barry: Your go-to motif accountant 0.0-1

Generated by Doxygen 1.9.1

1	Main Page	1
2	Module Index	5
	2.1 Modules	5
3	Hierarchical Index	7
	3.1 Class Hierarchy	7
4	Class Index	9
	4.1 Class List	9
5	File Index	11
	5.1 File List	11
6	Module Documentation	13
	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
	0.0.4.0 ruie_uyri_iiiriit_Ghanges()	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
<b>6.4.2.2</b> counter_ctriads() [1/2]	22
<b>6.4.2.3</b> 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
<b>6.4.2.9</b> counter_idegree() [1/2]	24
<b>6.4.2.10</b> counter_idegree() [2/2]	24
6.4.2.11 counter_idegree15() [1/2]	24
6.4.2.12 counter_idegree15() [2/2]	24
<b>6.4.2.13</b> counter_isolates() [1/2]	25
<b>6.4.2.14</b> counter_isolates() [2/2]	25
6.4.2.15 counter_istar2() [1/2]	25
<b>6.4.2.16</b> 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
<b>6.4.2.23</b> counter_odegree() [1/2]	27
<b>6.4.2.24</b> counter_odegree() [2/2]	27
6.4.2.25 counter_odegree15() [1/2]	27
<b>6.4.2.26</b> counter_odegree15() [2/2]	27
6.4.2.27 counter_ones()	27
6.4.2.28 counter_ostar2() [1/2]	28
<b>6.4.2.29</b> counter_ostar2() [2/2]	28
6.4.2.30 counter_transition()	28
6.4.2.31 counter_transition_formula()	29
<b>6.4.2.32</b> counter_ttriads() [1/2]	29
<b>6.4.2.33</b> 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
7 Namespace Documentation	39
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 UKNOWN	 . 42
8 Class Documentation	43
8.1 BArray < Cell_Type, Data_Type > Class Template Reference	
8.1.1 Detailed Description	
8.1.2 Constructor & Destructor Documentation	
8.1.2.1 BArray() [1/6]	
8.1.2.2 BArray() [2/6]	
8.1.2.3 BArray() [3/6]	
8.1.2.4 BArray() [4/6]	
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
<b>8.1.3.15 insert_cell()</b> [1/3]	 . 50
<b>8.1.3.16 insert_cell()</b> [2/3]	 . 50
<b>8.1.3.17 insert_cell()</b> [3/3]	 . 50
8.1.3.18 is_dense()	
8.1.3.19 is_empty()	
8.1.3.20 ncol()	
8.1.3.21 nnozero()	 . 51

8.1.3.22 nrow()	51
8.1.3.23 operator()() [1/2]	51
8.1.3.24 operator()() [2/2]	51
8.1.3.25 operator*=()	52
<b>8.1.3.26</b> operator+=() [1/3]	52
<b>8.1.3.27 operator+=()</b> [2/3]	52
<b>8.1.3.28 operator+=()</b> [3/3]	52
8.1.3.29 operator-=() [1/3]	52
8.1.3.30 operator-=() [2/3]	52
8.1.3.31 operator-=() [3/3]	53
8.1.3.32 operator/=()	53
8.1.3.33 operator=() [1/2]	53
8.1.3.34 operator=() [2/2]	53
8.1.3.35 operator==()	53
8.1.3.36 out_of_range()	53
8.1.3.37 print()	54
8.1.3.38 reserve()	54
8.1.3.39 resize()	54
8.1.3.40 rm_cell()	54
8.1.3.41 row()	54
8.1.3.42 set_data()	54
8.1.3.43 swap_cells()	55
8.1.3.44 swap_cols()	55
8.1.3.45 swap_rows()	55
8.1.3.46 toggle_cell()	55
8.1.3.47 toggle_lock()	56
8.1.3.48 transpose()	56
8.1.3.49 zero_col()	56
8.1.3.50 zero_row()	56
8.1.4 Friends And Related Function Documentation	56
8.1.4.1 BArrayCell< Cell_Type, Data_Type >	56
8.1.4.2 BArrayCell_const< Cell_Type, Data_Type >	57
8.1.5 Member Data Documentation	57
8.1.5.1 visited	57
8.2 BArrayCell< Cell_Type, Data_Type > Class Template Reference	57
8.2.1 Detailed Description	57
8.2.2 Constructor & Destructor Documentation	58
8.2.2.1 BArrayCell()	58
8.2.2.2 ~BArrayCell()	58
8.2.3 Member Function Documentation	58
8.2.3.1 operator Cell_Type()	58
8.2.3.2 operator*=()	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
<b>8.4.2.2 BArrayDense()</b> [2/6]	65
<b>8.4.2.3 BArrayDense()</b> [3/6]	66
<b>8.4.2.4 BArrayDense()</b> [4/6]	66
<b>8.4.2.5 BArrayDense()</b> [5/6]	66
<b>8.4.2.6 BArrayDense()</b> [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
<b>8.4.3.3 col()</b> [2/2]	67
8.4.3.4 colsum()	67
8.4.3.5 D() [1/2]	68
<b>8.4.3.6 D()</b> [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 get_entries()	69
8.4.3.15 get_row_vec() [1/2]	69
<b>8.4.3.16 get_row_vec()</b> [2/2]	70
8.4.3.17 insert_cell() [1/2]	70
8.4.3.18 insert_cell() [2/2]	70
8.4.3.19 is_dense()	70
8.4.3.20 is_empty()	70
8.4.3.21 ncol()	71
8.4.3.22 nnozero()	71
8.4.3.23 nrow()	71
8.4.3.24 operator()() [1/2]	71
<b>8.4.3.25 operator()()</b> [2/2]	71
8.4.3.26 operator*=()	71
8.4.3.27 operator+=() [1/3]	72
<b>8.4.3.28</b> operator+=() [2/3]	72
<b>8.4.3.29</b> operator+=() [3/3]	72
8.4.3.30 operator-=() [1/3]	72
8.4.3.31 operator-=() [2/3]	72
8.4.3.32 operator-=() [3/3]	72
8.4.3.33 operator/=()	73
8.4.3.34 operator=() [1/2]	73
<b>8.4.3.35</b> operator=() [2/2]	73
8.4.3.36 operator==()	73
8.4.3.37 out_of_range()	73
8.4.3.38 print()	73
8.4.3.39 reserve()	74
8.4.3.40 resize()	74
8.4.3.41 rm_cell()	74
<b>8.4.3.42 row()</b> [1/2]	74
<b>8.4.3.43 row()</b> [2/2]	74
8.4.3.44 rowsum()	74
8.4.3.45 set_data()	75
8.4.3.46 swap_cells()	75
8.4.3.47 swap_cols()	75
8.4.3.48 swap_rows()	75
8.4.3.49 toggle_cell()	76
8.4.3.50 toggle_lock()	76
8.4.3.51 transpose()	76
8.4.3.52 zero_col()	76
8.4.3.53 zero_row()	76
Friends And Related Function Documentation	76
8.4.4.1 BArrayDenseCell< Cell_Type, Data_Type >	77

8.4.4

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()
8.14.3.2 end()
8.14.3.3 is_col()
8.14.3.4 is_row()
8.14.3.5 operator std::vector < Cell_Type >()
8.14.3.6 operator"!=()
8.14.3.7 operator<()
8.14.3.8 operator<=()
8.14.3.9 operator==()
8.14.3.10 operator>()
8.14.3.11 operator>=()
8.14.3.12 size()
8.15 Cell< Cell_Type > Class Template Reference
8.15.1 Detailed Description
8.15.2 Constructor & Destructor Documentation
8.15.2.1 Cell() [1/7]
8.15.2.2 Cell() [2/7]
8.15.2.3 ~Cell()
8.15.2.4 Cell() [3/7]
8.15.2.5 Cell() [4/7]
8.15.2.6 Cell() [5/7]
8.15.2.7 Cell() [6/7]
8.15.2.8 Cell() [7/7]
8.15.3 Member Function Documentation
8.15.3.1 add() [1/4]
8.15.3.2 add() [2/4]
8.15.3.3 add() [3/4]
8.15.3.4 add() [4/4] 100
8.15.3.5 operator Cell_Type()
8.15.3.6 operator"!=()
8.15.3.7 operator=() [1/2]
8.15.3.8 operator=() [2/2]
8.15.3.9 operator==()
8.15.4 Member Data Documentation
8.15.4.1 active
8.15.4.2 value
8.15.4.3 visited
8.16 Cell_const< Cell_Type > Class Template Reference
8.16.1 Detailed Description
8.17 ConstBArrayRowlter< Cell_Type, Data_Type > Class Template Reference
8.17.1 Detailed Description
8.17.2 Constructor & Destructor Documentation 10

8.17.2.1 ConstBArrayRowlter()	09
$8.17.2.2 \sim ConstBArrayRowlter() \ \dots \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ $	09
8.17.3 Member Data Documentation	09
8.17.3.1 Array	10
8.17.3.2 current_col	10
8.17.3.3 current_row	10
8.17.3.4 iter	10
8.18 Counter< Array_Type, Data_Type > Class Template Reference	10
8.18.1 Detailed Description	11
8.18.2 Constructor & Destructor Documentation	12
8.18.2.1 Counter() [1/4]	12
8.18.2.2 Counter() [2/4]	12
8.18.2.3 Counter() [3/4]	12
8.18.2.4 Counter() [4/4]	12
8.18.2.5 ~Counter()	13
8.18.3 Member Function Documentation	13
8.18.3.1 count()	13
8.18.3.2 get_description()	13
8.18.3.3 get_hasher()	13
8.18.3.4 get_name()	13
8.18.3.5 init()	13
8.18.3.6 operator=() [1/2]	14
8.18.3.7 operator=() [2/2]	14
8.18.3.8 set_hasher()	14
8.18.4 Member Data Documentation	14
8.18.4.1 count_fun	14
8.18.4.2 data	15
8.18.4.3 desc	15
8.18.4.4 hasher_fun	15
8.18.4.5 init_fun	15
8.18.4.6 name	15
8.19 Counters < Array_Type, Data_Type > Class Template Reference	16
8.19.1 Detailed Description	16
8.19.2 Constructor & Destructor Documentation	16
8.19.2.1 Counters() [1/3]	17
8.19.2.2 ~Counters()	17
8.19.2.3 Counters() [2/3]	17
8.19.2.4 Counters() [3/3]	17
8.19.3 Member Function Documentation	17
8.19.3.1 add_counter() [1/2]	18
8.19.3.2 add_counter() [2/2]	18
8.19.3.3 add_hash()	18

8.19.3.4 gen_hash()	8
8.19.3.5 get_descriptions()	9
8.19.3.6 get_names()	9
8.19.3.7 operator=() [1/2]	9
8.19.3.8 operator=() [2/2]	9
8.19.3.9 operator[]()	20
8.19.3.10 size()	20
8.20 DEFM Class Reference	1:1
8.20.1 Detailed Description	2
8.20.2 Constructor & Destructor Documentation	2
8.20.2.1 DEFM()	2
8.20.3 Member Function Documentation	2
8.20.3.1 get_ID()	2
8.20.3.2 get_m_order()	2
8.20.3.3 get_model()	23
8.20.3.4 get_n_covars()	23
8.20.3.5 get_n_obs()	23
8.20.3.6 get_n_rows()	23
8.20.3.7 get_n_y()	23
8.20.3.8 get_X()	23
8.20.3.9 get_X_names()	4
8.20.3.10 get_Y()	4
8.20.3.11 get_Y_names()	4
8.20.3.12 init()	4
8.20.3.13 is_motif()	:4
8.20.3.14 likelihood()	4
8.20.3.15 logodds()	:5
8.20.3.16 motif_census()	:5
8.20.3.17 print()	:5
8.20.3.18 set_names()	:5
8.20.3.19 simulate()	:5
8.21 DEFMCounterData Class Reference	:6
8.21.1 Detailed Description	:6
8.21.2 Constructor & Destructor Documentation	:6
8.21.2.1 DEFMCounterData() [1/2]	:6
8.21.2.2 DEFMCounterData() [2/2]	:7
8.21.2.3 ~DEFMCounterData()	:7
8.21.3 Member Function Documentation	:7
8.21.3.1 idx()	:7
8.21.3.2 is_true()	:7
8.21.3.3 num()	:7
8 21 4 Mambar Data Documentation	Ω

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 DEFMData Class Reference	129
8.22.1 Detailed Description	130
8.22.2 Constructor & Destructor Documentation	130
8.22.2.1 DEFMData() [1/2]	130
8.22.2.2 DEFMData() [2/2]	130
8.22.2.3 ~DEFMData()	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
<b>8.23.2.1 DEFMRuleData()</b> [1/3]	133
<b>8.23.2.2 DEFMRuleData()</b> [2/3]	133
<b>8.23.2.3 DEFMRuleData()</b> [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	37
8.25.1 Detailed Description	37
8.25.2 Constructor & Destructor Documentation	38
8.25.2.1 Entries() [1/2]	38
8.25.2.2 Entries() [2/2]	38
8.25.2.3 ~Entries()	38
8.25.3 Member Function Documentation	38
8.25.3.1 resize()	38
8.25.4 Member Data Documentation	39
8.25.4.1 source	39
8.25.4.2 target	39
8.25.4.3 val	39
8.26 Flock Class Reference	39
8.26.1 Detailed Description	40
8.26.2 Constructor & Destructor Documentation	41
8.26.2.1 Flock()	41
8.26.2.2 ~Flock()	41
8.26.3 Member Function Documentation	41
8.26.3.1 add_data()	41
8.26.3.2 colnames()	42
8.26.3.3 get_counters()	42
8.26.3.4 get_model()	42
8.26.3.5 get_stats_support()	42
8.26.3.6 get_stats_target()	42
8.26.3.7 get_support_fun()	42
8.26.3.8 init()	43
8.26.3.9 likelihood_joint()	43
8.26.3.10 nfuns()	43
8.26.3.11 nleafs()	43
8.26.3.12 nnodes()	44
8.26.3.13 nterms()	44
8.26.3.14 ntrees()	44
8.26.3.15 operator()()	44
8.26.3.16 parse_polytomies()	44
8.26.3.17 print()	45
8.26.3.18 set_seed()	45
8.26.3.19 support_size()	45
8.26.4 Member Data Documentation	45
8.26.4.1 dat	45
8.26.4.2 initialized	46
8.26.4.3 model	46
8.26.4.4 nfunctions	46

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

8.28.3.18 nleafs()	 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_rengine	 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	
plate Reference	163
8.29.1 Detailed Description	167
8.29.2 Constructor & Destructor Documentation	
8.29.2.1 Model() [1/3]	168
8.29.2.2 Model() [2/3]	
8.29.2.3 Model() [3/3]	168
8.29.2.4 ~Model()	168
8.29.3 Member Function Documentation	
8.29.3.1 add_array()	
8.29.3.2 add_counter() [1/2]	
8.29.3.3 add_counter() [2/2]	169
8.29.3.4 add_hasher()	
<b>8.29.3.5 add_rule()</b> [1/2]	 169

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

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_rengine	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
<b>8.31.2.3 NetworkData()</b> [3/3]	188

8.31.2.4 ~NetworkData()
8.31.3 Member Data Documentation
8.31.3.1 directed
8.31.3.2 vertex_attr
8.32 Node Class Reference
8.32.1 Detailed Description
8.32.2 Constructor & Destructor Documentation
8.32.2.1 Node() [1/5]
8.32.2.2 Node() [2/5]
8.32.2.3 Node() [3/5]
8.32.2.4 Node() [4/5]
8.32.2.5 Node() [5/5]
8.32.2.6 ~Node()
8.32.3 Member Function Documentation
8.32.3.1 get_parent()
8.32.3.2 is_leaf()
8.32.3.3 noffspring()
8.32.4 Member Data Documentation
8.32.4.1 annotations
8.32.4.2 array
8.32.4.3 arrays
8.32.4.4 arrays_valid
8.32.4.5 duplication
8.32.4.6 id
8.32.4.7 narray
8.32.4.8 offspring
8.32.4.9 ord
8.32.4.10 parent
8.32.4.11 probability
8.32.4.12 subtree_prob
8.32.4.13 visited
8.33 NodeData Class Reference
8.33.1 Detailed Description
8.33.2 Constructor & Destructor Documentation
8.33.2.1 NodeData()
8.33.3 Member Data Documentation
8.33.3.1 blengths
8.33.3.2 duplication
8.33.3.3 states
8.34 PhyloCounterData Class Reference
8.34.1 Detailed Description
8 3/4 2 Constructor & Destructor Documentation

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 Data Documentation	200
8.35.3.1 counts	200
8.35.3.2 duplication	200
8.35.3.3 lb	201
8.35.3.4 pos	201
8.35.3.5 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
<b>8.36.3.1 add_rule()</b> [1/2]	204
<b>8.36.3.2 add_rule()</b> [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
<b>8.38.3.2</b> get_description() [1/2]		211
<b>8.38.3.3 get_description()</b> [2/2]		211
<b>8.38.3.4 get_name()</b> [1/2]		211
<b>8.38.3.5 get_name()</b> [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()		213
8.39.3 Member Function Documentation		

8.39.3.2 add_rule() [2/2]	214
8.39.3.3 get_descriptions()	214
8.39.3.4 get_names()	214
8.39.3.5 get_seq()	214
8.39.3.6 operator()()	215
8.39.3.7 operator=()	215
8.39.3.8 size()	216
8.40 StatsCounter< Array_Type, Data_Type > Class Template Reference	216
8.40.1 Detailed Description	216
8.40.2 Constructor & Destructor Documentation	217
8.40.2.1 StatsCounter() [1/3]	217
8.40.2.2 StatsCounter() [2/3]	217
8.40.2.3 StatsCounter() [3/3]	217
8.40.2.4 ~StatsCounter()	217
8.40.3 Member Function Documentation	218
8.40.3.1 add_counter()	218
8.40.3.2 count_all()	218
8.40.3.3 count_current()	218
8.40.3.4 count_init()	218
8.40.3.5 get_counters()	218
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()	219
8.40.3.10 size()	219
8.41 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference	220
8.41.1 Detailed Description	221
8.41.2 Constructor & Destructor Documentation	222
8.41.2.1 Support() [1/3]	222
8.41.2.2 Support() [2/3]	222
8.41.2.3 Support() [3/3]	222
8.41.2.4 ~Support()	222
8.41.3 Member Function Documentation	223
8.41.3.1 add_counter()	223
8.41.3.2 add_rule() [1/2] 2	223
8.41.3.3 add_rule() [2/2] 2	223
8.41.3.4 add_rule_dyn() [1/2]	223
8.41.3.5 add_rule_dyn() [2/2]	223
8.41.3.6 calc()	224
8.41.3.7 eval_rules_dyn()	224

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

9.3.1.5 ROW	233
9.3.2 Function Documentation	233
9.3.2.1 BARRAY_TEMPLATE() [1/6]	234
9.3.2.2 BARRAY_TEMPLATE() [2/6]	234
9.3.2.3 BARRAY_TEMPLATE() [3/6]	234
9.3.2.4 BARRAY_TEMPLATE() [4/6]	234
9.3.2.5 BARRAY_TEMPLATE() [5/6]	234
9.3.2.6 BARRAY_TEMPLATE() [6/6]	235
9.3.2.7 BARRAY_TEMPLATE_ARGS()	235
9.3.2.8 BARRAY_TYPE()	235
9.3.2.9 for()	235
9.3.2.10 operator()()	235
9.3.3 Variable Documentation	235
9.3.3.1 rhs	236
9.3.3.2 this	236
9.4 include/barry/barray-meat.hpp File Reference	236
9.4.1 Macro Definition Documentation	238
9.4.1.1 BARRAY_TEMPLATE	238
9.4.1.2 BARRAY_TEMPLATE_ARGS	238
9.4.1.3 BARRAY_TYPE	239
9.4.1.4 COL	239
9.4.1.5 ROW	239
9.4.2 Function Documentation	239
9.4.2.1 ans()	239
9.4.2.2 BARRAY_TEMPLATE() [1/24]	239
9.4.2.3 BARRAY_TEMPLATE() [2/24]	240
9.4.2.4 BARRAY_TEMPLATE() [3/24]	240
9.4.2.5 BARRAY_TEMPLATE() [4/24]	240
9.4.2.6 BARRAY_TEMPLATE() [5/24]	240
9.4.2.7 BARRAY_TEMPLATE() [6/24]	240
9.4.2.8 BARRAY_TEMPLATE() [7/24]	240
9.4.2.9 BARRAY_TEMPLATE() [8/24]	241
9.4.2.10 BARRAY_TEMPLATE() [9/24]	241
9.4.2.11 BARRAY_TEMPLATE() [10/24]	241
9.4.2.12 BARRAY_TEMPLATE() [11/24]	241
9.4.2.13 BARRAY_TEMPLATE() [12/24]	241
9.4.2.14 BARRAY_TEMPLATE() [13/24]	242
9.4.2.15 BARRAY_TEMPLATE() [14/24]	242
9.4.2.16 BARRAY_TEMPLATE() [15/24]	242
9.4.2.17 BARRAY_TEMPLATE() [16/24]	242
9.4.2.18 BARRAY_TEMPLATE() [17/24]	242
9.4.2.19 BARRAY_TEMPLATE() [18/24]	242

