#define ZERO_CELL static_cast<Cell_Type>(0.0)
```

Definition at line 38 of file bararraydense-meat.hpp.

## 9.9.2 Function Documentation

### 9.9.2.1 ans()

```
std::vector< Cell_Type > ans (  
    ncol() ,  
    static_cast< Cell_Type > false )
```

### 9.9.2.2 BDENSE\_TEMPLATE() [1/39]

```
BDENSE_TEMPLATE (  
    BArrayDense ) && [noexcept]
```

Definition at line 240 of file bararraydense-meat.hpp.

### 9.9.2.3 BDENSE\_TEMPLATE() [2/39]

```
BDENSE_TEMPLATE (
    BArrayDense ) const &
```

### 9.9.2.4 BDENSE\_TEMPLATE() [3/39]

```
BDENSE_TEMPLATE (
    BArrayDense )
```

### 9.9.2.5 BDENSE\_TEMPLATE() [4/39]

```
BDENSE_TEMPLATE (
    ~ BArrayDense )
```

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

### 9.9.2.6 BDENSE\_TEMPLATE() [5/39]

```
BDENSE_TEMPLATE (
    BDENSE_TYPE() & ,
    operator+ ) const
```

Definition at line 566 of file barraydense-meat.hpp.

### 9.9.2.7 BDENSE\_TEMPLATE() [6/39]

```
BDENSE_TEMPLATE (
    BDENSE_TYPE() & ,
    operator- ) const
```

Definition at line 584 of file barraydense-meat.hpp.

### 9.9.2.8 BDENSE\_TEMPLATE() [7/39]

```
BDENSE_TEMPLATE (
    BDENSE_TYPE() & ,
    operator ) && [noexcept]
```

Definition at line 257 of file barraydense-meat.hpp.

#### 9.9.2.9 BDENSE\_TEMPLATE() [8/39]

```
BDENSE_TEMPLATE (
    BDENSE_TYPE() & ,
    operator ) const &
```

Definition at line 194 of file bararraydense-meat.hpp.

#### 9.9.2.10 BDENSE\_TEMPLATE() [9/39]

```
BDENSE_TEMPLATE (
    bool ,
    is_empty )
```

#### 9.9.2.11 BDENSE\_TEMPLATE() [10/39]

```
BDENSE_TEMPLATE (
    bool ,
    operator == ) const &
```

Definition at line 300 of file bararraydense-meat.hpp.

#### 9.9.2.12 BDENSE\_TEMPLATE() [11/39]

```
BDENSE_TEMPLATE (
    Cell< Cell_Type > ,
    default_val ) const
```

Definition at line 562 of file bararraydense-meat.hpp.

#### 9.9.2.13 BDENSE\_TEMPLATE() [12/39]

```
BDENSE_TEMPLATE (
    Cell_Type ,
    get_cell )
```

**9.9.2.14 BDENSE\_TEMPLATE() [13/39]**

```
BDENSE_TEMPLATE (
    const Cell_Type,
    colsum ) const
```

Definition at line 999 of file barraydense-meat.hpp.

**9.9.2.15 BDENSE\_TEMPLATE() [14/39]**

```
BDENSE_TEMPLATE (
    const Cell_Type,
    rowsum ) const
```

Definition at line 994 of file barraydense-meat.hpp.

**9.9.2.16 BDENSE\_TEMPLATE() [15/39]**

```
BDENSE_TEMPLATE (
    const Data_Type & ,
    D ) const
```

Definition at line 353 of file barraydense-meat.hpp.

**9.9.2.17 BDENSE\_TEMPLATE() [16/39]**

```
BDENSE_TEMPLATE (
    const Data_Type * ,
    D_ptr ) const
```

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

**9.9.2.18 BDENSE\_TEMPLATE() [17/39]**

```
BDENSE_TEMPLATE (
    const std::vector< Cell_Type > & ,
    get_data ) const
```

Definition at line 989 of file barraydense-meat.hpp.

**9.9.2.19 BDENSE\_TEMPLATE() [18/39]**

```
BDENSE_TEMPLATE (
    Data_Type & ,
    D )
```

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

**9.9.2.20 BDENSE\_TEMPLATE() [19/39]**

```
BDENSE_TEMPLATE (
    Data_Type * ,
    D_ptr )
```

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

**9.9.2.21 BDENSE\_TEMPLATE() [20/39]**

```
BDENSE_TEMPLATE (
    Entries< Cell_Type > ,
    get_entries ) const
```

Definition at line 502 of file bararraydense-meat.hpp.

**9.9.2.22 BDENSE\_TEMPLATE() [21/39]**

```
BDENSE_TEMPLATE (
    size_t ,
    ncol ) const [noexcept]
```

Definition at line 548 of file bararraydense-meat.hpp.

**9.9.2.23 BDENSE\_TEMPLATE() [22/39]**

```
BDENSE_TEMPLATE (
    size_t ,
    nnozero ) const [noexcept]
```

Definition at line 552 of file bararraydense-meat.hpp.

**9.9.2.24 BDENSE\_TEMPLATE()** [23/39]

```
BDENSE_TEMPLATE (
    size_t ,
    nrow ) const [noexcept]
```

Definition at line 544 of file barraydense-meat.hpp.

**9.9.2.25 BDENSE\_TEMPLATE()** [24/39]

```
BDENSE_TEMPLATE (
    std::vector< Cell_Type > ,
    get_row_vec )
```

**9.9.2.26 BDENSE\_TEMPLATE()** [25/39]

```
BDENSE_TEMPLATE (
    void ,
    clear )
```

Definition at line 896 of file barraydense-meat.hpp.

**9.9.2.27 BDENSE\_TEMPLATE()** [26/39]

```
BDENSE_TEMPLATE (
    void ,
    get_row_vec )
```

Definition at line 402 of file barraydense-meat.hpp.

**9.9.2.28 BDENSE\_TEMPLATE()** [27/39]

```
BDENSE_TEMPLATE (
    void ,
    insert_cell )
```

**9.9.2.29 BDENSE\_TEMPLATE()** [28/39]

```
BDENSE_TEMPLATE (
    void ,
    out_of_range )
```

**9.9.2.30 BDENSE\_TEMPLATE()** [29/39]

```
BDENSE_TEMPLATE (
    void ,
    print ) const
```

**9.9.2.31 BDENSE\_TEMPLATE()** [30/39]

```
BDENSE_TEMPLATE (
    void ,
    reserve )
```

Definition at line 946 of file `barraydense-meat.hpp`.

**9.9.2.32 BDENSE\_TEMPLATE()** [31/39]

```
BDENSE_TEMPLATE (
    void ,
    resize )
```

**9.9.2.33 BDENSE\_TEMPLATE()** [32/39]

```
BDENSE_TEMPLATE (
    void ,
    set_data )
```

**9.9.2.34 BDENSE\_TEMPLATE()** [33/39]

```
BDENSE_TEMPLATE (
    void ,
    swap_cells )
```



**9.9.2.35 BDENSE\_TEMPLATE()** [34/39]

```
BDENSE_TEMPLATE (
    void ,
    swap_cols )
```

**9.9.2.36 BDENSE\_TEMPLATE()** [35/39]

```
BDENSE_TEMPLATE (
    void ,
    swap_rows )
```

**9.9.2.37 BDENSE\_TEMPLATE()** [36/39]

```
BDENSE_TEMPLATE (
    void ,
    toggle_cell )
```

**9.9.2.38 BDENSE\_TEMPLATE()** [37/39]

```
BDENSE_TEMPLATE (
    void ,
    transpose )
```

Definition at line 868 of file barraydense-meat.hpp.

**9.9.2.39 BDENSE\_TEMPLATE()** [38/39]

```
BDENSE_TEMPLATE (
    void ,
    zero_col )
```

**9.9.2.40 BDENSE\_TEMPLATE()** [39/39]

```
BDENSE_TEMPLATE (
    void ,
    zero_row )
```

#### 9.9.2.41 for()

```
for ( )
```

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

#### 9.9.2.42 if() [1/4]

```
if (
    (i0==i1) && (j0==j1) )
```

#### 9.9.2.43 if() [2/4]

```
if (
    el [POS(i, j)] == BARRY_ZERO_DENSE )
```

Definition at line 663 of file bararraydense-meat.hpp.

#### 9.9.2.44 if() [3/4]

```
if (
    el_colsums [j] == ZERO_CELL )
```

#### 9.9.2.45 if() [4/4]

```
if (
    el_rowsums [i] == ZERO_CELL )
```

#### 9.9.2.46 insert\_cell() [1/2]

```
insert_cell (
    i0 ,
    j0 ,
    val ,
    false ,
    false )
```

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

```
insert_cell (
    i1 ,
    j1 ,
    val0 ,
    false ,
    false )
```

**9.9.2.48 M()**

```
bool M (
    Array_. M )
```

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

**9.9.2.49 printf\_barry()**

```
printf_barry (
    fmt ,
    args )
```

**9.9.2.50 resize()** [1/6]

```
el_colsums resize (
    M ,
    ZERO_CELL )
```

**9.9.2.51 resize()** [2/6]

```
el_colsums resize (
    M_ ,
    ZERO_CELL )
```

**9.9.2.52 resize()** [3/6]

```
el resize (
    N * M,
    ZERO_CELL )
```

**9.9.2.53** `resize()` [4/6]

```
el_rowsums resize (
    N ,
    ZERO_CELL )
```

**9.9.2.54** `resize()` [5/6]

```
el resize (
    N_ * M_,
    ZERO_CELL )
```

**9.9.2.55** `resize()` [6/6]

```
el_rowsums resize (
    N_ ,
    ZERO_CELL )
```

**9.9.2.56** `rm_cell()` [1/3]

```
else rm_cell (
    i ,
    j ,
    false ,
    false )
```

**9.9.2.57** `rm_cell()` [2/3]

```
rm_cell (
    i0 ,
    j0 ,
    false ,
    false )
```

**9.9.2.58** `rm_cell()` [3/3]

```
rm_cell (
    i1 ,
    j1 ,
    false ,
    false )
```

### 9.9.2.59 va\_end()

```
va_end (
    args )
```

### 9.9.2.60 va\_start()

```
va_start (
    args ,
    fmt )
```

## 9.9.3 Variable Documentation

### 9.9.3.1 add

```
size_t const std::vector< size_t > const std::vector< size_t > bool add
```

#### Initial value:

```
{
    if (source.size() != target.size())
        throw std::length_error("-source- and -target- don't match on length.")
```

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

### 9.9.3.2 ans

```
return ans
```

Definition at line 398 of file barraydense-meat.hpp.

### 9.9.3.3 check\_bounds

```
bool check_bounds
```

#### Initial value:

```
{
    if (check_bounds)
    {
        out_of_range(i0,0u);
        out_of_range(i1,0u);
    }

    for (size_t j = 0u; j < M; ++j)
        std::swap(el[POS(i0, j)], el[POS(i1, j)])
```

Definition at line 654 of file barraydense-meat.hpp.

#### 9.9.3.4 check\_exists

```
size_t bool int check_exists
```

##### Initial value:

```
{  
    if (check_bounds)  
        out_of_range(i, j)
```

Definition at line 655 of file bararraydense-meat.hpp.

#### 9.9.3.5 col

```
col
```

Definition at line 843 of file bararraydense-meat.hpp.

#### 9.9.3.6 const

```
const
```

##### Initial value:

```
{  
    if (i >= N)  
        throw std::range_error("The row is out of range.")
```

Definition at line 360 of file bararraydense-meat.hpp.

#### 9.9.3.7 copy\_data

```
bool copy_data
```

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

#### 9.9.3.8 data

```
data = data_
```

Definition at line 334 of file bararraydense-meat.hpp.

### 9.9.3.9 delete\_data

```
delete_data = delete_data_
```

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

### 9.9.3.10 delete\_data\_

```
bool delete_data_
```

#### Initial value:

```
{  
    if ((data != nullptr) && delete_data)  
        delete data
```

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

### 9.9.3.11 el

```
return el == ZERO_CELL
```

Definition at line 381 of file barraydense-meat.hpp.

### 9.9.3.12 el\_colsums

```
el_colsums[j] = (v.value - old)
```

Definition at line 675 of file barraydense-meat.hpp.

### 9.9.3.13 el\_rowsums

```
el_rowsums[i] = (v.value - old)
```

Definition at line 674 of file barraydense-meat.hpp.

**9.9.3.14 else**

```
else (
    void )
```

**Initial value:**

```
{
    Cell_Type old = el[POS(i,j)]
```

Definition at line 670 of file bararraydense-meat.hpp.

**9.9.3.15 false**

```
false
```

Definition at line 767 of file bararraydense-meat.hpp.

**9.9.3.16 i1**

```
size_t i1
```

Definition at line 721 of file bararraydense-meat.hpp.

**9.9.3.17 j**

```
j
```

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

**9.9.3.18 j0**

```
size_t j0
```

Definition at line 720 of file bararraydense-meat.hpp.

**9.9.3.19 j1**

```
size_t j1
```

Definition at line 721 of file bararraydense-meat.hpp.



**9.9.3.20 M**

M = [M\\_](#)

Definition at line 57 of file barraydense-meat.hpp.

**9.9.3.21 M\_**

size\_t M\_

**Initial value:**

```
{
    std::vector< Cell_Type > el_tmp(el)
```

Definition at line 43 of file barraydense-meat.hpp.

**9.9.3.22 N**

N = [N\\_](#)

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

**9.9.3.23 report**

size\_t size\_t size\_t bool int int\* report

**Initial value:**

```
{
    if (check\_bounds) {
        out_of_range(i0, j0);
        out_of_range(i1, j1);
    }

    if (report != nullptr)
        (*report) = EXISTS::BOTH
```

Definition at line 724 of file barraydense-meat.hpp.

**9.9.3.24 return**

return

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

#### 9.9.3.25 source

```
size_t const std::vector< size_t >& source
```

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

#### 9.9.3.26 target

```
size_t const std::vector< size_t > const std::vector< size_t >& target
```

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

#### 9.9.3.27 v

```
size_t Cell_Type v
```

Definition at line 653 of file baraydense-meat.hpp.

#### 9.9.3.28 val0

```
Cell_Type val0 = el[POS(i0,j0)]
```

Definition at line 742 of file baraydense-meat.hpp.

#### 9.9.3.29 val1

```
Cell_Type val1 = el[POS(i1,j1)]
```

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

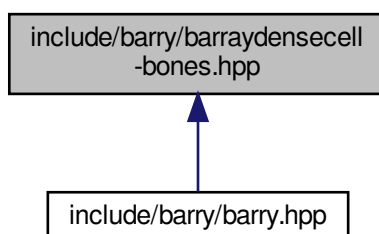
#### 9.9.3.30 value

```
size_t const std::vector< size_t > const std::vector< size_t > const std::vector< Cell_Type  
>& value
```

Definition at line 46 of file baraydense-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

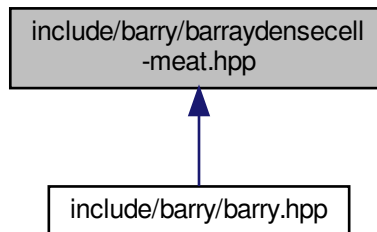
#### 9.10.1.1 POS

```
#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

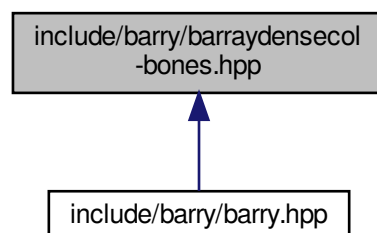
##### 9.11.1.1 POS

```
#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`.

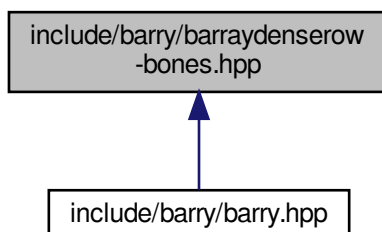
#### 9.12.1.3 ZERO\_CELL

```
#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 bararrayrow-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 bararrayrow-meat.hpp.

### 9.15.1.3 BROW\_TYPE

```
#define BROW_TYPE( ) BArrayRow<Cell_Type, Data_Type>
```

Definition at line 4 of file bararrayrow-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 bararrayrow-meat.hpp.

### 9.15.2.2 BROW\_TEMPLATE() [2/5]

```
BROW_TEMPLATE (  
    void ,  
    operator+ ) const &
```

Definition at line 25 of file bararrayrow-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

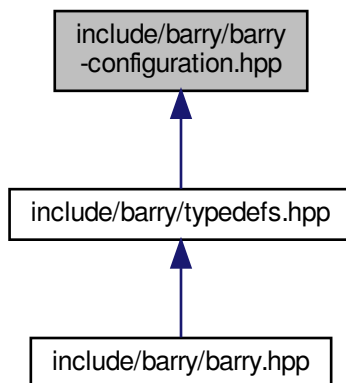
### Classes

- 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

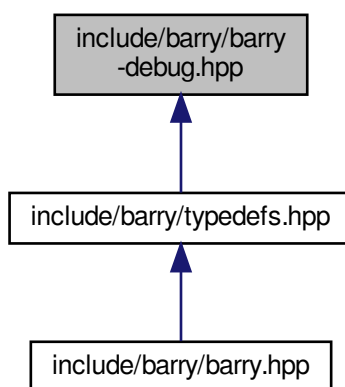
### 9.18.2.1 Map

```
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

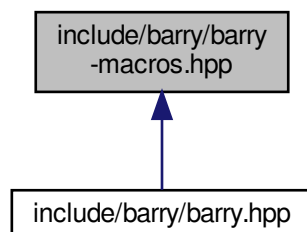
#### 9.19.1.1 BARRY\_DEBUG\_LEVEL

```
#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);`

### 9.20.1 Macro Definition Documentation

#### 9.20.1.1 BARRY\_ONE

```
#define BARRY_ONE Cell<Cell_Type>(1.0)
```

Definition at line 7 of file barry-macros.hpp.

#### 9.20.1.2 BARRY\_ONE\_DENSE

```
#define BARRY_ONE_DENSE static_cast<Cell_Type>(1.0)
```

Definition at line 8 of file barry-macros.hpp.

### 9.20.1.3 BARRY\_UNUSED

```
#define BARRY_UNUSED(  
    expr ) do { (void)(expr); } while (0);
```

Definition at line 10 of file barry-macros.hpp.

### 9.20.1.4 BARRY\_ZERO

```
#define BARRY_ZERO Cell<Cell_Type>(0.0)
```

Definition at line 4 of file barry-macros.hpp.

### 9.20.1.5 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 <cstdio>  
#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 "counters/phylo.hpp"
#include "counters/defm.hpp"
```

## Namespaces

- [barry](#)  
*barry: Your go-to motif accountant*
- [barry::counters](#)  
*Tree class and Treeliterator class.*
- [barry::counters::network](#)
- [barry::counters::phylo](#)
- [barry::counters::defm](#)

## 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 25 of file barry.hpp.

### 9.21.1.2 BARRY\_VERSION

```
#define BARRY_VERSION BARRY_VERSION_MAYOR ## . ## BARRY_VERSION_MINOR
```

Definition at line 29 of file barry.hpp.

### 9.21.1.3 BARRY\_VERSION\_MAYOR

```
#define BARRY_VERSION_MAYOR 0
```

Definition at line 27 of file barry.hpp.

### 9.21.1.4 BARRY\_VERSION\_MINOR

```
#define BARRY_VERSION_MINOR 1
```

Definition at line 28 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 96 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 99 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 103 of file barry.hpp.

### 9.21.1.8 RULE\_LAMBDA

```
#define RULE_LAMBDA(  
    a )
```

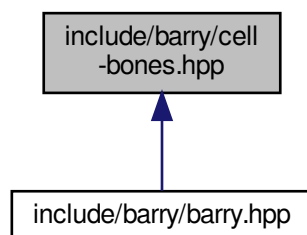
#### Value:

```
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 106 of file barry.hpp.

## 9.22 include/barry/cell-bones.hpp File Reference

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

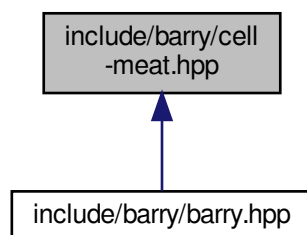


## Classes

- class [Cell< Cell\\_Type >](#)  
*Entries in [BArray](#). For now, it only has two members:*

## 9.23 include/barry/cell-meat.hpp File Reference

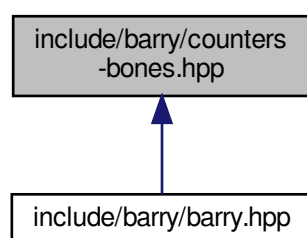
This graph shows which files directly or indirectly include this file:



## 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:

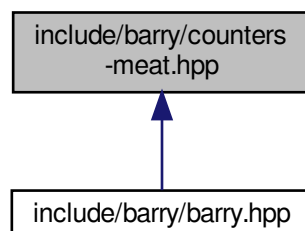


## Classes

- 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](#) = [locator->second](#)
- [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 212 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 231 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 246 of file counters-meat.hpp.

**9.26.2.28 if()** [2/3]

```
if (  
    hasher )
```

Definition at line 253 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 225 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\_Rule\_Dyn\_Type Data\_Rule\_Dyn\_Type data\_

#### Initial value:

```
{  
    rules_dyn->add_rule(  
        rule_fun_,  
        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.push_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 268 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**

```
const std::vector< double > size_t i = locator->second
```

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 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 263 of file counters-meat.hpp.

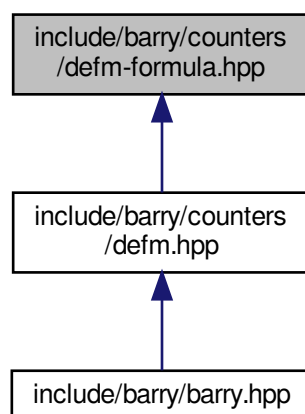
### 9.26.3.16 return

```
return
```

Definition at line 175 of file counters-meat.hpp.

## 9.27 include/barry/counters/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.*

### 9.27.1 Function Documentation

### 9.27.1.1 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>	

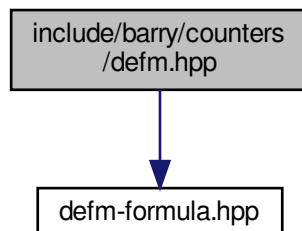
Definition at line 46 of file `defm-formula.hpp`.



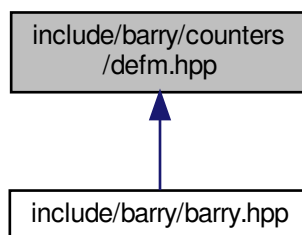
## 9.28 include/barry/counters/defm.hpp File Reference

```
#include "defm-formula.hpp"
```

Include dependency graph for defm.hpp:



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



### Classes

- class [DEFMData](#)
- class [DEFMCounterData](#)
  - Data class used to store arbitrary size\_t or double vectors.*
- class [DEFMRuleData](#)
- class [DEFMRuleDynData](#)

### 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)`

## Typedefs

- `typedef BArrayDense< int, DEFMData > DEFMArray`

### Convenient typedefs for network objects.

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

## 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 <i>DEFMRules</i> object ( <i>Rules</i> < <i>DEFMArray</i> , <i>bool</i> >).
-------	--

- 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.

**9.28.1 Macro Definition Documentation****9.28.1.1 UNI\_SUB**

```
#define UNI_SUB(  
    a )
```

**Value:**

```
(\
  ((a) == 0) ? "\u2080" : (\
  ((a) == 1) ? "\u2081" : (\
  ((a) == 2) ? "\u2082" : (\
  ((a) == 3) ? "\u2083" : (\
  ((a) == 4) ? "\u2084" : (\
  ((a) == 5) ? "\u2085" : (\
  ((a) == 6) ? "\u2086" : (\
  ((a) == 7) ? "\u2087" : (\
  ((a) == 8) ? "\u2088" : (\
  "\u2089")))))))\
)
```

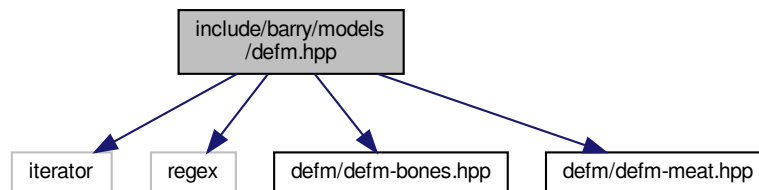
**9.28.2 Typedef Documentation****9.28.2.1 DEFMArray**

```
typedef BArrayDense<int, DEFMData> DEFMArray
```

Definition at line 25 of file defm.hpp.

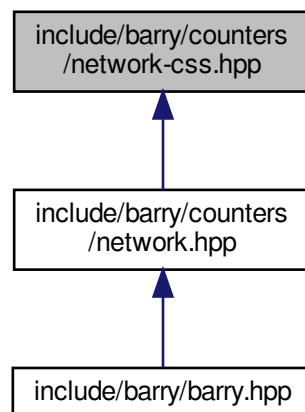
## 9.29 include/barry/models/defm.hpp File Reference

```
#include <iterator>
#include <regex>
#include "defm/defm-bones.hpp"
#include "defm/defm-meat.hpp"
Include dependency graph for defm.hpp:
```



## 9.30 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_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\_)  
*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\_)  
*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\_)  
*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\_)  
*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\_)  
*Counts mixed reciprocity errors.*
- template<typename Tnet = Network>  
void `counter_css_census01` (NetCounters< Tnet > \*counters, size\_t netsize, const std::vector< size\_t > &end\_)
- template<typename Tnet = Network>  
void `counter_css_census02` (NetCounters< Tnet > \*counters, size\_t netsize, const std::vector< size\_t > &end\_)
- template<typename Tnet = Network>  
void `counter_css_census03` (NetCounters< Tnet > \*counters, size\_t netsize, const std::vector< size\_t > &end\_)
- template<typename Tnet = Network>  
void `counter_css_census04` (NetCounters< Tnet > \*counters, size\_t netsize, const std::vector< size\_t > &end\_)
- template<typename Tnet = Network>  
void `counter_css_census05` (NetCounters< Tnet > \*counters, size\_t netsize, const std::vector< size\_t > &end\_)
- template<typename Tnet = Network>  
void `counter_css_census06` (NetCounters< Tnet > \*counters, size\_t netsize, const std::vector< size\_t > &end\_)
- template<typename Tnet = Network>  
void `counter_css_census07` (NetCounters< Tnet > \*counters, size\_t netsize, const std::vector< size\_t > &end\_)
- template<typename Tnet = Network>  
void `counter_css_census08` (NetCounters< Tnet > \*counters, size\_t netsize, const std::vector< size\_t > &end\_)

- `template<typename Tnet = Network>`  
`void counter_css_census09 (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_)`
- `template<typename Tnet = Network>`  
`void counter_css_census10 (NetCounters< Tnet > *counters, size_t netsize, const std::vector< size_t > &end_)`

## 9.30.1 Macro Definition Documentation

### 9.30.1.1 CSS\_APPEND

```
#define CSS_APPEND(  
    name )
```

#### Value:

```
std::string name_ = (name);\nfor (size_t i = 0u; i < end_.size(); ++i) { \nstd::string tmpname = name_ + " (" + std::to_string(i) + ")";\ncounters->add_counter(tmp_count, tmp_init, nullptr, \n    NetCounterData({netsize, i == 0u ? netsize : end_[i-1], end_[i]}, {}),\n    tmpname);}
```

Definition at line 42 of file network-css.hpp.

### 9.30.1.2 CSS\_CASE\_ELSE

```
#define CSS_CASE_ELSE( )
```

Definition at line 27 of file network-css.hpp.

### 9.30.1.3 CSS\_CASE\_PERCEIVED

```
#define CSS_CASE_PERCEIVED( ) else if (((i >= s) && (i < e)) & ((j >= s) && (j < e)))
```

Definition at line 20 of file network-css.hpp.

### 9.30.1.4 CSS\_CASE\_TRUTH

```
#define CSS_CASE_TRUTH( ) if ((i < n) && (j < n))
```

Definition at line 13 of file network-css.hpp.

### 9.30.1.5 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 37 of file network-css.hpp.

### 9.30.1.6 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 31 of file network-css.hpp.

### 9.30.1.7 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 49 of file network-css.hpp.

### 9.30.1.8 CSS\_PERCEIVED\_CELLS

```
#define CSS_PERCEIVED_CELLS( )
```

**Value:**

```
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 21 of file network-css.hpp.

### 9.30.1.9 CSS\_SIZE

```
#define CSS_SIZE( )
```

#### Value:

```
size_t n = data.indices[0u]; \
size_t s = data.indices[1u]; \
size_t e = data.indices[2u];
```

Definition at line 7 of file network-css.hpp.

### 9.30.1.10 CSS\_TRUE\_CELLS

```
#define CSS_TRUE_CELLS( )
```

#### Value:

```
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 14 of file network-css.hpp.

## 9.30.2 Function Documentation

### 9.30.2.1 counter\_css\_census01()

```
template<typename Tnet = Network>
void counter_css_census01 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_ ) [inline]
```

Definition at line 275 of file network-css.hpp.

### 9.30.2.2 counter\_css\_census02()

```
template<typename Tnet = Network>
void counter_css_census02 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_ ) [inline]
```

Definition at line 325 of file network-css.hpp.



### 9.30.2.3 counter\_css\_census03()

```
template<typename Tnet = Network>
void counter_css_census03 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_ ) [inline]
```

Definition at line 364 of file network-css.hpp.

### 9.30.2.4 counter\_css\_census04()

```
template<typename Tnet = Network>
void counter_css_census04 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_ ) [inline]
```

Definition at line 403 of file network-css.hpp.

### 9.30.2.5 counter\_css\_census05()

```
template<typename Tnet = Network>
void counter_css_census05 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_ ) [inline]
```

Definition at line 442 of file network-css.hpp.

### 9.30.2.6 counter\_css\_census06()

```
template<typename Tnet = Network>
void counter_css_census06 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_ ) [inline]
```

Definition at line 481 of file network-css.hpp.

### 9.30.2.7 counter\_css\_census07()

```
template<typename Tnet = Network>
void counter_css_census07 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_ ) [inline]
```

Definition at line 520 of file network-css.hpp.

### 9.30.2.8 counter\_css\_census08()

```
template<typename Tnet = Network>
void counter_css_census08 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_ ) [inline]
```

Definition at line 559 of file network-css.hpp.

### 9.30.2.9 counter\_css\_census09()

```
template<typename Tnet = Network>
void counter_css_census09 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_ ) [inline]
```

Definition at line 598 of file network-css.hpp.

### 9.30.2.10 counter\_css\_census10()

```
template<typename Tnet = Network>
void counter_css_census10 (
    NetCounters< Tnet > * counters,
    size_t netsize,
    const std::vector< size_t > & end_ ) [inline]
```

Definition at line 637 of file network-css.hpp.

#### 9.30.2.11 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_ ) [inline]
```

Counts completely false reciprocity (comission)

Definition at line 154 of file network-css.hpp.

#### 9.30.2.12 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_ ) [inline]
```

Counts completely false reciprocity (omission)

Definition at line 194 of file network-css.hpp.

#### 9.30.2.13 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_ ) [inline]
```

Counts mixed reciprocity errors.

Definition at line 234 of file network-css.hpp.

#### 9.30.2.14 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_ ) [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)
—	

The *end\_* parameter should be of length *N* of *networks* - 1. It is assumed that the first network ends at *netsize*.

Definition at line 63 of file *network-css.hpp*.

### 9.30.2.15 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_ ) [inline]
```

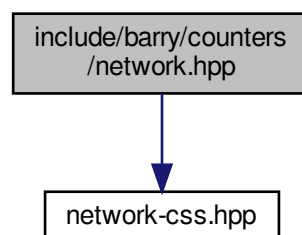
Counts errors of omission.

Definition at line 110 of file *network-css.hpp*.

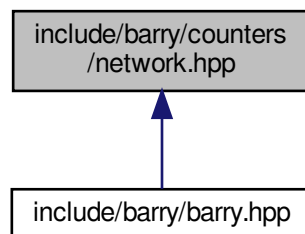
## 9.31 include/barry/counters/network.hpp File Reference

```
#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 >`

## 9.31.1 Macro Definition Documentation

### 9.31.1.1 BARRY\_ZERO\_NETWORK

```
#define BARRY_ZERO_NETWORK 0.0
```

Definition at line 85 of file network.hpp.

### 9.31.1.2 BARRY\_ZERO\_NETWORK\_DENSE

```
#define BARRY_ZERO_NETWORK_DENSE 0
```

Definition at line 86 of file network.hpp.

### 9.31.1.3 NET\_C\_DATA\_IDX

```
#define NET_C_DATA_IDX(  
    i ) (data.indices[i])
```

Definition at line 74 of file network.hpp.

### 9.31.1.4 NET\_C\_DATA\_NUM

```
#define NET_C_DATA_NUM(  
    i ) (data.numbers[i])
```

Definition at line 75 of file network.hpp.

### 9.31.1.5 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.

### 9.31.1.6 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.



### 9.31.1.7 NETWORK\_RULE

```
#define NETWORK_RULE(  
    a )
```

**Value:**

```
template<typename Tnet = Network>\ninline 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.

### 9.31.1.8 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.

### 9.31.1.9 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.

## 9.31.2 Typedef Documentation

### 9.31.2.1 NetCounter

```
template<typename Tnet = Network>\nusing NetCounter = Counter<Tnet, NetCounterData >
```

Definition at line 89 of file network.hpp.

### 9.31.2.2 NetCounters

```
template<typename Tnet = Network>  
using NetCounters = Counters<Tnet, NetCounterData>
```

Definition at line 92 of file network.hpp.

### 9.31.2.3 NetModel

```
template<typename Tnet >  
using NetModel = Model<Tnet, NetCounterData>
```

Definition at line 101 of file network.hpp.

### 9.31.2.4 NetRule

```
template<typename Tnet = Network>  
using NetRule = Rule<Tnet, bool>
```

Definition at line 104 of file network.hpp.

### 9.31.2.5 NetRules

```
template<typename Tnet = Network>  
using NetRules = Rules<Tnet, bool>
```

Definition at line 107 of file network.hpp.

### 9.31.2.6 NetStatsCounter

```
template<typename Tnet = Network>  
using NetStatsCounter = StatsCounter<Tnet, NetCounterData>
```

Definition at line 98 of file network.hpp.

### 9.31.2.7 NetSupport

```
template<typename Tnet = Network>  
using NetSupport = Support<Tnet, NetCounterData >
```

Definition at line 95 of file network.hpp.

### 9.31.2.8 Network

```
typedef BArray<double, NetworkData> Network
```

Definition at line 82 of file network.hpp.

### 9.31.2.9 NetworkDense

```
typedef BArrayDense<int, NetworkData> NetworkDense
```

Definition at line 83 of file network.hpp.

## 9.31.3 Function Documentation

### 9.31.3.1 rules\_zerodiag()

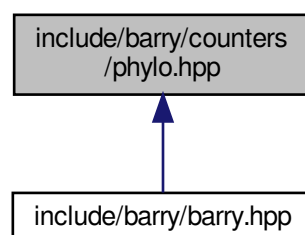
```
template<typename Tnet = Network>
void rules_zerodiag (
    NetRules< Tnet > * rules ) [inline]
```

Number of edges.

Definition at line 1381 of file network.hpp.

## 9.32 include/barry/counters/phylo.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` [DEFAULT\\_DUPLICATION](#) 1u
- `#define` [DUPL\\_SPEC](#) 0u
- `#define` [DUPL\\_DUPL](#) 1u
- `#define` [DUPL\\_EITH](#) 2u
- `#define` [MAKE\\_DUPL\\_VARS](#)()
- `#define` [IS\\_EITHER](#)() (DATA\_AT == [DUPL\\_EITH](#))
- `#define` [IS\\_DUPLICATION](#)() ((DATA\_AT == [DUPL\\_DUPL](#)) & (DPL))
- `#define` [IS\\_SPECIATION](#)() ((DATA\_AT == [DUPL\\_SPEC](#)) & (!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](#)()

## Typedefs

- `typedef` `std::vector< std::pair< size_t, size_t > >` [PhyloRuleData](#)

### Convenient typedefs for Node objects.

- `typedef` [BArrayDense](#)< size\_t, [NodeData](#) > [PhyloArray](#)
- `typedef` [Counter](#)< [PhyloArray](#), [PhyloCounterData](#) > [PhyloCounter](#)
- `typedef` [Counters](#)< [PhyloArray](#), [PhyloCounterData](#) > [PhyloCounters](#)
- `typedef` [Rule](#)< [PhyloArray](#), [PhyloRuleData](#) > [PhyloRule](#)
- `typedef` [Rules](#)< [PhyloArray](#), [PhyloRuleData](#) > [PhyloRules](#)
- `typedef` [Rule](#)< [PhyloArray](#), [PhyloRuleDynData](#) > [PhyloRuleDyn](#)
- `typedef` [Rules](#)< [PhyloArray](#), [PhyloRuleDynData](#) > [PhyloRulesDyn](#)
- `typedef` [Support](#)< [PhyloArray](#), [PhyloCounterData](#), [PhyloRuleData](#), [PhyloRuleDynData](#) > [PhyloSupport](#)
- `typedef` [StatsCounter](#)< [PhyloArray](#), [PhyloCounterData](#) > [PhyloStatsCounter](#)
- `typedef` [Model](#)< [PhyloArray](#), [PhyloCounterData](#), [PhyloRuleData](#), [PhyloRuleDynData](#) > [PhyloModel](#)
- `typedef` [PowerSet](#)< [PhyloArray](#), [PhyloRuleData](#) > [PhyloPowerSet](#)

## Functions

- `std::string get_last_name (size_t d)`
- `void counter_overall_gains (PhyloCounters *counters, size_t duplication=DEFAULT_DUPLICATION)`  
*Overall functional gains.*
- `void counter_gains (PhyloCounters *counters, std::vector< size_t > nfun, size_t duplication=DEFAULT_DUPLICATION)`  
*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=DEFAULT_DUPLICATION)`  
*k genes gain function nfun*
- `void counter_genes_changing (PhyloCounters *counters, size_t duplication=DEFAULT_DUPLICATION)`  
*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=DEFAULT_DUPLICATION)`  
*Keeps track of how many pairs of genes preserve pseudostate.*
- `void counter_prop_genes_changing (PhyloCounters *counters, size_t duplication=DEFAULT_DUPLICATION)`  
*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=DEFAULT_DUPLICATION)`  
*Overall functional loss.*
- `void counter_maxfuns (PhyloCounters *counters, size_t lb, size_t ub, size_t duplication=DEFAULT_DUPLICATION)`  
*Cap the number of functions per gene.*
- `void counter_loss (PhyloCounters *counters, std::vector< size_t > nfun, size_t duplication=DEFAULT_DUPLICATION)`  
*Total count of losses for an specific function.*
- `void counter_overall_changes (PhyloCounters *counters, size_t duplication=DEFAULT_DUPLICATION)`  
*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=DEFAULT_DUPLICATION)`  
*Total count of Sub-functionalization events.*
- `void counter_cogain (PhyloCounters *counters, size_t nfunA, size_t nfunB, size_t duplication=DEFAULT_DUPLICATION)`  
*Co-evolution (joint gain or loss)*
- `void counter_longest (PhyloCounters *counters, size_t duplication=DEFAULT_DUPLICATION)`  
*Longest branch mutates (either by gain or by loss)*
- `void counter_neofun (PhyloCounters *counters, size_t nfunA, size_t nfunB, size_t duplication=DEFAULT_DUPLICATION)`  
*Total number of neofunctionalization events.*
- `void counter_pairwise_neofun_singlefun (PhyloCounters *counters, size_t nfunA, size_t duplication=DEFAULT_DUPLICATION)`  
*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=DEFAULT_DUPLICATION)`  
*Total number of neofunctionalization events.*
- `void counter_co_opt (PhyloCounters *counters, size_t nfunA, size_t nfunB, size_t duplication=DEFAULT_DUPLICATION)`  
*Function co-opting.*
- `void counter_k_genes_changing (PhyloCounters *counters, size_t k, size_t duplication=DEFAULT_DUPLICATION)`  
*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=DEFAULT_DUPLICATION)`  
*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=DEFAULT_DUPLICATION)`  
*Used when all the functions are in 0 (like the root node prob.)*
- `void counter_overall_gains_from_0 (PhyloCounters *counters, size_t duplication=DEFAULT_DUPLICATION)`  
*Used when all the functions are in 0 (like the root node prob.)*
- `void counter_pairwise_overall_change (PhyloCounters *counters, size_t duplication=DEFAULT_DUPLICATION)`  
*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=DEFAULT_DUPLICATION)`

*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=DEFAULT\_DUPLICATION)

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

- void `rule_dyn_limit_changes` (`PhyloSupport` \*support, size\_t pos, size\_t lb, size\_t ub, size\_t duplication=DEFAULT\_DUPLICATION)

*Overall functional gains.*

## 9.32.1 Macro Definition Documentation

### 9.32.1.1 DEFAULT\_DUPLICATION

```
#define DEFAULT_DUPLICATION 1u
```

Definition at line 5 of file phylo.hpp.

### 9.32.1.2 DUPL\_DUPL

```
#define DUPL_DUPL 1u
```

Definition at line 7 of file phylo.hpp.

### 9.32.1.3 DUPL\_EITH

```
#define DUPL_EITH 2u
```

Definition at line 8 of file phylo.hpp.

### 9.32.1.4 DUPL\_SPEC

```
#define DUPL_SPEC 0u
```

Definition at line 6 of file phylo.hpp.

### 9.32.1.5 IF\_MATCHES

```
#define IF_MATCHES( )
```

**Value:**

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

Definition at line 19 of file phylo.hpp.

### 9.32.1.6 IF\_NOTMATCHES

```
#define IF_NOTMATCHES( )
```

**Value:**

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

Definition at line 21 of file phylo.hpp.

### 9.32.1.7 IS\_DUPLICATION

```
#define IS_DUPLICATION( ) ((DATA_AT == DUPL_DUPL) & (DPL))
```

Definition at line 16 of file phylo.hpp.

### 9.32.1.8 IS\_EITHER

```
#define IS_EITHER( ) (DATA_AT == DUPL_EITH)
```

Definition at line 15 of file phylo.hpp.

### 9.32.1.9 IS\_SPECIATION

```
#define IS_SPECIATION( ) ((DATA_AT == DUPL_SPEC) & (!DPL))
```

Definition at line 17 of file phylo.hpp.

### 9.32.1.10 MAKE\_DUPL\_VARS

```
#define MAKE_DUPL_VARS( )
```

**Value:**

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

Definition at line 11 of file phylo.hpp.

### 9.32.1.11 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 139 of file phylo.hpp.

### 9.32.1.12 PHYLO\_COUNTER\_LAMBDA

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

**Value:**

```
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 133 of file phylo.hpp.

### 9.32.1.13 PHYLO\_RULE\_DYN\_LAMBDA

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

**Value:**

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

Definition at line 136 of file phylo.hpp.



## 9.32.2 Typedef Documentation

### 9.32.2.1 PhyloArray

```
typedef BArrayDense<size_t, NodeData> PhyloArray
```

Definition at line 106 of file phylo.hpp.

### 9.32.2.2 PhyloCounter

```
typedef Counter<PhyloArray, PhyloCounterData > PhyloCounter
```

Definition at line 107 of file phylo.hpp.

### 9.32.2.3 PhyloCounters

```
typedef Counters< PhyloArray, PhyloCounterData> PhyloCounters
```

Definition at line 108 of file phylo.hpp.

### 9.32.2.4 PhyloModel

```
typedef Model<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel
```

Definition at line 118 of file phylo.hpp.

### 9.32.2.5 PhyloPowerSet

```
typedef PowerSet<PhyloArray, PhyloRuleData> PhyloPowerSet
```

Definition at line 119 of file phylo.hpp.

#### 9.32.2.6 PhyloRule

```
typedef Rule<PhyloArray,PhyloRuleData> PhyloRule
```

Definition at line 110 of file phylo.hpp.

#### 9.32.2.7 PhyloRuleData

```
typedef std::vector< std::pair< size_t, size_t > > PhyloRuleData
```

Definition at line 99 of file phylo.hpp.

#### 9.32.2.8 PhyloRuleDyn

```
typedef Rule<PhyloArray,PhyloRuleDynData> PhyloRuleDyn
```

Definition at line 113 of file phylo.hpp.

#### 9.32.2.9 PhyloRules

```
typedef Rules<PhyloArray,PhyloRuleData> PhyloRules
```

Definition at line 111 of file phylo.hpp.

#### 9.32.2.10 PhyloRulesDyn

```
typedef Rules<PhyloArray,PhyloRuleDynData> PhyloRulesDyn
```

Definition at line 114 of file phylo.hpp.

#### 9.32.2.11 PhyloStatsCounter

```
typedef StatsCounter<PhyloArray, PhyloCounterData> PhyloStatsCounter
```

Definition at line 117 of file phylo.hpp.

### 9.32.2.12 PhyloSupport

```
typedef Support<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport
```

Definition at line 116 of file phylo.hpp.

## 9.32.3 Function Documentation

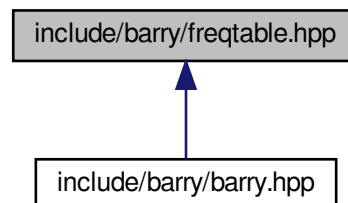
### 9.32.3.1 get\_last\_name()

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

Definition at line 142 of file phylo.hpp.