	9.4.2.20 BARRAY_TEMPLATE() [19/24]	13
	9.4.2.21 BARRAY_TEMPLATE() [20/24] 24	l3
	9.4.2.22 BARRAY_TEMPLATE() [21/24]	ŀ3
	9.4.2.23 BARRAY_TEMPLATE() [22/24]	l3
	<b>9.4.2.24 BARRAY_TEMPLATE()</b> [23/24] <b>24</b>	ŀ3
	9.4.2.25 BARRAY_TEMPLATE() [24/24] 24	l3
	9.4.2.26 COL()	14
	9.4.2.27 for() [1/3]	14
	9.4.2.28 for() [2/3]	14
	9.4.2.29 for() [3/3]	14
	9.4.2.30 if() [1/17]	14
	9.4.2.31 if() [2/17]	14
	9.4.2.32 if() [3/17]	ŀ5
	9.4.2.33 if() [4/17]	ŀ5
	9.4.2.34 if() [5/17]	ŀ5
	9.4.2.35 if() [6/17]	ŀ5
	9.4.2.36 if() [7/17]	ŀ5
	9.4.2.37 if() [8/17]	ŀ5
	9.4.2.38 if() [9/17]	<b>l</b> 6
	9.4.2.39 if() [10/17]	<b>l</b> 6
	9.4.2.40 if() [11/17]	<b>l</b> 6
	9.4.2.41 if() [12/17]	ŀ6
	9.4.2.42 if() [13/17]	<b>l</b> 6
	9.4.2.43 if() [14/17]	<b>l</b> 6
	9.4.2.44 if() [15/17]	16
	9.4.2.45 if() [16/17]	<b>!</b> 7
	9.4.2.46 if() [17/17]	١7
	9.4.2.47 M()	١7
	9.4.2.48 resize() [1/2]	١7
	9.4.2.49 resize() [2/2]	١7
	9.4.2.50 return()	
	9.4.2.51 ROW() [1/2]	18
	9.4.2.52 ROW() [2/2]	18
9.4.3 \	riable Documentation	18
	9.4.3.1 add	18
	9.4.3.2 ans	18
	9.4.3.3 Array	18
	9.4.3.4 check_bounds	19
	9.4.3.5 check_exists	19
	9.4.3.6 col0	19
	9.4.3.7 const	
	9.4.3.8 copy_data	50

9.4.3.9 data	250
9.4.3.10 delete_data	250
9.4.3.11 delete_data	250
9.4.3.12 else	250
9.4.3.13 false	251
9.4.3.14 first	251
9.4.3.15 i1	251
9.4.3.16 j	251
9.4.3.17 j0	251
9.4.3.18 j1	251
9.4.3.19 M	252
9.4.3.20 M	252
9.4.3.21 N	252
9.4.3.22 NCells	252
9.4.3.23 report	252
9.4.3.24 return	253
9.4.3.25 row0	253
9.4.3.26 search	253
9.4.3.27 source	253
9.4.3.28 target	253
9.4.3.29 v	253
9.4.3.30 value	254
9.5 include/barry/barraycell-bones.hpp File Reference	254
9.6 include/barry/barraycell-meat.hpp File Reference	254
9.7 include/barry/barraydense-bones.hpp File Reference	255
9.8 include/barry/barraydense-meat-operators.hpp File Reference	255
9.8.1 Macro Definition Documentation	256
9.8.1.1 BDENSE_TEMPLATE	256
9.8.1.2 BDENSE_TEMPLATE_ARGS	256
9.8.1.3 BDENSE_TYPE	256
9.8.1.4 COL	256
9.8.1.5 POS	257
9.8.1.6 POS_N	257
9.8.1.7 ROW	257
9.8.2 Function Documentation	257
9.8.2.1 BDENSE_TEMPLATE() [1/4]	257
9.8.2.2 BDENSE_TEMPLATE() [2/4]	257
<b>9.8.2.3 BDENSE_TEMPLATE()</b> [3/4]	258
9.8.2.4 BDENSE_TEMPLATE() [4/4]	258
9.8.2.5 BDENSE_TEMPLATE_ARGS()	258
9.8.2.6 BDENSE_TYPE()	258
9.9 include/barry/barraydense-meat.hpp File Reference	258

9.9.1 Macro Definition Documentation
9.9.1.1 BDENSE_TEMPLATE
9.9.1.2 BDENSE_TEMPLATE_ARGS
9.9.1.3 BDENSE_TYPE
9.9.1.4 COL
9.9.1.5 POS
9.9.1.6 POS_N
9.9.1.7 ROW
9.9.1.8 ZERO_CELL
9.9.2 Function Documentation
9.9.2.1 ans()
9.9.2.2 BDENSE_TEMPLATE() [1/39]
9.9.2.3 BDENSE_TEMPLATE() [2/39]
9.9.2.4 BDENSE_TEMPLATE() [3/39]
9.9.2.5 BDENSE_TEMPLATE() [4/39]
9.9.2.6 BDENSE_TEMPLATE() [5/39]
9.9.2.7 BDENSE_TEMPLATE() [6/39]
9.9.2.8 BDENSE_TEMPLATE() [7/39]
9.9.2.9 BDENSE_TEMPLATE() [8/39]
9.9.2.10 BDENSE_TEMPLATE() [9/39]
9.9.2.11 BDENSE_TEMPLATE() [10/39]
9.9.2.12 BDENSE_TEMPLATE() [11/39]
9.9.2.13 BDENSE_TEMPLATE() [12/39]
9.9.2.14 BDENSE_TEMPLATE() [13/39]
9.9.2.15 BDENSE_TEMPLATE() [14/39]
9.9.2.16 BDENSE_TEMPLATE() [15/39]
9.9.2.17 BDENSE_TEMPLATE() [16/39]
9.9.2.18 BDENSE_TEMPLATE() [17/39]
9.9.2.19 BDENSE_TEMPLATE() [18/39]
9.9.2.20 BDENSE_TEMPLATE() [19/39]
9.9.2.21 BDENSE_TEMPLATE() [20/39]
9.9.2.22 BDENSE_TEMPLATE() [21/39]
9.9.2.23 BDENSE_TEMPLATE() [22/39]
9.9.2.24 BDENSE_TEMPLATE() [23/39]
9.9.2.25 BDENSE_TEMPLATE() [24/39]
<b>9.9.2.26 BDENSE_TEMPLATE()</b> [25/39]
9.9.2.27 BDENSE_TEMPLATE() [26/39]
9.9.2.28 BDENSE_TEMPLATE() [27/39]
9.9.2.29 BDENSE_TEMPLATE() [28/39]
9.9.2.30 BDENSE_TEMPLATE() [29/39]
9.9.2.31 BDENSE_TEMPLATE() [30/39]
9.9.2.32 BDENSE_TEMPLATE() [31/39]

9.9.2.33 BD	ENSE_TEMPLATE()	[32/39]	 	 	 	268
9.9.2.34 BD	ENSE_TEMPLATE()	[33/39]	 	 	 	268
9.9.2.35 BD	ENSE_TEMPLATE()	[34/39]	 	 	 	269
9.9.2.36 BD	ENSE_TEMPLATE()	[35/39]	 	 	 	269
9.9.2.37 BD	ENSE_TEMPLATE()	[36/39]	 	 	 	269
9.9.2.38 BD	ENSE_TEMPLATE()	[37/39]	 	 	 	269
9.9.2.39 BD	ENSE_TEMPLATE()	[38/39]	 	 	 	269
9.9.2.40 BD	ENSE_TEMPLATE()	[39/39]	 	 	 	269
9.9.2.41 for	)		 	 	 	270
9.9.2.42 if()	[1/4]		 	 	 	270
9.9.2.43 if()	[2/4]		 	 	 	270
9.9.2.44 if()	[3/4]		 	 	 	270
9.9.2.45 if()	[4/4]		 	 	 	270
9.9.2.46 ins	ert_cell() [1/2]		 	 	 	270
9.9.2.47 ins	ert_cell() [2/2]		 	 	 	271
9.9.2.48 M()			 	 	 	271
9.9.2.49 prii	ntf_barry()		 	 	 	271
9.9.2.50 res	ize() [1/6]		 	 	 	271
9.9.2.51 res	<b>ize()</b> [2/6]		 	 	 	271
9.9.2.52 res	<b>ize()</b> [3/6]		 	 	 	271
9.9.2.53 res	<b>ize()</b> [4/6]		 	 	 	272
9.9.2.54 res	<b>ize()</b> [5/6]		 	 	 	272
9.9.2.55 res	<b>ize()</b> [6/6]		 	 	 	272
9.9.2.56 rm	_cell() [1/3]		 	 	 	272
9.9.2.57 rm	_cell() [2/3]		 	 	 	272
9.9.2.58 rm	_cell() [3/3]		 	 	 	272
9.9.2.59 va_	_end()		 	 	 	273
9.9.2.60 va_	_start()		 	 	 	273
9.9.3 Variable Docu	mentation		 	 	 	273
9.9.3.1 add			 	 	 	273
9.9.3.2 ans			 	 	 	273
9.9.3.3 ched	ck_bounds		 	 	 	273
9.9.3.4 ched	ck_exists		 	 	 	274
9.9.3.5 col .			 	 	 	274
9.9.3.6 cons	st		 	 	 	274
9.9.3.7 copy	/_data		 	 	 	274
9.9.3.8 data			 	 	 	274
9.9.3.9 dele	te_data		 	 	 	275
9.9.3.10 del	ete_data		 	 	 	275
9.9.3.11 el .			 	 	 	275
9.9.3.12 el_	colsums		 	 	 	275
9.9.3.13 el_	rowsums		 	 	 	275

9.9.3.14 else	76
9.9.3.15 false	76
9.9.3.16 i1	76
9.9.3.17 j	76
9.9.3.18 j0	76
9.9.3.19 j1	76
9.9.3.20 M	77
9.9.3.21 M	77
9.9.3.22 N	77
9.9.3.23 report	77
9.9.3.24 return	77
9.9.3.25 source	78
9.9.3.26 target	78
9.9.3.27 v	78
9.9.3.28 val0	78
9.9.3.29 val1	78
9.9.3.30 value	78
9.10 include/barry/barraydensecell-bones.hpp File Reference	79
9.10.1 Macro Definition Documentation	79
9.10.1.1 POS	79
9.11 include/barry/barraydensecell-meat.hpp File Reference	80
9.11.1 Macro Definition Documentation	80
9.11.1.1 POS	80
9.12 include/barry/barraydensecol-bones.hpp File Reference	80
9.12.1 Macro Definition Documentation	81
9.12.1.1 POS	81
9.12.1.2 POS_N	81
9.12.1.3 ZERO_CELL	81
9.13 include/barry/barraydenserow-bones.hpp File Reference	82
9.13.1 Macro Definition Documentation	82
9.13.1.1 POS	82
9.13.1.2 POS_N	83
9.13.1.3 ZERO_CELL	83
9.14 include/barry/barrayrow-bones.hpp File Reference	83
9.15 include/barry/barrayrow-meat.hpp File Reference	83
9.15.1 Macro Definition Documentation	83
9.15.1.1 BROW_TEMPLATE	84
9.15.1.2 BROW_TEMPLATE_ARGS	84
9.15.1.3 BROW_TYPE	84
9.15.2 Function Documentation	84
9.15.2.1 BROW_TEMPLATE() [1/5]	84
9.15.2.2 BROW_TEMPLATE() [2/5]	84

<b>9.15.2.3 BROW_TEMPLATE()</b> [3/5]
9.15.2.4 BROW_TEMPLATE() [4/5]
9.15.2.5 BROW_TEMPLATE() [5/5]
9.16 include/barry/barrayvector-bones.hpp File Reference
9.17 include/barry/barrayvector-meat.hpp File Reference
9.18 include/barry/barry-configuration.hpp File Reference
9.18.1 Macro Definition Documentation
9.18.1.1 BARRY_CHECK_SUPPORT
9.18.1.2 BARRY_ISFINITE
9.18.1.3 BARRY_MAX_NUM_ELEMENTS
9.18.1.4 BARRY_SAFE_EXP
9.18.1.5 printf_barry
9.18.2 Typedef Documentation
9.18.2.1 Map
9.19 include/barry/barry-debug.hpp File Reference
9.19.1 Macro Definition Documentation
9.19.1.1 BARRY_DEBUG_LEVEL
9.20 include/barry/barry-macros.hpp File Reference
9.20.1 Macro Definition Documentation
9.20.1.1 BARRY_ONE
9.20.1.2 BARRY_ONE_DENSE
9.20.1.3 BARRY_UNUSED
9.20.1.4 BARRY_ZERO
9.20.1.5 BARRY_ZERO_DENSE
9.21 include/barry/barry.hpp File Reference
9.21.1 Macro Definition Documentation
9.21.1.1 BARRY_HPP
9.21.1.2 BARRY_VERSION
9.21.1.3 BARRY_VERSION_MAYOR
9.21.1.4 BARRY_VERSION_MINOR
9.21.1.5 COUNTER_FUNCTION
9.21.1.6 COUNTER_LAMBDA
9.21.1.7 RULE_FUNCTION
9.21.1.8 RULE_LAMBDA
9.22 include/barry/cell-bones.hpp File Reference
9.23 include/barry/cell-meat.hpp File Reference
9.24 include/barry/col-bones.hpp File Reference
9.25 include/barry/counters-bones.hpp File Reference
9.26 include/barry/counters-meat.hpp File Reference
9.26.1 Macro Definition Documentation
9.26.1.1 COUNTER_TEMPLATE
9.26.1.2 COUNTER_TEMPLATE_ARGS

9.26.1.3 COUNTER_TYPE	<b>)</b> 7
9.26.1.4 COUNTERS_TEMPLATE	<b>)</b> 7
9.26.1.5 COUNTERS_TEMPLATE_ARGS	<b>)</b> 7
9.26.1.6 COUNTERS_TYPE	<b>)</b> 7
9.26.1.7 TMP_HASHER_CALL	<b>)</b> 7
9.26.2 Function Documentation	98
9.26.2.1 count_fun()	98
9.26.2.2 COUNTER_TEMPLATE() [1/9]	98
9.26.2.3 COUNTER_TEMPLATE() [2/9]	98
9.26.2.4 COUNTER_TEMPLATE() [3/9]	98
9.26.2.5 COUNTER_TEMPLATE() [4/9]	98
9.26.2.6 COUNTER_TEMPLATE() [5/9]	99
9.26.2.7 COUNTER_TEMPLATE() [6/9]	99
9.26.2.8 COUNTER_TEMPLATE() [7/9]	99
9.26.2.9 COUNTER_TEMPLATE() [8/9]	99
9.26.2.10 COUNTER_TEMPLATE() [9/9]	9
9.26.2.11 COUNTERS_TEMPLATE() [1/9]	)0
9.26.2.12 COUNTERS_TEMPLATE() [2/9]	)0
9.26.2.13 COUNTERS_TEMPLATE() [3/9]	)0
9.26.2.14 COUNTERS_TEMPLATE() [4/9]	)0
9.26.2.15 COUNTERS_TEMPLATE() [5/9]	)0
9.26.2.16 COUNTERS_TEMPLATE() [6/9]	)0
9.26.2.17 COUNTERS_TEMPLATE() [7/9]	)1
9.26.2.18 COUNTERS_TEMPLATE() [8/9]	)1
9.26.2.19 COUNTERS_TEMPLATE() [9/9]	)1
9.26.2.20 data()	)1
9.26.2.21 desc()	)1
9.26.2.22 for()	)1
9.26.2.23 hasher() [1/2]	)2
9.26.2.24 hasher() [2/2]	)2
9.26.2.25 hasher_fun() [1/2]	)2
9.26.2.26 hasher_fun() [2/2]	)2
9.26.2.27 if() [1/3]	)2
9.26.2.28 if() [2/3]	)2
9.26.2.29 if() [3/3]	)3
9.26.2.30 init_fun() [1/3]	)3
9.26.2.31 init_fun() [2/3]	)3
9.26.2.32 init_fun() [3/3]	)3
9.26.2.33 name()	)3
9.26.3 Variable Documentation	)3
9.26.3.1 add_dims	)3
9.26.3.2 count fun	۱/

9.26.3.3 counter	04
9.26.3.4 counter	04
9.26.3.5 data	04
9.26.3.6 desc	05
9.26.3.7 fun	05
9.26.3.8 fun	05
9.26.3.9 hasher_fun	05
9.26.3.10 i	06
9.26.3.11 init_fun	06
9.26.3.12 j	06
9.26.3.13 name	06
9.26.3.14 noexcept	06
9.26.3.15 res	07
9.26.3.16 return	07
9.27 include/barry/counters/defm-formula.hpp File Reference	07
9.27.1 Function Documentation	07
9.27.1.1 defm_motif_parser()	80
9.28 include/barry/counters/defm.hpp File Reference	09
9.28.1 Macro Definition Documentation	11
9.28.1.1 UNI_SUB	11
9.28.2 Typedef Documentation	11
9.28.2.1 DEFMArray	11
9.29 include/barry/models/defm.hpp File Reference	12
9.30 include/barry/counters/network-css.hpp File Reference	12
9.30.1 Macro Definition Documentation	14
9.30.1.1 CSS_APPEND	14
9.30.1.2 CSS_CASE_ELSE	14
9.30.1.3 CSS_CASE_PERCEIVED	14
9.30.1.4 CSS_CASE_TRUTH	14
9.30.1.5 CSS_CHECK_SIZE	15
9.30.1.6 CSS_CHECK_SIZE_INIT	15
9.30.1.7 CSS_NET_COUNTER_LAMBDA_INIT	15
9.30.1.8 CSS_PERCEIVED_CELLS	15
9.30.1.9 CSS_SIZE	16
9.30.1.10 CSS_TRUE_CELLS	16
9.30.2 Function Documentation	16
9.30.2.1 counter_css_census01()	16
9.30.2.2 counter_css_census02()	16
9.30.2.3 counter_css_census03()	17
9.30.2.4 counter_css_census04()	17
9.30.2.5 counter_css_census05()	17
9.30.2.6 counter_css_census06()	17

9.30.2.7 counter_css_census07()	318
9.30.2.8 counter_css_census08()	318
9.30.2.9 counter_css_census09()	318
9.30.2.10 counter_css_census10()	318
9.30.2.11 counter_css_completely_false_recip_comiss()	319
9.30.2.12 counter_css_completely_false_recip_omiss()	319
9.30.2.13 counter_css_mixed_recip()	319
9.30.2.14 counter_css_partially_false_recip_commi()	319
9.30.2.15 counter_css_partially_false_recip_omiss()	320
9.31 include/barry/counters/network.hpp File Reference	320
9.31.1 Macro Definition Documentation	323
9.31.1.1 BARRY_ZERO_NETWORK	323
9.31.1.2 BARRY_ZERO_NETWORK_DENSE	324
9.31.1.3 NET_C_DATA_IDX	324
9.31.1.4 NET_C_DATA_NUM	324
9.31.1.5 NETWORK_COUNTER	324
9.31.1.6 NETWORK_COUNTER_LAMBDA	324
9.31.1.7 NETWORK_RULE	325
9.31.1.8 NETWORK_RULE_LAMBDA	325
9.31.1.9 NETWORKDENSE_COUNTER_LAMBDA	325
9.31.2 Typedef Documentation	325
9.31.2.1 NetCounter	325
9.31.2.2 NetCounters	326
9.31.2.3 NetModel	326
9.31.2.4 NetRule	326
9.31.2.5 NetRules	326
9.31.2.6 NetStatsCounter	326
9.31.2.7 NetSupport	326
9.31.2.8 Network	327
9.31.2.9 NetworkDense	327
9.31.3 Function Documentation	327
9.31.3.1 rules_zerodiag()	327
9.32 include/barry/counters/phylo.hpp File Reference	327
9.32.1 Macro Definition Documentation	330
9.32.1.1 DEFAULT_DUPLICATION	330
9.32.1.2 DUPL_DUPL	330
9.32.1.3 DUPL_EITH	330
9.32.1.4 DUPL_SPEC	330
9.32.1.5 IF_MATCHES	331
9.32.1.6 IF_NOTMATCHES	331
9.32.1.7 IS_DUPLICATION	331
9.32.1.8 IS_EITHER	331

9.32.1.9 IS_SPECIATION	. 331
9.32.1.10 MAKE_DUPL_VARS	. 332
9.32.1.11 PHYLO_CHECK_MISSING	. 332
9.32.1.12 PHYLO_COUNTER_LAMBDA	. 332
9.32.1.13 PHYLO_RULE_DYN_LAMBDA	. 332
9.32.2 Typedef Documentation	. 333
9.32.2.1 PhyloArray	. 333
9.32.2.2 PhyloCounter	. 333
9.32.2.3 PhyloCounters	. 333
9.32.2.4 PhyloModel	. 333
9.32.2.5 PhyloPowerSet	. 333
9.32.2.6 PhyloRule	. 334
9.32.2.7 PhyloRuleData	. 334
9.32.2.8 PhyloRuleDyn	. 334
9.32.2.9 PhyloRules	. 334
9.32.2.10 PhyloRulesDyn	. 334
9.32.2.11 PhyloStatsCounter	. 334
9.32.2.12 PhyloSupport	. 335
9.32.3 Function Documentation	. 335
9.32.3.1 get_last_name()	. 335
9.33 include/barry/freqtable.hpp File Reference	. 335
9.34 include/barry/model-bones.hpp File Reference	. 336
9.35 include/barry/model-meat.hpp File Reference	. 336
9.35.1 Macro Definition Documentation	. 338
9.35.1.1 MODEL_TEMPLATE	. 338
9.35.1.2 MODEL_TEMPLATE_ARGS	. 339
9.35.1.3 MODEL_TYPE	. 339
9.35.2 Function Documentation	. 339
9.35.2.1 for()	. 339
9.35.2.2 if() [1/4]	. 339
9.35.2.3 if() [2/4]	. 339
<b>9.35.2.4 if()</b> [3/4]	. 340
9.35.2.5 if() [4/4]	. 340
9.35.2.6 insert_cell()	. 340
9.35.2.7 likelihood_()	. 340
<b>9.35.2.8 MODEL_TEMPLATE()</b> [1/33]	. 340
<b>9.35.2.9 MODEL_TEMPLATE()</b> [2/33]	. 341
<b>9.35.2.10 MODEL_TEMPLATE()</b> [3/33]	. 341
<b>9.35.2.11 MODEL_TEMPLATE()</b> [4/33]	. 341
<b>9.35.2.12 MODEL_TEMPLATE()</b> [5/33]	. 341
<b>9.35.2.13 MODEL_TEMPLATE()</b> [6/33]	. 341
<b>9.35.2.14 MODEL_TEMPLATE()</b> [7/33]	. 342