## 9.33 include/barry/freqtable.hpp File Reference

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

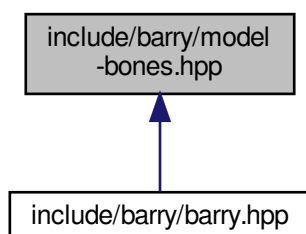


## Classes

- class `FreqTable< T >`  
*Frequency table of vectors.*

## 9.34 include/barry/model-bones.hpp File Reference

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



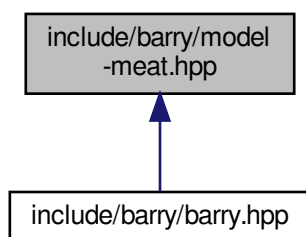
### Classes

- 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:*

## 9.35 include/barry/model-meat.hpp File Reference

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



### Macros

- #define [MODEL\\_TYPE\(\)](#)
- #define [MODEL\\_TEMPLATE\\_ARGS\(\)](#)
- #define [MODEL\\_TEMPLATE\(a, b\)](#) template [MODEL\\_TEMPLATE\\_ARGS\(\)](#) inline [a MODEL\\_TYPE\(\)](#)::b

## Functions

- double [update\\_normalizing\\_constant](#) (const 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\_<sub>default</sub>=false)
- [MODEL\\_TEMPLATE](#) (void, store\_psets)() noexcept
- [MODEL\\_TEMPLATE](#) (std::vector< double >, gen\_key)(const Array\_Type &Array\_)
- [MODEL\\_TEMPLATE](#) (void, add\_counter)(Counter< Array\_Type
- [MODEL\\_TEMPLATE](#) (void, set\_counters)(Counters< Array\_Type
- support\_fun [set\\_counters](#) (counters)
- [MODEL\\_TEMPLATE](#) (void, add\_hasher)(Hasher\_fun\_type< Array\_Type
- [MODEL\\_TEMPLATE](#) (void, add\_rule)(Rule< Array\_Type
- [MODEL\\_TEMPLATE](#) (void, set\_rules)(Rules< Array\_Type
- support\_fun [set\\_rules](#) (rules)
- [MODEL\\_TEMPLATE](#) (void, add\_rule\_dyn)(Rule< Array\_Type
- [MODEL\\_TEMPLATE](#) (void, set\_rules\_dyn)(Rules< Array\_Type
- support\_fun [set\\_rules\\_dyn](#) (rules\_dyn)
- [MODEL\\_TEMPLATE](#) (size\_t, add\_array)(const Array\_Type &Array\_
- if (transform\_model\_fun) = transform\_model\_fun(&tmp\_counts[0u], tmp\_counts.size())
- else stats\_target [push\\_back](#) (counter\_fun.count\_all())
- if (force\_new|(locator==keys2support.end()))
- arrays2support [push\\_back](#) (locator->second)
- return arrays2support [size](#) () - 1u
- [MODEL\\_TEMPLATE](#) (double, likelihood)(const std
- [MODEL\\_TEMPLATE](#) (double, likelihood\_total)(const std
- [MODEL\\_TEMPLATE](#) (double, get\_norm\_const)(const std
- [MODEL\\_TEMPLATE](#) (const std::vector< Array\_Type > \*, get\_pset)(const size\_t &i)
- [MODEL\\_TEMPLATE](#) (const std::vector< double > \*, get\_pset\_stats)(const size\_t &i)
- [MODEL\\_TEMPLATE](#) (void, print\_stats)(size\_t i) const
- [MODEL\\_TEMPLATE](#) (size\_t, [size](#))() const noexcept
- [MODEL\\_TEMPLATE](#) (size\_t, size\_unique)() const noexcept
- [MODEL\\_TEMPLATE](#) (size\_t, nterms)() const noexcept
- [MODEL\\_TEMPLATE](#) (size\_t, nrules)() const noexcept
- [MODEL\\_TEMPLATE](#) (size\_t, nrules\_dyn)() const noexcept
- [MODEL\\_TEMPLATE](#) (size\_t, support\_size)() const noexcept
- [MODEL\\_TEMPLATE](#) (std::vector< std::string >, colnames)() const
- [MODEL\\_TEMPLATE](#) (Array\_Type, sample)(const Array\_Type &Array\_
- if (locator==keys2support.end())
- std::uniform\_real\_distribution [urand](#) (0, 1)
- if ((probs.size() > 0u) &&(vec\_equal\_approx(params, params\_last[a])))
- std::vector< double > [temp\\_stats](#) (params.size())
- for (size\_t array=0u;array< [probs.size](#)();++array)
- [MODEL\\_TEMPLATE](#) (double, conditional\_prob)(const Array\_Type &Array\_
- A [insert\\_cell](#) (i, j, A.default\_val(), true, false)
- std::vector< double > [tmp\\_counts](#) (counters->size())
- return (1.0+std::exp(-vec\_inner\_prod< double > (&params[0u], &tmp\_counts[0u], params.size())))
- [MODEL\\_TEMPLATE](#) (const std::mt19937 \*, get\_engine)() const
- [MODEL\\_TEMPLATE](#) (std::vector< std::vector< double > > \*, get\_stats\_target)()
- [MODEL\\_TEMPLATE](#) (std::vector< std::vector< double > > \*, get\_stats\_support)()
- [MODEL\\_TEMPLATE](#) (std::vector< size\_t > \*, get\_arrays2support)()
- [MODEL\\_TEMPLATE](#) (std::vector< std::vector< Array\_Type > > \*, get\_pset\_arrays)()
- [MODEL\\_TEMPLATE](#) (std::vector< std::vector< double > > \*, get\_pset\_stats)()
- [MODEL\\_TEMPLATE](#) (std::vector< std::vector< double > > \*, get\_pset\_probs)()
- [MODEL\\_TEMPLATE](#) (void, set\_transform\_model)(std

## Variables

- [Data\\_Counter\\_Type](#) & [counter](#)
- [return](#)
- [Data\\_Counter\\_Type](#) [count\\_fun\\_](#)
- [Data\\_Counter\\_Type](#) [Counter\\_fun\\_type](#)< [Array\\_Type](#), [Data\\_Counter\\_Type](#) > [init\\_fun\\_](#)
- [Data\\_Counter\\_Type](#) [Counter\\_fun\\_type](#)< [Array\\_Type](#), [Data\\_Counter\\_Type](#) > [Data\\_Counter\\_Type](#) [data\\_](#)
- [Data\\_Counter\\_Type](#) \* [counters\\_](#)
- [Data\\_Counter\\_Type](#) [fun\\_](#)
- [Data\\_Rule\\_Type](#) & [rules](#)
- [Data\\_Rule\\_Type](#) \* [rules\\_](#)
- [this](#) [delete\\_rules](#) = [false](#)
- [Data\\_Rule\\_Dyn\\_Type](#) [rule\\_fun\\_](#)
- [this](#) [rules\\_dyn](#) = [rules\\_](#)
- [this](#) [delete\\_rules\\_dyn](#) = [false](#)
- [bool](#) [force\\_new](#)
- [std::vector](#)< [double](#) > [key](#) = [counters->gen\\_hash\(Array\\_\)](#)
- [MapVec\\_type](#)< [double](#), [size\\_t](#) >::const\_iterator [locator](#) = [keys2support.find\(key\)](#)
- [stats\\_support\\_n\\_arrays](#) [[locator->second](#)]
- [const](#) [std::vector](#)< [double](#) > & [params](#)
- [size\\_t](#) [i](#) = [locator->second](#)
- [size\\_t](#) [a](#) = [arrays2support\[i\]](#)
- [double](#) [r](#) = [urand\(\\*rengine\)](#)
- [double](#) [cumprob](#) = [0.0](#)
- [size\\_t](#) [k](#) = [params.size\(\)](#)
- [size\\_t](#) [j](#) = [0u](#)
- [std::vector](#)< [double](#) > & [probs](#) = [pset\\_probs\[a\]](#)
- [else](#)
- [const](#) [std::vector](#)< [double](#) > & [stats](#) = [pset\\_stats\[a\]](#)
- [int](#) [i\\_matches](#) = [-1](#)
- [return](#) [this](#) [pset\\_arrays](#) [[a](#)][[j](#)]
- [template](#) [Data\\_Counter\\_Type](#)
- [template](#) [Data\\_Rule\\_Type](#)

## 9.35.1 Macro Definition Documentation

### 9.35.1.1 MODEL\_TEMPLATE

```
#define MODEL_TEMPLATE(
    a,
    b ) template MODEL_TEMPLATE_ARGS() inline a MODEL_TYPE()::b
```

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

### 9.35.1.2 MODEL\_TEMPLATE\_ARGS

```
template MODEL_TEMPLATE_ARGS( )
```

**Value:**

```
<typename Array_Type, typename Data_Counter_Type,\
  typename Data_Rule_Type, typename Data_Rule_Dyn_Type>
```

Definition at line 120 of file model-meat.hpp.

### 9.35.1.3 MODEL\_TYPE

```
template Data_Rule_Dyn_Type * MODEL_TYPE( )
```

**Value:**

```
Model<Array_Type, Data_Counter_Type, Data_Rule_Type,\
  Data_Rule_Dyn_Type>
```

Definition at line 117 of file model-meat.hpp.

## 9.35.2 Function Documentation

### 9.35.2.1 for()

```
for ( )
```

Definition at line 1307 of file model-meat.hpp.

### 9.35.2.2 if() [1/4]

```
if (
    (probs.size() > 0u) && (vec_equal_approx(params, params_last[a])) )
```

Definition at line 1290 of file model-meat.hpp.

### 9.35.2.3 if() [2/4]

```
if (
    force_new| locator==keys2support.end() )
```

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

Definition at line 449 of file model-meat.hpp.

**9.35.2.4 if() [3/4]**

```
if (
    locator == keys2support.end() )
```

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

Definition at line 1192 of file model-meat.hpp.

**9.35.2.5 if() [4/4]**

```
if (
    transform_model_fun ) = transform_model_fun(&tmp_counts[0u], tmp_counts.size())
```

Definition at line 434 of file model-meat.hpp.

**9.35.2.6 insert\_cell()**

```
A insert_cell (
    i ,
    j ,
    A. default_val(),
    true ,
    false )
```

**9.35.2.7 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 59 of file model-meat.hpp.

**9.35.2.8 MODEL\_TEMPLATE() [1/33]**

```
MODEL_TEMPLATE (
    Array_Type ,
    sample ) const &
```



### 9.35.2.9 MODEL\_TEMPLATE() [2/33]

```
MODEL_TEMPLATE (
    const std::mt19937 * ,
    get_rengine ) const
```

Definition at line 1376 of file model-meat.hpp.

### 9.35.2.10 MODEL\_TEMPLATE() [3/33]

```
MODEL_TEMPLATE (
    const std::vector< Array_Type > * ,
    get_pset ) const &
```

Definition at line 918 of file model-meat.hpp.

### 9.35.2.11 MODEL\_TEMPLATE() [4/33]

```
MODEL_TEMPLATE (
    const std::vector< double > * ,
    get_pset_stats ) const &
```

Definition at line 930 of file model-meat.hpp.

### 9.35.2.12 MODEL\_TEMPLATE() [5/33]

```
MODEL_TEMPLATE (
    double ,
    conditional_prob ) const &
```

### 9.35.2.13 MODEL\_TEMPLATE() [6/33]

```
MODEL_TEMPLATE (
    double ,
    get_norm_const ) const
```

Definition at line 882 of file model-meat.hpp.

**9.35.2.14 MODEL\_TEMPLATE()** [7/33]

```
MODEL_TEMPLATE (
    double ,
    likelihood ) const
```

Definition at line 561 of file model-meat.hpp.

**9.35.2.15 MODEL\_TEMPLATE()** [8/33]

```
MODEL_TEMPLATE (
    double ,
    likelihood_total ) const
```

Definition at line 816 of file model-meat.hpp.

**9.35.2.16 MODEL\_TEMPLATE()** [9/33]

```
MODEL_TEMPLATE (
    size_t ,
    add_array ) const &
```

**9.35.2.17 MODEL\_TEMPLATE()** [10/33]

```
MODEL_TEMPLATE (
    size_t ,
    nrules ) const [noexcept]
```

Definition at line 1056 of file model-meat.hpp.

**9.35.2.18 MODEL\_TEMPLATE()** [11/33]

```
MODEL_TEMPLATE (
    size_t ,
    nrules_dyn ) const [noexcept]
```

Definition at line 1063 of file model-meat.hpp.

**9.35.2.19 MODEL\_TEMPLATE()** [12/33]

```
MODEL_TEMPLATE (
    size_t ,
    nterms ) const [noexcept]
```

Definition at line 1046 of file model-meat.hpp.

**9.35.2.20 MODEL\_TEMPLATE()** [13/33]

```
MODEL_TEMPLATE (
    size_t ,
    size ) const [noexcept]
```

Definition at line 1031 of file model-meat.hpp.

**9.35.2.21 MODEL\_TEMPLATE()** [14/33]

```
MODEL_TEMPLATE (
    size_t ,
    size_unique ) const [noexcept]
```

Definition at line 1038 of file model-meat.hpp.

**9.35.2.22 MODEL\_TEMPLATE()** [15/33]

```
MODEL_TEMPLATE (
    size_t ,
    support_size ) const [noexcept]
```

Definition at line 1070 of file model-meat.hpp.

**9.35.2.23 MODEL\_TEMPLATE()** [16/33]

```
MODEL_TEMPLATE (
    std::vector< double > ,
    gen_key ) const &
```

Definition at line 304 of file model-meat.hpp.

**9.35.2.24 MODEL\_TEMPLATE() [17/33]**

```
MODEL_TEMPLATE (
    std::vector< size_t > * ,
    get_arrays2support )
```

Definition at line 1411 of file model-meat.hpp.

**9.35.2.25 MODEL\_TEMPLATE() [18/33]**

```
MODEL_TEMPLATE (
    std::vector< std::string > ,
    colnames ) const
```

Definition at line 1082 of file model-meat.hpp.

**9.35.2.26 MODEL\_TEMPLATE() [19/33]**

```
MODEL_TEMPLATE (
    std::vector< std::vector< Array_Type > > * ,
    get_pset_arrays )
```

Definition at line 1416 of file model-meat.hpp.

**9.35.2.27 MODEL\_TEMPLATE() [20/33]**

```
MODEL_TEMPLATE (
    std::vector< std::vector< double > > * ,
    get_pset_probs )
```

Definition at line 1424 of file model-meat.hpp.

**9.35.2.28 MODEL\_TEMPLATE() [21/33]**

```
MODEL_TEMPLATE (
    std::vector< std::vector< double > > * ,
    get_pset_stats )
```

Definition at line 1420 of file model-meat.hpp.

**9.35.2.29 MODEL\_TEMPLATE()** [22/33]

```
MODEL_TEMPLATE (
    std::vector< std::vector< double > > * ,
    get_stats_support )
```

Definition at line 1406 of file model-meat.hpp.

**9.35.2.30 MODEL\_TEMPLATE()** [23/33]

```
MODEL_TEMPLATE (
    std::vector< std::vector< double > > * ,
    get_stats_target )
```

Definition at line 1401 of file model-meat.hpp.

**9.35.2.31 MODEL\_TEMPLATE()** [24/33]

```
MODEL_TEMPLATE (
    void ,
    add_counter )
```

**9.35.2.32 MODEL\_TEMPLATE()** [25/33]

```
MODEL_TEMPLATE (
    void ,
    add_hasher )
```

**9.35.2.33 MODEL\_TEMPLATE()** [26/33]

```
MODEL_TEMPLATE (
    void ,
    add_rule )
```

**9.35.2.34 MODEL\_TEMPLATE()** [27/33]

```
MODEL_TEMPLATE (
    void ,
    add_rule_dyn )
```

**9.35.2.35 MODEL\_TEMPLATE()** [28/33]

```
MODEL_TEMPLATE (
    void ,
    print_stats ) const
```

Definition at line 941 of file model-meat.hpp.

**9.35.2.36 MODEL\_TEMPLATE()** [29/33]

```
MODEL_TEMPLATE (
    void ,
    set_counters )
```

**9.35.2.37 MODEL\_TEMPLATE()** [30/33]

```
MODEL_TEMPLATE (
    void ,
    set_rules )
```

**9.35.2.38 MODEL\_TEMPLATE()** [31/33]

```
MODEL_TEMPLATE (
    void ,
    set_rules_dyn )
```

**9.35.2.39 MODEL\_TEMPLATE()** [32/33]

```
MODEL_TEMPLATE (
    void ,
    set_transform_model )
```

Definition at line 1428 of file model-meat.hpp.

**9.35.2.40 MODEL\_TEMPLATE()** [33/33]

```
MODEL_TEMPLATE (
    void ,
    store_psets ) [noexcept]
```

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

**9.35.2.41 push\_back() [1/2]**

```
else stats_target push_back (
    counter_fun. count_all() )
```

**9.35.2.42 push\_back() [2/2]**

```
arrays2support push_back (
    locator-> second )
```

**9.35.2.43 return()**

```
return (
    1.0+ std::exp-vec_inner_prod< double > (&params[0u], &tmp_counts[0u], params.↵
size()) )
```

**9.35.2.44 set\_counters()**

```
counter_fun set_counters (
    counters )
```

**9.35.2.45 set\_rules()**

```
support_fun set_rules (
    rules )
```

**9.35.2.46 set\_rules\_dyn()**

```
support_fun set_rules_dyn (
    rules_dyn )
```

**9.35.2.47 size()**

```
return arrays2support size ( )
```

#### 9.35.2.48 temp\_stats()

```
std::vector< double > temp_stats (
    params. size() )
```

#### 9.35.2.49 tmp\_counts()

```
std::vector< double > tmp_counts (
    counters-> size() )
```

#### 9.35.2.50 update\_normalizing\_constant()

```
double update_normalizing_constant (
    const double * params,
    const double * support,
    size_t k,
    size_t n ) [inline]
```

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

#### 9.35.2.51 urand()

```
std::uniform_real_distribution urand (
    0 ,
    1 )
```

### 9.35.3 Variable Documentation

#### 9.35.3.1 a

```
size_t a = arrays2support[i]
```

Definition at line 1278 of file model-meat.hpp.



### 9.35.3.2 count\_fun\_

`Data_Counter_Type` count\_fun\_

Definition at line 319 of file model-meat.hpp.

### 9.35.3.3 counter

`Data_Counter_Type&` counter

**Initial value:**

```
{  
    counters->add_counter(counter, Data_Counter_Type())
```

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

### 9.35.3.4 counters\_

`Data_Counter_Type*` counters\_

**Initial value:**

```
{  
    if (delete_counters) {  
        delete counters;  
        delete_counters = false;  
    }  
  
    this->counters = counters_
```

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

### 9.35.3.5 cumprob

`double` cumprob = 0.0

Definition at line 1283 of file model-meat.hpp.

### 9.35.3.6 data\_

`Data_Rule_Dyn_Type` Data\_Rule\_Dyn\_Type data\_

**Initial value:**

```
{  
  
    counters->add_counter(  
        count_fun_,  
        init_fun_,  
        data_  
    )
```

Definition at line 321 of file model-meat.hpp.

### 9.35.3.7 Data\_Counter\_Type

```
template Data_Counter_Type
```

Definition at line 1396 of file model-meat.hpp.

### 9.35.3.8 Data\_Rule\_Type

```
template Data_Rule_Type
```

Definition at line 1396 of file model-meat.hpp.

### 9.35.3.9 delete\_rules

```
this delete_rules = false
```

Definition at line 378 of file model-meat.hpp.

### 9.35.3.10 delete\_rules\_dyn

```
this delete_rules_dyn = false
```

Definition at line 417 of file model-meat.hpp.

### 9.35.3.11 else

```
else (  
    void )
```

#### Initial value:

```
{  
    probs.resize(pset_arrays[a].size())
```

Definition at line 1300 of file model-meat.hpp.

### 9.35.3.12 force\_new

```
bool force_new
```

#### Initial value:

```
{  
  
    counter_fun.reset_array(&Array_)
```

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

### 9.35.3.13 fun\_

```
Data_Counter_Type fun_
```

#### Initial value:

```
{  
    counters->add_hash(fun_)
```

Definition at line 352 of file model-meat.hpp.

### 9.35.3.14 i

```
const std::vector< double > size_t i = locator->second
```

Definition at line 1186 of file model-meat.hpp.

### 9.35.3.15 i\_matches

```
int i_matches = -1
```

Definition at line 1306 of file model-meat.hpp.

### 9.35.3.16 init\_fun\_

```
Data_Counter_Type Counter_fun_type<Array_Type,Data_Counter_Type> init_fun_
```

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

### 9.35.3.17 j

```
const std::vector< double > size_t size_t j = 0u
```

Definition at line 1288 of file model-meat.hpp.

### 9.35.3.18 k

```
size_t k = params.size()
```

Definition at line 1285 of file model-meat.hpp.

### 9.35.3.19 key

```
std::vector< double > key = counters->gen_hash(Array_)
```

Definition at line 447 of file model-meat.hpp.

### 9.35.3.20 locator

```
MapVec_type< double, size_t >::const_iterator locator = keys2support.find(key)
```

Definition at line 448 of file model-meat.hpp.

### 9.35.3.21 params

```
const std::vector< double > & params
```

#### Initial value:

```
{  
    if (!this->with_pset)  
        throw std::logic_error("Sampling is only available when store_pset() is active.")
```

Definition at line 1179 of file model-meat.hpp.

### 9.35.3.22 probs

```
std::vector< double >& probs = pset_probs[a]
```

Definition at line 1289 of file model-meat.hpp.

### 9.35.3.23 pset\_arrays

```
return this pset_arrays[a][j]
```

Definition at line 1340 of file model-meat.hpp.

### 9.35.3.24 r

```
double r = urand(*rengine)
```

Definition at line 1282 of file model-meat.hpp.

### 9.35.3.25 return

```
return
```

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

### 9.35.3.26 rule\_fun\_

```
Data_Rule_Dyn_Type rule_fun_
```

Definition at line 396 of file model-meat.hpp.

### 9.35.3.27 rules

```
this rules
```

#### Initial value:

```
{  
    rules->add_rule(rules, Data_Rule_Type())
```

Definition at line 362 of file model-meat.hpp.

### 9.35.3.28 rules\_

```
Data_Rule_Dyn_Type * rules_
```

#### Initial value:

```
{  
    if (delete_rules)  
        delete rules
```

Definition at line 371 of file model-meat.hpp.

### 9.35.3.29 rules\_dyn

```
this rules_dyn = rules_
```

Definition at line 416 of file model-meat.hpp.

### 9.35.3.30 stats

```
const std::vector< double >& stats = pset_stats[a]
```

Definition at line 1304 of file model-meat.hpp.

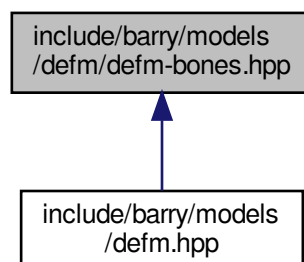
### 9.35.3.31 stats\_support\_n\_arrays

```
stats_support_n_arrays[locator->second]
```

Definition at line 552 of file model-meat.hpp.

## 9.36 include/barry/models/defm/defm-bones.hpp File Reference

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

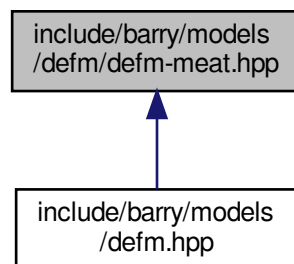


## Classes

- class [DEFM](#)

## 9.37 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 defmcounters::DEFMArray &Array\_, defmcounters::DEFMCounterData *data)`

### 9.37.1 Macro Definition Documentation

#### 9.37.1.1 DEFM\_LOOP\_ARRAYS

```
#define DEFM_LOOP_ARRAYS(
    a ) for (size_t a = 0u; a < (nobs_i - M_order); ++a)
```

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

### 9.37.1.2 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 30 of file defm-meat.hpp.

## 9.37.2 Function Documentation

### 9.37.2.1 keygen\_defm()

```
std::vector< double > keygen_defm (  
    const defmcounters::DEFMArray & Array_,  
    defmcounters::DEFMCounterData * data ) [inline]
```

Definition at line 4 of file defm-meat.hpp.