9.35.2.15 MODEL_TEMPLATE()	[8/33].	 	 	 	 342
9.35.2.16 MODEL_TEMPLATE()	[9/33] .	 	 	 	 342
9.35.2.17 MODEL_TEMPLATE()	[10/33]	 	 	 	 342
9.35.2.18 MODEL_TEMPLATE()	[11/33]	 	 	 	 342
9.35.2.19 MODEL_TEMPLATE()	[12/33]	 	 	 	 343
9.35.2.20 MODEL_TEMPLATE()	[13/33]	 	 	 	 343
9.35.2.21 MODEL_TEMPLATE()	[14/33]	 	 	 	 343
9.35.2.22 MODEL_TEMPLATE()	[15/33]	 	 	 	 343
9.35.2.23 MODEL_TEMPLATE()	[16/33]	 	 	 	 343
9.35.2.24 MODEL_TEMPLATE()	[17/33]	 	 	 	 344
9.35.2.25 MODEL_TEMPLATE()	[18/33]	 	 	 	 344
9.35.2.26 MODEL_TEMPLATE()	[19/33]	 	 	 	 344
9.35.2.27 MODEL_TEMPLATE()	[20/33]	 	 	 	 344
9.35.2.28 MODEL_TEMPLATE()	[21/33]	 	 	 	 344
9.35.2.29 MODEL_TEMPLATE()	[22/33]	 	 	 	 345
9.35.2.30 MODEL_TEMPLATE()	[23/33]	 	 	 	 345
9.35.2.31 MODEL_TEMPLATE()	[24/33]	 	 	 	 345
9.35.2.32 MODEL_TEMPLATE()	[25/33]	 	 	 	 345
9.35.2.33 MODEL_TEMPLATE()	[26/33]	 	 	 	 345
9.35.2.34 MODEL_TEMPLATE()	[27/33]	 	 	 	 345
9.35.2.35 MODEL_TEMPLATE()	[28/33]	 	 	 	 346
9.35.2.36 MODEL_TEMPLATE()	[29/33]	 	 	 	 346
9.35.2.37 MODEL_TEMPLATE()	[30/33]	 	 	 	 346
9.35.2.38 MODEL_TEMPLATE()	[31/33]	 	 	 	 346
9.35.2.39 MODEL_TEMPLATE()	[32/33]	 	 	 	 346
9.35.2.40 MODEL_TEMPLATE()	[33/33]	 	 	 	 346
9.35.2.41 push_back() [1/2]		 	 	 	 347
9.35.2.42 push_back() [2/2]		 	 	 	 347
9.35.2.43 return()		 	 	 	 347
9.35.2.44 set_counters()		 	 	 	 347
9.35.2.45 set_rules()		 	 	 	 347
9.35.2.46 set_rules_dyn()		 	 	 	 347
9.35.2.47 size()					
9.35.2.48 temp_stats()					
9.35.2.49 tmp_counts()		 	 	 	 348
9.35.2.50 update_normalizing_c	onstant()	 	 	 	 348
9.35.2.51 urand()		 	 	 	 348
9.35.3 Variable Documentation					
9.35.3.1 a					
9.35.3.2 count_fun		 	 	 	 349
9.35.3.3 counter		 	 	 	 349
9.35.3.4 counters_		 	 	 	 349

9.35.3.5 cumprob	349
9.35.3.6 data	349
9.35.3.7 Data_Counter_Type	350
9.35.3.8 Data_Rule_Type	350
9.35.3.9 delete_rules	350
9.35.3.10 delete_rules_dyn	350
9.35.3.11 else	350
9.35.3.12 force_new	
9.35.3.13 fun	351
9.35.3.14 i	351
9.35.3.15 i_matches	351
9.35.3.16 init_fun	
9.35.3.17 j	
9.35.3.18 k	352
9.35.3.19 key	352
9.35.3.20 locator	352
9.35.3.21 params	352
9.35.3.22 probs	352
9.35.3.23 pset_arrays	353
9.35.3.24 r	353
9.35.3.25 return	353
9.35.3.26 rule_fun	353
9.35.3.27 rules	353
9.35.3.28 rules	354
9.35.3.29 rules_dyn	354
9.35.3.30 stats	354
9.35.3.31 stats_support_n_arrays	354
9.36 include/barry/models/defm/defm-bones.hpp File Reference	354
9.37 include/barry/models/defm/defm-meat.hpp File Reference	355
9.37.1 Macro Definition Documentation	355
9.37.1.1 DEFM_LOOP_ARRAYS	355
9.37.1.2 DEFM_RANGES	356
9.37.2 Function Documentation	356
9.37.2.1 keygen_defm()	356
9.38 include/barry/models/geese.hpp File Reference	356
9.39 include/barry/models/geese/flock-bones.hpp File Reference	357
9.40 include/barry/models/geese/flock-meat.hpp File Reference	357
9.41 include/barry/models/geese/geese-bones.hpp File Reference	358
9.41.1 Macro Definition Documentation	358
9.41.1.1 INITIALIZED	359
9.41.2 Function Documentation	359
9.41.2.1 keygen_full()	359

9.41.2.2 RULE_FUNCTION()	359
9.41.2.3 vec_diff()	359
9.41.2.4 vector_caster()	359
9.42 include/barry/models/geese/geese-meat-constructors.hpp File Reference	360
9.43 include/barry/models/geese/geese-meat-likelihood.hpp File Reference	360
9.44 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference	361
9.45 include/barry/models/geese/geese-meat-predict.hpp File Reference	362
9.46 include/barry/models/geese/geese-meat-predict_exhaust.hpp File Reference	362
9.47 include/barry/models/geese/geese-meat-predict_sim.hpp File Reference	363
9.48 include/barry/models/geese/geese-meat-simulate.hpp File Reference	363
9.49 include/barry/models/geese/geese-meat.hpp File Reference	364
9.50 include/barry/models/geese/geese-node-bones.hpp File Reference	364
9.51 include/barry/powerset-bones.hpp File Reference	365
9.52 include/barry/powerset-meat.hpp File Reference	365
9.53 include/barry/progress.hpp File Reference	366
9.53.1 Macro Definition Documentation	366
9.53.1.1 BARRY_PROGRESS_BAR_WIDTH	366
9.54 include/barry/rules-bones.hpp File Reference	367
9.54.1 Function Documentation	367
9.54.1.1 rule_fun_default()	367
9.55 include/barry/rules-meat.hpp File Reference	368
9.56 include/barry/statscounter-bones.hpp File Reference	368
9.57 include/barry/statscounter-meat.hpp File Reference	369
9.57.1 Macro Definition Documentation	370
9.57.1.1 STATSCOUNTER_TEMPLATE	370
9.57.1.2 STATSCOUNTER_TEMPLATE_ARGS	370
9.57.1.3 STATSCOUNTER_TYPE	370
9.57.2 Function Documentation	370
9.57.2.1 clear()	370
9.57.2.2 for()	370
9.57.2.3 resize()	371
9.57.2.4 STATSCOUNTER_TEMPLATE() [1/9]	371
9.57.2.5 STATSCOUNTER_TEMPLATE() [2/9]	371
9.57.2.6 STATSCOUNTER_TEMPLATE() [3/9]	371
9.57.2.7 STATSCOUNTER_TEMPLATE() [4/9]	371
9.57.2.8 STATSCOUNTER_TEMPLATE() [5/9]	371
9.57.2.9 STATSCOUNTER_TEMPLATE() [6/9]	372
9.57.2.10 STATSCOUNTER_TEMPLATE() [7/9]	372
9.57.2.11 STATSCOUNTER_TEMPLATE() [8/9]	372
9.57.2.12 STATSCOUNTER_TEMPLATE() [9/9]	372
9.57.3 Variable Documentation	372
9.57.3.1 counter	372

9.57.3.2 counter_deleted	73
9.57.3.3 counters	73
9.57.3.4 counters	73
9.57.3.5 current_stats	73
9.57.3.6 EmptyArray	73
9.57.3.7 f	74
9.57.3.8 j	74
9.57.3.9 return	74
9.58 include/barry/support-bones.hpp File Reference	74
9.59 include/barry/support-meat.hpp File Reference	75
9.59.1 Macro Definition Documentation	76
9.59.1.1 BARRY_SUPPORT_MEAT_HPP	76
9.59.1.2 SUPPORT_TEMPLATE	76
9.59.1.3 SUPPORT_TEMPLATE_ARGS	77
9.59.1.4 SUPPORT_TYPE	77
9.59.2 Function Documentation	77
9.59.2.1 calc_backend_dense()	77
9.59.2.2 calc_backend_sparse()	77
9.59.2.3 for()	77
9.59.2.4 if() [1/3]	78
9.59.2.5 if() [2/3]	78
9.59.2.6 if() [3/3]	78
9.59.2.7 insert_cell() [1/2]	78
9.59.2.8 insert_cell() [2/2]	78
9.59.2.9 rm_cell()	79
9.59.2.10 SUPPORT_TEMPLATE() [1/17]	79
9.59.2.11 SUPPORT_TEMPLATE() [2/17]	79
9.59.2.12 SUPPORT_TEMPLATE() [3/17]	79
9.59.2.13 SUPPORT_TEMPLATE() [4/17]	79
9.59.2.14 SUPPORT_TEMPLATE() [5/17]	80
9.59.2.15 SUPPORT_TEMPLATE() [6/17]	80
9.59.2.16 SUPPORT_TEMPLATE() [7/17]	80
9.59.2.17 SUPPORT_TEMPLATE() [8/17]	80
9.59.2.18 SUPPORT_TEMPLATE() [9/17]	80
9.59.2.19 SUPPORT_TEMPLATE() [10/17]	80
9.59.2.20 SUPPORT_TEMPLATE() [11/17]	81
9.59.2.21 SUPPORT_TEMPLATE() [12/17]	81
9.59.2.22 SUPPORT_TEMPLATE() [13/17]	81
9.59.2.23 SUPPORT_TEMPLATE() [14/17]	81
9.59.2.24 SUPPORT_TEMPLATE() [15/17]	81
9.59.2.25 SUPPORT_TEMPLATE() [16/17]	82
9.59.2.26 SUPPORT_TEMPLATE() [17/17]	82

9.59.3 Variable Documentation	382
9.59.3.1 array_bank	382
9.59.3.2 change_stats_different	382
9.59.3.3 coord_i	382
9.59.3.4 coord_j	382
9.59.3.5 counters	383
9.59.3.6 counters	383
9.59.3.7 delete_counters	383
9.59.3.8 delete_rules	383
9.59.3.9 delete_rules_dyn	383
9.59.3.10 else	384
9.59.3.11 f	384
9.59.3.12 hashes	384
9.59.3.13 return	384
9.59.3.14 rules	384
9.59.3.15 rules	385
9.59.3.16 rules_dyn	385
9.59.3.17 stats_bank	385
9.59.3.18 tmp_chng	385
9.60 include/barry/typedefs.hpp File Reference	386
9.60.1 Typedef Documentation	388
9.60.1.1 Col_type	388
9.60.1.2 Counter_fun_type	388
9.60.1.3 Counts_type	388
9.60.1.4 Hasher_fun_type	388
9.60.1.5 MapVec_type	389
9.60.1.6 Row_type	389
9.60.1.7 Rule_fun_type	389
9.60.2 Function Documentation	389
9.60.2.1 sort_array()	389
9.60.2.2 vec_equal()	390
9.60.2.3 vec_equal_approx()	390
9.60.2.4 vec_inner_prod() [1/2]	391
9.60.2.5 vec_inner_prod() [2/2]	391
9.61 README.md File Reference	391
Index	393

# 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."

2 Main Page

# **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 <ostream>
#include "../include/barry.hpp"
typedef std::vector< unsigned int > vuint;
int main() {
  // Creating network of size six with five ties
  netcounters::Network net(
       {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 freing 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();
    "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
  2,]
                   1
  3,] .
4,] 1
          . .
  5,]
New view
  0,]
           1
              1
       1
  1,]
                   1
  2,] 1
  3,] . . .
4,] 1 . 1
  5,] . .
Edges
Transitive triads : 3
Isolates
C triads
                  : 3
Mutuals
```

# **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 <a href="here">here</a> and in the PDF version of the documentation <a href="here">here</a>.

# **Code of Conduct**

Please note that the <code>barry</code> project is released with a <code>Contributor</code> Code of Conduct. By contributing to this project, you agree to abide by its terms.

4 Main Page

# **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

6 Module Index

# **Hierarchical Index**

# 3.1 Class Hierarchy

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

BArray < Cell_Type, Data_Type >
BArray < bool, bool >
BArray < Cell_Type, Data_Type >
BArrayCell < Cell_Type, Data_Type >
BArrayCell_const< Cell_Type, Data_Type >
BArrayDense < Cell_Type, Data_Type >
BArrayDense < bool, bool >
BArrayDenseCell< Cell_Type, Data_Type >
BArrayDenseCell_const< Cell_Type, Data_Type >
BArrayDenseCol < Cell_Type, Data_Type >
BArrayDenseCol_const< Cell_Type, Data_Type >
BArrayDenseRow < Cell_Type, Data_Type >
BArrayDenseRow_const< Cell_Type, Data_Type >
BArrayRow< Cell_Type, Data_Type >
BArrayRow_const< Cell_Type, Data_Type >
BArrayVector< Cell_Type, Data_Type >
BArrayVector_const< Cell_Type, Data_Type >
Cell< Cell_Type >
Cell< bool >
Cell_const< Cell_Type >
ConstBArrayRowIter< Cell_Type, Data_Type >
Counter< Array_Type, Data_Type >
Counters < Array_Type, Data_Type >
Counters < Array_Type, Data_Type >
Counters < BArray < bool, bool > , bool >
Counters < BArray <>, bool >
DEFMCounterData
DEFMData
barry::counters::defm::DEFMModel
DEFM
DEFMRuleData
DEFMRuleDynData
Entries < Cell_Type >
Flock
$FreqTable < T > \dots \dots$

8 Hierarchical Index

# **Class Index**

# 4.1 Class List

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

BArray< Cell_Type, Data_Type >
Baseline class for binary arrays
BArrayCell< Cell_Type, Data_Type >
BArrayCell_const< Cell_Type, Data_Type >
BArrayDense < Cell_Type, Data_Type >
Baseline class for binary arrays
BArrayDenseCell< Cell_Type, Data_Type >
BArrayDenseCell_const< Cell_Type, Data_Type >
BArrayDenseCol < Cell_Type, Data_Type >
BArrayDenseCol_const< Cell_Type, Data_Type >
BArrayDenseRow < Cell_Type, Data_Type >
BArrayDenseRow_const < Cell_Type, Data_Type >
BArrayRow < Cell_Type, Data_Type >
BArrayRow_const < Cell_Type, Data_Type >
BArrayVector< Cell_Type, Data_Type >
Row or column of a BArray
BArrayVector_const < Cell_Type, Data_Type >
Cell< Cell_Type >
Entries in BArray. For now, it only has two members:
Cell_const< Cell_Type >
ConstBArrayRowlter < Cell_Type, Data_Type >
Counter< Array_Type, Data_Type >
A counter function based on change statistics
Counters< Array_Type, Data_Type >
Vector of counters
DEFM
DEFMCounterData
Data class used to store arbitrary size_t or double vectors
DEFMData
DEFMRuleData
DEFMRuleDynData
Entries < Cell_Type >
A wrapper class to store source, target, val from a BArray object 137
Flock
A Flock is a group of Geese
A Flock is a group of Geese

10 Class Index

FreqTable < T >	
Frequency table of vectors	146
Geese	
Annotated Phylo Model	150
Model < Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >	
General framework for discrete exponential models. This class allows generating discrete expo-	
nential models in the form of a linear exponential model:	163
NetCounterData	
Data class used to store arbitrary size_t or double vectors	185
NetworkData	
Data class for Networks	187
Node	
A single node for the model	189
NodeData	
Data definition for the PhyloArray class	195
PhyloCounterData	196
PhyloRuleDynData	199
PowerSet < Array_Type, Data_Rule_Type >	
Powerset of a binary array	201
Progress	
A simple progress bar	208
Rule < Array_Type, Data_Type >	
Rule for determining if a cell should be included in a sequence	209
Rules< Array_Type, Data_Type >	
Vector of objects of class Rule	212
StatsCounter< Array_Type, Data_Type >	
Count stats for a single Array	216
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >	
Compute the support of sufficient statistics	220
vecHasher< T >	230

# File Index

# 5.1 File List

Here is a list of all files with brief descriptions:

include/barry/barray-bones.hpp
include/barry/barray-iterator.hpp
include/barry/barray-meat-operators.hpp
include/barry/barray-meat.hpp
include/barry/barraycell-bones.hpp
include/barry/barraycell-meat.hpp
include/barry/barraydense-bones.hpp
include/barry/barraydense-meat-operators.hpp
include/barry/barraydense-meat.hpp
include/barry/barraydensecell-bones.hpp
include/barry/barraydensecell-meat.hpp
include/barry/barraydensecol-bones.hpp
include/barry/barraydenserow-bones.hpp
include/barry/barrayrow-bones.hpp
include/barry/barrayrow-meat.hpp
include/barry/barrayvector-bones.hpp
include/barry/barrayvector-meat.hpp
include/barry/barry-configuration.hpp
include/barry/barry-debug.hpp
include/barry/barry-macros.hpp
include/barry/barry.hpp
include/barry/cell-bones.hpp
include/barry/cell-meat.hpp
include/barry/col-bones.hpp
include/barry/counters-bones.hpp
include/barry/counters-meat.hpp
include/barry/freqtable.hpp
include/barry/model-bones.hpp
include/barry/model-meat.hpp
include/barry/powerset-bones.hpp
include/barry/powerset-meat.hpp
include/barry/progress.hpp
include/barry/rules-bones.hpp
include/barry/rules-meat.hpp
include/barry/statscounter-bones.hpp

12 File Index

include/barry/statscounter-meat.hpp
include/barry/support-bones.hpp
include/barry/support-meat.hpp
include/barry/typedefs.hpp
include/barry/counters/defm-formula.hpp
include/barry/counters/defm.hpp
include/barry/counters/network-css.hpp
include/barry/counters/network.hpp
include/barry/counters/phylo.hpp
include/barry/models/defm.hpp
include/barry/models/geese.hpp
include/barry/models/defm/defm-bones.hpp
include/barry/models/defm/defm-meat.hpp
include/barry/models/geese/flock-bones.hpp
include/barry/models/geese/flock-meat.hpp
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
include/barry/models/geese/geese-node-bones.hpp

# **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 emty 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) colum (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

6.3 Phylo rules 15

# **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)

### 6.3.1 Detailed Description

Rules for phylogenetic modeling.

#### **Parameters**

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

#### 6.3.2 Macro Definition Documentation

#### 6.3.2.1 DEFM\_COUNTER

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( \scriptstyle a )
```

#### Value:

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

Lambda function for definition of a network counter function

Definition at line 218 of file defm.hpp.

#### 6.3.2.3 DEFM\_RULE

Function for definition of a network counter function

Definition at line 228 of file defm.hpp.

#### 6.3.2.4 DEFM RULE LAMBDA

Lambda function for definition of a network counter function

Definition at line 232 of file defm.hpp.

#### 6.3.2.5 DEFM\_RULEDYN\_LAMBDA

Lambda function for definition of a network counter function

Definition at line 238 of file defm.hpp.

#### 6.3.2.6 MAKE DEFM HASHER

Definition at line 197 of file defm.hpp.

6.3 Phylo rules

# 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 CountersDEFMArray, 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 RuleDEFMArray, 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()()

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

#### **Parameters**

i	
j	

#### Returns

double

Definition at line 170 of file defm.hpp.

# 6.3.4.4 print()

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

Definition at line 183 of file defm.hpp.

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

Overall functional gains.

#### **Parameters**

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

#### Returns

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

Definition at line 2177 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)
- 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\_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.

#### Returns true if the cell is free

#### **Parameters**

rules | A pointer to a DEFMRules object (Rules < 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**

counters	A pointer to a DEFMCounters object (Counters < DEFMArray, DEFMCounterData > ).
counters	A pointer to a NetCounters object (Counters < Network, NetCounterData > ).

### 6.4.2 Function Documentation

#### 6.4.2.1 counter\_absdiff()

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]

Definition at line 665 of file network.hpp.

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

Definition at line 610 of file network.hpp.

# 6.4.2.4 counter\_degree()

Counts number of vertices with a given out-degree.

Definition at line 1326 of file network.hpp.

#### 6.4.2.5 counter\_density()

Definition at line 729 of file network.hpp.

# 6.4.2.6 counter\_diff()

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()

Number of edges.

Definition at line 152 of file network.hpp.

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

Prevalence of ones.

#### **Parameters**

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

Definition at line 785 of file defm.hpp.

#### 6.4.2.9 counter\_idegree() [1/2]

Definition at line 1170 of file network.hpp.

### 6.4.2.10 counter\_idegree() [2/2]

Counts number of vertices with a given in-degree.

Definition at line 1123 of file network.hpp.

# 6.4.2.11 counter\_idegree15() [1/2]

Definition at line 785 of file network.hpp.

#### 6.4.2.12 counter\_idegree15() [2/2]

Definition at line 757 of file network.hpp.

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

Definition at line 215 of file network.hpp.

#### 6.4.2.14 counter\_isolates() [2/2]

Number of isolated vertices.

Definition at line 175 of file network.hpp.

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

Definition at line 338 of file network.hpp.

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

Definition at line 312 of file network.hpp.

#### 6.4.2.17 counter\_logit\_intercept()

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

Definition at line 326 of file defm.hpp.

#### 6.4.2.18 counter\_mutual()

Number of mutual ties.

Definition at line 256 of file network.hpp.

#### 6.4.2.19 counter\_nodecov()

Definition at line 1066 of file network.hpp.

#### 6.4.2.20 counter\_nodeicov()

Definition at line 1016 of file network.hpp.

### 6.4.2.21 counter\_nodematch()

Definition at line 1091 of file network.hpp.

# 6.4.2.22 counter\_nodeocov()

Definition at line 1041 of file network.hpp.

#### 6.4.2.23 counter\_odegree() [1/2]

Definition at line 1271 of file network.hpp.

# 6.4.2.24 counter\_odegree() [2/2]

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]

Definition at line 862 of file network.hpp.

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

Definition at line 834 of file network.hpp.

#### 6.4.2.27 counter\_ones()

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

Prevalence of ones.