## 9.38 include/barry/models/geese.hpp File Reference

```
#include "geese/geese-node-bones.hpp"  
#include "geese/geese-bones.hpp"  
#include "geese/geese-meat.hpp"  
#include "geese/geese-meat-constructors.hpp"  
#include "geese/geese-meat-likelihood.hpp"  
#include "geese/geese-meat-likelihood_exhaust.hpp"  
#include "geese/geese-meat-simulate.hpp"  
#include "geese/geese-meat-predict.hpp"  
#include "geese/geese-meat-predict_exhaust.hpp"  
#include "geese/geese-meat-predict_sim.hpp"  
#include "geese/flock-bones.hpp"  
#include "geese/flock-meat.hpp"
```

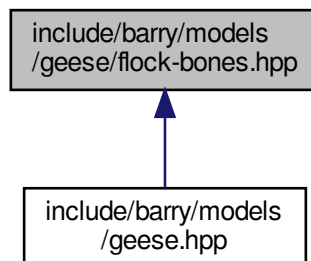
Include dependency graph for geese.hpp:





## 9.39 include/barry/models/geese/flock-bones.hpp File Reference

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



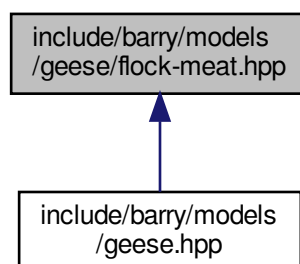
### Classes

- class [Flock](#)

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

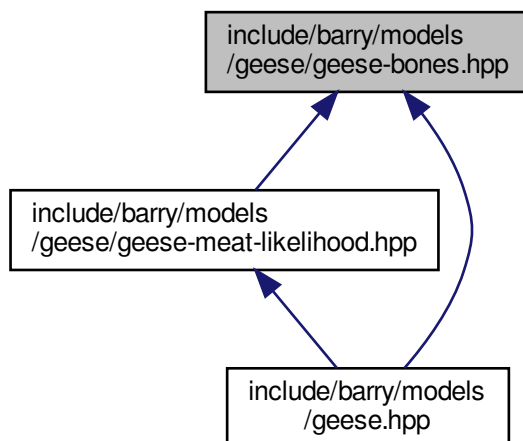
## 9.40 include/barry/models/geese/flock-meat.hpp File Reference

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



## 9.41 include/barry/models/geese/geese-bones.hpp File Reference

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



### Classes

- class [Geese](#)  
*Annotated Phylo [Model](#).*

### Macros

- #define [INITIALIZED\(\)](#)

### Functions

- template<typename Ta , typename Tb >  
std::vector< Ta > [vector\\_caster](#) (const std::vector< Tb > &x)
- [RULE\\_FUNCTION](#) (rule\_empty\_free)
- std::vector< double > [keygen\\_full](#) (const phylocounters::PhyloArray &array, const phylocounters::PhyloCounterData \*d)
- bool [vec\\_diff](#) (const std::vector< size\_t > &s, const std::vector< size\_t > &a)

#### 9.41.1 Macro Definition Documentation

### 9.41.1.1 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.

## 9.41.2 Function Documentation

### 9.41.2.1 keygen\_full()

```
std::vector< double > keygen_full (   
    const phylocounters::PhyloArray & array,   
    const phylocounters::PhyloCounterData * d ) [inline]
```

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

### 9.41.2.2 RULE\_FUNCTION()

```
RULE_FUNCTION (   
    rule_empty_free )
```

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

### 9.41.2.3 vec\_diff()

```
bool vec_diff (   
    const std::vector< size_t > & s,   
    const std::vector< size_t > & a ) [inline]
```

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

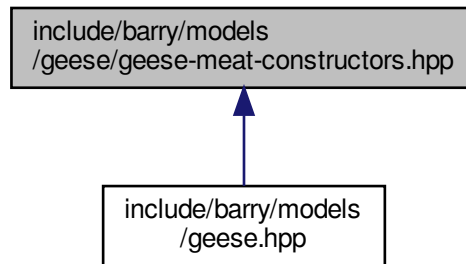
### 9.41.2.4 vector\_caster()

```
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.

## 9.42 include/barry/models/geese/geese-meat-constructors.hpp File Reference

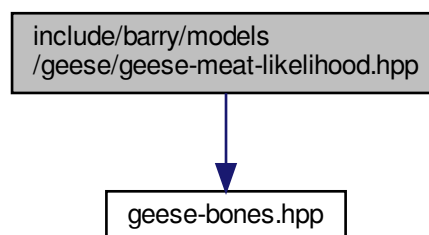
This graph shows which files directly or indirectly include this file:



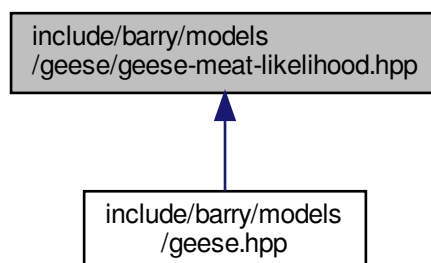
## 9.43 include/barry/models/geese/geese-meat-likelihood.hpp File Reference

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

Include dependency graph for geese-meat-likelihood.hpp:

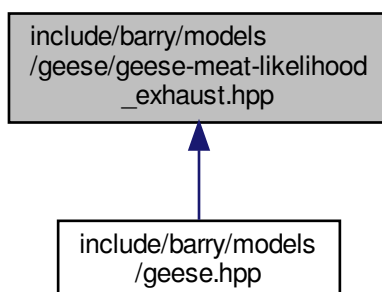


This graph shows which files directly or indirectly include this file:



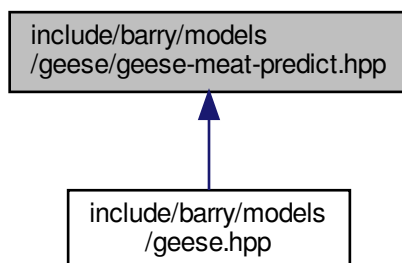
## 9.44 include/barry/models/geese/geese-meat-likelihood\_exhaust.hpp File Reference

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



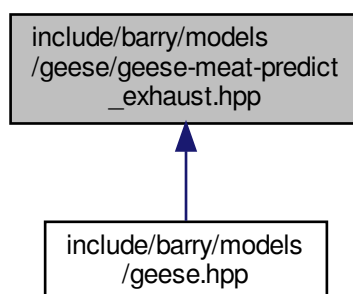
## 9.45 include/barry/models/geese/geese-meat-predict.hpp File Reference

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



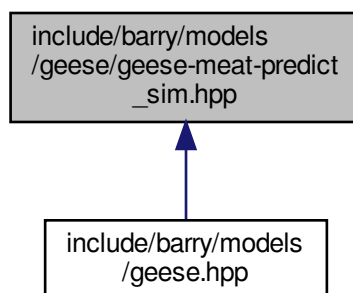
## 9.46 include/barry/models/geese/geese-meat-predict\_exhaust.hpp File Reference

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



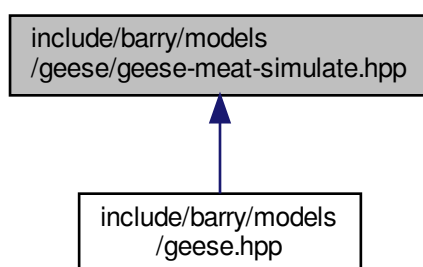
## 9.47 include/barry/models/geese/geese-meat-predict\_sim.hpp File Reference

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



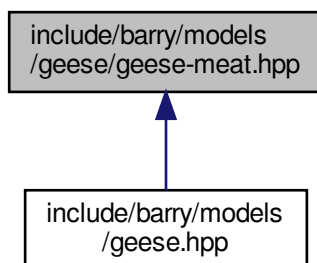
## 9.48 include/barry/models/geese/geese-meat-simulate.hpp File Reference

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



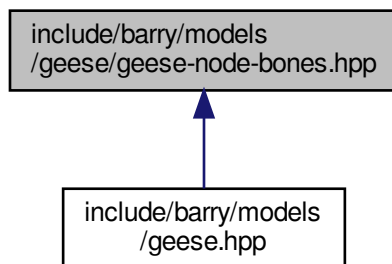
## 9.49 include/barry/models/geese/geese-meat.hpp File Reference

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



## 9.50 include/barry/models/geese/geese-node-bones.hpp File Reference

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



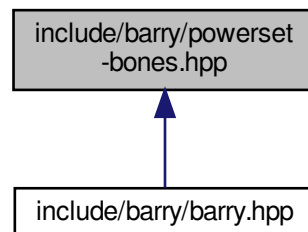
### Classes

- class [Node](#)  
*A single node for the model.*



## 9.51 include/barry/powerset-bones.hpp File Reference

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

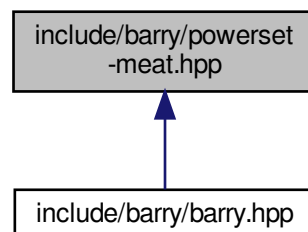


### Classes

- class [PowerSet< Array\\_Type, Data\\_Rule\\_Type >](#)  
*Powerset of a binary array.*

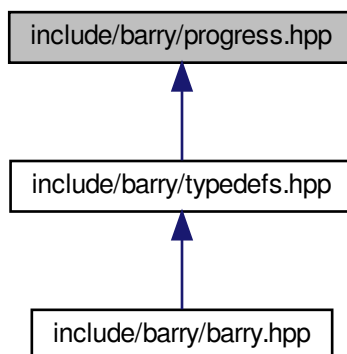
## 9.52 include/barry/powerset-meat.hpp File Reference

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



## 9.53 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`

### 9.53.1 Macro Definition Documentation

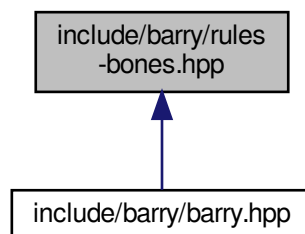
#### 9.53.1.1 BARRY\_PROGRESS\_BAR\_WIDTH

```
#define BARRY_PROGRESS_BAR_WIDTH 80
```

Definition at line 5 of file progress.hpp.

## 9.54 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)`

#### 9.54.1 Function Documentation

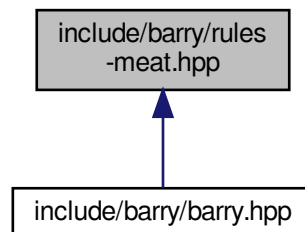
##### 9.54.1.1 rule\_fun\_default()

```
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.

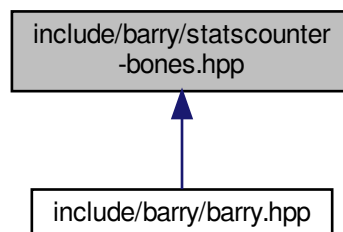
## 9.55 include/barry/rules-meat.hpp File Reference

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



## 9.56 include/barry/statscounter-bones.hpp File Reference

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

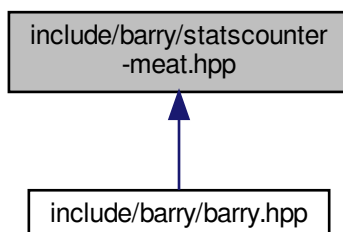


## Classes

- class [StatsCounter< Array\\_Type, Data\\_Type >](#)  
*Count stats for a single Array.*

## 9.57 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`

## 9.57.1 Macro Definition Documentation

### 9.57.1.1 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.

### 9.57.1.2 STATSCOUNTER\_TEMPLATE\_ARGS

```
template STATSCOUNTER_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

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

### 9.57.1.3 STATSCOUNTER\_TYPE

```
template Data_Type * STATSCOUNTER_TYPE( ) StatsCounter<Array_Type, Data_Type>
```

Definition at line 4 of file statscounter-meat.hpp.

## 9.57.2 Function Documentation

### 9.57.2.1 clear()

```
EmptyArray clear ( )
```

### 9.57.2.2 for()

```
for (  
    size_t n = 0u; n < counters->size(); ++n )
```

### 9.57.2.3 `resize()`

```
current_stats resize (
    counters-> size(),
    0. 0 )
```

### 9.57.2.4 `STATSCOUNTER_TEMPLATE()` [1/9]

```
STATSCOUNTER_TEMPLATE (
    StatsCounter ) const
```

### 9.57.2.5 `STATSCOUNTER_TEMPLATE()` [2/9]

```
STATSCOUNTER_TEMPLATE (
    ~ StatsCounter )
```

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

### 9.57.2.6 `STATSCOUNTER_TEMPLATE()` [3/9]

```
STATSCOUNTER_TEMPLATE (
    std::vector< std::string > ,
    get_descriptions ) const
```

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

### 9.57.2.7 `STATSCOUNTER_TEMPLATE()` [4/9]

```
STATSCOUNTER_TEMPLATE (
    std::vector< std::string > ,
    get_names ) const
```

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

### 9.57.2.8 `STATSCOUNTER_TEMPLATE()` [5/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    add_counter )
```

#### 9.57.2.9 STATSCOUNTER\_TEMPLATE() [6/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    count_current )
```

#### 9.57.2.10 STATSCOUNTER\_TEMPLATE() [7/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    count_init )
```

#### 9.57.2.11 STATSCOUNTER\_TEMPLATE() [8/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    reset_array ) const
```

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

#### 9.57.2.12 STATSCOUNTER\_TEMPLATE() [9/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    set_counters )
```

### 9.57.3 Variable Documentation

#### 9.57.3.1 counter

Data\_Type& counter

##### Initial value:

```
{
    Array      = counter.Array
```

Definition at line 12 of file statscounter-meat.hpp.



### 9.57.3.2 counter\_deleted

```
counter_deleted = false
```

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

### 9.57.3.3 counters

```
counters = new Counters<Array_Type,Data_Type> ((*counter.counters))
```

Definition at line 22 of file statscounter-meat.hpp.

### 9.57.3.4 counters\_

```
Data_Type* counters_
```

#### Initial value:

```
{  
  
    if (!counter_deleted)  
        delete counters
```

Definition at line 53 of file statscounter-meat.hpp.

### 9.57.3.5 current\_stats

```
current_stats = counter.current_stats
```

Definition at line 19 of file statscounter-meat.hpp.

### 9.57.3.6 EmptyArray

```
EmptyArray = *Array
```

Definition at line 17 of file statscounter-meat.hpp.

### 9.57.3.7 f\_

Data\_Rule\_Dyn\_Type f\_

#### Initial value:

```
{  
    counters->add_counter(f_)
```

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

### 9.57.3.8 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.

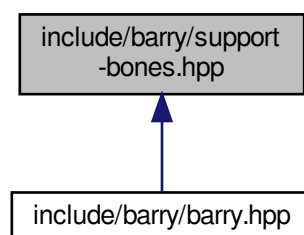
### 9.57.3.9 return

return

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

## 9.58 include/barry/support-bones.hpp File Reference

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

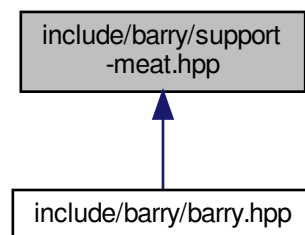


## Classes

- class [Support](#)< [Array\\_Type](#), [Data\\_Counter\\_Type](#), [Data\\_Rule\\_Type](#), [Data\\_Rule\\_Dyn\\_Type](#) >  
*Compute the support of sufficient statistics.*

## 9.59 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`
- `#define SUPPORT_TEMPLATE_ARGS()`
- `#define SUPPORT_TYPE()`
- `#define SUPPORT_TEMPLATE(a, b)`

## Functions

- `SUPPORT_TEMPLATE` (void, init\_support)(std
- `SUPPORT_TEMPLATE` (void, reset\_array)()
- `SUPPORT_TEMPLATE` (void, reset\_array)(const [Array\\_Type](#) &[Array\\_](#))
- `SUPPORT_TEMPLATE` (void, [calc\\_backend\\_sparse](#))(size\_t pos
- [calc\\_backend\\_sparse](#) (pos+1u, [array\\_bank](#), [stats\\_bank](#))
- [EmptyArray](#) insert\_cell ([coord\\_i](#), [coord\\_j](#), [EmptyArray.default\\_val\(\).value](#), false, false)
- for (size\_t n=0u;n< n\_counters;++n)
- if ([rules\\_dyn->size\(\)](#) > 0u)
- if ([array\\_bank != nullptr](#)) [array\\_bank](#) -> push\_back([EmptyArray](#))
- [EmptyArray](#) rm\_cell ([coord\\_i](#), [coord\\_j](#), false, false)
- if ([change\\_stats\\_different](#) > 0u)
- `SUPPORT_TEMPLATE` (void, [calc\\_backend\\_dense](#))(size\_t pos
- [calc\\_backend\\_dense](#) (pos+1u, [array\\_bank](#), [stats\\_bank](#))
- [EmptyArray](#) insert\_cell ([coord\\_i](#), [coord\\_j](#), 1, false, false)
- `SUPPORT_TEMPLATE` (void, calc)(std
- `SUPPORT_TEMPLATE` (void, add\_counter)([Counter](#)< [Array\\_Type](#)
- `SUPPORT_TEMPLATE` (void, set\_counters)([Counters](#)< [Array\\_Type](#)

- `SUPPORT_TEMPLATE` (void, add\_rule)(Rule< Array\_Type
- `SUPPORT_TEMPLATE` (void, set\_rules)(Rules< Array\_Type
- `SUPPORT_TEMPLATE` (void, add\_rule\_dyn)(Rule< Array\_Type
- `SUPPORT_TEMPLATE` (void, set\_rules\_dyn)(Rules< Array\_Type
- `SUPPORT_TEMPLATE` (bool, eval\_rules\_dyn)(const std
- `SUPPORT_TEMPLATE` (std::vector< double >, get\_counts)() const
- `SUPPORT_TEMPLATE` (std::vector< double > \*, get\_current\_stats)()
- `SUPPORT_TEMPLATE` (void, print)() const
- `SUPPORT_TEMPLATE` (const FreqTable< double > &, get\_data)() const

## Variables

- std::vector< Array\_Type > \* `array_bank`
- std::vector< Array\_Type > std::vector< double > \* `stats_bank`
- const size\_t & `coord_i` = coordinates\_free[pos \* 2u]
- const size\_t & `coord_j` = coordinates\_free[pos \* 2u + 1u]
- double `tmp_chng`
- size\_t `change_stats_different` = hashes\_initialized[pos] ? 0u : 1u
- else
- & `hashes` [pos]
- return
- Data\_Counter\_Type f\_
- Data\_Counter\_Type \* `counters_`
- delete\_counters = false
- counters = counters\_
- Data\_Rule\_Type \* `rules_`
- delete\_rules = false
- rules = rules\_
- delete\_rules\_dyn = false
- rules\_dyn = rules\_

## 9.59.1 Macro Definition Documentation

### 9.59.1.1 BARRY\_SUPPORT\_MEAT\_HPP

```
#define BARRY_SUPPORT_MEAT_HPP 1
```

Definition at line 2 of file support-meat.hpp.

### 9.59.1.2 SUPPORT\_TEMPLATE

```
#define SUPPORT_TEMPLATE(
    a,
    b )
```

#### Value:

```
template SUPPORT_TEMPLATE_ARGS() \
inline a SUPPORT_TYPE()::b
```

Definition at line 10 of file support-meat.hpp.

### 9.59.1.3 SUPPORT\_TEMPLATE\_ARGS

```
template SUPPORT_TEMPLATE_ARGS( )
```

**Value:**

```
<typename Array_Type, typename \  
Data_Counter_Type, typename Data_Rule_Type, typename Data_Rule_Dyn_Type>
```

Definition at line 4 of file support-meat.hpp.

### 9.59.1.4 SUPPORT\_TYPE

```
template Data_Rule_Dyn_Type * SUPPORT_TYPE( )
```

**Value:**

```
Support<Array_Type, Data_Counter_Type, Data_Rule_Type, \  
Data_Rule_Dyn_Type>
```

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

## 9.59.2 Function Documentation

### 9.59.2.1 calc\_backend\_dense()

```
calc_backend_dense (   
    pos+ 1u,   
    array_bank ,   
    stats_bank )
```

### 9.59.2.2 calc\_backend\_sparse()

```
calc_backend_sparse (   
    pos+ 1u,   
    array_bank ,   
    stats_bank )
```

### 9.59.2.3 for()

```
for ( )
```

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

**9.59.2.4 if() [1/3]**

```
if (
    array_bank !    = nullptr ) -> push_back(EmptyArray)
```

**9.59.2.5 if() [2/3]**

```
if (
    change_stats_different ,
    Ou )
```

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

**9.59.2.6 if() [3/3]**

```
if (
    rules_dyn-> size(),
    Ou )
```

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

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

```
EmptyArray insert_cell (
    coord_i ,
    coord_j ,
    1 ,
    false ,
    false )
```

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

```
EmptyArray insert_cell (
    coord_i ,
    coord_j ,
    EmptyArray.default_val(). value,
    false ,
    false )
```

### 9.59.2.9 rm\_cell()

```
EmptyArray rm_cell (
    coord_i ,
    coord_j ,
    false ,
    false )
```

### 9.59.2.10 SUPPORT\_TEMPLATE() [1/17]

```
SUPPORT_TEMPLATE (
    bool ,
    eval_rules_dyn ) const
```

Definition at line 489 of file support-meat.hpp.

### 9.59.2.11 SUPPORT\_TEMPLATE() [2/17]

```
SUPPORT_TEMPLATE (
    const FreqTable< double > & ,
    get_data ) const
```

Definition at line 558 of file support-meat.hpp.

### 9.59.2.12 SUPPORT\_TEMPLATE() [3/17]

```
SUPPORT_TEMPLATE (
    std::vector< double > * ,
    get_current_stats )
```

Definition at line 543 of file support-meat.hpp.

### 9.59.2.13 SUPPORT\_TEMPLATE() [4/17]

```
SUPPORT_TEMPLATE (
    std::vector< double > ,
    get_counts ) const
```

Definition at line 531 of file support-meat.hpp.

**9.59.2.14 SUPPORT\_TEMPLATE()** [5/17]

```
SUPPORT_TEMPLATE (
    void ,
    add_counter )
```

**9.59.2.15 SUPPORT\_TEMPLATE()** [6/17]

```
SUPPORT_TEMPLATE (
    void ,
    add_rule )
```

**9.59.2.16 SUPPORT\_TEMPLATE()** [7/17]

```
SUPPORT_TEMPLATE (
    void ,
    add_rule_dyn )
```

**9.59.2.17 SUPPORT\_TEMPLATE()** [8/17]

```
SUPPORT_TEMPLATE (
    void ,
    calc )
```

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

**9.59.2.18 SUPPORT\_TEMPLATE()** [9/17]

```
SUPPORT_TEMPLATE (
    void ,
    calc_backend_dense )
```

**9.59.2.19 SUPPORT\_TEMPLATE()** [10/17]

```
SUPPORT_TEMPLATE (
    void ,
    calc_backend_sparse )
```



**9.59.2.20 SUPPORT\_TEMPLATE()** [11/17]

```
SUPPORT_TEMPLATE (
    void ,
    init_support )
```

Definition at line 13 of file support-meat.hpp.