#### **Parameters**

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

Definition at line 256 of file defm.hpp.

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

Definition at line 404 of file network.hpp.

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

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**

counters	Pointer ot a vector of counters
covar_index	If $>=$ 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**

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

Definition at line 754 of file defm.hpp.

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

Definition at line 531 of file network.hpp.

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

Definition at line 441 of file network.hpp.

#### 6.4.2.34 NETWORK\_COUNTER()

Definition at line 997 of file network.hpp.

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

Blocks switching a one to zero.

#### **Parameters**

rules	
ids	lds of the variables that will follow this rule.

Definition at line 853 of file defm.hpp.

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

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.

6.5 Phylo counters 31

• 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->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.
- 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\_DUPLICA
- 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.)

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** 

counters | A pointer to a PhyloCounters object (Counters<PhyloArray, PhyloCounterData>).

#### 6.5.2 Function Documentation

32 Module Documentation

#### 6.5.2.1 counter\_co\_opt()

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()

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()

Functional gains for a specific function (nfun).

Definition at line 193 of file phylo.hpp.

6.5 Phylo counters 33

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

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()

k genes gain function nfun

Definition at line 253 of file phylo.hpp.

#### 6.5.2.6 counter genes changing()

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()

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

Definition at line 1491 of file phylo.hpp.

34 Module Documentation

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

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()

Longest branch mutates (either by gain or by loss)

Definition at line 945 of file phylo.hpp.

### 6.5.2.10 counter\_loss()

Total count of losses for an specific function.

Definition at line 688 of file phylo.hpp.

#### 6.5.2.11 counter\_maxfuns()

Cap the number of functions per gene.

Definition at line 626 of file phylo.hpp.

6.5 Phylo counters 35

#### 6.5.2.12 counter\_neofun()

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()

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()

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()

Overall functional gains.

Total number of gains (irrespective of the function).

Definition at line 155 of file phylo.hpp.

36 Module Documentation

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

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()

Overall functional loss.

Definition at line 578 of file phylo.hpp.

#### 6.5.2.18 counter pairwise first gain()

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

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

Definition at line 2045 of file phylo.hpp.

#### 6.5.2.19 counter pairwise neofun singlefun()

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->1} = 1 - 2 \* x(w,a)

Definition at line 1196 of file phylo.hpp.

6.5 Phylo counters 37

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

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()

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

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

Definition at line 1906 of file phylo.hpp.

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

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()

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.

38 Module Documentation

# 6.5.2.24 counter\_subfun()

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 Treelterator class.

# 7.1.1 Detailed Description

barry: Your go-to motif accountant

# 7.2 barry::counters Namespace Reference

Tree class and Treelterator class.

#### **Namespaces**

- defm
- network
- phylo

# 7.2.1 Detailed Description

Tree class and Treelterator 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 UKNOWN = -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 UKNOWN

```
const int EXISTS::UKNOWN = -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\_)
- $\sim$ 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.

Generated by Doxygen

• BArray ()

Zero-size array.

BArray (size\_t N\_, size\_t M\_)

Empty array.

BArray (size\_t N\_, size\_t M\_, const std::vector< size\_t > &source, const std::vector< size\_t > &target, const std::vector< Cell\_Type > &value, bool add=true)

Edgelist with data.

BArray (size\_t N\_, size\_t M\_, const std::vector< size\_t > &source, const std::vector< size\_t > &target, bool add=true)

Edgelist with no data (simpler)

BArray (const BArray< Cell\_Type, Data\_Type > &Array\_, bool copy\_data=false)

Copy constructor.

- BArray < Cell\_Type, Data\_Type > & operator= (const BArray < Cell\_Type, Data\_Type > &Array\_)
   Assignment constructor.
- BArray (BArray< Cell\_Type, Data\_Type > &&x) noexcept

Move operator.

- BArray < Cell\_Type, Data\_Type > & operator= (BArray < Cell\_Type, Data\_Type > &&x) noexcept
   Move assignment.
- void set\_data (Data\_Type \*data\_, bool delete\_data\_=false)

Set the data object.

- Data\_Type \* D\_ptr ()
- const Data Type \* D ptr () const
- Data\_Type & D ()
- const Data\_Type & D () const
- void flush\_data ()

#### Queries

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

#### **Parameters**

i,j	Coordinates
check_bounds	If false 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

i,j	Row,column
check_bounds	When true and out of range, the function throws an error.
check_exists Wither check if the cell exists (before trying to delete/add), or, in the case of swap_cells, check if either of both cells exists/don't exist.	
	Swap_cciis, direct il cittel di botti cella exista, dolli exist.

- BArray< Cell\_Type, Data\_Type > & operator+= (const std::pair< size\_t, size\_t > &coords)
- BArray< Cell\_Type, Data\_Type > & operator-= (const std::pair< size\_t, size\_t > &coords)
- BArrayCell< Cell\_Type, Data\_Type > operator() (size\_t i, size\_t j, bool check\_bounds=true)
- const Cell\_Type operator() (size\_t i, size\_t j, bool check\_bounds=true) const
- void rm\_cell (size\_t i, size\_t j, bool check\_bounds=true, bool check\_exists=true)
- void insert\_cell (size\_t i, size\_t j, const Cell< Cell\_Type > &v, bool check\_bounds, bool check\_exists)
- void insert cell (size ti, size ti, 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::UKNOWN)
- 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.  $std::vector < std::unordered_maps < size_t, Cell_Type > >$ .

#### **Template Parameters**

Cell_Type	Type of cell (any type).
Data_Type	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]

Empty array.

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

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

Edgelist with data.

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

Edgelist with no data (simpler)

#### 8.1.2.5 BArray() [5/6]

Copy constructor.

#### 8.1.2.6 BArray() [6/6]

Move operator.

### 8.1.2.7 $\sim$ 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()

#### 8.1.3.2 col()

#### 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()

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

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

# 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]

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

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

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

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

#### 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()

#### 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]

#### 8.1.3.24 operator()() [2/2]

#### 8.1.3.25 operator\*=()

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

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

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

#### 8.1.3.29 operator-=() [1/3]

#### 8.1.3.30 operator-=() [2/3]

#### 8.1.3.31 operator-=() [3/3]

# 8.1.3.32 operator/=()

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

Move assignment.

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

Assignment constructor.

#### 8.1.3.35 operator==()

#### 8.1.3.36 out\_of\_range()

#### 8.1.3.37 print()

#### 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()

#### 8.1.3.40 rm\_cell()

#### 8.1.3.41 row()

# 8.1.3.42 set\_data()

Set the data object.

#### **Parameters**

data_	
delete_←	
data_	

#### 8.1.3.43 swap\_cells()

#### 8.1.3.44 swap\_cols()

# 8.1.3.45 swap\_rows()

#### 8.1.3.46 toggle\_cell()

#### 8.1.3.47 toggle\_lock()

#### 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()

# 8.1.3.50 zero\_row()

#### 8.1.4 Friends And Related Function Documentation

# ${\bf 8.1.4.1}\quad {\bf BArrayCell}{<}\ {\bf Cell\_Type,\,Data\_Type}>$

```
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()

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\*=()

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

#### 8.2.3.3 operator+=()

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

#### 8.2.3.4 operator-=()

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

#### 8.2.3.5 operator/=()

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

#### 8.2.3.6 operator=()

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

# 8.2.3.7 operator==()

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()

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"!=()

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

#### 8.3.3.3 operator<()

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

#### 8.3.3.4 operator<=()

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

#### 8.3.3.5 operator==()

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

#### 8.3.3.6 operator>()

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

#### 8.3.3.7 operator>=()

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 ti, size tj, 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 ti) const
- const Cell\_Type colsum (size\_t i) const

#### Constructors

#### **Parameters**

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

 $\bullet \ \ \mathsf{BArrayDense} < \mathsf{Cell\_Type}, \ \mathsf{Data\_Type} > \&x) \ \mathsf{noexcept} \\$ 

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 false 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**

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

- 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 i, 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 ti, size ti, 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 i, bool check bounds=true, int check exists=EXISTS::UKNOWN)
- 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 i0, size t i1, 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**

Cell_Type	Type of cell (any type).
Data_Type	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]

Empty array.

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

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

Edgelist with data.

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

Edgelist with no data (simpler)

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

Copy constructor.

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

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()

# 8.4.3.2 col() [1/2]

Definition at line 490 of file barraydense-meat.hpp.

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

Definition at line 476 of file barraydense-meat.hpp.

#### 8.4.3.4 colsum()

# 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()

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

#### 8.4.3.12 get col vec() [2/2]

# 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]

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

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

#### 8.4.3.18 insert cell() [2/2]

# 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()

# 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]

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

# 8.4.3.26 operator \*= ()

#### 8.4.3.27 operator+=() [1/3]

#### 8.4.3.28 operator+=() [2/3]

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

## 8.4.3.30 operator-=() [1/3]

# 8.4.3.31 operator-=() [2/3]

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

#### 8.4.3.33 operator/=()

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

Move assignment.

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

Assignment constructor.

#### 8.4.3.36 operator==()

# 8.4.3.37 out\_of\_range()

# 8.4.3.38 print()

### 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()

# 8.4.3.41 rm\_cell()

# 8.4.3.42 row() [1/2]

# 8.4.3.43 row() [2/2]

# 8.4.3.44 rowsum()

#### 8.4.3.45 set\_data()

Set the data object.

#### **Parameters**

data_	
delete_←	
data_	

# 8.4.3.46 swap\_cells()

#### 8.4.3.47 swap\_cols()

# 8.4.3.48 swap\_rows()

# 8.4.3.49 toggle\_cell()

# 8.4.3.50 toggle\_lock()

#### 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()

# 8.4.3.53 zero\_row()

# 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**

- $\bullet \ \ \, \mathsf{BArrayDenseCell} \ (\mathsf{BArrayDense} < \mathsf{Cell\_Type}, \mathsf{Data\_Type} > *\mathsf{Array\_}, \mathsf{size\_t} \ i\_, \mathsf{size\_t} \ j\_, \mathsf{bool} \ \mathsf{check\_bounds} = \mathsf{true})$
- 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()

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\*=()

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

#### 8.5.3.3 operator+=()

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

#### 8.5.3.4 operator-=()

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

#### 8.5.3.5 operator/=()

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

# 8.5.3.6 operator=() [1/2]

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

# 8.5.3.7 operator=() [2/2]

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

#### 8.5.3.8 operator==()

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
- $\bullet \ \ {\it class BArrayDenseCell\_const} < {\it Cell\_Type}, \ {\it Data\_Type} >$

# 8.7.1 Detailed Description

```
\label{template} \mbox{typename Cell_Type = bool, typename Data_Type = bool} > \mbox{class BArrayDenseCol} < \mbox{Cell_Type, Data_Type} > \mbox{}
```

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

## 8.7.2 Constructor & Destructor Documentation

# 8.7.2.1 BArrayDenseCol()

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()()

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()

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()()

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()

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

# 8.9.3 Member Function Documentation

# 8.9.3.1 begin()

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

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

# 8.9.3.2 end()

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

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

## 8.9.3.3 operator()()

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

# 8.9.3.4 size()

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

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

# 8.9.4 Friends And Related Function Documentation

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

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

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

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

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

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

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

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

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

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

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

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

#include <barraydenserow-bones.hpp>

#### **Public Member Functions**

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

#### **Friends**

- class BArrayDenseCell
   Cell\_Type, Data\_Type
- class BArrayDenseCell\_const< Cell\_Type, Data\_Type >

# 8.10.1 Detailed Description

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

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

# 8.10.2 Constructor & Destructor Documentation

#### 8.10.2.1 BArrayDenseRow\_const()

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()()

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)
- $\bullet \ \ \mathsf{void} \ \mathsf{operator} *= (\mathsf{const} \ \mathsf{BArrayRow} < \mathsf{Cell\_Type}, \ \mathsf{Data\_Type} > \mathsf{\&val})$
- 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()

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

#### 8.11.2.2 $\sim$ BArrayRow()

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

Definition at line 26 of file barrayrow-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\*=()

# 8.11.3.3 operator+=()

#### 8.11.3.4 operator-=()

#### 8.11.3.5 operator/=()

#### 8.11.3.6 operator=()

# 8.11.3.7 operator==()

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 barrayrow-bones.hpp.

# 8.12.2 Constructor & Destructor Documentation

#### 8.12.2.1 BArrayRow\_const()

Definition at line 49 of file barrayrow-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 barrayrow-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"!=()

#### 8.12.3.3 operator<()

#### 8.12.3.4 operator<=()

#### 8.12.3.5 operator==()

# 8.12.3.6 operator>()

# 8.12.3.7 operator>=()

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** 

Cell_Type	
Data_Type	

Definition at line 11 of file barrayvector-bones.hpp.

# 8.13.2 Constructor & Destructor Documentation

# 8.13.2.1 BArrayVector()

Construct a new BArrayVector object.

# **Parameters**

Array_	Pointer to a BArray object
dim_	Dimension. 0 means row and 1 means column.
<u>i_</u>	Element to point.
check_bounds	When true, 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 \*= ()

Definition at line 133 of file barrayvector-meat.hpp.

#### 8.13.3.7 operator+=()

Definition at line 91 of file barrayvector-meat.hpp.

#### 8.13.3.8 operator-=()

Definition at line 112 of file barrayvector-meat.hpp.

#### 8.13.3.9 operator/=()

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

# 8.13.3.10 operator=()

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

# 8.13.3.11 operator==()

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</li>
- 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()

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"!=()

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

# 8.14.3.7 operator<()

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

# 8.14.3.8 operator<=()

Definition at line 281 of file barrayvector-meat.hpp.

# 8.14.3.9 operator==()

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

# 8.14.3.10 operator>()

Definition at line 308 of file barrayvector-meat.hpp.

#### 8.14.3.11 operator>=()

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

```
\label{eq:class} \begin{tabular}{ll} template < class Cell\_Type > \\ class Cell < Cell\_Type > \\ \end{tabular}
```

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]

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

# 8.15.2.3 $\sim$ 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]

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

# 8.15.2.5 Cell() [4/7]

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]

# 8.15.3.2 add() [2/4]

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]

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"!=()

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

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

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

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

Definition at line 5 of file cell-meat.hpp.

## 8.15.3.9 operator==()

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.

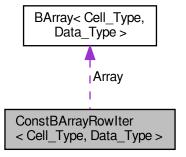
The documentation for this class was generated from the following file:

• include/barry/barray-meat.hpp

# 8.17 ConstBArrayRowlter< Cell\_Type, Data\_Type > Class Template Reference

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

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



#### **Public Member Functions**

- ConstBArrayRowlter (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

```
\label{template} \begin{tabular}{ll} template < typename Cell_Type, typename Data_Type > \\ class ConstBArrayRowlter < Cell_Type, Data_Type > \\ \end{tabular}
```

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

#### 8.17.2 Constructor & Destructor Documentation

# 8.17.2.1 ConstBArrayRowIter()

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

#### 8.17.2.2 ∼ConstBArrayRowlter()

```
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 BArray as the needed variables (see BArray::data).
data_	Data to be used with the counter.
delete_← data_	When true, the destructor will delete the pointer in the main data.

- 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]

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

# 8.18.2.3 Counter() [3/4]

Copy constructor.

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

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()

#### 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()

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

Copy assignment.

# 8.18.3.7 operator=() [2/2]

Move assignment.

#### 8.18.3.8 set\_hasher()

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** 

fun

#### 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)
- 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

```
\label{template} $$ \ensuremath{\sf template}$$ < 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]

Copy constructor.

# **Parameters**



#### 8.19.2.4 Counters() [3/3]

Move constructor.

#### **Parameters**

counters⇔

#### 8.19.3 Member Function Documentation

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

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

#### 8.19.3.3 add hash()

#### 8.19.3.4 gen\_hash()

Generates a hash for the given array according to the counters.

#### **Parameters**

array	
add_dims	When true (default) the dimmension 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]

Copy assignment constructor.

#### **Parameters**



#### Returns

Counters<Array Type,Data Type>

# 8.19.3.8 operator=() [2/2]

Move assignment constructor.

#### **Parameters**



#### Returns

Counters<Array\_Type,Data\_Type>&

# 8.19.3.9 operator[]()

Returns a pointer to a particular counter.

#### **Parameters**

```
idx 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:

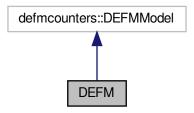
• include/barry/counters-bones.hpp

8.20 DEFM Class Reference 121

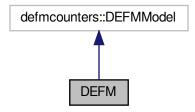
# 8.20 DEFM Class Reference

#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()

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 DEFM Class Reference 123

# 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()

8.20 DEFM Class Reference 125

#### 8.20.3.15 logodds()

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 > \mathit{idx} ) \quad [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 > \textit{Y\_names\_,} \\ std::vector < std::string > \textit{X\_names\_} ) \quad [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 ti) 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()

Definition at line 92 of file defm.hpp.

# 8.21.3.2 is\_true()

```
bool DEFMCounterData::is_true ( \label{eq:size_ti} \mbox{size\_t $i$ ) const [inline]}
```

Definition at line 94 of file defm.hpp.

#### 8.21.3.3 num()

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.

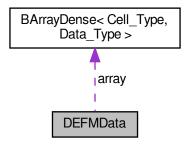
The documentation for this class was generated from the following file:

• include/barry/counters/defm.hpp

# 8.22 DEFMData Class Reference

#include <defm.hpp>

Collaboration diagram for DEFMData:



#### **Public Member Functions**

• DEFMData ()

Vector indicating which covariates are included in the model.

DEFMData (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) colum (j) data.

- double at (size\_t i, size\_t j) const
- size\_t ncol () const
- size\_t nrow () const
- · void print () const
- ∼DEFMData ()

# **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 **DEFMData()** [1/2]

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

Vector indicating which covariates are included in the model.

Definition at line 38 of file defm.hpp.

#### 8.22.2.2 DEFMData() [2/2]

Constructor.

#### **Parameters**

covariates←	Pointer to the attribute data.
_	
obs_← start_	Location of the current observation in the covariates vector
X_ncol_	Number of columns (covariates.)

Definition at line 47 of file defm.hpp.

# 8.22.2.3 $\sim$ DEFMData()

```
\texttt{DEFMData::}{\sim} \texttt{DEFMData ( ) } \quad \texttt{[inline]}
```

Definition at line 69 of file defm.hpp.

# 8.22.3 Member Function Documentation

#### 8.22.3.1 at()

#### 8.22.4 Member Data Documentation

#### 8.22.4.1 array

```
DEFMArray* DEFMData::array
```

Definition at line 30 of file defm.hpp.

#### 8.22.4.2 covar\_sort

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

Definition at line 35 of file defm.hpp.

#### 8.22.4.3 covar used

```
std::vector< size_t > DEFMData::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* DEFMData::covariates
```

Vector of covariates (complete vector)

Definition at line 31 of file defm.hpp.

#### 8.22.4.5 obs\_start

```
size_t DEFMData::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 DEFMData::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 DEFMData::X_nrow
```

Number of rows in the array of covariates.

Definition at line 34 of file defm.hpp.

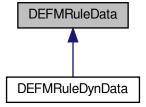
The documentation for this class was generated from the following file:

• 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 ti) 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]

Definition at line 115 of file defm.hpp.

## 8.23.2.3 DEFMRuleData() [3/3]

Definition at line 121 of file defm.hpp.

#### 8.23.3 Member Function Documentation

#### 8.23.3.1 idx()

Definition at line 110 of file defm.hpp.

#### 8.23.3.2 is\_true()

Definition at line 111 of file defm.hpp.

# 8.23.3.3 num()

```
double DEFMRuleData::num ( \label{eq:size_ti} \mbox{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.

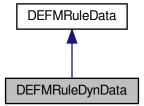
The documentation for this class was generated from the following file:

• include/barry/counters/defm.hpp

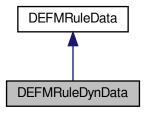
# 8.24 DEFMRuleDynData Class Reference

```
#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

```
\label{template} \begin{tabular}{ll} template < typename Cell_Type > \\ class Entries < Cell_Type > \\ \end{tabular}
```

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

# **Template Parameters**

```
Cell_Type 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]

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()

Definition at line 94 of file typedefs.hpp.

8.26 Flock Class Reference 139

## 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 ()

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 nfuns () 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
- $\bullet \ \ \mathsf{size\_t} \ \mathsf{parse\_polytomies} \ (\mathsf{bool} \ \mathsf{verb=true}, \ \mathsf{std} :: \mathsf{vector} < \mathsf{size\_t} > * \mathsf{dist=nullptr}) \ \mathsf{const} \ \mathsf{noexcept}$

Check polytomies and return the largest.

· void print () const

#### **Public Attributes**

- std::vector< Geese > dat
- size t nfunctions = 0u
- bool initialized = false
- std::mt19937 rengine
- phylocounters::PhyloModel model = phylocounters::PhyloModel()

# 8.26.1 Detailed Description

A Flock is a group of Geese.

This object buils 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 Flock Class Reference 141

## 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::\simFlock ( ) [inline]
```

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

## 8.26.3 Member Function Documentation

## 8.26.3.1 add\_data()

Add a tree to the flock.

#### **Parameters**

annotations	see Geese::Geese.
geneid	see Geese.
parent	see Geese.
duplication	see Geese.

#### 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 Flock Class Reference 143

## 8.26.3.8 init()

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

## 8.26.3.9 likelihood\_joint()

Returns the joint likelihood of the model.

#### **Parameters**

par	Vector of model parameters.	
as_log	When true it will return the value as log.	
use_reduced_sequence	use_reduced_sequence When true (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 nfuns()

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

Definition at line 167 of file flock-meat.hpp.

## 8.26.3.11 nleafs()

```
\verb|std::vector| < \verb|size_t| > \verb|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()()

Access the i-th geese element.

#### **Parameters**

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

Returns

Geese\*

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

## 8.26.3.16 parse\_polytomies()

8.26 Flock Class Reference 145

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()

Set the seed of the model.