**9.59.2.21 SUPPORT\_TEMPLATE()** [12/17]

```
SUPPORT_TEMPLATE (
    void ,
    print ) const
```

Definition at line 547 of file support-meat.hpp.

**9.59.2.22 SUPPORT\_TEMPLATE()** [13/17]

```
SUPPORT_TEMPLATE (
    void ,
    reset_array )
```

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

**9.59.2.23 SUPPORT\_TEMPLATE()** [14/17]

```
SUPPORT_TEMPLATE (
    void ,
    reset_array ) const &
```

Definition at line 120 of file support-meat.hpp.

**9.59.2.24 SUPPORT\_TEMPLATE()** [15/17]

```
SUPPORT_TEMPLATE (
    void ,
    set_counters )
```

#### 9.59.2.25 SUPPORT\_TEMPLATE() [16/17]

```
SUPPORT_TEMPLATE (
    void ,
    set_rules )
```

#### 9.59.2.26 SUPPORT\_TEMPLATE() [17/17]

```
SUPPORT_TEMPLATE (
    void ,
    set_rules_dyn )
```

### 9.59.3 Variable Documentation

#### 9.59.3.1 array\_bank

```
std::vector< Array_Type > * array_bank
```

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

#### 9.59.3.2 change\_stats\_different

```
size_t change_stats_different = hashes_initialized[pos] ? 0u : 1u
```

Definition at line 158 of file support-meat.hpp.

#### 9.59.3.3 coord\_i

```
const size_t & coord_i = coordinates_free[pos * 2u]
```

Definition at line 144 of file support-meat.hpp.

#### 9.59.3.4 coord\_j

```
const size_t & coord_j = coordinates_free[pos * 2u + 1u]
```

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

### 9.59.3.5 counters

```
counters = counters_
```

Definition at line 417 of file support-meat.hpp.

### 9.59.3.6 counters\_

```
Data_Counter_Type* counters_
```

**Initial value:**

```
{  
  
    if (delete_counters)  
        delete counters
```

Definition at line 410 of file support-meat.hpp.

### 9.59.3.7 delete\_counters

```
delete_counters = false
```

Definition at line 416 of file support-meat.hpp.

### 9.59.3.8 delete\_rules

```
delete_rules = false
```

Definition at line 450 of file support-meat.hpp.

### 9.59.3.9 delete\_rules\_dyn

```
delete_rules_dyn = false
```

Definition at line 482 of file support-meat.hpp.

#### 9.59.3.10 else

```
else (  
    void )
```

##### Initial value:

```
{  
    if (change_stats_different > 0u)  
        hashes[pos] = data.add(current_stats, nullptr)
```

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

#### 9.59.3.11 f\_

```
Data_Rule_Dyn_Type f_
```

##### Initial value:

```
{  
    counters->add_counter(f_)
```

Definition at line 401 of file support-meat.hpp.

#### 9.59.3.12 hashes

```
& hashes
```

Definition at line 217 of file support-meat.hpp.

#### 9.59.3.13 return

```
return
```

Definition at line 249 of file support-meat.hpp.

#### 9.59.3.14 rules

```
rules = rules_
```

Definition at line 451 of file support-meat.hpp.

### 9.59.3.15 rules\_

Data\_Rule\_Dyn\_Type\* rules\_

#### Initial value:

```
{  
  
    if (delete_rules)  
        delete rules
```

Definition at line 444 of file support-meat.hpp.

### 9.59.3.16 rules\_dyn

rules\_dyn = rules\_

Definition at line 483 of file support-meat.hpp.

### 9.59.3.17 stats\_bank

std::vector< Array\_Type > std::vector< double > \* stats\_bank

#### Initial value:

```
{  
  
    if (pos >= coordiantes_n_free)  
        return
```

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

### 9.59.3.18 tmp\_chng

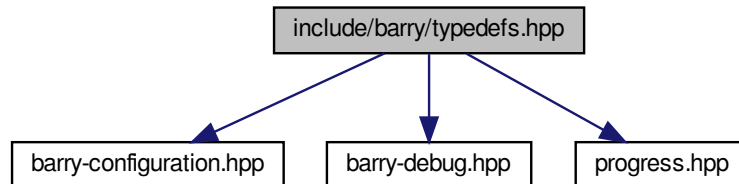
double tmp\_chng

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

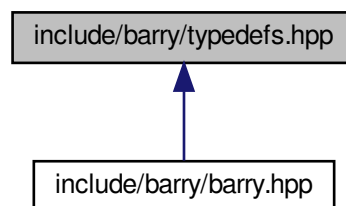
## 9.60 include/barry/typedefs.hpp File Reference

```
#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

## 9.60.1 Typedef Documentation

### 9.60.1.1 Col\_type

```
template<typename Cell_Type >
using Col_type = Map< size_t, Cell<Cell_Type>* >
```

Definition at line 70 of file typedefs.hpp.

### 9.60.1.2 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.

### 9.60.1.3 Counts\_type

```
typedef std::vector< std::pair< std::vector<double>, size_t > > Counts_type
```

Definition at line 51 of file typedefs.hpp.

### 9.60.1.4 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.

### 9.60.1.5 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.

### 9.60.1.6 Row\_type

```
template<typename Cell_Type >
using Row_type = Map< size_t, Cell<Cell_Type> >
```

Definition at line 67 of file typedefs.hpp.

### 9.60.1.7 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.

## 9.60.2 Function Documentation

### 9.60.2.1 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`.

9.60.2.2 `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`.

9.60.2.3 `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 228 of file `typedefs.hpp`.

#### 9.60.2.4 `vec_inner_prod()` [1/2]