#### **Parameters**

```
s Passed to the rengine.seed() 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 rengine

```
std::mt19937 Flock::rengine
```

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()

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()

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()

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	t Id of the parent gene. Also of length $N$	
duplication	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 true it will print out information about the encountered polytomies.

• size t nfuns () 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 true, it will compute the predictions using leave-one-out, thus the prediction will be repeated nleaf times.
only_annotated	When true, it will make the predictions only on the induced sub-tree with annotated leafs.
use_reduced_sequence	Passed to the likelihood method.
preorder	For the tree traversal.

When res\_prob is specified, the function will attach the member vector probabilities from the Nodes objects. This contains the probability that the ith node has either of the possible states.

## Returns

std::vector< double > Returns the posterior probability

- std::vector< std::vector< double >> predict (const std::vector< double > &par, std::vector< std::vector< double >> \*res\_prob=nullptr, bool leave\_one\_out=false, bool only\_annotated=false, bool use\_reduced
   \_sequence=true)
- std::vector < std::vector < double > > predict\_backend (const std::vector < double > &par, bool use\_←
  reduced\_sequence, const std::vector < 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_rengine() 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_rengine()
• 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< size\_t >>> pset\_loc

Locations of columns.

- std::vector< size\_t > sequence
- std::vector< size t > reduced sequence
- bool initialized = false
- bool delete\_rengine = 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]

Definition at line 214 of file geese-meat-constructors.hpp.

## 8.28.2.4 Geese() [4/4]

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

## 8.28.2.5 ∼Geese()

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

Definition at line 101 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 340 of file geese-meat.hpp.

#### 8.28.3.2 calc\_sequence()

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

Definition at line 296 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 462 of file geese-meat.hpp.

#### 8.28.3.4 get\_annotated\_nodes()

```
\verb|std::vector| < \verb|size_t| > \verb|Geese::get_annotated_nodes| ( ) | const | [inline]|
```

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

Definition at line 678 of file geese-meat.hpp.

## 8.28.3.5 get\_counters()

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

Definition at line 661 of file geese-meat.hpp.

## 8.28.3.6 get\_model()

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

Definition at line 666 of file geese-meat.hpp.

# 8.28.3.7 get\_probabilities()

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

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

#### 8.28.3.8 get\_rengine()

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

Definition at line 656 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  $\mathbb{P}$  functions, there will be  $2^{\mathbb{P}}$  possible combinations.

Returns

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

Definition at line 674 of file geese-meat.hpp.

## 8.28.3.10 get\_support\_fun()

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

Definition at line 670 of file geese-meat.hpp.

## 8.28.3.11 inherit\_support()

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

## 8.28.3.12 init()

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

#### 8.28.3.13 init\_node()

```
void Geese::init_node ( \label{eq:node} \mbox{Node \& $n$ ) [inline]}
```

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

## 8.28.3.14 likelihood()

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

## 8.28.3.15 likelihood\_exhaust()

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 453 of file geese-meat.hpp.

#### 8.28.3.17 nfuns()

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

Number of functions analyzed.

Definition at line 409 of file geese-meat.hpp.

## 8.28.3.18 nleafs()

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

Number of leaf.

Definition at line 423 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 416 of file geese-meat.hpp.

#### 8.28.3.20 nterms()

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

Number of terms included.

Definition at line 435 of file geese-meat.hpp.

## 8.28.3.21 observed\_counts()

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

Definition at line 504 of file geese-meat.hpp.

# 8.28.3.22 operator=() [1/2]

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

#### 8.28.3.24 parse\_polytomies()

Check polytomies and return the largest.

Definition at line 469 of file geese-meat.hpp.

## 8.28.3.25 predict()

Definition at line 243 of file geese-meat-predict.hpp.

## 8.28.3.26 predict\_backend()

< True if the array belongs to the set

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

#### 8.28.3.27 predict\_exhaust()

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

#### 8.28.3.28 predict\_exhaust\_backend()

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 638 of file geese-meat.hpp.

#### 8.28.3.31 print\_observed\_counts()

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

Definition at line 575 of file geese-meat.hpp.

#### 8.28.3.32 set\_seed()

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

#### 8.28.3.33 simulate()

```
\label{eq: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 443 of file geese-meat.hpp.

## 8.28.3.35 update\_annotations()

Definition at line 267 of file geese-meat.hpp.

## 8.28.4 Member Data Documentation

## 8.28.4.1 delete\_rengine

```
bool Geese::delete_rengine = 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< 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:

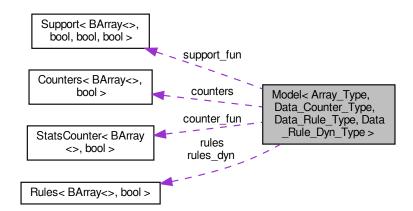
- include/barry/models/geese/geese-bones.hpp
- include/barry/models/geese/geese-meat-constructors.hpp
- include/barry/models/geese/geese-meat-likelihood.hpp
- include/barry/models/geese/geese-meat-likelihood\_exhaust.hpp
- include/barry/models/geese/geese-meat-predict.hpp
- include/barry/models/geese/geese-meat-predict\_exhaust.hpp
- 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 rengine (std::mt19937 \*rengine\_, bool delete\_=false)
- void set seed (size ts)
- 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 rengine () 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</t> 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	Vector of parameters	
as_log	When true, the function returns the log-likelihood.	

- double likelihood (const std::vector< double > &params, const size\_t &i, 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	Index relative to the array in the model.	
params	A new vector of model parameters to compute the normalizing constant.	
as_log When true returns the logged version of the normalizing constant.		

- 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\_\top frequency contains the frequency with which each of the target stats\_target (arrays) shows in the support. array2support maps array indices (0, 1, ...) to the corresponding support.

Each vector of  $stats\_support$  has the data stored in a row-wise order, with each row starting with the weights, e.g., in a model with k terms the first k + 1 elements of  $stats\_support$  would be:

- weights
- term 1
- term 2
- ..
- term k
- std::vector< std::vector< double >> stats\_support

Sufficient statistics of the model (support)

std::vector< size\_t > stats\_support\_n\_arrays

Number of arrays included per support.

 $\bullet \ \, {\sf std::vector} < {\sf std::vector} < {\sf double} > > {\sf stats\_target}$ 

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:

$$\frac{\exp\left(\theta^{\dagger}c(A)\right)}{\sum_{A'\in\mathcal{A}}\exp\left(\theta^{\dagger}c(A')\right)}$$

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**

Array_Type	Class of BArray object.
Data_Counter_Type	Any type.
Data_Rule_Type	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]

Definition at line 166 of file model-meat.hpp.

# 8.29.2.3 Model() [3/3]

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()

Adds an array to the support of not already included.

#### **Parameters**

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

#### Returns

The number of the array.

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

#### 8.29.3.3 add counter() [2/2]

## 8.29.3.4 add\_hasher()

#### 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]

## 8.29.3.7 add\_rule\_dyn() [1/2]

#### 8.29.3.8 add rule dyn() [2/2]

## 8.29.3.9 colnames()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_\times
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::string > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_\times
Dyn_Type >::colnames ( ) const
```

## 8.29.3.10 conditional\_prob()

Conditional probability ("Gibbs sampler")

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

#### **Parameters**

Array⊷	Array to check
_	
params	Vector of parameters
i	Row entry
j	Column entry

#### Returns

double The conditional probability

#### 8.29.3.11 gen\_key()

#### 8.29.3.12 get\_arrays2support()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_\times
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_\times
Type >::get_arrays2support ()
```

## 8.29.3.13 get\_counters()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_\times
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()

## 8.29.3.15 get\_pset()

#### 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]

# 8.29.3.20 get\_rengine()

```
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_rengine ( ) 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  $(k + 1) \times n \text{ unique elements}$ , 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]

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

## 8.29.3.28 likelihood() [3/4]

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

#### 8.29.3.30 likelihood total()

#### 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_\times
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=()

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 965 of file model-meat.hpp.

#### 8.29.3.36 print\_stats()

# 8.29.3.37 sample() [1/2]

# 8.29.3.38 sample() [2/2]

Definition at line 1092 of file model-meat.hpp.

#### 8.29.3.39 set counters()

#### 8.29.3.40 set\_rengine()

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()

Definition at line 139 of file model-bones.hpp.

# 8.29.3.44 set\_transform\_model()

Set the transform model fun object.

The transform\_model function is used to transform the data

#### **Parameters**

data	
target	
n_arrays	
arrays2support	

# 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()

# 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_\(\cong \)

Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>

StatsCounter<Array_Type, Data_Counter_Type> Model< Array_Type, Data_Counter_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_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_\times
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_\times
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::string > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_\times
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_\leftarray 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]

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 ()
- $\bullet \ \ {\tt NetworkData} \ ({\tt std::vector} < {\tt double} > {\tt vertex\_attr\_, bool \ directed\_=true}) \\$

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]

Constructor using a single attribute.

#### **Parameters**

vertex_←	Double vector of length equal to the number of vertices in the data.
attr_	
directed_	When true 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**

vertex_←	Vector of double vectors. The size equals to the number of attributes to be created. Each
attr_	individual vector should be of length equal to the number of vertices.
directed_	When true 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.

8.32 Node Class Reference 189

# 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 tid
```

Id of the node (as specified in the input)

· size tord

Order in which the node was created.

- · phylocounters::PhyloArray array
- 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 Node Class Reference 191

# 8.32.2.2 Node() [2/5]

Definition at line 59 of file geese-node-bones.hpp.

# 8.32.2.3 Node() [3/5]

Definition at line 65 of file geese-node-bones.hpp.

# 8.32.2.4 Node() [4/5]

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

# 8.32.2.5 Node() [5/5]

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
```

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

8.32 Node Class Reference 193

# 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()

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**

```
\bullet \  \  \, \mathsf{PhyloCounterData} \ (\mathsf{std}::\mathsf{vector} < \mathsf{size\_t} > \mathsf{data\_}, \ \mathsf{std}::\mathsf{vector} < \mathsf{double} > *\mathsf{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]

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()

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()()

Definition at line 83 of file phylo.hpp.

# 8.34.3.7 operator[]()

Definition at line 84 of file phylo.hpp.

# 8.34.3.8 push\_back()

Definition at line 86 of file phylo.hpp.

# 8.34.3.9 reserve()

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 )
- ∼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()

Definition at line 2155 of file phylo.hpp.

# 8.35.2.2 ~PhyloRuleDynData()

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

Definition at line 2164 of file phylo.hpp.

# 8.35.3 Member Data Documentation

# 8.35.3.1 counts

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

Definition at line 2149 of file phylo.hpp.

#### 8.35.3.2 duplication

```
size_t PhyloRuleDynData::duplication
```

Definition at line 2153 of file phylo.hpp.

#### 8.35.3.3 lb

size\_t PhyloRuleDynData::lb

Definition at line 2151 of file phylo.hpp.

#### 8.35.3.4 pos

size\_t PhyloRuleDynData::pos

Definition at line 2150 of file phylo.hpp.

# 8.35.3.5 ub

size\_t PhyloRuleDynData::ub

Definition at line 2152 of file phylo.hpp.

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

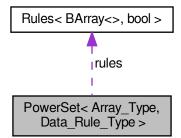
• include/barry/counters/phylo.hpp

# 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** 

Array_Type	
Data_Rule_Type	

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]

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

#### 8.36.2.3 PowerSet() [3/3]

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]

Definition at line 173 of file powerset-meat.hpp.

# 8.36.3.2 add\_rule() [2/2]

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[]()

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

#### 8.36.3.10 reset()

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 Rule_fun_type.
dat_	Data pointer to be passed to fun_
delete_←	When true, the Rule destructor will delete the pointer, if defined.
dat_	

- Rule ()
- Rule (Rule\_fun\_type< Array\_Type, Data\_Type > fun\_, Data\_Type dat\_, std::string name\_="", std::string desc ="")

# 8.38.1 Detailed Description

```
\label{template} $$ \ensuremath{\sf template}$$ < typename \ensuremath{\sf Array\_Type}$ = BArray<>, typename \ensuremath{\sf Data\_Type}$ = bool> class \ensuremath{\sf Rule}< Array\_Type, \ensuremath{\sf 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**

Array_Type	An object of class BArray.
Data_Type	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]

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

Read/Write access to the data.

#### 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 48 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 60 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 42 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 54 of file rules-meat.hpp.

#### 8.38.3.6 operator()()

Definition at line 37 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

### 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**

Array_Type	An object of class BArray
Data_Type	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]

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]

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

#### 8.39.3.2 add\_rule() [2/2]

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

#### 8.39.3.3 get\_descriptions()

```
template<typename Array_Type , typename Data_Type >
std::vector< std::string > Rules< Array_Type, Data_Type >::get_descriptions [inline]
```

Definition at line 173 of file rules-meat.hpp.

#### 8.39.3.4 get\_names()

```
template<typename Array_Type , typename Data_Type >
std::vector< std::string > Rules< Array_Type, Data_Type >::get_names [inline]
```

Definition at line 161 of file rules-meat.hpp.

#### 8.39.3.5 get\_seq()

Computes the sequence of free and locked cells in an BArray.

#### **Parameters**

а	An object of class BArray.
free	Pointer to a vector of pairs (i, j) listing the free cells.
locked	(optional) Pointer to a vector of pairs (i, j) listing the locked cells.

#### Returns

Nothing.

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

### 8.39.3.6 operator()()

Check whether a given cell is free or locked.

### **Parameters**

а	A BArray object
i	row position
j	col position

#### Returns

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

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

### 8.39.3.7 operator=()

Definition at line 19 of file rules-meat.hpp.

#### 8.39.3.8 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 ti, size tj)
- 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

```
\label{template} \begin{tabular}{ll} template < typename \ Array\_Type, \ typename \ Data\_Type > \\ class \ Stats Counter < Array\_Type, \ Data\_Type > \\ \end{tabular}
```

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]

Creator of a StatsCounter

#### **Parameters**

Array←	A const pointer to a BArray.	
_		

Definition at line 37 of file statscounter-bones.hpp.

### 8.40.2.2 StatsCounter() [2/3]

Copy constructor.

**Parameters** 

counter

### 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()

### 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()

### 8.40.3.4 count\_init()

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()

Changes the reference array for the counting.

#### **Parameters**

```
Array

→ A pointer to an array of class Array_Type.
```

### 8.40.3.9 set\_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 couter 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 realizaton).

#### Resets the support calculator

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

### Parameters

Array←	New array over which the support will be computed.
_	

- void reset\_array ()
- void reset\_array (const Array\_Type &Array\_)

#### Manage counters

#### **Parameters**

f_	A counter to be added.
counters←	A vector of counters to be added.

- void add\_counter (Counter< Array\_Type, Data\_Counter\_Type > f\_)
- void set\_counters (Counters < Array\_Type, Data\_Counter\_Type > \*counters\_)

#### Manage rules

#### **Parameters**

f_	A rule to be added.
counters←	A vector of rules to be added.
_	

- void add\_rule (Rule < Array\_Type, Data\_Rule\_Type > \*f\_)
- void add\_rule (Rule < Array\_Type, Data\_Rule\_Type > f\_)
- void set rules (Rules < Array Type, Data Rule Type > \*rules )
- void add\_rule\_dyn (Rule< Array\_Type, Data\_Rule\_Dyn\_Type > \*f\_)
   void add\_rule\_dyn (Rule< Array\_Type, Data\_Rule\_Dyn\_Type > f\_)
- void set rules dyn (Rules< Array Type, Data Rule Dyn Type > \*rules )
- bool eval rules dyn (const std::vector< double > &counts, const size t &i, const size t &i)

#### **Public Attributes**

- size t N
- size t M
- bool delete counters = true
- bool delete rules = true
- bool delete\_rules\_dyn = true
- size t max num elements = BARRY MAX NUM ELEMENTS
- std::vector< double > current\_stats
- std::vector < size\_t > coordinates\_free
- std::vector< size t > coordinates locked
- · size t coordiantes n free
- · size t coordiantes n locked
- std::vector< double > change\_stats
- std::vector< size\_t > hashes
- std::vector< bool > hashes initialized
- size\_t n\_counters

### 8.41.1 Detailed Description

template < typename Array\_Type = BArray < bool, bool >, typename Data\_Counter\_Type = bool, typename Data\_Rule\_Type = bool, typename Data\_Rule\_Dyn\_Type = bool> class Support < Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >

Compute the support of sufficient statistics.

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

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

In the case of rule\_dyn, the function will 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]

Constructor passing a reference Array.

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

#### 8.41.2.2 Support() [2/3]

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()

#### 8.41.3.2 add rule() [1/2]

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

### 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_\leftrightarrow dyn ( Rule< Array_Type, Data_Rule_Dyn_Type > f_)
```

#### 8.41.3.6 calc()

Computes the entire support.

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

#### **Parameters**

array_bank	If specified, the counter will add to the vector each possible state of the array, as it counts.
stats_bank	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_Dyn_Type >::get_counters ()
```

Vector of couter 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 realizaton).

#### 8.41.3.14 init\_support()

### 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]

### 8.41.3.18 set\_counters()

#### 8.41.3.19 set\_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> 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 \cdot _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()()

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.

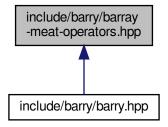
# 9.2 include/barry/barray-iterator.hpp File Reference

### 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

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

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

#### 9.3.1.5 ROW

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]

### 9.3.2.3 BARRAY\_TEMPLATE() [3/6]

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]

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()()

### 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.

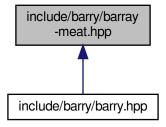
#### 9.3.3.2 this

```
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)</li>

    Data Type bool M (Array .M)

    BARRAY_TEMPLATE (BARRAY_TYPE() &, operator=)(const BArray< Cell_Type</li>

    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 ti

    BARRAY TEMPLATE (Cell Type, get cell)(size ti

    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 ti

• 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_</li>
```

#### **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

 \bullet \  \, \text{size\_t const std::vector} < \  \, \text{size\_t} > \  \, \text{const std::vector} < \  \, \text{cize\_t} > \  \, \text{const std::vector} < \  \, \text{Cell\_Type} > \& \  \, \text{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 j1
• 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

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

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

#### 9.4.1.5 ROW

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

### 9.4.2 Function Documentation

### 9.4.2.1 ans()

### 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 ( \sim \textit{BArray} \ )
```

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

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

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]

### 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]

### 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]

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]

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

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

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

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

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

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

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

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

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

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

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 (
```

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

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

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

### 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]

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 ( \label{eq:col} \texttt{COL(j0).size()} \quad = = 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 (  {\tt report !} \quad = {\tt nullptr} \; ) \\
```

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

```
if ( \label{eq:row_row_row} \mbox{ROW(i).size()} = = 0u \mbox{ )}
```

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

```
if ( \label{eq:row_row_row} \mbox{ROW(iO).size()} \ = \ = \mbox{$0$$u$} \ )
```

# 9.4.2.45 if() [16/17]

```
if ( \label{eq:row_row_row} \text{ROW(i1).size()} \quad = = 0u \text{ )}
```

# 9.4.2.46 if() [17/17]

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

## 9.4.2.47 M()

```
Data_Type bool M ( \label{eq:continuous} \text{Array}.\quad \textit{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()

# 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

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\_

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

#### 9.4.3.12 else

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_{\underline{}}
```

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

## 9.4.3.20 M\_

```
size_t M_
```

#### Initial value:

ilitiai ve

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

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

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

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

Definition at line 17 of file barraydense-meat-operators.hpp.

#### 9.8.1.7 ROW

Definition at line 14 of file barraydense-meat-operators.hpp.

#### 9.8.2 Function Documentation

## 9.8.2.1 BDENSE\_TEMPLATE() [1/4]

Definition at line 90 of file barraydense-meat-operators.hpp.

## 9.8.2.2 BDENSE\_TEMPLATE() [2/4]

Definition at line 34 of file barraydense-meat-operators.hpp.

## 9.8.2.3 BDENSE\_TEMPLATE() [3/4]

Definition at line 61 of file barraydense-meat-operators.hpp.

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

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

#### 9.8.2.5 BDENSE\_TEMPLATE\_ARGS()

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

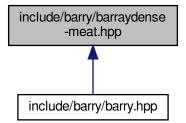
#### 9.8.2.6 BDENSE\_TYPE()

```
template const BDENSE_TYPE ( ) &
```

Definition at line 22 of file barraydense-meat-operators.hpp.

# 9.9 include/barry/barraydense-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)</li>
- 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, nnozero)() const noexcept
- BDENSE\_TEMPLATE (Cell< Cell\_Type >, default\_val)() const

   DENSE\_TEMPLATE (CELS TYPE);

   Type (Cels Type );

   Ty
- 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**

col

```
 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< 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 i 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 i1

    size_t size_t bool int int * report

    Cell_Type val0 = el[POS(i0,j0)]

Cell_Type val1 = el[POS(i1,j1)]
· false
```

#### 9.9.1 Macro Definition Documentation

## 9.9.1.1 BDENSE\_TEMPLATE

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

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

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

#### 9.9.1.7 ROW

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

#### 9.9.1.8 ZERO CELL

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

Definition at line 38 of file barraydense-meat.hpp.

# 9.9.2 Function Documentation

#### 9.9.2.1 ans()

## **9.9.2.2 BDENSE\_TEMPLATE()** [1/39]

Definition at line 240 of file barraydense-meat.hpp.

## 9.9.2.3 BDENSE\_TEMPLATE() [2/39]

## 9.9.2.4 BDENSE\_TEMPLATE() [3/39]

```
BDENSE_TEMPLATE (

BArrayDense )
```

#### 9.9.2.5 BDENSE\_TEMPLATE() [4/39]