```
template<>
double vec_inner_prod (
    const double * a,
    const double * b,
    size_t n ) [inline]
```

Definition at line 274 of file typedefs.hpp.

#### 9.60.2.5 `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 251 of file typedefs.hpp.

## 9.61 README.md File Reference



# Index

- ~BArray
  - BArray< Cell\_Type, Data\_Type >, [47](#)
- ~BArrayCell
  - BArrayCell< Cell\_Type, Data\_Type >, [58](#)
- ~BArrayCell\_const
  - BArrayCell\_const< Cell\_Type, Data\_Type >, [60](#)
- ~BArrayDense
  - BArrayDense< Cell\_Type, Data\_Type >, [66](#)
- ~BArrayDenseCell
  - BArrayDenseCell< Cell\_Type, Data\_Type >, [79](#)
- ~BArrayRow
  - BArrayRow< Cell\_Type, Data\_Type >, [92](#)
- ~BArrayRow\_const
  - BArrayRow\_const< Cell\_Type, Data\_Type >, [94](#)
- ~BArrayVector
  - BArrayVector< Cell\_Type, Data\_Type >, [97](#)
- ~BArrayVector\_const
  - BArrayVector\_const< Cell\_Type, Data\_Type >, [100](#)
- ~Cell
  - Cell< Cell\_Type >, [104](#)
- ~ConstBArrayRowIter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [109](#)
- ~Counter
  - Counter< Array\_Type, Data\_Type >, [112](#)
- ~Counters
  - Counters< Array\_Type, Data\_Type >, [117](#)
- ~DEFMCounterData
  - DEFMCounterData, [127](#)
- ~DEFMData
  - DEFMData, [130](#)
- ~DEFMRuleDynData
  - DEFMRuleDynData, [136](#)
- ~Entries
  - Entries< Cell\_Type >, [138](#)
- ~Flock
  - Flock, [141](#)
- ~FreqTable
  - FreqTable< T >, [147](#)
- ~Geese
  - Geese, [154](#)
- ~Model
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [168](#)
- ~NetCounterData
  - NetCounterData, [186](#)
- ~NetworkData
  - NetworkData, [188](#)
- ~Node
  - Node, [191](#)
- ~PhyloRuleDynData
  - PhyloRuleDynData, [200](#)
- ~PowerSet
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [203](#)
- ~Progress
  - Progress, [208](#)
- ~Rule
  - Rule< Array\_Type, Data\_Type >, [210](#)
- ~Rules
  - Rules< Array\_Type, Data\_Type >, [213](#)
- ~StatsCounter
  - StatsCounter< Array\_Type, Data\_Type >, [218](#)
- ~Support
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [223](#)
- a
  - model-meat.hpp, [350](#)
- active
  - Cell< Cell\_Type >, [107](#)
- add
  - barray-meat.hpp, [250](#)
  - barraydense-meat.hpp, [275](#)
  - Cell< Cell\_Type >, [105](#), [106](#)
  - FreqTable< T >, [148](#)
- add\_array
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [168](#)
- add\_counter
  - Counters< Array\_Type, Data\_Type >, [117](#), [118](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [169](#)
  - StatsCounter< Array\_Type, Data\_Type >, [218](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [223](#)
- add\_data
  - Flock, [141](#)
- add\_dims
  - counters-meat.hpp, [305](#)
- add\_hash
  - Counters< Array\_Type, Data\_Type >, [118](#)
- add\_hasher

- Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
169
- add\_rule
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
169
  - PowerSet< Array\_Type, Data\_Rule\_Type >,  
203, 204
  - Rules< Array\_Type, Data\_Type >,  
214
  - Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
223, 224
- add\_rule\_dyn
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
170
  - Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
224
- annotations
  - Node, 192
- ans
  - barray-meat.hpp, 241, 250
  - barraydense-meat.hpp, 264, 275
- Array
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >,  
109
- array
  - DEFMData, 131
  - Node, 192
- Array\_
  - barray-meat.hpp, 250
- array\_bank
  - support-meat.hpp, 384
- arrays
  - Node, 192
- arrays2support
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
179
- arrays\_valid
  - Node, 193
- AS\_ONE
  - EXISTS, 41
- as\_vector
  - FreqTable< T >,  
148
- AS\_ZERO
  - EXISTS, 41
- at
  - DEFMData, 131
  - PhyloCounterData, 197
- BArray
  - BArray< Cell\_Type, Data\_Type >,  
46, 47
- BArray< Cell\_Type, Data\_Type >,  
43
  - ~BArray, 47
  - BArray, 46, 47
  - BArrayCell< Cell\_Type, Data\_Type >,  
56
  - BArrayCell\_const< Cell\_Type, Data\_Type >,  
56
  - clear, 47
  - col, 47
  - D, 48
  - D\_ptr, 48
  - default\_val, 48
  - flush\_data, 48
  - get\_cell, 48
  - get\_col\_vec, 49
  - get\_entries, 49
  - get\_row\_vec, 49
  - insert\_cell, 50
  - is\_dense, 50
  - is\_empty, 50
  - ncol, 51
  - nnozero, 51
  - nrow, 51
  - operator\*=  
operator(),  
operator+=,  
operator-=  
operator/=,  
operator=,  
operator==,  
out\_of\_range,  
print,  
reserve,  
resize,  
rm\_cell,  
row,  
set\_data,  
swap\_cells,  
swap\_cols,  
swap\_rows,  
toggle\_cell,  
toggle\_lock,  
transpose,  
visited,  
zero\_col,  
zero\_row,
- barray-meat-operators.hpp
  - BARRAY\_TEMPLATE, 234–236
  - BARRAY\_TEMPLATE\_ARGS, 235, 237
  - BARRAY\_TYPE, 235, 237
  - COL, 235
  - for, 237
  - operator(), 237
  - rhs, 237
  - ROW, 235
  - this, 238
- barray-meat.hpp
  - add, 250
  - ans, 241, 250
  - Array\_, 250
  - BARRAY\_TEMPLATE, 240–245
  - BARRAY\_TEMPLATE\_ARGS, 240
  - BARRAY\_TYPE, 240
  - check\_bounds, 250

- check\_exists, 251
- COL, 241, 245
- col0, 251
- const, 251
- copy\_data, 251
- data, 252
- delete\_data, 252
- delete\_data\_, 252
- else, 252
- false, 252
- first, 253
- for, 246
- i1, 253
- if, 246–249
- j, 253
- j0, 253
- j1, 253
- M, 249, 253
- M\_, 254
- N, 254
- NCells, 254
- report, 254
- resize, 249
- return, 249, 254
- ROW, 241, 249, 250
- row0, 255
- search, 255
- source, 255
- target, 255
- v, 255
- value, 255
- BARRAY\_TEMPLATE
  - barray-meat-operators.hpp, 234–236
  - barray-meat.hpp, 240–245
- BARRAY\_TEMPLATE\_ARGS
  - barray-meat-operators.hpp, 235, 237
  - barray-meat.hpp, 240
- BARRAY\_TYPE
  - barray-meat-operators.hpp, 235, 237
  - barray-meat.hpp, 240
- BArrayCell
  - BArrayCell< Cell\_Type, Data\_Type >, 58
- BArrayCell< Cell\_Type, Data\_Type >, 57
  - ~BArrayCell, 58
  - BArray< Cell\_Type, Data\_Type >, 56
  - BArrayCell, 58
  - operator Cell\_Type, 58
  - operator\*=, 58
  - operator+=, 58
  - operator-=, 59
  - operator/=: 59
  - operator=, 59
  - operator==, 59
- BArrayCell\_const
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 60
- BArrayCell\_const< Cell\_Type, Data\_Type >, 60
  - ~BArrayCell\_const, 60
  - BArray< Cell\_Type, Data\_Type >, 56
- BArrayCell\_const, 60
- operator Cell\_Type, 61
- operator!=, 61
- operator<, 61
- operator<=, 61
- operator>, 61
- operator>=, 62
- operator==, 61
- BArrayDense
  - BArrayDense< Cell\_Type, Data\_Type >, 65, 66
- BArrayDense< Cell\_Type, Data\_Type >, 62
  - ~BArrayDense, 66
  - BArrayDense, 65, 66
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 76, 81
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 77, 84
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 77
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 77, 88
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 77
  - clear, 67
  - col, 67
  - colsum, 67
  - D, 67, 68
  - D\_ptr, 68
  - default\_val, 68
  - get\_cell, 68
  - get\_col\_vec, 68, 69
  - get\_data, 69
  - get\_entries, 69
  - get\_row\_vec, 69
  - insert\_cell, 70
  - is\_dense, 70
  - is\_empty, 70
  - ncol, 70
  - nnozero, 71
  - nrow, 71
  - operator\*=, 71
  - operator(), 71
  - operator+=, 71, 72
  - operator-=, 72
  - operator/=: 72
  - operator=, 73
  - operator==, 73
  - out\_of\_range, 73
  - print, 73
  - reserve, 73
  - resize, 74
  - rm\_cell, 74
  - row, 74
  - rowsum, 74
  - set\_data, 74
  - swap\_cells, 75
  - swap\_cols, 75
  - swap\_rows, 75
  - toggle\_cell, 75

- toggle\_lock, 76
- transpose, 76
- visited, 77
- zero\_col, 76
- zero\_row, 76
- barraydense-meat-operators.hpp
  - BDENSE\_TEMPLATE, 258–260
  - BDENSE\_TEMPLATE\_ARGS, 258, 260
  - BDENSE\_TYPE, 258, 260
  - COL, 258
  - POS, 258
  - POS\_N, 259
  - ROW, 259
- barraydense-meat.hpp
  - add, 275
  - ans, 264, 275
  - BDENSE\_TEMPLATE, 263–271
  - BDENSE\_TEMPLATE\_ARGS, 263
  - BDENSE\_TYPE, 263
  - check\_bounds, 275
  - check\_exists, 275
  - COL, 263
  - col, 276
  - const, 276
  - copy\_data, 276
  - data, 276
  - delete\_data, 276
  - delete\_data\_, 277
  - el, 277
  - el\_colsums, 277
  - el\_rowsums, 277
  - else, 277
  - false, 278
  - for, 271
  - i1, 278
  - if, 272
  - insert\_cell, 272
  - j, 278
  - j0, 278
  - j1, 278
  - M, 273, 278
  - M\_, 279
  - N, 279
  - POS, 263
  - POS\_N, 263
  - printf\_barry, 273
  - report, 279
  - resize, 273, 274
  - return, 279
  - rm\_cell, 274
  - ROW, 264
  - source, 279
  - target, 280
  - v, 280
  - va\_end, 274
  - va\_start, 275
  - val0, 280
  - val1, 280
  - value, 280
  - ZERO\_CELL, 264
- BArrayDenseCell
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 79
- BArrayDenseCell< Cell\_Type, Data\_Type >, 78
  - ~BArrayDenseCell, 79
  - BArrayDense< Cell\_Type, Data\_Type >, 76, 81
  - BArrayDenseCell, 79
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 81, 84
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 81, 86
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 88
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 91
  - operator Cell\_Type, 79
  - operator\*=: 79
  - operator+=, 79
  - operator-=, 80
  - operator/=: 80
  - operator=, 80
  - operator==, 80
- barraydensecell-bones.hpp
  - POS, 281
- barraydensecell-meat.hpp
  - POS, 282
- BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 82
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 84
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 86
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 88
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 91
- BArrayDenseCol
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 82
- BArrayDenseCol< Cell\_Type, Data\_Type >, 82
  - BArrayDense< Cell\_Type, Data\_Type >, 77, 84
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 81, 84
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 84
  - BArrayDenseCol, 82
  - begin, 83
  - end, 83
  - operator(), 83
  - size, 83
- barraydensecol-bones.hpp
  - POS, 283
  - POS\_N, 283
  - ZERO\_CELL, 283
- BArrayDenseCol\_const
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 85
- BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 84
  - BArrayDense< Cell\_Type, Data\_Type >, 77
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 81, 86
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 86



- BArrayDenseCol\_const, 85
- begin, 85
- end, 85
- operator(), 85
- size, 86
- BArrayDenseRow
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 87
- BArrayDenseRow< Cell\_Type, Data\_Type >, 86
  - BArrayDense< Cell\_Type, Data\_Type >, 77, 88
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 88
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 88
  - BArrayDenseRow, 87
  - begin, 87
  - end, 87
  - operator(), 88
  - size, 88
- barraydenserow-bones.hpp
  - POS, 284
  - POS\_N, 284
  - ZERO\_CELL, 285
- BArrayDenseRow\_const
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 89
- BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 89
  - BArrayDense< Cell\_Type, Data\_Type >, 77
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 91
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 91
  - BArrayDenseRow\_const, 89
  - begin, 90
  - end, 90
  - operator(), 90
  - size, 90
- BArrayRow
  - BArrayRow< Cell\_Type, Data\_Type >, 92
- BArrayRow< Cell\_Type, Data\_Type >, 91
  - ~BArrayRow, 92
  - BArrayRow, 92
  - operator BArrayRow< Cell\_Type, Data\_Type >, 92
  - operator\*=, 92
  - operator+=, 92
  - operator-=, 92
  - operator/=: 93
  - operator=, 93
  - operator==, 93
- barrayrow-meat.hpp
  - BROW\_TEMPLATE, 285–287
  - BROW\_TEMPLATE\_ARGS, 286
  - BROW\_TYPE, 286
- BArrayRow\_const
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 94
- BArrayRow\_const< Cell\_Type, Data\_Type >, 93
  - ~BArrayRow\_const, 94
  - BArrayRow\_const, 94
  - operator BArrayRow\_const< Cell\_Type, Data\_Type >, 94
  - operator!=, 94
  - operator<, 94
  - operator<=: 95
  - operator>, 95
  - operator>=: 95
  - operator==, 95
- BArrayVector
  - BArrayVector< Cell\_Type, Data\_Type >, 96
- BArrayVector< Cell\_Type, Data\_Type >, 95
  - ~BArrayVector, 97
  - BArrayVector, 96
  - begin, 97
  - end, 97
  - is\_col, 97
  - is\_row, 97
  - operator std::vector< Cell\_Type >, 98
  - operator\*=, 98
  - operator+=, 98
  - operator-=, 98
  - operator/=: 98
  - operator=, 99
  - operator==, 99
  - size, 99
- BArrayVector\_const
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 100
- BArrayVector\_const< Cell\_Type, Data\_Type >, 99
  - ~BArrayVector\_const, 100
  - BArrayVector\_const, 100
  - begin, 101
  - end, 101
  - is\_col, 101
  - is\_row, 101
  - operator std::vector< Cell\_Type >, 101
  - operator!=, 101
  - operator<, 102
  - operator<=: 102
  - operator>, 102
  - operator>=: 102
  - operator==, 102
  - size, 103
- barry, 39
- barry-configuration.hpp
  - BARRY\_CHECK\_SUPPORT, 289
  - BARRY\_ISFINITE, 289
  - BARRY\_MAX\_NUM\_ELEMENTS, 289
  - BARRY\_SAFE\_EXP, 289
  - Map, 290
  - printf\_barry, 289
- barry-debug.hpp
  - BARRY\_DEBUG\_LEVEL, 290
- barry-macros.hpp
  - BARRY\_ONE, 291
  - BARRY\_ONE\_DENSE, 291
  - BARRY\_UNUSED, 291
  - BARRY\_ZERO, 292
  - BARRY\_ZERO\_DENSE, 292
- barry.hpp
  - BARRY\_HPP, 293

- BARRY\_VERSION, 294
- BARRY\_VERSION\_MAYOR, 294
- BARRY\_VERSION\_MINOR, 294
- COUNTER\_FUNCTION, 294
- COUNTER\_LAMBDA, 294
- RULE\_FUNCTION, 295
- RULE\_LAMBDA, 295
- barry::counters, 39
- barry::counters::defm, 40
- barry::counters::network, 40
- barry::counters::phylo, 40
- BARRY\_CHECK\_SUPPORT
  - barry-configuration.hpp, 289
- BARRY\_DEBUG\_LEVEL
  - barry-debug.hpp, 290
- BARRY\_HPP
  - barry.hpp, 293
- BARRY\_ISFINITE
  - barry-configuration.hpp, 289
- BARRY\_MAX\_NUM\_ELEMENTS
  - barry-configuration.hpp, 289
- BARRY\_ONE
  - barry-macros.hpp, 291
- BARRY\_ONE\_DENSE
  - barry-macros.hpp, 291
- BARRY\_PROGRESS\_BAR\_WIDTH
  - progress.hpp, 368
- BARRY\_SAFE\_EXP
  - barry-configuration.hpp, 289
- BARRY\_SUPPORT\_MEAT\_HPP
  - support-meat.hpp, 378
- BARRY\_UNUSED
  - barry-macros.hpp, 291
- BARRY\_VERSION
  - barry.hpp, 294
- BARRY\_VERSION\_MAYOR
  - barry.hpp, 294
- BARRY\_VERSION\_MINOR
  - barry.hpp, 294
- BARRY\_ZERO
  - barry-macros.hpp, 292
- BARRY\_ZERO\_DENSE
  - barry-macros.hpp, 292
- BARRY\_ZERO\_NETWORK
  - network.hpp, 325
- BARRY\_ZERO\_NETWORK\_DENSE
  - network.hpp, 325
- BDENSE\_TEMPLATE
  - barraydense-meat-operators.hpp, 258–260
  - barraydense-meat.hpp, 263–271
- BDENSE\_TEMPLATE\_ARGS
  - barraydense-meat-operators.hpp, 258, 260
  - barraydense-meat.hpp, 263
- BDENSE\_TYPE
  - barraydense-meat-operators.hpp, 258, 260
  - barraydense-meat.hpp, 263
- begin
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 83
- BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 85
- BArrayDenseRow< Cell\_Type, Data\_Type >, 87
- BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 90
- BArrayVector< Cell\_Type, Data\_Type >, 97
- BArrayVector\_const< Cell\_Type, Data\_Type >, 101
- PhyloCounterData, 197
- PowerSet< Array\_Type, Data\_Rule\_Type >, 204
- Rules< Array\_Type, Data\_Type >, 214
- blengths
  - NodeData, 196
- BOTH
  - CHECK, 40
  - EXISTS, 42
- BROW\_TEMPLATE
  - barrayrow-meat.hpp, 285–287
- BROW\_TEMPLATE\_ARGS
  - barrayrow-meat.hpp, 286
- BROW\_TYPE
  - barrayrow-meat.hpp, 286
- calc
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 204
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 224
- calc\_backend\_dense
  - support-meat.hpp, 379
- calc\_backend\_sparse
  - support-meat.hpp, 379
- calc\_reduced\_sequence
  - Geese, 154
- calc\_sequence
  - Geese, 154
- Cell
  - Cell< Cell\_Type >, 104, 105
- Cell< Cell\_Type >, 103
  - ~Cell, 104
  - active, 107
  - add, 105, 106
  - Cell, 104, 105
  - operator Cell\_Type, 106
  - operator!=, 106
  - operator=, 106, 107
  - operator==, 107
  - value, 107
  - visited, 107
- Cell\_const< Cell\_Type >, 108
- change\_stats
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 227
- change\_stats\_different
  - support-meat.hpp, 384
- CHECK, 40
  - BOTH, 40
  - NONE, 40

- ONE, [40](#)
- TWO, [41](#)
- check\_bounds
  - barray-meat.hpp, [250](#)
  - barraydense-meat.hpp, [275](#)
- check\_exists
  - barray-meat.hpp, [251](#)
  - barraydense-meat.hpp, [275](#)
- clear
  - BArray< Cell\_Type, Data\_Type >, [47](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [67](#)
  - FreqTable< T >, [148](#)
  - statscounter-meat.hpp, [372](#)
- COL
  - barray-meat-operators.hpp, [235](#)
  - barray-meat.hpp, [241](#), [245](#)
  - barraydense-meat-operators.hpp, [258](#)
  - barraydense-meat.hpp, [263](#)
- col
  - BArray< Cell\_Type, Data\_Type >, [47](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [67](#)
  - barraydense-meat.hpp, [276](#)
- col0
  - barray-meat.hpp, [251](#)
- Col\_type
  - typedefs.hpp, [390](#)
- colnames
  - Flock, [141](#)
  - Geese, [155](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [170](#)
- colsum
  - BArrayDense< Cell\_Type, Data\_Type >, [67](#)
- conditional\_prob
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [170](#)
- const
  - barray-meat.hpp, [251](#)
  - barraydense-meat.hpp, [276](#)
- ConstBArrayRowIter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [109](#)
- ConstBArrayRowIter< Cell\_Type, Data\_Type >, [108](#)
  - ~ConstBArrayRowIter, [109](#)
  - Array, [109](#)
  - ConstBArrayRowIter, [109](#)
  - current\_col, [110](#)
  - current\_row, [110](#)
  - iter, [110](#)
- coord\_i
  - support-meat.hpp, [384](#)
- coord\_j
  - support-meat.hpp, [384](#)
- cooriantes\_n\_free
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [228](#)
- cooriantes\_n\_locked
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [228](#)
- coordinates\_free
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [206](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [228](#)
- coordinates\_locked
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [206](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [228](#)
- copy\_data
  - barray-meat.hpp, [251](#)
  - barraydense-meat.hpp, [276](#)
- count
  - Counter< Array\_Type, Data\_Type >, [113](#)
- count\_all
  - StatsCounter< Array\_Type, Data\_Type >, [218](#)
- count\_current
  - StatsCounter< Array\_Type, Data\_Type >, [219](#)
- count\_fun
  - Counter< Array\_Type, Data\_Type >, [114](#)
  - counters-meat.hpp, [300](#)
- count\_fun\_
  - counters-meat.hpp, [305](#)
  - model-meat.hpp, [350](#)
- count\_init
  - StatsCounter< Array\_Type, Data\_Type >, [219](#)
- Counter
  - Counter< Array\_Type, Data\_Type >, [112](#)
- counter
  - counters-meat.hpp, [306](#)
  - model-meat.hpp, [351](#)
  - statscounter-meat.hpp, [374](#)
- Counter< Array\_Type, Data\_Type >, [110](#)
  - ~Counter, [112](#)
  - count, [113](#)
  - count\_fun, [114](#)
  - Counter, [112](#)
  - data, [114](#)
  - desc, [115](#)
  - get\_description, [113](#)
  - get\_hasher, [113](#)
  - get\_name, [113](#)
  - hasher\_fun, [115](#)
  - init, [113](#)
  - init\_fun, [115](#)
  - name, [115](#)
  - operator=, [113](#), [114](#)
  - set\_hasher, [114](#)
- counter\_
  - counters-meat.hpp, [306](#)
- counter\_absdiff
  - DEFMArray counters, [21](#)

- counter\_co\_opt
  - Phylo counters, [31](#)
- counter\_cogain
  - Phylo counters, [32](#)
- counter\_css\_census01
  - network-css.hpp, [318](#)
- counter\_css\_census02
  - network-css.hpp, [318](#)
- counter\_css\_census03
  - network-css.hpp, [318](#)
- counter\_css\_census04
  - network-css.hpp, [319](#)
- counter\_css\_census05
  - network-css.hpp, [319](#)
- counter\_css\_census06
  - network-css.hpp, [319](#)
- counter\_css\_census07
  - network-css.hpp, [319](#)
- counter\_css\_census08
  - network-css.hpp, [320](#)
- counter\_css\_census09
  - network-css.hpp, [320](#)
- counter\_css\_census10
  - network-css.hpp, [320](#)
- counter\_css\_completely\_false\_recip\_comiss
  - network-css.hpp, [320](#)
- counter\_css\_completely\_false\_recip\_omiss
  - network-css.hpp, [321](#)
- counter\_css\_mixed\_recip
  - network-css.hpp, [321](#)
- counter\_css\_partially\_false\_recip\_commi
  - network-css.hpp, [321](#)
- counter\_css\_partially\_false\_recip\_omiss
  - network-css.hpp, [322](#)
- counter\_ctriads
  - DEFMArray counters, [22](#)
- counter\_degree
  - DEFMArray counters, [22](#)
- counter\_deleted
  - statscounter-meat.hpp, [374](#)
- counter\_density
  - DEFMArray counters, [22](#)
- counter\_diff
  - DEFMArray counters, [23](#)
- counter\_edges
  - DEFMArray counters, [23](#)
- counter\_fixed\_effect
  - DEFMArray counters, [23](#)
- counter\_fun
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [179](#)
- Counter\_fun\_type
  - typedefs.hpp, [390](#)
- COUNTER\_FUNCTION
  - barry.hpp, [294](#)
- counter\_gains
  - Phylo counters, [32](#)
- counter\_gains\_from\_0
  - Phylo counters, [32](#)
- counter\_gains\_k\_offspring
  - Phylo counters, [33](#)
- counter\_genes\_changing
  - Phylo counters, [33](#)
- counter\_idegree
  - DEFMArray counters, [24](#)
- counter\_idegree15
  - DEFMArray counters, [24](#)
- counter\_isolates
  - DEFMArray counters, [24](#), [25](#)
- counter\_istar2
  - DEFMArray counters, [25](#)
- counter\_k\_genes\_changing
  - Phylo counters, [33](#)
- COUNTER\_LAMBDA
  - barry.hpp, [294](#)
- counter\_less\_than\_p\_prop\_genes\_changing
  - Phylo counters, [33](#)
- counter\_logit\_intercept
  - DEFMArray counters, [25](#)
- counter\_longest
  - Phylo counters, [34](#)
- counter\_loss
  - Phylo counters, [34](#)
- counter\_maxfun
  - Phylo counters, [34](#)
- counter\_mutual
  - DEFMArray counters, [25](#)
- counter\_neofun
  - Phylo counters, [34](#)
- counter\_neofun\_a2b
  - Phylo counters, [35](#)
- counter\_nodecov
  - DEFMArray counters, [26](#)
- counter\_nodeicov
  - DEFMArray counters, [26](#)
- counter\_nodematch
  - DEFMArray counters, [26](#)
- counter\_nodeocov
  - DEFMArray counters, [26](#)
- counter\_odegree
  - DEFMArray counters, [26](#), [27](#)
- counter\_odegree15
  - DEFMArray counters, [27](#)
- counter\_ones
  - DEFMArray counters, [27](#)
- counter\_ostar2
  - DEFMArray counters, [28](#)
- counter\_overall\_changes
  - Phylo counters, [35](#)
- counter\_overall\_gains
  - Phylo counters, [35](#)
- counter\_overall\_gains\_from\_0
  - Phylo counters, [35](#)
- counter\_overall\_loss
  - Phylo counters, [36](#)

- counter\_pairwise\_first\_gain
  - Phylo counters, 36
- counter\_pairwise\_neofun\_singlefun
  - Phylo counters, 36
- counter\_pairwise\_overall\_change
  - Phylo counters, 36
- counter\_pairwise\_preserving
  - Phylo counters, 37
- counter\_preserve\_pseudogene
  - Phylo counters, 37
- counter\_prop\_genes\_changing
  - Phylo counters, 37
- counter\_subfun
  - Phylo counters, 37
- COUNTER\_TEMPLATE
  - counters-meat.hpp, 298, 300, 301
- COUNTER\_TEMPLATE\_ARGS
  - counters-meat.hpp, 298
- counter\_transition
  - DEFMArray counters, 28
- counter\_transition\_formula
  - DEFMArray counters, 29
- counter\_ttriads
  - DEFMArray counters, 29
- COUNTER\_TYPE
  - counters-meat.hpp, 299
- Counters
  - Counters< Array\_Type, Data\_Type >, 116, 117
- counters
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 179
  - statscounter-meat.hpp, 375
  - support-meat.hpp, 384
- Counters< Array\_Type, Data\_Type >, 116
  - ~Counters, 117
  - add\_counter, 117, 118
  - add\_hash, 118
  - Counters, 116, 117
  - gen\_hash, 118
  - get\_descriptions, 118
  - get\_names, 119
  - operator=, 119
  - operator[], 120
  - size, 120
- counters-meat.hpp
  - add\_dims, 305
  - count\_fun, 300
  - count\_fun\_, 305
  - counter, 306
  - counter\_, 306
  - COUNTER\_TEMPLATE, 298, 300, 301
  - COUNTER\_TEMPLATE\_ARGS, 298
  - COUNTER\_TYPE, 299
  - COUNTERS\_TEMPLATE, 299, 301–303
  - COUNTERS\_TEMPLATE\_ARGS, 299
  - COUNTERS\_TYPE, 299
  - data, 303
  - data\_, 306
  - desc, 303
  - desc\_, 306
  - for, 303
  - fun, 307
  - fun\_, 307
  - hasher, 303, 304
  - hasher\_fun, 304
  - hasher\_fun\_, 307
  - i, 307
  - if, 304
  - init\_fun, 305
  - init\_fun\_, 308
  - j, 308
  - name, 305
  - name\_, 308
  - noexcept, 308
  - res, 308
  - return, 309
  - TMP\_HASHER\_CALL, 299
- counters\_
  - model-meat.hpp, 351
  - statscounter-meat.hpp, 375
  - support-meat.hpp, 385
- COUNTERS\_TEMPLATE
  - counters-meat.hpp, 299, 301–303
- COUNTERS\_TEMPLATE\_ARGS
  - counters-meat.hpp, 299
- COUNTERS\_TYPE
  - counters-meat.hpp, 299
- Counting, 13
- counts
  - DEFMRuleDynData, 137
  - PhyloRuleDynData, 200
- Counts\_type
  - typedefs.hpp, 390
- covar\_sort
  - DEFMData, 131