```
BDENSE_TEMPLATE ( \sim \textit{BArrayDense} \ )
```

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

#### 9.9.2.6 BDENSE\_TEMPLATE() [5/39]

Definition at line 566 of file barraydense-meat.hpp.

#### 9.9.2.7 BDENSE\_TEMPLATE() [6/39]

Definition at line 584 of file barraydense-meat.hpp.

#### 9.9.2.8 BDENSE\_TEMPLATE() [7/39]

Definition at line 257 of file barraydense-meat.hpp.

# 9.9.2.9 BDENSE\_TEMPLATE() [8/39]

Definition at line 194 of file barraydense-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 barraydense-meat.hpp.

## 9.9.2.12 BDENSE\_TEMPLATE() [11/39]

Definition at line 562 of file barraydense-meat.hpp.

#### 9.9.2.13 BDENSE\_TEMPLATE() [12/39]

# **9.9.2.14 BDENSE\_TEMPLATE()** [13/39]

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

## 9.9.2.15 BDENSE\_TEMPLATE() [14/39]

Definition at line 994 of file barraydense-meat.hpp.

#### 9.9.2.16 BDENSE\_TEMPLATE() [15/39]

Definition at line 353 of file barraydense-meat.hpp.

#### 9.9.2.17 BDENSE\_TEMPLATE() [16/39]

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

# 9.9.2.18 BDENSE\_TEMPLATE() [17/39]

Definition at line 989 of file barraydense-meat.hpp.

# 9.9.2.19 BDENSE\_TEMPLATE() [18/39]

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

## 9.9.2.20 BDENSE\_TEMPLATE() [19/39]

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

#### 9.9.2.21 BDENSE\_TEMPLATE() [20/39]

Definition at line 502 of file barraydense-meat.hpp.

#### 9.9.2.22 BDENSE\_TEMPLATE() [21/39]

Definition at line 548 of file barraydense-meat.hpp.

# 9.9.2.23 BDENSE\_TEMPLATE() [22/39]

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

# 9.9.2.24 BDENSE\_TEMPLATE() [23/39]

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]

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

#### 9.9.2.28 BDENSE\_TEMPLATE() [27/39]

# 9.9.2.29 BDENSE\_TEMPLATE() [28/39]

# 9.9.2.30 BDENSE\_TEMPLATE() [29/39]

# 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]

# 9.9.2.34 BDENSE\_TEMPLATE() [33/39]

# 9.9.2.35 BDENSE\_TEMPLATE() [34/39]

# 9.9.2.36 BDENSE\_TEMPLATE() [35/39]

# 9.9.2.37 BDENSE\_TEMPLATE() [36/39]

#### 9.9.2.38 BDENSE\_TEMPLATE() [37/39]

Definition at line 868 of file barraydense-meat.hpp.

# 9.9.2.39 BDENSE\_TEMPLATE() [38/39]

# 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 barraydense-meat.hpp.

el [POS(i, j)] = = BARRY\_ZERO\_DENSE)

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

```
9.9.2.44 if() [3/4]
```

if (

```
if ( {\tt el\_colsums}~[j] ~=~ {\tt ZERO\_CELL}~)
```

# 9.9.2.45 if() [4/4]

```
if (
    el_rowsums [i] = =ZERO_CELL )
```

# 9.9.2.46 insert\_cell() [1/2]

# 9.9.2.47 insert\_cell() [2/2]

## 9.9.2.48 M()

```
bool M ( \label{eq:Array_.} \text{Array}\_. \quad \textit{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]

# 9.9.2.51 resize() [2/6]

# **9.9.2.52** resize() [3/6]

```
el resize ( \label{eq:N*M, ZERO_CELL} \mbox{N * $M$,}
```

# 9.9.2.53 resize() [4/6]

## 9.9.2.54 resize() [5/6]

```
el resize ( \label{eq:n_*_M_*_M_*} {\tt N_*_* M_*_*} {\tt ZERO_CELL} )
```

## 9.9.2.55 resize() [6/6]

## 9.9.2.56 rm\_cell() [1/3]

# 9.9.2.57 rm\_cell() [2/3]

## 9.9.2.58 rm\_cell() [3/3]

## 9.9.2.59 va\_end()

```
va_end (
          args )
```

#### 9.9.2.60 va\_start()

#### 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)])</pre>
```

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 barraydense-meat.hpp.

#### 9.9.3.5 col

col

Definition at line 843 of file barraydense-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 barraydense-meat.hpp.

# 9.9.3.7 copy\_data

```
bool copy_data
```

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

## 9.9.3.8 data

```
data = data_
```

Definition at line 334 of file barraydense-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:

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

Definition at line 670 of file barraydense-meat.hpp.

## 9.9.3.15 false

false

Definition at line 767 of file barraydense-meat.hpp.

## 9.9.3.16 i1

```
size_t i1
```

Definition at line 721 of file barraydense-meat.hpp.

#### 9.9.3.17 j

j

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

#### 9.9.3.18 j0

```
size_t j0
```

Definition at line 720 of file barraydense-meat.hpp.

# 9.9.3.19 j1

```
size_t j1
```

Definition at line 721 of file barraydense-meat.hpp.

#### 9.9.3.20 M

```
M = M_{\underline{}}
```

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 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 barraydense-meat.hpp.

#### 9.9.3.26 target

```
\verb|size_t| const| std::vector<|size_t| > const| std::vector<|size_t| > \& target|
```

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

#### 9.9.3.27 v

```
size_t Cell_Type v
```

Definition at line 653 of file barraydense-meat.hpp.

#### 9.9.3.28 val0

```
Cell_Type val0 = el[POS(i0,j0)]
```

Definition at line 742 of file barraydense-meat.hpp.

## 9.9.3.29 val1

```
Cell_Type val1 = el[POS(i1,j1)]
```

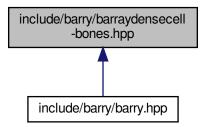
Definition at line 743 of file barraydense-meat.hpp.

#### 9.9.3.30 value

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

# 9.10 include/barry/barraydensecell-bones.hpp File Reference

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



## Classes

class BArrayDenseCell
 Cell\_Type, Data\_Type >

#### **Macros**

• #define POS(a, b) (a) + (b) \* N

# 9.10.1 Macro Definition Documentation

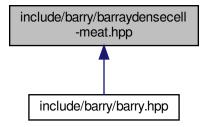
#### 9.10.1.1 POS

```
#define POS( \label{eq:a_posterior} \textbf{a}, \\ \textbf{b} \text{ ) (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
```

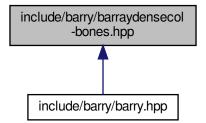
#### 9.11.1 Macro Definition Documentation

#### 9.11.1.1 POS

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 >
- $\bullet \ \, {\sf class\ BArrayDenseCol\_const} {<\ Cell\_Type,\ Data\_Type} > \\$

#### **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

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 \text{ }) \text{ } \text{ } (b) \text{ } * \text{ } \text{N } \text{ } + \text{ } (a)
```

Definition at line 4 of file barraydenserow-bones.hpp.

## 9.13.1.2 POS N

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

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

# 9.15.1.2 BROW\_TEMPLATE\_ARGS

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

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

# 9.15.1.3 BROW\_TYPE

```
#define BROW_TYPE( ) BArrayRow<Cell_Type, Data_Type>
```

Definition at line 4 of file barrayrow-meat.hpp.

# 9.15.2 Function Documentation

# 9.15.2.1 BROW\_TEMPLATE() [1/5]

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

## 9.15.2.2 BROW\_TEMPLATE() [2/5]

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

# 9.15.2.3 BROW\_TEMPLATE() [3/5]

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

# 9.15.2.4 BROW\_TEMPLATE() [4/5]

Definition at line 55 of file barrayrow-meat.hpp.

## 9.15.2.5 BROW\_TEMPLATE() [5/5]

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

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

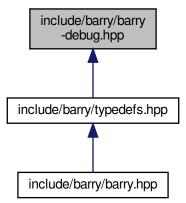
#### 9.18.2.1 Map

```
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

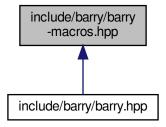
# 9.19.1.1 BARRY\_DEBUG\_LEVEL

#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 CellCell_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

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 <cstdarg>
#include <vector>
#include <unordered_map>
#include <functional>
#include <stdexcept>
#include <cmath>
#include <map>
#include <algorithm>
#include <utility>
#include <random>
#include <climits>
#include <cfloat>
#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 Treelterator 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

Definition at line 99 of file barry.hpp.

## 9.21.1.7 RULE\_FUNCTION

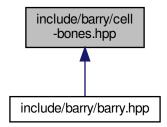
Definition at line 103 of file barry.hpp.

#### 9.21.1.8 RULE LAMBDA

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:



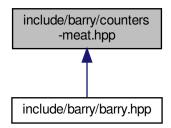
# **Classes**

- 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 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>

# **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</li>

- 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</li>
- COUNTERS\_TEMPLATE (COUNTERS\_TYPE() &, operator=)(Counters< Array\_Type
- COUNTERS TEMPLATE (void, add counter)(Counter< Array Type</li>
- 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</li>

#### **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

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

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()

# 9.26.2.2 COUNTER\_TEMPLATE() [1/9]

```
COUNTER_TEMPLATE (

Counter ) const
```

# 9.26.2.3 COUNTER\_TEMPLATE() [2/9]

# **9.26.2.4 COUNTER\_TEMPLATE()** [3/9]

# 9.26.2.5 COUNTER\_TEMPLATE() [4/9]

 $< {\sf Move \ assignment}$ 

# **9.26.2.6 COUNTER\_TEMPLATE()** [5/9]

# 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 ,

qet_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]

# 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]

# 9.26.2.19 **COUNTERS\_TEMPLATE()** [9/9]

# 9.26.2.20 data()

# 9.26.2.21 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]

Definition at line 141 of file counters-meat.hpp.

# 9.26.2.24 hasher() [2/2]

Definition at line 144 of file counters-meat.hpp.

# 9.26.2.25 hasher\_fun() [1/2]

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

## 9.26.2.26 hasher\_fun() [2/2]

# 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 ( {\tt res.} \quad {\tt size() = =0u \ )}
```

# 9.26.2.30 init\_fun() [1/3]

# 9.26.2.31 init\_fun() [2/3]

# **9.26.2.32** init\_fun() [3/3]

# 9.26.2.33 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);
    }
}
```

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.

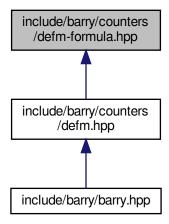
#### 9.26.3.16 return

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()

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.,  $\{\ldots\}$  >  $\{\ldots\}$ . The first set of brackets, which we call LHS, can only hold row id that are less than m\_order.

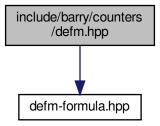
#### **Parameters**

formula	
locations	
signs	
m_order	
y_ncol	

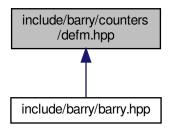
Definition at line 46 of file defm-formula.hpp.

# 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
- $\bullet \ \ typedef \ Rule < DEFMArray, \ DEFMRule DynData > DEFMRule Dyn\\$
- 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)

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 DEFMRules object (Rules < 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.

## 9.28.1 Macro Definition Documentation

# 9.28.1.1 UNI\_SUB

## 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

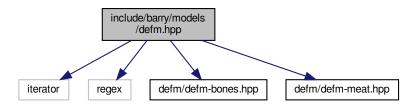
## 9.28.2.1 DEFMArray

```
typedef BArrayDense<int, DEFMData> DEFMArray
```

Definition at line 25 of file defm.hpp.

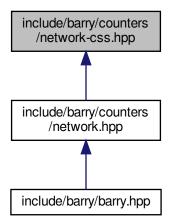
# 9.29 include/barry/models/defm.hpp File Reference

```
#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**

 $\label{template} $$ \end{template} $$ $$ \end{template} $$ \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:

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.");}</pre>
```

Definition at line 37 of file network-css.hpp.

## 9.30.1.6 CSS\_CHECK\_SIZE\_INIT

Definition at line 31 of file network-css.hpp.

## 9.30.1.7 CSS\_NET\_COUNTER\_LAMBDA\_INIT

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()

Definition at line 275 of file network-css.hpp.

# 9.30.2.2 counter\_css\_census02()

Definition at line 325 of file network-css.hpp.

## 9.30.2.3 counter\_css\_census03()

Definition at line 364 of file network-css.hpp.

## 9.30.2.4 counter\_css\_census04()

Definition at line 403 of file network-css.hpp.

# 9.30.2.5 counter\_css\_census05()

Definition at line 442 of file network-css.hpp.

# 9.30.2.6 counter\_css\_census06()

Definition at line 481 of file network-css.hpp.

## 9.30.2.7 counter\_css\_census07()

Definition at line 520 of file network-css.hpp.

## 9.30.2.8 counter\_css\_census08()

Definition at line 559 of file network-css.hpp.

# 9.30.2.9 counter\_css\_census09()

Definition at line 598 of file network-css.hpp.

# 9.30.2.10 counter\_css\_census10()

Definition at line 637 of file network-css.hpp.

#### 9.30.2.11 counter\_css\_completely\_false\_recip\_comiss()

Counts completely false reciprocity (comission)

Definition at line 154 of file network-css.hpp.

#### 9.30.2.12 counter\_css\_completely\_false\_recip\_omiss()

Counts completely false reciprocity (omission)

Definition at line 194 of file network-css.hpp.

#### 9.30.2.13 counter css mixed recip()

Counts mixed reciprocity errors.

Definition at line 234 of file network-css.hpp.

## 9.30.2.14 counter\_css\_partially\_false\_recip\_commi()

Counts errors of commission.

#### **Parameters**

netsize	Size of the reference (true) network
end⊷	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()

Counts errors of omission.

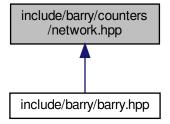
Definition at line 110 of file network-css.hpp.

# 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 (Rules < Network, bool > ).

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

Definition at line 74 of file network.hpp.

#### 9.31.1.4 NET\_C\_DATA\_NUM

Definition at line 75 of file network.hpp.

### 9.31.1.5 NETWORK\_COUNTER

#### 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

#### 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

Function for definition of a network counter function

Definition at line 133 of file network.hpp.

#### 9.31.1.8 NETWORK RULE LAMBDA

Lambda function for definition of a network counter function

Definition at line 138 of file network.hpp.

### 9.31.1.9 NETWORKDENSE\_COUNTER\_LAMBDA

Definition at line 123 of file network.hpp.

## 9.31.2 Typedef Documentation

#### 9.31.2.1 NetCounter

```
template<typename Tnet = Network>
using 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

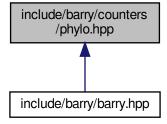
### 9.31.3.1 rules\_zerodiag()

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

- $\bullet \ \ void\ counter\_genes\_changing\ (PhyloCounters\ *counters,\ size\_t\ duplication=DEFAULT\_DUPLICATION)$ 
  - 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->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 Ou

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

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

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.

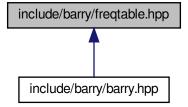
#### 9.32.3 Function Documentation

## 9.32.3.1 get\_last\_name()

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:



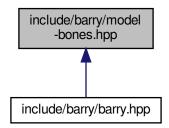
#### **Classes**

class FreqTableT >

Frequency table of vectors.

# 9.34 include/barry/model-bones.hpp File Reference

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



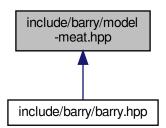
#### **Classes**

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 =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</li>

    MODEL_TEMPLATE (void, set_counters)(Counters < Array_Type</li>

    support_fun set_counters (counters)

    MODEL TEMPLATE (void, add hasher)(Hasher fun type< Array Type</li>

• MODEL TEMPLATE (void, add rule)(Rule < Array Type

    MODEL_TEMPLATE (void, set_rules)(Rules< Array_Type</li>

    support_fun set_rules (rules)

• MODEL_TEMPLATE (void, add_rule_dyn)(Rule< Array_Type

    MODEL TEMPLATE (void, set rules dyn)(Rules< Array Type</li>

    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)</li>

• 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_rengine)() 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]
• 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

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

#### 9.35.1.2 MODEL\_TEMPLATE\_ARGS

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 1301 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 1284 of file model-meat.hpp.

# 9.35.2.3 if() [2/4]

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 1186 of file model-meat.hpp.

```
9.35.2.5 if() [4/4]
```

```
if (
    transform_model_fun ) = transform_model_fun(&tmp_counts[Ou], tmp_counts.size())
```

Definition at line 434 of file model-meat.hpp.

# 9.35.2.6 insert\_cell()

## 9.35.2.7 likelihood\_()

Definition at line 59 of file model-meat.hpp.

# 9.35.2.8 MODEL\_TEMPLATE() [1/33]

## 9.35.2.9 MODEL\_TEMPLATE() [2/33]

Definition at line 1370 of file model-meat.hpp.

## 9.35.2.10 MODEL\_TEMPLATE() [3/33]

Definition at line 912 of file model-meat.hpp.

#### 9.35.2.11 MODEL TEMPLATE() [4/33]

Definition at line 924 of file model-meat.hpp.

# 9.35.2.12 MODEL\_TEMPLATE() [5/33]

#### 9.35.2.13 MODEL TEMPLATE() [6/33]

Definition at line 876 of file model-meat.hpp.

## 9.35.2.14 MODEL\_TEMPLATE() [7/33]

Definition at line 561 of file model-meat.hpp.

## 9.35.2.15 MODEL\_TEMPLATE() [8/33]

Definition at line 810 of file model-meat.hpp.

#### 9.35.2.16 MODEL TEMPLATE() [9/33]

# 9.35.2.17 MODEL\_TEMPLATE() [10/33]

Definition at line 1050 of file model-meat.hpp.

#### 9.35.2.18 MODEL\_TEMPLATE() [11/33]

Definition at line 1057 of file model-meat.hpp.

#### 9.35.2.19 MODEL\_TEMPLATE() [12/33]

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

## 9.35.2.20 MODEL\_TEMPLATE() [13/33]

Definition at line 1025 of file model-meat.hpp.

#### 9.35.2.21 MODEL\_TEMPLATE() [14/33]

Definition at line 1032 of file model-meat.hpp.

### 9.35.2.22 MODEL\_TEMPLATE() [15/33]

Definition at line 1064 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 1405 of file model-meat.hpp.

## 9.35.2.25 MODEL\_TEMPLATE() [18/33]

Definition at line 1076 of file model-meat.hpp.

#### 9.35.2.26 MODEL\_TEMPLATE() [19/33]

Definition at line 1410 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 1418 of file model-meat.hpp.

## **9.35.2.28** MODEL\_TEMPLATE() [21/33]

```
\label{eq:model_template} \mbox{MODEL\_TEMPLATE (} $$ std::vector < std::vector < double >> * , $$ get_pset_stats )
```

Definition at line 1414 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 1400 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 1395 of file model-meat.hpp.

## 9.35.2.31 MODEL\_TEMPLATE() [24/33]

#### **9.35.2.32** MODEL\_TEMPLATE() [25/33]

## 9.35.2.33 MODEL\_TEMPLATE() [26/33]

```
MODEL_TEMPLATE (
          void ,
          add_rule )
```

# 9.35.2.34 MODEL\_TEMPLATE() [27/33]

## 9.35.2.35 MODEL\_TEMPLATE() [28/33]

Definition at line 935 of file model-meat.hpp.

## 9.35.2.36 MODEL\_TEMPLATE() [29/33]

## 9.35.2.37 MODEL\_TEMPLATE() [30/33]

#### 9.35.2.38 MODEL\_TEMPLATE() [31/33]

# **9.35.2.39** MODEL\_TEMPLATE() [32/33]

Definition at line 1422 of file model-meat.hpp.

# 9.35.2.40 MODEL\_TEMPLATE() [33/33]

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

# 9.35.2.41 push\_back() [1/2]

# 9.35.2.42 push\_back() [2/2]

# 9.35.2.43 return()

```
return (  1.0+ \ std::exp-vec\_inner\_prod < \ double > (\&params[0u], \ \&tmp\_counts[0u], \ params. \leftarrow \\ size()) \ )
```

#### 9.35.2.44 set\_counters()

#### 9.35.2.45 set\_rules()

```
support_fun set_rules (
    rules )
```

## 9.35.2.46 set\_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()

#### 9.35.2.50 update\_normalizing\_constant()

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

#### 9.35.2.51 urand()

```
std::uniform_real_distribution urand ( \begin{array}{c} 0 \text{ ,} \\ 1 \end{array} )
```

# 9.35.3 Variable Documentation

# 9.35.3.1 a

```
size_t a = arrays2support[i]
```

Definition at line 1272 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 1277 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 1390 of file model-meat.hpp.

# 9.35.3.8 Data\_Rule\_Type

```
template Data_Rule_Type
```

Definition at line 1390 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

Definition at line 1294 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 1180 of file model-meat.hpp.

### 9.35.3.15 i\_matches

```
int i_matches = -1
```

Definition at line 1300 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 1282 of file model-meat.hpp.

#### 9.35.3.18 k

```
size_t k = params.size()
```

Definition at line 1279 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 1173 of file model-meat.hpp.

#### 9.35.3.22 probs

```
std::vector< double >& probs = pset_probs[a]
```

Definition at line 1283 of file model-meat.hpp.

# 9.35.3.23 pset\_arrays

```
return this pset_arrays[a][j]
```

Definition at line 1334 of file model-meat.hpp.

## 9.35.3.24 r

```
double r = urand(*rengine)
```

Definition at line 1276 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

```
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 1298 of file model-meat.hpp.

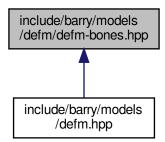
# 9.35.3.31 stats\_support\_n\_arrays

```
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:

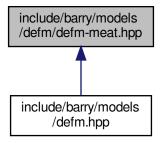


#### **Classes**

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

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

#### 9.37.1.2 **DEFM RANGES**

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

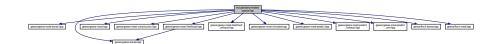
#### 9.37.2 Function Documentation

#### 9.37.2.1 keygen\_defm()

Definition at line 4 of file defm-meat.hpp.