- covar\_used
  - DEFMData, 131
- covariates
  - DEFMData, 131
- CSS\_APPEND
  - network-css.hpp, 316
- CSS\_CASE\_ELSE
  - network-css.hpp, 316
- CSS\_CASE\_PERCEIVED
  - network-css.hpp, 316
- CSS\_CASE\_TRUTH
  - network-css.hpp, 316
- CSS\_CHECK\_SIZE
  - network-css.hpp, 316
- CSS\_CHECK\_SIZE\_INIT
  - network-css.hpp, 317
- CSS\_NET\_COUNTER\_LAMBDA\_INIT
  - network-css.hpp, 317
- CSS\_PERCEIVED\_CELLS
  - network-css.hpp, 317

- CSS\_SIZE
  - network-css.hpp, 317
- CSS\_TRUE\_CELLS
  - network-css.hpp, 318
- cumprob
  - model-meat.hpp, 351
- current\_col
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, 110
- current\_row
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, 110
- current\_stats
  - statscounter-meat.hpp, 375
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 228
- D
  - BArray< Cell\_Type, Data\_Type >, 48
  - BArrayDense< Cell\_Type, Data\_Type >, 67, 68
  - Rule< Array\_Type, Data\_Type >, 211
- D\_ptr
  - BArray< Cell\_Type, Data\_Type >, 48
  - BArrayDense< Cell\_Type, Data\_Type >, 68
- dat
  - Flock, 145
- data
  - barray-meat.hpp, 252
  - barraydense-meat.hpp, 276
  - Counter< Array\_Type, Data\_Type >, 114
  - counters-meat.hpp, 303
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 206
- data\_
  - counters-meat.hpp, 306
  - model-meat.hpp, 351
- Data\_Counter\_Type
  - model-meat.hpp, 351
- Data\_Rule\_Type
  - model-meat.hpp, 352
- DEFAULT\_DUPLICATION
  - phylo.hpp, 332
- default\_val
  - BArray< Cell\_Type, Data\_Type >, 48
  - BArrayDense< Cell\_Type, Data\_Type >, 68
- DEFM, 121
  - DEFM, 122
  - get\_ID, 122
  - get\_m\_order, 122
  - get\_model, 122
  - get\_n\_covars, 123
  - get\_n\_obs, 123
  - get\_n\_rows, 123
  - get\_n\_y, 123
  - get\_X, 123
  - get\_X\_names, 123
  - get\_Y, 124
  - get\_Y\_names, 124
  - init, 124
  - is\_motif, 124
  - likelihood, 124
  - logodds, 124
  - motif\_census, 125
  - print, 125
  - set\_names, 125
  - simulate, 125
- defm-formula.hpp
  - defm\_motif\_parser, 309
- defm-meat.hpp
  - DEFM\_LOOP\_ARRAYS, 357
  - DEFM\_RANGES, 357
  - keygen\_defm, 358
- defm.hpp
  - DEFMArray, 313
  - UNI\_SUB, 313
- DEFM\_COUNTER
  - Phylo rules, 15
- DEFM\_COUNTER\_LAMBDA
  - Phylo rules, 15
- DEFM\_LOOP\_ARRAYS
  - defm-meat.hpp, 357
- defm\_motif\_parser
  - defm-formula.hpp, 309
- DEFM\_RANGES
  - defm-meat.hpp, 357
- DEFM\_RULE
  - Phylo rules, 15
- DEFM\_RULE\_LAMBDA
  - Phylo rules, 16
- DEFM\_RULEDYN\_LAMBDA
  - Phylo rules, 16
- DEFMArray
  - defm.hpp, 313
- DEFMArray counters, 19
  - counter\_absdiff, 21
  - counter\_ctriads, 22
  - counter\_degree, 22
  - counter\_density, 22
  - counter\_diff, 23
  - counter\_edges, 23
  - counter\_fixed\_effect, 23
  - counter\_iddegree, 24
  - counter\_iddegree15, 24
  - counter\_isolates, 24, 25
  - counter\_istar2, 25
  - counter\_logit\_intercept, 25
  - counter\_mutual, 25
  - counter\_nodecov, 26
  - counter\_nodeicov, 26
  - counter\_nodematch, 26
  - counter\_nodeocov, 26
  - counter\_odegree, 26, 27
  - counter\_odegree15, 27
  - counter\_ones, 27
  - counter\_ostar2, 28
  - counter\_transition, 28
  - counter\_transition\_formula, 29

- counter\_ttriads, [29](#)
- NETWORK\_COUNTER, [29](#)
- rules\_dont\_become\_zero, [29](#)
- rules\_markov\_fixed, [30](#)
- DEFMCounter
  - Phylo rules, [17](#)
- DEFMCounterData, [126](#)
  - ~DEFMCounterData, [127](#)
  - DEFMCounterData, [126](#)
  - idx, [127](#)
  - indices, [128](#)
  - is\_motif, [128](#)
  - is\_true, [127](#)
  - logical, [128](#)
  - num, [127](#)
  - numbers, [128](#)
- DEFMCounters
  - Phylo rules, [17](#)
- DEFMData, [129](#)
  - ~DEFMData, [130](#)
  - array, [131](#)
  - at, [131](#)
  - covar\_sort, [131](#)
  - covar\_used, [131](#)
  - covariates, [131](#)
  - DEFMData, [130](#)
  - obs\_start, [131](#)
  - X\_ncol, [132](#)
  - X\_nrow, [132](#)
- DEFMModel
  - Phylo rules, [17](#)
- DEFMRule
  - Phylo rules, [17](#)
- DEFMRuleData, [132](#)
  - DEFMRuleData, [133](#)
  - idx, [134](#)
  - indices, [134](#)
  - init, [134](#)
  - is\_true, [134](#)
  - logical, [135](#)
  - num, [134](#)
  - numbers, [135](#)
- DEFMRuleDyn
  - Phylo rules, [17](#)
- DEFMRuleDynData, [135](#)
  - ~DEFMRuleDynData, [136](#)
  - counts, [137](#)
  - DEFMRuleDynData, [136](#)
- DEFMRules
  - Phylo rules, [17](#)
- DEFMRulesDyn
  - Phylo rules, [18](#)
- DEFMStatsCounter
  - Phylo rules, [18](#)
- DEFMSupport
  - Phylo rules, [18](#)
- delete\_counters
  - Model< Array\_Type, Data\_Counter\_Type,
  - Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [180](#)
  - Support< Array\_Type, Data\_Counter\_Type,
  - Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [229](#)
  - support-meat.hpp, [385](#)
- delete\_data
  - barray-meat.hpp, [252](#)
  - barraydense-meat.hpp, [276](#)
- delete\_data\_
  - barray-meat.hpp, [252](#)
  - barraydense-meat.hpp, [277](#)
- delete\_rengine
  - Geese, [161](#)
  - Model< Array\_Type, Data\_Counter\_Type,
  - Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [180](#)
- delete\_rules
  - Model< Array\_Type, Data\_Counter\_Type,
  - Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [180](#)
  - model-meat.hpp, [352](#)
  - Support< Array\_Type, Data\_Counter\_Type,
  - Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [229](#)
  - support-meat.hpp, [385](#)
- delete\_rules\_dyn
  - Model< Array\_Type, Data\_Counter\_Type,
  - Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [180](#)
  - model-meat.hpp, [352](#)
  - Support< Array\_Type, Data\_Counter\_Type,
  - Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [229](#)
  - support-meat.hpp, [385](#)
- delete\_support
  - Geese, [161](#)
- desc
  - Counter< Array\_Type, Data\_Type >, [115](#)
  - counters-meat.hpp, [303](#)
- desc\_
  - counters-meat.hpp, [306](#)
- directed
  - NetworkData, [188](#)
- DUPL\_DUPL
  - phylo.hpp, [332](#)
- DUPL\_EITH
  - phylo.hpp, [332](#)
- DUPL\_SPEC
  - phylo.hpp, [332](#)
- duplication
  - Node, [193](#)
  - NodeData, [196](#)
  - PhyloRuleDynData, [200](#)
- el
  - barraydense-meat.hpp, [277](#)
- el\_colsums
  - barraydense-meat.hpp, [277](#)

- el\_rowsums
  - barraydense-meat.hpp, 277
- else
  - barray-meat.hpp, 252
  - barraydense-meat.hpp, 277
  - model-meat.hpp, 352
  - support-meat.hpp, 385
- empty
  - PhyloCounterData, 197
- EmptyArray
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 206
  - statscounter-meat.hpp, 375
- end
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 83
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 85
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 87
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 90
  - BArrayVector< Cell\_Type, Data\_Type >, 97
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 101
  - PhyloCounterData, 198
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 204
  - Progress, 209
  - Rules< Array\_Type, Data\_Type >, 214
- Entries
  - Entries< Cell\_Type >, 138
- Entries< Cell\_Type >, 137
  - ~Entries, 138
  - Entries, 138
  - resize, 138
  - source, 139
  - target, 139
  - val, 139
- eval\_rules\_dyn
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 225
- EXISTS, 41
  - AS\_ONE, 41
  - AS\_ZERO, 41
  - BOTH, 42
  - NONE, 42
  - ONE, 42
  - TWO, 42
  - UNKNOWN, 42
- f\_
  - statscounter-meat.hpp, 375
  - support-meat.hpp, 386
- false
  - barray-meat.hpp, 252
  - barraydense-meat.hpp, 278
- first
  - barray-meat.hpp, 253
- first\_calc\_done
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 180
- Flock, 139
  - ~Flock, 141
  - add\_data, 141
  - colnames, 141
  - dat, 145
  - Flock, 141
  - get\_counters, 142
  - get\_model, 142
  - get\_stats\_support, 142
  - get\_stats\_target, 142
  - get\_support\_fun, 142
  - init, 142
  - initialized, 145
  - likelihood\_joint, 143
  - model, 146
  - nfunctions, 146
  - nfuncs, 143
  - nleafs, 143
  - nnodes, 143
  - nterms, 144
  - ntrees, 144
  - operator(), 144
  - parse\_polytomies, 144
  - print, 145
  - rengine, 146
  - set\_seed, 145
  - support\_size, 145
- flush\_data
  - BArray< Cell\_Type, Data\_Type >, 48
- for
  - barray-meat-operators.hpp, 237
  - barray-meat.hpp, 246
  - barraydense-meat.hpp, 271
  - counters-meat.hpp, 303
  - model-meat.hpp, 341
  - statscounter-meat.hpp, 372
  - support-meat.hpp, 379
- force\_new
  - model-meat.hpp, 352
- FreqTable
  - FreqTable< T >, 147
- FreqTable< T >, 146
  - ~FreqTable, 147
  - add, 148
  - as\_vector, 148
  - clear, 148
  - FreqTable, 147
  - get\_data, 148
  - get\_index, 148
  - make\_hash, 149
  - print, 149
  - reserve, 149
  - size, 149
- fun
  - counters-meat.hpp, 307



- fun\_
  - counters-meat.hpp, [307](#)
  - model-meat.hpp, [353](#)
- Geese, [150](#)
  - ~Geese, [154](#)
  - calc\_reduced\_sequence, [154](#)
  - calc\_sequence, [154](#)
  - colnames, [155](#)
  - delete\_rengine, [161](#)
  - delete\_support, [161](#)
  - Geese, [153](#), [154](#)
  - get\_annotated\_nodes, [155](#)
  - get\_counters, [155](#)
  - get\_model, [155](#)
  - get\_probabilities, [155](#)
  - get\_rengine, [155](#)
  - get\_states, [156](#)
  - get\_support\_fun, [156](#)
  - inherit\_support, [156](#)
  - init, [156](#)
  - init\_node, [156](#)
  - initialized, [161](#)
  - likelihood, [157](#)
  - likelihood\_exhaust, [157](#)
  - map\_to\_nodes, [162](#)
  - nannotations, [157](#)
  - nfunctions, [162](#)
  - nfuncs, [157](#)
  - nleaves, [157](#)
  - nnodes, [158](#)
  - nodes, [162](#)
  - nterms, [158](#)
  - observed\_counts, [158](#)
  - operator=, [158](#)
  - parse\_polytomies, [158](#)
  - predict, [159](#)
  - predict\_backend, [159](#)
  - predict\_exhaust, [159](#)
  - predict\_exhaust\_backend, [159](#)
  - predict\_sim, [160](#)
  - print, [160](#)
  - print\_observed\_counts, [160](#)
  - pset\_loc, [162](#)
  - reduced\_sequence, [162](#)
  - sequence, [162](#)
  - set\_seed, [160](#)
  - simulate, [160](#)
  - support\_size, [161](#)
  - update\_annotations, [161](#)
- geese-bones.hpp
  - INITIALIZED, [360](#)
  - keygen\_full, [361](#)
  - RULE\_FUNCTION, [361](#)
  - vec\_diff, [361](#)
  - vector\_caster, [361](#)
- gen\_hash
  - Counters< Array\_Type, Data\_Type >, [118](#)
- gen\_key
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [171](#)
- get\_annotated\_nodes
  - Geese, [155](#)
- get\_arrays2support
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [171](#)
- get\_cell
  - BArray< Cell\_Type, Data\_Type >, [48](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [68](#)
- get\_col\_vec
  - BArray< Cell\_Type, Data\_Type >, [49](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [68](#), [69](#)
- get\_counters
  - Flock, [142](#)
  - Geese, [155](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [171](#)
  - PhyloCounterData, [198](#)
  - StatsCounter< Array\_Type, Data\_Type >, [219](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [225](#)
- get\_counts
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [225](#)
- get\_current\_stats
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [225](#)
- get\_data
  - BArrayDense< Cell\_Type, Data\_Type >, [69](#)
  - FreqTable< T >, [148](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [204](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [225](#)
- get\_data\_ptr
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [205](#)
- get\_description
  - Counter< Array\_Type, Data\_Type >, [113](#)
  - Rule< Array\_Type, Data\_Type >, [211](#)
- get\_descriptions
  - Counters< Array\_Type, Data\_Type >, [118](#)
  - Rules< Array\_Type, Data\_Type >, [214](#)
  - StatsCounter< Array\_Type, Data\_Type >, [219](#)
- get\_entries
  - BArray< Cell\_Type, Data\_Type >, [49](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [69](#)
- get\_hasher
  - Counter< Array\_Type, Data\_Type >, [113](#)
- get\_ID
  - DEFM, [122](#)
- get\_index

- FreqTable< T >, 148
- get\_last\_name
  - phylo.hpp, 337
- get\_m\_order
  - DEFM, 122
- get\_model
  - DEFM, 122
  - Flock, 142
  - Geese, 155
- get\_n\_covars
  - DEFM, 123
- get\_n\_obs
  - DEFM, 123
- get\_n\_rows
  - DEFM, 123
- get\_n\_y
  - DEFM, 123
- get\_name
  - Counter< Array\_Type, Data\_Type >, 113
  - Rule< Array\_Type, Data\_Type >, 211
- get\_names
  - Counters< Array\_Type, Data\_Type >, 119
  - Rules< Array\_Type, Data\_Type >, 215
  - StatsCounter< Array\_Type, Data\_Type >, 219
- get\_norm\_const
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 171
- get\_parent
  - Node, 191
- get\_probabilities
  - Geese, 155
- get\_pset
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 171
- get\_pset\_arrays
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 172
- get\_pset\_probs
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 172
- get\_pset\_stats
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 172
- get\_rengine
  - Geese, 155
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 172
- get\_row\_vec
  - BArray< Cell\_Type, Data\_Type >, 49
  - BArrayDense< Cell\_Type, Data\_Type >, 69
- get\_rules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 173
- Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 226
- get\_rules\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 173
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 226
- get\_seq
  - Rules< Array\_Type, Data\_Type >, 215
- get\_states
  - Geese, 156
- get\_stats\_support
  - Flock, 142
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 173
- get\_stats\_target
  - Flock, 142
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 173
- get\_support\_fun
  - Flock, 142
  - Geese, 156
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 173
- get\_X
  - DEFM, 123
- get\_X\_names
  - DEFM, 123
- get\_Y
  - DEFM, 124
- get\_Y\_names
  - DEFM, 124
- hasher
  - counters-meat.hpp, 303, 304
- hasher\_fun
  - Counter< Array\_Type, Data\_Type >, 115
  - counters-meat.hpp, 304
- hasher\_fun\_
  - counters-meat.hpp, 307
- Hasher\_fun\_type
  - typedefs.hpp, 390
- hashes
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 229
  - support-meat.hpp, 386
- hashes\_initialized
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 229

- i
  - counters-meat.hpp, 307
  - model-meat.hpp, 353
- i1
  - barray-meat.hpp, 253
  - barraydense-meat.hpp, 278
- i\_matches
  - model-meat.hpp, 353
- id
  - Node, 193
- idx
  - DEFMCounterData, 127
  - DEFMRuleData, 134
- if
  - barray-meat.hpp, 246–249
  - barraydense-meat.hpp, 272
  - counters-meat.hpp, 304
  - model-meat.hpp, 341, 342
  - support-meat.hpp, 379, 380
- IF\_MATCHES
  - phylo.hpp, 332
- IF\_NOTMATCHES
  - phylo.hpp, 333
- include/barry/barray-bones.hpp, 233
- include/barry/barray-iterator.hpp, 233
- include/barry/barray-meat-operators.hpp, 234
- include/barry/barray-meat.hpp, 238
- include/barry/barraycell-bones.hpp, 256
- include/barry/barraycell-meat.hpp, 256
- include/barry/barraydense-bones.hpp, 257
- include/barry/barraydense-meat-operators.hpp, 257
- include/barry/barraydense-meat.hpp, 260
- include/barry/barraydensecell-bones.hpp, 281
- include/barry/barraydensecell-meat.hpp, 282
- include/barry/barraydensecol-bones.hpp, 282
- include/barry/barraydenserow-bones.hpp, 284
- include/barry/barrayrow-bones.hpp, 285
- include/barry/barrayrow-meat.hpp, 285
- include/barry/barrayvector-bones.hpp, 287
- include/barry/barrayvector-meat.hpp, 288
- include/barry/barray-configuration.hpp, 288
- include/barry/barray-debug.hpp, 290
- include/barry/barray-macros.hpp, 291
- include/barry/barray.hpp, 292
- include/barry/cell-bones.hpp, 295
- include/barry/cell-meat.hpp, 296
- include/barry/col-bones.hpp, 296
- include/barry/counters-bones.hpp, 296
- include/barry/counters-meat.hpp, 297
- include/barry/counters/defm-formula.hpp, 309
- include/barry/counters/defm.hpp, 311
- include/barry/counters/network-css.hpp, 314
- include/barry/counters/network.hpp, 322
- include/barry/counters/phylo.hpp, 329
- include/barry/freqtable.hpp, 337
- include/barry/model-bones.hpp, 338
- include/barry/model-meat.hpp, 338
- include/barry/models/defm.hpp, 314
- include/barry/models/defm/defm-bones.hpp, 356
- include/barry/models/defm/defm-meat.hpp, 357
- include/barry/models/geese.hpp, 358
- include/barry/models/geese/flock-bones.hpp, 359
- include/barry/models/geese/flock-meat.hpp, 359
- include/barry/models/geese/geese-bones.hpp, 360
- include/barry/models/geese/geese-meat-constructors.hpp, 362
- include/barry/models/geese/geese-meat-likelihood.hpp, 362
- include/barry/models/geese/geese-meat-likelihood\_exhaust.hpp, 363
- include/barry/models/geese/geese-meat-predict.hpp, 364
- include/barry/models/geese/geese-meat-predict\_exhaust.hpp, 364
- include/barry/models/geese/geese-meat-predict\_sim.hpp, 365
- include/barry/models/geese/geese-meat-simulate.hpp, 365
- include/barry/models/geese/geese-meat.hpp, 366
- include/barry/models/geese/geese-node-bones.hpp, 366
- include/barry/powerset-bones.hpp, 367
- include/barry/powerset-meat.hpp, 367
- include/barry/progress.hpp, 368
- include/barry/rules-bones.hpp, 369
- include/barry/rules-meat.hpp, 370
- include/barry/statscounter-bones.hpp, 370
- include/barry/statscounter-meat.hpp, 371
- include/barry/support-bones.hpp, 376
- include/barry/support-meat.hpp, 377
- include/barry/typedefs.hpp, 388
- indices
  - DEFMCounterData, 128
  - DEFMRuleData, 134
  - NetCounterData, 186
- inherit\_support
  - Geese, 156
- init
  - Counter< Array\_Type, Data\_Type >, 113
  - DEFM, 124
  - DEFMRuleData, 134
  - Flock, 142
  - Geese, 156
- init\_fun
  - Counter< Array\_Type, Data\_Type >, 115
  - counters-meat.hpp, 305
- init\_fun\_
  - counters-meat.hpp, 308
  - model-meat.hpp, 353
- init\_node
  - Geese, 156
- init\_support
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 205
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 226

- INITIALIZED
  - geese-bones.hpp, 360
- initialized
  - Flock, 145
  - Geese, 161
- insert\_cell
  - BArray< Cell\_Type, Data\_Type >, 50
  - BArrayDense< Cell\_Type, Data\_Type >, 70
  - barraydense-meat.hpp, 272
  - model-meat.hpp, 342
  - support-meat.hpp, 380
- is\_col
  - BArrayVector< Cell\_Type, Data\_Type >, 97
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 101
- is\_dense
  - BArray< Cell\_Type, Data\_Type >, 50
  - BArrayDense< Cell\_Type, Data\_Type >, 70
- IS\_DUPLICATION
  - phylo.hpp, 333
- IS\_EITHER
  - phylo.hpp, 333
- is\_empty
  - BArray< Cell\_Type, Data\_Type >, 50
  - BArrayDense< Cell\_Type, Data\_Type >, 70
- is\_leaf
  - Node, 192
- is\_motif
  - DEFM, 124
  - DEFMCounterData, 128
- is\_row
  - BArrayVector< Cell\_Type, Data\_Type >, 97
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 101
- IS\_SPECIATION
  - phylo.hpp, 333
- is\_true
  - DEFMCounterData, 127
  - DEFMRuleData, 134
- iter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, 110
- j
  - barray-meat.hpp, 253
  - barraydense-meat.hpp, 278
  - counters-meat.hpp, 308
  - model-meat.hpp, 353
  - statscounter-meat.hpp, 376
- j0
  - barray-meat.hpp, 253
  - barraydense-meat.hpp, 278
- j1
  - barray-meat.hpp, 253
  - barraydense-meat.hpp, 278
- k
  - model-meat.hpp, 354
- key
  - model-meat.hpp, 354
- keygen\_defm
  - defm-meat.hpp, 358
- keygen\_full
  - geese-bones.hpp, 361
- keys2support
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 181
- lb
  - PhyloRuleDynData, 201
- likelihood
  - DEFM, 124
  - Geese, 157
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 174
- likelihood\_
  - model-meat.hpp, 342
- likelihood\_exhaust
  - Geese, 157
- likelihood\_joint
  - Flock, 143
- likelihood\_total
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 175
- locator
  - model-meat.hpp, 354
- logical
  - DEFMCounterData, 128
  - DEFMRuleData, 135
- logodds
  - DEFM, 124
- M
  - barray-meat.hpp, 249, 253
  - barraydense-meat.hpp, 273, 278
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 206
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 230
- M\_
  - barray-meat.hpp, 254
  - barraydense-meat.hpp, 279
- MAKE\_DEFM\_HASHER
  - Phylo rules, 16
- MAKE\_DUPL\_VARS
  - phylo.hpp, 333
- make\_hash
  - FreqTable< T >, 149
- Map
  - barry-configuration.hpp, 290
- map\_to\_nodes
  - Geese, 162
- MapVec\_type
  - typedefs.hpp, 391
- max\_num\_elements

- Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
230
- Model
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
167, 168
- model
  - Flock, 146
- Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type,  
Data\_Rule\_Dyn\_Type >, 163
  - ~Model, 168
  - add\_array, 168
  - add\_counter, 169
  - add\_hasher, 169
  - add\_rule, 169
  - add\_rule\_dyn, 170
  - arrays2support, 179
  - colnames, 170
  - conditional\_prob, 170
  - counter\_fun, 179
  - counters, 179
  - delete\_counters, 180
  - delete\_engine, 180
  - delete\_rules, 180
  - delete\_rules\_dyn, 180
  - first\_calc\_done, 180
  - gen\_key, 171
  - get\_arrays2support, 171
  - get\_counters, 171
  - get\_norm\_const, 171
  - get\_pset, 171
  - get\_pset\_arrays, 172
  - get\_pset\_probs, 172
  - get\_pset\_stats, 172
  - get\_engine, 172
  - get\_rules, 173
  - get\_rules\_dyn, 173
  - get\_stats\_support, 173
  - get\_stats\_target, 173
  - get\_support\_fun, 173
  - keys2support, 181
  - likelihood, 174
  - likelihood\_total, 175
  - Model, 167, 168
  - normalizing\_constants, 181
  - nrules, 175
  - nrules\_dyn, 175
  - nterms, 175
  - operator=, 175
  - params\_last, 181
  - print, 176
  - print\_stats, 176
  - pset\_arrays, 181
  - pset\_probs, 182
  - pset\_stats, 182
  - rengine, 182
  - rules, 182
  - rules\_dyn, 183
  - sample, 176
  - set\_counters, 177
  - set\_engine, 177
  - set\_rules, 177
  - set\_rules\_dyn, 177
  - set\_seed, 177
  - set\_transform\_model, 178
  - size, 178
  - size\_unique, 178
  - stats\_support, 183
  - stats\_support\_n\_arrays, 183
  - stats\_target, 183
  - store\_psets, 178
  - support\_fun, 184
  - support\_size, 179
  - transform\_model, 179
  - transform\_model\_fun, 184
  - transform\_model\_term\_names, 184
  - with\_pset, 185
- model-meat.hpp
  - a, 350
  - count\_fun\_, 350
  - counter, 351
  - counters\_, 351
  - cumprob, 351
  - data\_, 351
  - Data\_Counter\_Type, 351
  - Data\_Rule\_Type, 352
  - delete\_rules, 352
  - delete\_rules\_dyn, 352
  - else, 352
  - for, 341
  - force\_new, 352
  - fun\_, 353
  - i, 353
  - i\_matches, 353
  - if, 341, 342
  - init\_fun\_, 353
  - insert\_cell, 342
  - j, 353
  - k, 354
  - key, 354
  - likelihood\_, 342
  - locator, 354
  - MODEL\_TEMPLATE, 340, 342–348
  - MODEL\_TEMPLATE\_ARGS, 340
  - MODEL\_TYPE, 341
  - params, 354
  - probs, 354
  - pset\_arrays, 354
  - push\_back, 348, 349
  - r, 355
  - return, 349, 355
  - rule\_fun\_, 355
  - rules, 355
  - rules\_, 355
  - rules\_dyn, 356

- set\_counters, 349
- set\_rules, 349
- set\_rules\_dyn, 349
- size, 349
- stats, 356
- stats\_support\_n\_arrays, 356
- temp\_stats, 349
- tmp\_counts, 350
- update\_normalizing\_constant, 350
- urand, 350
- MODEL\_TEMPLATE
  - model-meat.hpp, 340, 342–348
- MODEL\_TEMPLATE\_ARGS
  - model-meat.hpp, 340
- MODEL\_TYPE
  - model-meat.hpp, 341
- motif\_census
  - DEFM, 125
- N
  - barray-meat.hpp, 254
  - barraydense-meat.hpp, 279
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 207
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 230
- n\_counters
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 230
- n\_free
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 207
- n\_locked
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 207
- name
  - Counter< Array\_Type, Data\_Type >, 115
  - counters-meat.hpp, 305
- name\_
  - counters-meat.hpp, 308
- nannotations
  - Geese, 157
- narray
  - Node, 193
- NCells
  - barray-meat.hpp, 254
- ncol
  - BArray< Cell\_Type, Data\_Type >, 51
  - BArrayDense< Cell\_Type, Data\_Type >, 70
  - Phylo rules, 18
- NET\_C\_DATA\_IDX
  - network.hpp, 326
- NET\_C\_DATA\_NUM
  - network.hpp, 326
- NetCounter
  - network.hpp, 327
- NetCounterData, 185
  - ~NetCounterData, 186
  - indices, 186
  - NetCounterData, 186
  - numbers, 186
- NetCounters
  - network.hpp, 327
- NetModel
  - network.hpp, 328
- NetRule
  - network.hpp, 328
- NetRules
  - network.hpp, 328
- NetStatsCounter
  - network.hpp, 328
- NetSupport
  - network.hpp, 328
- Network
  - network.hpp, 328
- network-css.hpp
  - counter\_css\_census01, 318
  - counter\_css\_census02, 318
  - counter\_css\_census03, 318
  - counter\_css\_census04, 319
  - counter\_css\_census05, 319