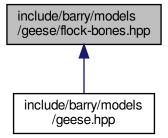
# 9.38 include/barry/models/geese.hpp File Reference

```
#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:

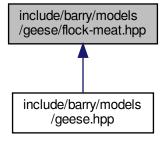


#### **Classes**

· class Flock

A Flock is a group of Geese.

# 9.40 include/barry/models/geese/flock-meat.hpp File Reference



# 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::Phylo
   — CounterData \*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()

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

#### 9.41.2.2 RULE\_FUNCTION()

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

#### 9.41.2.3 vec\_diff()

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

#### 9.41.2.4 vector\_caster()

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:

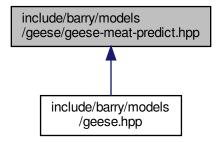


# 9.44 include/barry/models/geese/geese-meat-likelihood\_exhaust.hpp File Reference



# 9.45 include/barry/models/geese/geese-meat-predict.hpp File Reference

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

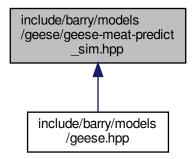


# 9.46 include/barry/models/geese/geese-meat-predict\_exhaust.hpp File Reference



# 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



# 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:



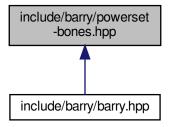
#### Classes

• class Node

A single node for the model.

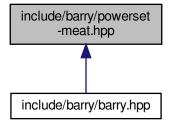
# 9.51 include/barry/powerset-bones.hpp File Reference

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



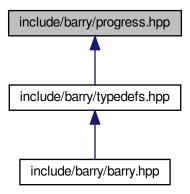
#### **Classes**

# 9.52 include/barry/powerset-meat.hpp File Reference



# 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

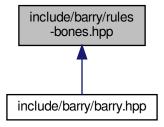
#### 9.53.1.1 BARRY\_PROGRESS\_BAR\_WIDTH

#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)
```

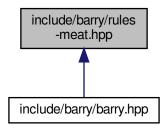
#### 9.54.1 Function Documentation

#### 9.54.1.1 rule\_fun\_default()

Definition at line 5 of file rules-bones.hpp.

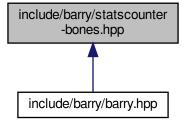
# 9.55 include/barry/rules-meat.hpp File Reference

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:

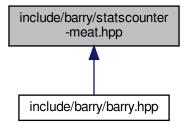


#### Classes

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 ti
- 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

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

#### 9.57.1.2 STATSCOUNTER\_TEMPLATE\_ARGS

```
template STATSCOUNTER_TEMPLATE_ARGS() <typename Array_Type</pre>, typename Data_Type>
```

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

#### 9.57.1.3 STATSCOUNTER\_TYPE

```
{\tt template\ Data\_Type*\ STATSCOUNTER\_TYPE()\ StatsCounter} < {\tt Array\_Type,Data\_Type} > {\tt template\ Data\_Type} > {\tt te
```

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()

#### 9.57.2.3 resize()

#### 9.57.2.4 STATSCOUNTER\_TEMPLATE() [1/9]

#### 9.57.2.5 STATSCOUNTER\_TEMPLATE() [2/9]

```
STATSCOUNTER_TEMPLATE ( \sim \textit{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]

#### 9.57.2.9 STATSCOUNTER\_TEMPLATE() [6/9]

#### 9.57.2.10 STATSCOUNTER\_TEMPLATE() [7/9]

#### 9.57.2.11 STATSCOUNTER\_TEMPLATE() [8/9]

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

#### 9.57.2.12 STATSCOUNTER\_TEMPLATE() [9/9]

#### 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 knowning what to count!")
```

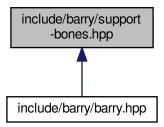
Definition at line 66 of file statscounter-meat.hpp.

#### 9.57.3.9 return

return

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

# 9.58 include/barry/support-bones.hpp File Reference

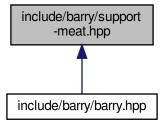


#### Classes

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)</li>
- 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</li>
- SUPPORT\_TEMPLATE (void, set\_counters)(Counters < Array\_Type</li>

```
• 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

#### 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

```
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

```
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]

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

```
9.59.2.6 if() [3/3]
```

```
if (
    rules_dyn-> size(),
    0u )
```

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

#### 9.59.2.7 insert\_cell() [1/2]

#### 9.59.2.8 insert\_cell() [2/2]

#### 9.59.2.9 rm\_cell()

#### 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]

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]

#### 9.59.2.16 SUPPORT\_TEMPLATE() [7/17]

#### 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]

#### 9.59.2.19 SUPPORT\_TEMPLATE() [10/17]

#### 9.59.2.20 SUPPORT\_TEMPLATE() [11/17]

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

#### 9.59.2.21 SUPPORT\_TEMPLATE() [12/17]

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]

Definition at line 120 of file support-meat.hpp.

#### 9.59.2.24 SUPPORT\_TEMPLATE() [15/17]

#### 9.59.2.25 SUPPORT\_TEMPLATE() [16/17]

#### 9.59.2.26 SUPPORT\_TEMPLATE() [17/17]

#### 9.59.3 Variable Documentation

#### 9.59.3.1 array\_bank

```
\verb|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] ? Ou : 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

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)
```

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

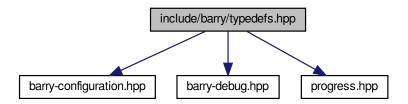
#### 9.59.3.18 tmp\_chng

```
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

## **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, 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::UKNOWN = -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**

Array_Type	a BArray
unit,size⇔	Focal cell
_t	
Data_Type	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** 

```
Array_Type
```

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()

Ascending sorting an array.

It will sort an array solving ties using the next column. Data is stored column-wise.

#### **Template Parameters**



#### **Parameters**



#### Returns

std::vector<size\_t> The sorting index.

Definition at line 141 of file typedefs.hpp.

#### 9.60.2.2 vec\_equal()

Compares if -a- and -b- are equal.

#### **Parameters**

a,b 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()

Definition at line 228 of file typedefs.hpp.

## 9.60.2.4 vec\_inner\_prod() [1/2]

Definition at line 274 of file typedefs.hpp.

## 9.60.2.5 vec\_inner\_prod() [2/2]

Definition at line 251 of file typedefs.hpp.

## 9.61 README.md File Reference

392 File Documentation

## Index

```
\simBArray
                                                           NetworkData, 188
    BArray< Cell Type, Data Type >, 47
                                                      \simNode
\simBArrayCell
                                                           Node, 191
    BArrayCell< Cell_Type, Data_Type >, 58
                                                      \simPhyloRuleDynData
~BArrayCell const
                                                           PhyloRuleDynData, 200
    BArrayCell_const< Cell_Type, Data_Type >, 60
                                                      \simPowerSet
{\sim}\mathsf{BArrayDense}
                                                           PowerSet < Array_Type, Data_Rule_Type >, 203
    BArrayDense < Cell_Type, Data_Type >, 66
                                                      \simProgress
                                                           Progress, 208
\simBArrayDenseCell
    BArrayDenseCell< Cell_Type, Data_Type >, 79
                                                      \simRule
\simBArrayRow
                                                           Rule < Array_Type, Data_Type >, 210
    BArrayRow< Cell Type, Data Type >, 92
                                                      \simRules
\simBArrayRow const
                                                           Rules < Array Type, Data Type >, 213
    BArrayRow const< Cell Type, Data Type >, 94
                                                      \simStatsCounter
\simBArrayVector
                                                           StatsCounter < Array_Type, Data_Type >, 217
    BArrayVector< Cell Type, Data Type >, 97
                                                      \simSupport
                                                                                       Data Counter Type,
~BArrayVector const
                                                           Support<
                                                                        Array_Type,
    BArrayVector_const< Cell_Type, Data_Type >,
                                                               Data_Rule_Type, Data_Rule_Dyn_Type >,
                                                               222
\simCell
                                                      а
    Cell< Cell_Type >, 104
                                                           model-meat.hpp, 348
\simConstBArrayRowIter
    ConstBArrayRowlter< Cell_Type, Data_Type >,
                                                      active
                                                           Cell< Cell_Type >, 107
         109
                                                      add
\simCounter
                                                           barray-meat.hpp, 248
    Counter< Array_Type, Data_Type >, 112
                                                           barraydense-meat.hpp, 273
\simCounters
                                                           Cell < Cell Type >, 105, 106
    Counters < Array_Type, Data_Type >, 117
                                                           FreqTable < T >, 148
\simDEFMCounterData
                                                      add_array
    DEFMCounterData, 127
                                                           Model <
                                                                       Array_Type,
                                                                                       Data Counter Type,
\simDEFMData
                                                               Data_Rule_Type, Data_Rule_Dyn_Type >,
    DEFMData, 130
\simDEFMRuleDynData
                                                      add counter
    DEFMRuleDynData, 136
                                                           Counters < Array_Type, Data_Type >, 117, 118
\simEntries
                                                                       Array_Type,
                                                                                       Data_Counter_Type,
                                                           Model<
    Entries < Cell Type >, 138
                                                               Data_Rule_Type, Data_Rule_Dyn_Type >,
\simFlock
    Flock, 141
                                                           StatsCounter< Array_Type, Data_Type >, 218
\simFreqTable
                                                                                       Data_Counter_Type,
                                                           Support<
                                                                       Array_Type,
    FreqTable < T >, 147
                                                               Data_Rule_Type, Data_Rule_Dyn_Type >,
\simGeese
                                                               223
    Geese, 154
                                                      add data
\simModel
                                                           Flock, 141
    Model<
                Array Type,
                                Data Counter Type,
         Data_Rule_Type, Data_Rule_Dyn_Type >,
                                                      add dims
                                                           counters-meat.hpp, 303
         168
                                                      add hash
\simNetCounterData
                                                           Counters < Array_Type, Data_Type >, 118
    NetCounterData, 186
                                                      add hasher
\simNetworkData
```

Data Rule Type, Data Rule Dyn Type > 189  PowerSet < Array_Type, Data_Rule_Type > 203, 204  Rules < Array_Type, Data_Type > 214  Support < Array_Type, Data_Rule_Dyn_Type > 223  add_rule_dyn Model < Array_Type, Data_Rule_Dyn_Type > 170  Support < Array_Type, Data_Tule_Dyn_Type > 170  Support < Array_Type, Data_Type > 170  array_meat.hpp, 239, 248  barraydense-meat.hpp, 262, 273  Array  DEFMData, 131  Node, 192  Array_ Barray-meat.hpp, 248  array_Barray	Model< Array_Type, Data_Counter_Type,     Data_Rule_Type, Data_Rule_Dyn_Type >,     169  add_rule     Model< Array_Type, Data_Counter_Type,	BArrayCell_const< Cell_Type, Data_Type >, 56 clear, 47 col, 47 D, 48
PowerSet< Array_Type, Data_Rule_Type >, 203, 204	Data_Rule_Type, Data_Rule_Dyn_Type >,	
Rules< Array_Type, Data_Type >, 214 Support< Array_Type, Data_Counter_Type,     Data_Rule_Type, Data_Rule_Dyn_Type >,     Data_Rule_Type, Data_Rule_Dyn_Type >,     T70  Model< Array_Type, Data_Rule_Dyn_Type >,     T70 Support< Array_Type, Data_Rule_Dyn_Type >,     T70 Support< Array_Type, Data_Rule_Dyn_Type >,     Data_Rule_Type, Data_Type >,     Topstatray-meat.hpp, 239, 248     Datray-meat.hpp, 262, 273 Array DEFMData, 131 Node, 192 Array_Dank Support—meat.hpp, 382 Array_Dank Support—meat.hpp, 382 Array_Data_Rule_Type, Data_Rule_Dyn_Type >,     T19 Array_Stalid Node, 192 Array_Stalid Node, 193 AS_ONE EXISTS, 41 SEXISTS, 41 SEXISTS, 41 SEXISTS, 41 SEXISTS, 41 BARRAY_TEMPLATE_ARGS, 233, 235 BARRAY_TEMPLATE_ARGS, 233, 235 BARRAY_TEMPLATE_ARGS, 233 SONE EXISTS, 41 BARRAY_Cell_Type, Data_Type >, 46, 47 BARRAY_Cell_Type, Data_Type >, 48 BARRAY_TEMPLATE_ARGS, 238 BARRAY_TEMPLATE_AR		get_cell, 48
Support		<b>-</b> - ·
Data_Rule_Type, Data_Rule_Dyn_Type >, insert_cell, 50 is_dense, 50 is_		<b>5</b> =  ,
223		<del>-</del>
Model		
Data_Rule_Type, Data_Rule_Dyn_Type >, 170  Support< Array_Type, Data_Counter_Type, Operator*=, 51 Operator*=, 52 Operator*=, 52 Operator*=, 53 Operator*=, 52 Operator*=, 52 Operator*=, 52 Operator*=, 52 Operator*=, 52 Operator*=, 52 Operator*=, 53 Operator*=, 52 Operator*=, 5	add_rule_dyn	is_empty, 50
170		
Support		
Data_Rule_Type, Data_Rule_Dyn_Type >, 223 annotations		
223 annotations Node, 192 ans barray-meat.hpp, 239, 248 barraydense-meat.hpp, 262, 273 Array ConstBArrayRowlter< Cell_Type, Data_Type > 109 array DEFMData, 131 Node, 192 array DEFMData, 192 Array DEFMData, 192 array DEFMData, 192 array Dank support-meat.hpp, 248 array_bank support-meat.hpp, 382 arrays_vanid Node, 192 Array Node, 192 Array Node, 192 Array Node, 192 BARRAY_TEMPLATE, 232, 233 AS_ONE EXISTS, 41 as_vector FreqTable< T >, 148 AS_ZERO EXISTS, 41 at DEFMData, 131 PhyloCounterData, 197 BArray BArray BArray Cell_Type, Data_Type >, 46, 47 BArray BArray BArray BArray BArray, 46, 47  poperator'=, 52 operator'=, 53 operator'e, 53 operator'e, 53 out_of_range, 53 purl, t, 53 operator'=, 53 operator'=, 53 operator'e, 54 operator'e,		•
annotations		•
Node, 192         operator/=, 53           ans         operator/=, 53           barray-meat.hpp, 239, 248         operator==, 53           barraydense-meat.hpp, 262, 273         out_of_range, 53           Array         print, 53           ConstBArrayRowlter         Cell_Type, Data_Type >, reserve, 54           109         resize, 54           array         rm_cell, 54           DEFMData, 131         row, 54           Node, 192         swap_cells, 55           array-bank         swap_rows, 55           support-meat.hpp, 382         toggle_lock, 55           arrays         toggle_lock, 55           Node, 192         transpose, 56           arraysesupport         visited, 57           Model         Array_Type, Data_Rule_Dyn_Type         zero_col, 56           2ero_row, 56         zero_row, 56           barray-walid         BARRAY_TEMPLATE, 232-234           Node, 193         BARRAY_TEMPLATE, 232-234           BASCRO         BARRAY_TEMPLATE, 233, 235           EXISTS, 41         COL, 233           at         DEFMData, 131         phyloCounterData, 197           BArray         BARRAY_Template, 238           BArray         Cell_Type, Data_Type >, 46, 47		•
barray-meat.hpp, 239, 248	Node, 192	•
barraydense-meat.hpp, 262, 273 Array  ConstBArrayRowlter < Cell_Type, Data_Type >, reserve, 54 resize, 54 rray  DEFMData, 131 Node, 192 Array_bank support-meat.hpp, 382 arrays Node, 192 arrays2support Model < Array_Type, Data_Counter_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 < Cell_Type, Data_Type >, 46, 47 BArray, 48, 47 BArray, 48, 47 BArray, 48, 47  BARRAY_TEMPLATE, 238-243 BARRAY_TEMPLATE, 238-243 BARRAY_TEMPLATE, 238-248 BARRAY_TYPE, 238 check bounds, 248	ans	operator=, 53
Array		•
ConstBArrayRowlter		·
109 array DEFMData, 131 Node, 192 Array_ barray-meat.hpp, 248 array_bank support-meat.hpp, 382 arrays2support Model	•	•
array		
DEFMData, 131		
Node, 192  Array_	•	
barray-meat.hpp, 248 array_bank support-meat.hpp, 382 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< Cell_Type, Data_Type >, 46, 47 BArray, 46, 47  BArray, 46, 47  swap_cols, 55 swap_rows, 55 toggle_cell, 55 toggle_cell, 55 toggle_lock, 55 toggle_olck, 55 toggle_olck for all supplementaries togg		set_data, 54
array_bank	Array_	swap_cells, 55
support-meat.hpp, 382 arrays Node, 192 arrays2support Model		• —
arrays		• —
Node, 192  arrays2support  Model		
arrays2support  Model < Array_Type, Data_Counter_Type,         Data_Rule_Type, Data_Rule_Dyn_Type >,         179  arrays_valid         Node, 193  AS_ONE         EXISTS, 41         S_ZERO         EXISTS, 41  at         DEFMData, 131         PhyloCounterData, 197  BArray < Cell_Type, Data_Type >, 46, 47  BArray < Cell_Type, Data_Type >, 43         ~BArray, 46, 47  BArray, 46, 47  BArray, 46, 47  Data_Rule_Type, Data_Dype, Data_Counter_Type, zero_col, 56         zero_crow, 56         barray-meat-operators.hpp  BARRAY_TEMPLATE, 232–234  BARRAY_TYPE, 233, 235  COL, 233         for, 235         coperator(), 235         rhs, 235         perator(), 235         rhs, 235         ans, 239, 248         Array_meat.hpp         add, 248         ans, 239, 248         Array_, 248         BARRAY_TEMPLATE, 238–243         BARRAY_TEMPLATE, 238–243         BARRAY_TEMPLATE, 238–243         BARRAY_TEMPLATE, 238–243         BARRAY_TYPE, 238         check bounds, 248	•	
Model       Array_Type, Data_Counter_Type, Data_Rule_Dyn_Type >, 179       zero_col, 56         179       zero_row, 56         barray-meat-operators.hpp         BARRAY_TEMPLATE, 232–234       BARRAY_TEMPLATE_ARGS, 233, 235         AS_ONE       BARRAY_TYPE, 233, 235         EXISTS, 41       COL, 233         as_vector       for, 235         FreqTable       rhs, 235         EXISTS, 41       ROW, 233         at       NOW, 233         at       barray-meat.hpp         BAPRAY_TYPE, 233, 235       AS_2ERO         EXISTS, 41       ROW, 235         at       at         DEFMData, 131       barray-meat.hpp         PhyloCounterData, 197       add, 248         BArray       Array_, 248         BARRAY_TEMPLATE, 238–243       BARRAY_TEMPLATE, 238–243         BARRAY_TEMPLATE, 238–243		·
179 arrays_valid     Node, 193  AS_ONE     EXISTS, 41  as_vector     FreqTable < T >, 148      DEFMData, 131     PhyloCounterData, 197  BArray     BArray  BArray < Cell_Type, Data_Type >, 46, 47  BArray, 47  BArray, 46, 47   barray-meat-operators.hpp  BARRAY_TEMPLATE, 232–234  BARRAY_TYPE, 233, 235  COL, 233  for, 235  cOL, 233  for, 235  rhs, 235  ROW, 233  this, 236  barray-meat.hpp  add, 248  ans, 239, 248  Array_ 248  BARRAY_TEMPLATE, 238–243  BARRAY_TEMPLATE, 238–243  BARRAY_TEMPLATE ARGS, 238  BARRAY_TYPE, 238  check bounds, 248		
arrays_valid	Data_Rule_Type, Data_Rule_Dyn_Type >,	zero_row, 56
Node, 193	179	• • • • • • • • • • • • • • • • • • • •
AS_ONE	• —	
EXISTS, 41  as_vector  FreqTable < T >, 148  AS_ZERO  EXISTS, 41  DEFMData, 131  PhyloCounterData, 197  BArray  BArray < Cell_Type, Data_Type >, 46, 47  BArray, 46, 47  EXISTS, 41  COL, 233  for, 235  for, 235  ROW, 235  rhs, 235  ROW, 233  this, 236  barray-meat.hpp  add, 248  ans, 239, 248  Array_, 248  BARRAY_TEMPLATE, 238–243  BARRAY_TYPE, 238  check bounds, 248		
as_vector	<del>_</del>	<del>-</del>
FreqTable < T >, 148  AS_ZERO  EXISTS, 41  ROW, 233  at  DEFMData, 131  PhyloCounterData, 197  BArray  BArray < Cell_Type, Data_Type >, 46, 47  BArray < Cell_Type, Data_Type >, 43  ~BArray, 46, 47  operator(), 235  rhs, 235  ROW, 233  this, 236  barray-meat.hpp  add, 248  ans, 239, 248  Array_, 248  BARRAY_TEMPLATE, 238–243  BARRAY_TEMPLATE_ARGS, 238  BARRAY_TYPE, 238  check bounds, 248	•	•
EXISTS, 41  at  DEFMData, 131  PhyloCounterData, 197  BArray  BArray  BArray  Cell_Type, Data_Type >, 46, 47  BArray, 47  BArray, 46, 47  BArray, 46, 47  BOW, 233  this, 236  barray-meat.hpp  add, 248  ans, 239, 248  Array_, 248  BARRAY_TEMPLATE, 238–243  BARRAY_TEMPLATE_ARGS, 238  BARRAY_TYPE, 238  check bounds, 248	<del>-</del>	
at this, 236  DEFMData, 131 barray-meat.hpp PhyloCounterData, 197 add, 248  BArray BArray Cell_Type, Data_Type >, 46, 47  BArray Cell_Type, Data_Type >, 43  ~BArray, 47  BArray, 46, 47  this, 236 barray-meat.hpp add, 248  Array, 248  BARRAY_TEMPLATE, 238–243  BARRAY_TEMPLATE_ARGS, 238  BARRAY_TYPE, 238  check bounds, 248	·	•
DEFMData, 131 PhyloCounterData, 197  BArray BArray Cell_Type, Data_Type >, 46, 47  BArray, 47 BArray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 46, 47  Barray, 48  Barray, 48  Barray, 48  Check bounds, 248	EXISTS, 41	
PhyloCounterData, 197  add, 248  ans, 239, 248  BArray  BArray  Cell_Type, Data_Type >, 46, 47  BARRAY_TEMPLATE, 238–243  BARRAY_TEMPLATE_ARGS, 238  BARRAY_TYPE, 238  BARRAY_TYPE, 238  check bounds, 248		
BArray ans, 239, 248  BArray Cell_Type, Data_Type >, 46, 47  BArray Cell_Type, Data_Type >, 43  BARRAY_TEMPLATE, 238–243  BARRAY_TEMPLATE_ARGS, 238  BARRAY_TYPE, 238  BARRAY_TYPE, 238  check bounds, 248		•
BArray BArray	PhyloCounterData, 197	
BArray < Cell_Type, Data_Type >, 46, 47  BARRAY_TEMPLATE, 238–243  BARRAY_TEMPLATE, 238–243  BARRAY_TEMPLATE_ARGS, 238  BARRAY_TYPE, 238  BARRAY_TYPE, 238  check bounds, 248	BArray	
BArray < Cell_Type, Data_Type >, 43  ~BArray, 47  BARRAY_TEMPLATE_ARGS, 238  BARRAY_TYPE, 238  check bounds, 248		· —
~BArray, 47  BARRAY_TYPE, 238  check bounds, 248		
GICON DOUNDS, 240	•	BARRAY_TYPE, 238
BArray(:ell< (:ell lyne l)ata lyne > 56	BArray, 46, 47 BArrayCell < Cell_Type, Data_Type >, 56	check_bounds, 248