  - counter\_css\_census06, 319
  - counter\_css\_census07, 319
  - counter\_css\_census08, 320
  - counter\_css\_census09, 320
  - counter\_css\_census10, 320
  - counter\_css\_completely\_false\_recip\_comiss, 320
  - counter\_css\_completely\_false\_recip\_omiss, 321
  - counter\_css\_mixed\_recip, 321
  - counter\_css\_partially\_false\_recip\_commi, 321
  - counter\_css\_partially\_false\_recip\_omiss, 322
  - CSS\_APPEND, 316
  - CSS\_CASE\_ELSE, 316
  - CSS\_CASE\_PERCEIVED, 316
  - CSS\_CASE\_TRUTH, 316
  - CSS\_CHECK\_SIZE, 316
  - CSS\_CHECK\_SIZE\_INIT, 317
  - CSS\_NET\_COUNTER\_LAMBDA\_INIT, 317
  - CSS\_PERCEIVED\_CELLS, 317
  - CSS\_SIZE, 317
  - CSS\_TRUE\_CELLS, 318
- network.hpp
  - BARRY\_ZERO\_NETWORK, 325
  - BARRY\_ZERO\_NETWORK\_DENSE, 325
  - NET\_C\_DATA\_IDX, 326
  - NET\_C\_DATA\_NUM, 326
  - NetCounter, 327
  - NetCounters, 327
  - NetModel, 328
  - NetRule, 328
  - NetRules, 328
  - NetStatsCounter, 328
  - NetSupport, 328
  - Network, 328
  - NETWORK\_COUNTER, 326
  - NETWORK\_COUNTER\_LAMBDA, 326
  - NETWORK\_RULE, 326
  - NETWORK\_RULE\_LAMBDA, 327

- NetworkDense, 329
- NETWORKDENSE\_COUNTER\_LAMBDA, 327
- rules\_zerodiag, 329
- NETWORK\_COUNTER
  - DEFMArray counters, 29
  - network.hpp, 326
- NETWORK\_COUNTER\_LAMBDA
  - network.hpp, 326
- NETWORK\_RULE
  - network.hpp, 326
- NETWORK\_RULE\_LAMBDA
  - network.hpp, 327
- NetworkData, 187
  - ~NetworkData, 188
  - directed, 188
  - NetworkData, 187, 188
  - vertex\_attr, 188
- NetworkDense
  - network.hpp, 329
- NETWORKDENSE\_COUNTER\_LAMBDA
  - network.hpp, 327
- next
  - Progress, 209
- nfuctions
  - Flock, 146
  - Geese, 162
- nfuns
  - Flock, 143
  - Geese, 157
- nleafs
  - Flock, 143
  - Geese, 157
- nnodes
  - Flock, 143
  - Geese, 158
- nnozero
  - BArray< Cell\_Type, Data\_Type >, 51
  - BArrayDense< Cell\_Type, Data\_Type >, 71
- Node, 189
  - ~Node, 191
  - annotations, 192
  - array, 192
  - arrays, 192
  - arrays\_valid, 193
  - duplication, 193
  - get\_parent, 191
  - id, 193
  - is\_leaf, 192
  - narray, 193
  - Node, 190, 191
  - noffspring, 192
  - offspring, 193
  - ord, 194
  - parent, 194
  - probability, 194
  - subtree\_prob, 194
  - visited, 194
- NodeData, 195
  - blengths, 196
  - duplication, 196
  - NodeData, 195
  - states, 196
- nodes
  - Geese, 162
- noexcept
  - counters-meat.hpp, 308
- noffspring
  - Node, 192
- NONE
  - CHECK, 40
  - EXISTS, 42
- normalizing\_constants
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 181
- nrow
  - BArray< Cell\_Type, Data\_Type >, 51
  - BArrayDense< Cell\_Type, Data\_Type >, 71
  - Phylo rules, 18
- nrules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 175
- nrules\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 175
- nterms
  - Flock, 144
  - Geese, 158
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 175
- ntrees
  - Flock, 144
- num
  - DEFMCounterData, 127
  - DEFMRuleData, 134
- numbers
  - DEFMCounterData, 128
  - DEFMRuleData, 135
  - NetCounterData, 186
- obs\_start
  - DEFMData, 131
- observed\_counts
  - Geese, 158
- offspring
  - Node, 193
- ONE
  - CHECK, 40
  - EXISTS, 42
- operator BArrayRow< Cell\_Type, Data\_Type >
  - BArrayRow< Cell\_Type, Data\_Type >, 92
- operator BArrayRow\_const< Cell\_Type, Data\_Type >
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 94
- operator Cell\_Type



- BArrayCell< Cell\_Type, Data\_Type >, 58
- BArrayCell\_const< Cell\_Type, Data\_Type >, 61
- BArrayDenseCell< Cell\_Type, Data\_Type >, 79
- Cell< Cell\_Type >, 106
- operator std::vector< Cell\_Type >
  - BArrayVector< Cell\_Type, Data\_Type >, 98
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 101
- operator!=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 61
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 94
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 101
  - Cell< Cell\_Type >, 106
- operator<
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 61
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 94
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 102
- operator<=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 61
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 95
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 102
- operator>
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 61
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 95
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 102
- operator>=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 62
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 95
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 102
- operator\*=
  - BArray< Cell\_Type, Data\_Type >, 51
  - BArrayCell< Cell\_Type, Data\_Type >, 58
  - BArrayDense< Cell\_Type, Data\_Type >, 71
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 79
  - BArrayRow< Cell\_Type, Data\_Type >, 92
  - BArrayVector< Cell\_Type, Data\_Type >, 98
- operator()
  - BArray< Cell\_Type, Data\_Type >, 51
  - barray-meat-operators.hpp, 237
  - BArrayDense< Cell\_Type, Data\_Type >, 71
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 83
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 85
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 88
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 90
  - Flock, 144
  - Phylo rules, 18
  - PhyloCounterData, 198
  - PhyloRuleDynData, 200
  - Rule< Array\_Type, Data\_Type >, 212
  - Rules< Array\_Type, Data\_Type >, 215
  - vecHasher< T >, 231
- operator+=
  - BArray< Cell\_Type, Data\_Type >, 52
  - BArrayCell< Cell\_Type, Data\_Type >, 58
  - BArrayDense< Cell\_Type, Data\_Type >, 71, 72
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 79
  - BArrayRow< Cell\_Type, Data\_Type >, 92
  - BArrayVector< Cell\_Type, Data\_Type >, 98
- operator-=
  - BArray< Cell\_Type, Data\_Type >, 52
  - BArrayCell< Cell\_Type, Data\_Type >, 59
  - BArrayDense< Cell\_Type, Data\_Type >, 72
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 80
  - BArrayRow< Cell\_Type, Data\_Type >, 92
  - BArrayVector< Cell\_Type, Data\_Type >, 98
- operator/=
  - BArray< Cell\_Type, Data\_Type >, 53
  - BArrayCell< Cell\_Type, Data\_Type >, 59
  - BArrayDense< Cell\_Type, Data\_Type >, 72
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 80
  - BArrayRow< Cell\_Type, Data\_Type >, 93
  - BArrayVector< Cell\_Type, Data\_Type >, 98
- operator=
  - BArray< Cell\_Type, Data\_Type >, 53
  - BArrayCell< Cell\_Type, Data\_Type >, 59
  - BArrayDense< Cell\_Type, Data\_Type >, 73
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 80
  - BArrayRow< Cell\_Type, Data\_Type >, 93
  - BArrayVector< Cell\_Type, Data\_Type >, 99
  - Cell< Cell\_Type >, 106, 107
  - Counter< Array\_Type, Data\_Type >, 113, 114
  - Counters< Array\_Type, Data\_Type >, 119
  - Geese, 158
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 175
  - Rules< Array\_Type, Data\_Type >, 216
- operator==
  - BArray< Cell\_Type, Data\_Type >, 53
  - BArrayCell< Cell\_Type, Data\_Type >, 59
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 61
  - BArrayDense< Cell\_Type, Data\_Type >, 73
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 80
  - BArrayRow< Cell\_Type, Data\_Type >, 93
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 95
  - BArrayVector< Cell\_Type, Data\_Type >, 99
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 102
  - Cell< Cell\_Type >, 107
- operator[]
  - Counters< Array\_Type, Data\_Type >, 120
  - PhyloCounterData, 198
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 205
- ord
  - Node, 194
- out\_of\_range
  - BArray< Cell\_Type, Data\_Type >, 53
  - BArrayDense< Cell\_Type, Data\_Type >, 73
- params



- model-meat.hpp, 354
- params\_last
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 181
- parent
  - Node, 194
- parse\_polytomies
  - Flock, 144
  - Geese, 158
- Phylo counters, 30
  - counter\_co\_opt, 31
  - counter\_cogain, 32
  - counter\_gains, 32
  - counter\_gains\_from\_0, 32
  - counter\_gains\_k\_offspring, 33
  - counter\_genes\_changing, 33
  - counter\_k\_genes\_changing, 33
  - counter\_less\_than\_p\_prop\_genes\_changing, 33
  - counter\_longest, 34
  - counter\_loss, 34
  - counter\_maxfun, 34
  - counter\_neofun, 34
  - counter\_neofun\_a2b, 35
  - counter\_overall\_changes, 35
  - counter\_overall\_gains, 35
  - counter\_overall\_gains\_from\_0, 35
  - counter\_overall\_loss, 36
  - counter\_pairwise\_first\_gain, 36
  - counter\_pairwise\_neofun\_singlefun, 36
  - counter\_pairwise\_overall\_change, 36
  - counter\_pairwise\_preserving, 37
  - counter\_preserve\_pseudogene, 37
  - counter\_prop\_genes\_changing, 37
  - counter\_subfun, 37
- Phylo rules, 14
  - DEFM\_COUNTER, 15
  - DEFM\_COUNTER\_LAMBDA, 15
  - DEFM\_RULE, 15
  - DEFM\_RULE\_LAMBDA, 16
  - DEFM\_RULEDYN\_LAMBDA, 16
  - DEFMCounter, 17
  - DEFMCounters, 17
  - DEFMModel, 17
  - DEFMRule, 17
  - DEFMRuleDyn, 17
  - DEFMRules, 17
  - DEFMRulesDyn, 18
  - DEFMStatsCounter, 18
  - DEFMSupport, 18
  - MAKE\_DEFM\_HASHER, 16
  - ncol, 18
  - nrow, 18
  - operator(), 18
  - print, 19
  - rule\_dyn\_limit\_changes, 19
- phylo.hpp
  - DEFAULT\_DUPLICATION, 332
  - DUPL\_DUPL, 332
  - DUPL\_EITH, 332
  - DUPL\_SPEC, 332
  - get\_last\_name, 337
  - IF\_MATCHES, 332
  - IF\_NOTMATCHES, 333
  - IS\_DUPLICATION, 333
  - IS\_EITHER, 333
  - IS\_SPECIATION, 333
  - MAKE\_DUPL\_VARS, 333
  - PHYLO\_CHECK\_MISSING, 334
  - PHYLO\_COUNTER\_LAMBDA, 334
  - PHYLO\_RULE\_DYN\_LAMBDA, 334
  - PhyloArray, 335
  - PhyloCounter, 335
  - PhyloCounters, 335
  - PhyloModel, 335
  - PhyloPowerSet, 335
  - PhyloRule, 335
  - PhyloRuleData, 336
  - PhyloRuleDyn, 336
  - PhyloRules, 336
  - PhyloRulesDyn, 336
  - PhyloStatsCounter, 336
  - PhyloSupport, 336
  - PHYLO\_CHECK\_MISSING
    - phylo.hpp, 334
  - PHYLO\_COUNTER\_LAMBDA
    - phylo.hpp, 334
  - PHYLO\_RULE\_DYN\_LAMBDA
    - phylo.hpp, 334
  - PhyloArray
    - phylo.hpp, 335
  - PhyloCounter
    - phylo.hpp, 335
  - PhyloCounterData, 196
    - at, 197
    - begin, 197
    - empty, 197
    - end, 198
    - get\_counters, 198
    - operator(), 198
    - operator[], 198
    - PhyloCounterData, 197
    - push\_back, 198
    - reserve, 198
    - shrink\_to\_fit, 199
    - size, 199
  - PhyloCounters
    - phylo.hpp, 335
  - PhyloModel
    - phylo.hpp, 335
  - PhyloPowerSet
    - phylo.hpp, 335
  - PhyloRule
    - phylo.hpp, 335
  - PhyloRuleData
    - phylo.hpp, 336

- PhyloRuleDyn
  - phylo.hpp, 336
- PhyloRuleDynData, 199
  - ~PhyloRuleDynData, 200
  - counts, 200
  - duplication, 200
  - lb, 201
  - operator(), 200
  - PhyloRuleDynData, 200
  - pos, 201
  - ub, 201
- PhyloRules
  - phylo.hpp, 336
- PhyloRulesDyn
  - phylo.hpp, 336
- PhyloStatsCounter
  - phylo.hpp, 336
- PhyloSupport
  - phylo.hpp, 336
- POS
  - barraydense-meat-operators.hpp, 258
  - barraydense-meat.hpp, 263
  - barraydensecell-bones.hpp, 281
  - barraydensecell-meat.hpp, 282
  - barraydensecol-bones.hpp, 283
  - barraydenserow-bones.hpp, 284
- pos
  - PhyloRuleDynData, 201
- POS\_N
  - barraydense-meat-operators.hpp, 259
  - barraydense-meat.hpp, 263
  - barraydensecol-bones.hpp, 283
  - barraydenserow-bones.hpp, 284
- PowerSet
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 203
- PowerSet< Array\_Type, Data\_Rule\_Type >, 201
  - ~PowerSet, 203
  - add\_rule, 203, 204
  - begin, 204
  - calc, 204
  - coordinates\_free, 206
  - coordinates\_locked, 206
  - data, 206
  - EmptyArray, 206
  - end, 204
  - get\_data, 204
  - get\_data\_ptr, 205
  - init\_support, 205
  - M, 206
  - N, 207
  - n\_free, 207
  - n\_locked, 207
  - operator[], 205
  - PowerSet, 203
  - reset, 205
  - rules, 207
  - rules\_deleted, 207
  - size, 205
- predict
  - Geese, 159
- predict\_backend
  - Geese, 159
- predict\_exhaust
  - Geese, 159
- predict\_exhaust\_backend
  - Geese, 159
- predict\_sim
  - Geese, 160
- print
  - BArray< Cell\_Type, Data\_Type >, 53
  - BArrayDense< Cell\_Type, Data\_Type >, 73
  - DEFM, 125
  - Flock, 145
  - FreqTable< T >, 149
  - Geese, 160
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 176
  - Phylo rules, 19
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 226
- print\_observed\_counts
  - Geese, 160
- print\_stats
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 176
- printf\_barry
  - barraydense-meat.hpp, 273
  - barry-configuration.hpp, 289
- probability
  - Node, 194
- probs
  - model-meat.hpp, 354
- Progress, 208
  - ~Progress, 208
  - end, 209
  - next, 209
  - Progress, 208
- progress.hpp
  - BARRY\_PROGRESS\_BAR\_WIDTH, 368
- pset\_arrays
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 181
  - model-meat.hpp, 354
- pset\_loc
  - Geese, 162
- pset\_probs
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 182
- pset\_stats
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 182

- 182
- push\_back
  - model-meat.hpp, 348, 349
  - PhyloCounterData, 198
- r
  - model-meat.hpp, 355
- README.md, 393
- reduced\_sequence
  - Geese, 162
- rengine
  - Flock, 146
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 182
- report
  - barray-meat.hpp, 254
  - barraydense-meat.hpp, 279
- res
  - counters-meat.hpp, 308
- reserve
  - BArray< Cell\_Type, Data\_Type >, 54
  - BArrayDense< Cell\_Type, Data\_Type >, 73
  - FreqTable< T >, 149
  - PhyloCounterData, 198
- reset
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 205
- reset\_array
  - StatsCounter< Array\_Type, Data\_Type >, 219
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 226, 227
- resize
  - BArray< Cell\_Type, Data\_Type >, 54
  - barray-meat.hpp, 249
  - BArrayDense< Cell\_Type, Data\_Type >, 74
  - barraydense-meat.hpp, 273, 274
  - Entries< Cell\_Type >, 138
  - statscounter-meat.hpp, 372
- return
  - barray-meat.hpp, 249, 254
  - barraydense-meat.hpp, 279
  - counters-meat.hpp, 309
  - model-meat.hpp, 349, 355
  - statscounter-meat.hpp, 376
  - support-meat.hpp, 386
- rhs
  - barray-meat-operators.hpp, 237
- rm\_cell
  - BArray< Cell\_Type, Data\_Type >, 54
  - BArrayDense< Cell\_Type, Data\_Type >, 74
  - barraydense-meat.hpp, 274
  - support-meat.hpp, 380
- ROW
  - barray-meat-operators.hpp, 235
  - barray-meat.hpp, 241, 249, 250
  - barraydense-meat-operators.hpp, 259
  - barraydense-meat.hpp, 264
- row
  - BArray< Cell\_Type, Data\_Type >, 54
  - BArrayDense< Cell\_Type, Data\_Type >, 74
- row0
  - barray-meat.hpp, 255
- Row\_type
  - typedefs.hpp, 391
- rowsum
  - BArrayDense< Cell\_Type, Data\_Type >, 74
- Rule
  - Rule< Array\_Type, Data\_Type >, 210
- Rule< Array\_Type, Data\_Type >, 209
  - ~Rule, 210
  - D, 211
  - get\_description, 211
  - get\_name, 211
  - operator(), 212
  - Rule, 210
- rule\_dyn\_limit\_changes
  - Phylo rules, 19
- rule\_fun\_
  - model-meat.hpp, 355
- rule\_fun\_default
  - rules-bones.hpp, 369
- Rule\_fun\_type
  - typedefs.hpp, 391
- RULE\_FUNCTION
  - barry.hpp, 295
  - geese-bones.hpp, 361
- RULE\_LAMBDA
  - barry.hpp, 295
- Rules
  - Rules< Array\_Type, Data\_Type >, 213
- rules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 182
  - model-meat.hpp, 355
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 207
  - support-meat.hpp, 386
- Rules< Array\_Type, Data\_Type >, 212
  - ~Rules, 213
  - add\_rule, 214
  - begin, 214
  - end, 214
  - get\_descriptions, 214
  - get\_names, 215
  - get\_seq, 215
  - operator(), 215
  - operator=, 216
  - Rules, 213
  - size, 216
- rules-bones.hpp
  - rule\_fun\_default, 369
- rules\_
  - model-meat.hpp, 355
  - support-meat.hpp, 386
- rules\_deleted
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 207

- rules\_dont\_become\_zero
  - DEFMArray counters, [29](#)
- rules\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [183](#)
  - model-meat.hpp, [356](#)
  - support-meat.hpp, [387](#)
- rules\_markov\_fixed
  - DEFMArray counters, [30](#)
- rules\_zerodiag
  - network.hpp, [329](#)
- sample
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [176](#)
- search
  - barray-meat.hpp, [255](#)
- sequence
  - Geese, [162](#)
- set\_counters
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [177](#)
  - model-meat.hpp, [349](#)
  - StatsCounter< Array\_Type, Data\_Type >, [220](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [227](#)
- set\_data
  - BArray< Cell\_Type, Data\_Type >, [54](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [74](#)
- set\_hasher
  - Counter< Array\_Type, Data\_Type >, [114](#)
- set\_names
  - DEFM, [125](#)
- set\_engine
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [177](#)
- set\_rules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [177](#)
  - model-meat.hpp, [349](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [227](#)
- set\_rules\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [177](#)
  - model-meat.hpp, [349](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [227](#)
- set\_seed
  - Flock, [145](#)
- Geese, [160](#)
- Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [177](#)
- set\_transform\_model
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [178](#)
- shrink\_to\_fit
  - PhyloCounterData, [199](#)
- simulate
  - DEFM, [125](#)
  - Geese, [160](#)
- size
  - BArrayDenseCol< Cell\_Type, Data\_Type >, [83](#)
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, [86](#)
  - BArrayDenseRow< Cell\_Type, Data\_Type >, [88](#)
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, [90](#)
  - BArrayVector< Cell\_Type, Data\_Type >, [99](#)
  - BArrayVector\_const< Cell\_Type, Data\_Type >, [103](#)
  - Counters< Array\_Type, Data\_Type >, [120](#)
  - FreqTable< T >, [149](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [178](#)
  - model-meat.hpp, [349](#)
  - PhyloCounterData, [199](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [205](#)
  - Rules< Array\_Type, Data\_Type >, [216](#)
  - StatsCounter< Array\_Type, Data\_Type >, [220](#)
- size\_unique
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [178](#)
- sort\_array
  - typedefs.hpp, [391](#)
- source
  - barray-meat.hpp, [255](#)
  - barraydense-meat.hpp, [279](#)
  - Entries< Cell\_Type >, [139](#)
- states
  - NodeData, [196](#)
- Statistical Models, [13](#)
- stats
  - model-meat.hpp, [356](#)
- stats\_bank
  - support-meat.hpp, [387](#)
- stats\_support
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [183](#)
- stats\_support\_n\_arrays
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [183](#)

- model-meat.hpp, 356
- stats\_target
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 183
- StatsCounter
  - StatsCounter< Array\_Type, Data\_Type >, 217, 218
- StatsCounter< Array\_Type, Data\_Type >, 216
  - ~StatsCounter, 218
  - add\_counter, 218
  - count\_all, 218
  - count\_current, 219
  - count\_init, 219
  - get\_counters, 219
  - get\_descriptions, 219
  - get\_names, 219
  - reset\_array, 219
  - set\_counters, 220
  - size, 220
  - StatsCounter, 217, 218
- statscounter-meat.hpp
  - clear, 372
  - counter, 374
  - counter\_deleted, 374
  - counters, 375
  - counters\_, 375
  - current\_stats, 375
  - EmptyArray, 375
  - f\_, 375
  - for, 372
  - j, 376
  - resize, 372
  - return, 376
  - STATSCOUNTER\_TEMPLATE, 372–374
  - STATSCOUNTER\_TEMPLATE\_ARGS, 372
  - STATSCOUNTER\_TYPE, 372
- STATSCOUNTER\_TEMPLATE
  - statscounter-meat.hpp, 372–374
- STATSCOUNTER\_TEMPLATE\_ARGS
  - statscounter-meat.hpp, 372
- STATSCOUNTER\_TYPE
  - statscounter-meat.hpp, 372
- store\_psets
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 178
- subtree\_prob
  - Node, 194
- Support
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 222, 223
- Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 220
  - ~Support, 223
  - add\_counter, 223
  - add\_rule, 223, 224
  - add\_rule\_dyn, 224
  - calc, 224
  - change\_stats, 227
  - cooriantes\_n\_free, 228
  - cooriantes\_n\_locked, 228
  - coordinates\_free, 228
  - coordinates\_locked, 228
  - current\_stats, 228
  - delete\_counters, 229
  - delete\_rules, 229
  - delete\_rules\_dyn, 229
  - eval\_rules\_dyn, 225
  - get\_counters, 225
  - get\_counts, 225
  - get\_current\_stats, 225
  - get\_data, 225
  - get\_rules, 226
  - get\_rules\_dyn, 226
  - hashes, 229
  - hashes\_initialized, 229
  - init\_support, 226
  - M, 230
  - max\_num\_elements, 230
  - N, 230
  - n\_counters, 230
  - print, 226
  - reset\_array, 226, 227
  - set\_counters, 227
  - set\_rules, 227
  - set\_rules\_dyn, 227
  - Support, 222, 223
- support-meat.hpp
  - array\_bank, 384
  - BARRY\_SUPPORT\_MEAT\_HPP, 378
  - calc\_backend\_dense, 379
  - calc\_backend\_sparse, 379
  - change\_stats\_different, 384
  - coord\_i, 384
  - coord\_j, 384
  - counters, 384
  - counters\_, 385
  - delete\_counters, 385
  - delete\_rules, 385
  - delete\_rules\_dyn, 385
  - else, 385
  - f\_, 386
  - for, 379
  - hashes, 386
  - if, 379, 380
  - insert\_cell, 380
  - return, 386
  - rm\_cell, 380
  - rules, 386
  - rules\_, 386
  - rules\_dyn, 387
  - stats\_bank, 387
  - SUPPORT\_TEMPLATE, 378, 381–384
  - SUPPORT\_TEMPLATE\_ARGS, 378

- SUPPORT\_TYPE, 379
- tmp\_chng, 387
- support\_fun
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 184
- support\_size
  - Flock, 145
  - Geese, 161
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 179
- SUPPORT\_TEMPLATE
  - support-meat.hpp, 378, 381–384
- SUPPORT\_TEMPLATE\_ARGS
  - support-meat.hpp, 378
- SUPPORT\_TYPE
  - support-meat.hpp, 379
- swap\_cells
  - BArray< Cell\_Type, Data\_Type >, 55
  - BArrayDense< Cell\_Type, Data\_Type >, 75
- swap\_cols
  - BArray< Cell\_Type, Data\_Type >, 55
  - BArrayDense< Cell\_Type, Data\_Type >, 75
- swap\_rows
  - BArray< Cell\_Type, Data\_Type >, 55
  - BArrayDense< Cell\_Type, Data\_Type >, 75
- target
  - barray-meat.hpp, 255
  - barraydense-meat.hpp, 280
  - Entries< Cell\_Type >, 139
- temp\_stats
  - model-meat.hpp, 349
- this
  - barray-meat-operators.hpp, 238
- tmp\_chng
  - support-meat.hpp, 387
- tmp\_counts
  - model-meat.hpp, 350
- TMP\_HASHER\_CALL
  - counters-meat.hpp, 299
- toggle\_cell
  - BArray< Cell\_Type, Data\_Type >, 55
  - BArrayDense< Cell\_Type, Data\_Type >, 75
- toggle\_lock
  - BArray< Cell\_Type, Data\_Type >, 55
  - BArrayDense< Cell\_Type, Data\_Type >, 76
- transform\_model
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 179
- transform\_model\_fun
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 184
- transform\_model\_term\_names
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 184
- transpose
  - BArray< Cell\_Type, Data\_Type >, 56
  - BArrayDense< Cell\_Type, Data\_Type >, 76
- TWO
  - CHECK, 41
  - EXISTS, 42
- typedefs.hpp
  - Col\_type, 390
  - Counter\_fun\_type, 390
  - Counts\_type, 390
  - Hasher\_fun\_type, 390
  - MapVec\_type, 391
  - Row\_type, 391
  - Rule\_fun\_type, 391
  - sort\_array, 391
  - vec\_equal, 392
  - vec\_equal\_approx, 392
  - vec\_inner\_prod, 392, 393
- ub
  - PhyloRuleDynData, 201
- UNKNOWN
  - EXISTS, 42
- UNI\_SUB
  - defm.hpp, 313
- update\_annotations
  - Geese, 161
- update\_normalizing\_constant
  - model-meat.hpp, 350
- urand
  - model-meat.hpp, 350
- v
  - barray-meat.hpp, 255
  - barraydense-meat.hpp, 280
- va\_end
  - barraydense-meat.hpp, 274
- va\_start
  - barraydense-meat.hpp, 275
- val
  - Entries< Cell\_Type >, 139
- val0
  - barraydense-meat.hpp, 280
- val1
  - barraydense-meat.hpp, 280
- value
  - barray-meat.hpp, 255
  - barraydense-meat.hpp, 280
  - Cell< Cell\_Type >, 107
- vec\_diff
  - geese-bones.hpp, 361
- vec\_equal
  - typedefs.hpp, 392
- vec\_equal\_approx
  - typedefs.hpp, 392
- vec\_inner\_prod
  - typedefs.hpp, 392, 393
- vecHasher< T >, 231

- operator(), [231](#)
- vector\_caster
  - geese-bones.hpp, [361](#)
- vertex\_attr
  - NetworkData, [188](#)
- visited
  - BArray< Cell\_Type, Data\_Type >, [57](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [77](#)
  - Cell< Cell\_Type >, [107](#)
  - Node, [194](#)
- with\_pset
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [185](#)
- X\_ncol
  - DEFMData, [132](#)
- X\_nrow
  - DEFMData, [132](#)
- ZERO\_CELL
  - barraydense-meat.hpp, [264](#)
  - barraydensecol-bones.hpp, [283](#)
  - barraydenserow-bones.hpp, [285](#)
- zero\_col
  - BArray< Cell\_Type, Data\_Type >, [56](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [76](#)
- zero\_row
  - BArray< Cell\_Type, Data\_Type >, [56](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [76](#)