check_exists, 249	BArrayCell_const, 60
COL, 239, 243	operator Cell_Type, 61
col0, 249	operator!=, 61
const, 249	operator<, 61
copy_data, 249	operator<=, 61
data, 250	operator>, 61
delete_data, 250	operator>=, 62
delete data , 250	operator==, 61
else, 250	BArrayDense
false, 250	BArrayDense< Cell_Type, Data_Type >, 65, 66
first, 251	BArrayDense < Cell_Type, Data_Type >, 62
for, 244	~BArrayDense, 66
i1, 251	BArrayDense, 65, 66
if, 244–247	BArrayDenseCell< Cell_Type, Data_Type >, 76,
j, 251	81
j0, 251	BArrayDenseCol< Cell_Type, Data_Type >, 77, 84
•	
j1, 251 M. 247, 251	BArrayDenseCol_const< Cell_Type, Data_Type >, 77
M, 247, 251	
M_, 252	BArrayDenseRow< Cell_Type, Data_Type >, 77,
N, 252	88
NCells, 252	BArrayDenseRow_const< Cell_Type, Data_Type
report, 252	>, 77
resize, 247	clear, 67
return, 247, 252	col, 67
ROW, 239, 247, 248	colsum, 67
row0, 253	D, 67, 68
search, 253	D_ptr, 68
source, 253	default_val, 68
target, 253	get_cell, 68
v, 253	get_col_vec, 68, 69
value, 253	get_data, 69
BARRAY_TEMPLATE	get_entries, 69
barray-meat-operators.hpp, 232-234	get_row_vec, 69
barray-meat.hpp, 238-243	insert_cell, 70
BARRAY_TEMPLATE_ARGS	is_dense, 70
barray-meat-operators.hpp, 233, 235	is_empty, 70
barray-meat.hpp, 238	ncol, 70
BARRAY_TYPE	nnozero, 71
barray-meat-operators.hpp, 233, 235	nrow, 71
barray-meat.hpp, 238	operator*=, 71
BArrayCell	operator(), 71
BArrayCell< Cell_Type, Data_Type >, 58	operator+=, 71, 72
BArrayCell< Cell_Type, Data_Type >, 57	operator-=, 72
∼BArrayCell, 58	operator/=, 72
BArray< Cell_Type, Data_Type >, 56	operator=, 73
BArrayCell, 58	operator==, 73
operator Cell_Type, 58	out of range, 73
operator*=, 58	print, 73
operator+=, 58	reserve, 73
operator-=, 59	resize, 74
operator/=, 59	rm_cell, 74
operator=, 59	row, 74
operator==, 59	rowsum, 74
BArrayCell_const	set_data, 74
BArrayCell_const< Cell_Type, Data_Type >, 60	swap_cells, 75
BArrayCell_const< Cell_Type, Data_Type >, 60	swap_cells, 75 swap_cols, 75
~BArrayCell_const, 60	swap_cois, 75 swap_rows, 75
BArray Cell_Type, Data_Type >, 56	toggle_cell, 75
DAITAY Cell_Type, Data_Type >, 30	toggie_teil, 70

toggle_lock, 76	value, 278
transpose, 76	ZERO_CELL, 262
visited, 77	BArrayDenseCell
zero_col, 76	BArrayDenseCell< Cell_Type, Data_Type >, 79
zero_row, 76	BArrayDenseCell< Cell Type, Data Type >, 78
barraydense-meat-operators.hpp	~BArrayDenseCell, 79
BDENSE_TEMPLATE, 256-258	BArrayDense< Cell_Type, Data_Type >, 76, 81
BDENSE_TEMPLATE_ARGS, 256, 258	BArrayDenseCell, 79
BDENSE_TYPE, 256, 258	BArrayDenseCol < Cell_Type, Data_Type >, 81, 84
COL, 256	BArrayDenseCol_const< Cell_Type, Data_Type >,
POS, 256	81, 86
POS_N, 257	BArrayDenseRow< Cell_Type, Data_Type >, 88
ROW, 257	BArrayDenseRow_const< Cell_Type, Data_Type
barraydense-meat.hpp	>, 91
add, 273	operator Cell_Type, 79
ans, 262, 273	operator*=, 79
BDENSE_TEMPLATE, 261-269	operator+=, 79
BDENSE_TEMPLATE_ARGS, 261	operator-=, 80
BDENSE_TYPE, 261	operator/=, 80
check_bounds, 273	operator=, 80
check_exists, 273	operator==, 80
COL, 261	barraydensecell-bones.hpp
col, 274	POS, 279
const, 274	barraydensecell-meat.hpp
copy_data, 274	POS, 280
data, 274	BArrayDenseCell_const< Cell_Type, Data_Type >, 82
delete_data, 274	BArrayDenseCol < Cell_Type, Data_Type >, 84
delete_data_, 275	BArrayDenseCol_const< Cell_Type, Data_Type >,
el, 275	86
el_colsums, 275	BArrayDenseRow< Cell_Type, Data_Type >, 88
el_rowsums, 275	BArrayDenseRow_const< Cell_Type, Data_Type
else, 275	>, 91
false, 276	BArrayDenseCol
for, 269	BArrayDenseCol< Cell_Type, Data_Type >, 82
i1, 276	BArrayDenseCol< Cell_Type, Data_Type >, 82
if, 270	BArrayDense< Cell_Type, Data_Type >, 77, 84
insert_cell, 270	BArrayDenseCell< Cell_Type, Data_Type >, 81,
j, 276	84
j0, 276	BArrayDenseCell_const< Cell_Type, Data_Type
j1, 276	>, 84
M, 271, 276	BArrayDenseCol, 82
M_, 277	begin, 83
N, 277	end, 83
POS, 261	operator(), 83
POS_N, 261	size, 83
printf_barry, 271	barraydensecol-bones.hpp
report, 277	POS, 281
resize, 271, 272	POS_N, 281
return, 277	ZERO_CELL, 281
rm_cell, 272	BArrayDenseCol_const
ROW, 262	BArrayDenseCol_const< Cell_Type, Data_Type >,
source, 277	85
target, 278	BArrayDenseCol_const< Cell_Type, Data_Type >, 84
v, 278	BArrayDense < Cell_Type, Data_Type >, 77
va_end, 272	BArrayDenseCell< Cell_Type, Data_Type >, 81,
va_start, 273	86
val0, 278	BArrayDenseCell_const< Cell_Type, Data_Type
val1, 278	>, 86

BArrayDenseCol_const, 85	operator<, 94
begin, 85	operator<=, 95
end, 85	operator>, 95
operator(), 85	operator>=, 95
size, 86	operator==, 95
BArrayDenseRow	BArray Vector
BArrayDenseRow< Cell_Type, Data_Type >, 87	BArrayVector < Cell_Type, Data_Type >, 96
BArrayDenseRow< Cell_Type, Data_Type >, 86	BArrayVector< Cell_Type, Data_Type >, 95
BArrayDense < Cell_Type, Data_Type >, 77, 88	~BArrayVector, 97
BArrayDenseCell< Cell_Type, Data_Type >, 88	BArrayVector, 96
BArrayDenseCell_const< Cell_Type, Data_Type	begin, 97
>, 88	end, 97
BArrayDenseRow, 87	is_col, 97
begin, 87	is_row, 97
end, 87	operator std::vector< Cell_Type >, 98
operator(), 88	operator*=, 98
size, 88	operator+=, 98
barraydenserow-bones.hpp	operator-=, 98
POS, 282	operator/=, 98
POS_N, 282	operator=, 99
ZERO_CELL, 283 BArrayDenseRow_const	operator==, 99 size, 99
BArrayDenseRow_const< Cell_Type, Data_Type	BArrayVector_const
>, 89	BArrayVector_const< Cell_Type, Data_Type >,
BArrayDenseRow_const< Cell_Type, Data_Type >, 89	100
BArrayDense< Cell_Type, Data_Type >, 77	BArrayVector_const< Cell_Type, Data_Type >, 99
BArrayDenseCell< Cell_Type, Data_Type >, 91	~BArrayVector_const, 100
BArrayDenseCell_const< Cell_Type, Data_Type  BArrayDenseCell_const< Cell_Type, Data_Type	BArrayVector_const, 100
>, 91	begin, 101
BArrayDenseRow_const, 89	end, 101
begin, 90	is_col, 101
end, 90	is_row, 101
operator(), 90	operator std::vector< Cell_Type >, 101
size, 90	operator!=, 101
BArrayRow	operator<, 102
BArrayRow< Cell_Type, Data_Type >, 92	operator<=, 102
BArrayRow< Cell_Type, Data_Type >, 91	operator>, 102
~BArrayRow, 92	operator>=, 102
BArrayRow, 92	operator==, 102
operator BArrayRow< Cell Type, Data Type >, 92	size, 103
operator*=, 92	barry, 39
operator+=, 92	barry-configuration.hpp
operator-=, 92	BARRY_CHECK_SUPPORT, 287
operator/=, 93	BARRY_ISFINITE, 287
operator=, 93	BARRY_MAX_NUM_ELEMENTS, 287
operator==, 93	BARRY_SAFE_EXP, 287
barrayrow-meat.hpp	Map, 288
BROW_TEMPLATE, 283-285	printf_barry, 287
BROW_TEMPLATE_ARGS, 284	barry-debug.hpp
BROW_TYPE, 284	BARRY_DEBUG_LEVEL, 288
BArrayRow_const	barry-macros.hpp
BArrayRow_const< Cell_Type, Data_Type >, 94	BARRY_ONE, 289
BArrayRow_const< Cell_Type, Data_Type >, 93	BARRY_ONE_DENSE, 289
$\sim$ BArrayRow_const, 94	BARRY_UNUSED, 289
BArrayRow_const, 94	BARRY_ZERO, 290
operator BArrayRow_const< Cell_Type, Data_Type	BARRY_ZERO_DENSE, 290
>, 94	barry.hpp
operator!=, 94	BARRY_HPP, 291

BARRY_VERSION, 292	BArrayDenseCol_const< Cell_Type, Data_Type >,
BARRY_VERSION_MAYOR, 292	85
BARRY_VERSION_MINOR, 292	BArrayDenseRow< Cell_Type, Data_Type >, 87
COUNTER_FUNCTION, 292	BArrayDenseRow_const< Cell_Type, Data_Type
COUNTER_LAMBDA, 292	>, 90
RULE_FUNCTION, 293	BArrayVector< Cell_Type, Data_Type >, 97
RULE_LAMBDA, 293	BArrayVector_const< Cell_Type, Data_Type >,
barry::counters, 39	101
barry::counters::defm, 40	PhyloCounterData, 197
barry::counters::network, 40	PowerSet < Array_Type, Data_Rule_Type >, 204
barry::counters::phylo, 40 BARRY_CHECK_SUPPORT	blengths NodeData, 196
barry-configuration.hpp, 287	BOTH
BARRY_DEBUG_LEVEL	CHECK, 40
barry-debug.hpp, 288	EXISTS, 42
BARRY HPP	BROW_TEMPLATE
barry.hpp, 291	barrayrow-meat.hpp, 283–285
BARRY_ISFINITE	BROW TEMPLATE ARGS
barry-configuration.hpp, 287	barrayrow-meat.hpp, 284
BARRY_MAX_NUM_ELEMENTS	BROW_TYPE
barry-configuration.hpp, 287	barrayrow-meat.hpp, 284
BARRY_ONE	
barry-macros.hpp, 289	calc
BARRY_ONE_DENSE	PowerSet < Array_Type, Data_Rule_Type >, 204
barry-macros.hpp, 289	Support< Array_Type, Data_Counter_Type,
BARRY_PROGRESS_BAR_WIDTH	Data_Rule_Type, Data_Rule_Dyn_Type >,
progress.hpp, 366	223
BARRY_SAFE_EXP	calc_backend_dense
barry-configuration.hpp, 287	support-meat.hpp, 377
BARRY_SUPPORT_MEAT_HPP	calc_backend_sparse
support-meat.hpp, 376	support-meat.hpp, 377
BARRY_UNUSED	calc_reduced_sequence
barry-macros.hpp, 289	Geese, 154
BARRY_VERSION	calc_sequence
barry.hpp, 292	Geese, 154
BARRY_VERSION_MAYOR	Cell
barry.hpp, 292	Cell < Cell_Type >, 104, 105
BARRY_VERSION_MINOR	Cell< Cell_Type >, 103
barry.hpp, 292	∼Cell, 104
BARRY_ZERO	active, 107
barry-macros.hpp, 290	add, 105, 106
BARRY_ZERO_DENSE	Cell, 104, 105
barry-macros.hpp, 290	operator Cell_Type, 106
BARRY_ZERO_NETWORK	operator!=, 106
network.hpp, 323	operator=, 106, 107 operator==, 107
BARRY_ZERO_NETWORK_DENSE	value, 107
network.hpp, 323	visited, 107
BDENSE_TEMPLATE	Cell_const< Cell_Type >, 108
barraydense-meat-operators.hpp, 256–258	change_stats
barraydense-meat.hpp, 261–269	Support< Array_Type, Data_Counter_Type,
BDENSE_TEMPLATE_ARGS	Data_Rule_Type, Data_Rule_Dyn_Type >,
barraydense-meat-operators.hpp, 256, 258	227
barraydense-meat.hpp, 261	change_stats_different
BDENSE_TYPE	support-meat.hpp, 382
barraydense-meat-operators.hpp, 256, 258	CHECK, 40
barraydense-meat.hpp, 261	BOTH, 40
begin  PArroy Donos Coll - Coll Type - Data - Type > 92	NONE, 40
BArrayDenseCol< Cell_Type, Data_Type >, 83	ONE, 40

TWO, 41	coordiantes_n_locked
check_bounds	Support< Array_Type, Data_Counter_Type,
barray-meat.hpp, 248	Data_Rule_Type, Data_Rule_Dyn_Type >,
barraydense-meat.hpp, 273	227
check_exists	coordinates_free
barray-meat.hpp, 249	PowerSet < Array_Type, Data_Rule_Type >, 206
barraydense-meat.hpp, 273	Support< Array_Type, Data_Counter_Type,
clear	Data_Rule_Type, Data_Rule_Dyn_Type >,
BArray< Cell_Type, Data_Type >, 47	227
BArrayDense < Cell_Type, Data_Type >, 67	coordinates_locked
FreqTable $< T >$ , 148	PowerSet < Array_Type, Data_Rule_Type >, 206
statscounter-meat.hpp, 370	Support< Array_Type, Data_Counter_Type,
COL	Data_Rule_Type, Data_Rule_Dyn_Type >,
barray-meat-operators.hpp, 233	228
barray-meat.hpp, 239, 243	copy_data
barraydense-meat-operators.hpp, 256	barray-meat.hpp, 249
barraydense-meat.hpp, 261	barraydense-meat.hpp, 274
col	count
BArray< Cell_Type, Data_Type >, 47	Counter< Array_Type, Data_Type >, 113
BArrayDense < Cell_Type, Data_Type >, 67	count_all
barraydense-meat.hpp, 274	StatsCounter< Array_Type, Data_Type >, 218
col0	count_current
barray-meat.hpp, 249	StatsCounter< Array_Type, Data_Type >, 218
Col_type	count_fun
typedefs.hpp, 388	Counter< Array_Type, Data_Type >, 114
colnames	counters-meat.hpp, 298
Flock, 141	count_fun_
Geese, 155	counters-meat.hpp, 303
Model< Array_Type, Data_Counter_Type,	model-meat.hpp, 348
Data_Rule_Type, Data_Rule_Dyn_Type >,	count_init
170	StatsCounter< Array_Type, Data_Type >, 218
colsum	Counter
BArrayDense < Cell_Type, Data_Type >, 67	Counter< Array_Type, Data_Type >, 112
conditional_prob	counter
Model < Array_Type, Data_Counter_Type,	counters-meat.hpp, 304
Data_Rule_Type, Data_Rule_Dyn_Type >,	model-meat.hpp, 349
170	statscounter-meat.hpp, 372
const	Counter< Array_Type, Data_Type >, 110
barray-meat.hpp, 249	$\sim$ Counter, 112
barraydense-meat.hpp, 274	count, 113
ConstBArrayRowlter	count_fun, 114
ConstBArrayRowIter< Cell_Type, Data_Type >,	Counter, 112
109	data, 114
ConstBArrayRowIter< Cell_Type, Data_Type >, 108	desc, 115
$\sim$ ConstBArrayRowlter, 109	get_description, 113
Array, 109	get_hasher, 113
ConstBArrayRowlter, 109	get_name, 113
current_col, 110	hasher_fun, 115
current_row, 110	init, 113
iter, 110	init_fun, 115
coord_i	name, 115
support-meat.hpp, 382	operator=, 113, 114
coord_j	set_hasher, 114
support-meat.hpp, 382	counter_
coordiantes_n_free	counters-meat.hpp, 304
Support< Array_Type, Data_Counter_Type,	counter_absdiff
Data_Rule_Type, Data_Rule_Dyn_Type >,	DEFMArray counters, 21
227	counter_co_opt

Phylo counters, 31	Phylo counters, 32
counter_cogain	counter_gains_k_offspring
Phylo counters, 32	Phylo counters, 33
counter_css_census01	counter_genes_changing
network-css.hpp, 316	Phylo counters, 33
counter_css_census02	counter_idegree
network-css.hpp, 316	DEFMArray counters, 24
counter_css_census03	counter_idegree15
network-css.hpp, 316	DEFMArray counters, 24
counter_css_census04	counter isolates
network-css.hpp, 317	DEFMArray counters, 24, 25
counter_css_census05	counter istar2
network-css.hpp, 317	DEFMArray counters, 25
counter_css_census06	counter_k_genes_changing
network-css.hpp, 317	Phylo counters, 33
counter_css_census07	COUNTER_LAMBDA
network-css.hpp, 317	barry.hpp, 292
counter css census08	counter_less_than_p_prop_genes_changing
network-css.hpp, 318	Phylo counters, 33
counter_css_census09	counter_logit_intercept
network-css.hpp, 318	DEFMArray counters, 25
counter_css_census10	counter_longest
network-css.hpp, 318	Phylo counters, 34
counter_css_completely_false_recip_comiss	counter_loss
network-css.hpp, 318	Phylo counters, 34
counter_css_completely_false_recip_omiss	counter_maxfuns
network-css.hpp, 319	Phylo counters, 34
counter_css_mixed_recip	counter_mutual
network-css.hpp, 319	DEFMArray counters, 25
counter_css_partially_false_recip_commi	counter_neofun
network-css.hpp, 319	Phylo counters, 34
counter_css_partially_false_recip_omiss	counter_neofun_a2b
network-css.hpp, 320	Phylo counters, 35
counter ctriads	counter nodecov
DEFMArray counters, 22	DEFMArray counters, 26
counter_degree	counter_nodeicov
DEFMArray counters, 22	DEFMArray counters, 26
counter_deleted	counter_nodematch
statscounter-meat.hpp, 372	DEFMArray counters, 26
counter_density	counter_nodeocov
DEFMArray counters, 22	DEFMArray counters, 26
counter_diff	counter_odegree
DEFMArray counters, 23	DEFMArray counters, 26, 27
counter_edges	counter odegree15
DEFMArray counters, 23	DEFMArray counters, 27
counter_fixed_effect	counter ones
DEFMArray counters, 23	DEFMArray counters, 27
counter_fun	counter_ostar2
Model< Array_Type, Data_Counter_Type,	DEFMArray counters, 28
Data_Rule_Type, Data_Rule_Dyn_Type >,	counter_overall_changes
179	Phylo counters, 35
Counter_fun_type	counter_overall_gains
typedefs.hpp, 388	Phylo counters, 35
COUNTER FUNCTION	counter_overall_gains_from_0
barry.hpp, 292	Phylo counters, 35
counter_gains	counter_overall_loss
Phylo counters, 32	Phylo counters, 36
counter_gains_from_0	counter_pairwise_first_gain
— <del>-</del> — — -	<del></del>

Phylo counters, 36	desc, 301
counter_pairwise_neofun_singlefun	desc_, 304
Phylo counters, 36	for, 301
counter_pairwise_overall_change	fun, 305
Phylo counters, 36	fun_, 305
counter_pairwise_preserving	hasher, 301, 302
Phylo counters, 37	hasher_fun, 302
counter_preserve_pseudogene	hasher_fun_, 305
Phylo counters, 37	i, 305
counter_prop_genes_changing	if, 302
Phylo counters, 37	init_fun, 303
counter_subfun	init_fun_, 306
Phylo counters, 37	j, 306
COUNTER_TEMPLATE	name, 303
counters-meat.hpp, 296, 298, 299	name_, 306
COUNTER_TEMPLATE_ARGS	noexcept, 306
counters-meat.hpp, 296	res, 306
counter_transition	return, 307
DEFMArray counters, 28	TMP_HASHER_CALL, 297
counter_transition_formula	counters_
DEFMArray counters, 29	model-meat.hpp, 349
counter_ttriads	statscounter-meat.hpp, 373
DEFMArray counters, 29	support-meat.hpp, 383
COUNTER_TYPE	COUNTERS_TEMPLATE
counters-meat.hpp, 297	counters-meat.hpp, 297, 299–301
Counters	COUNTERS_TEMPLATE_ARGS
Counters < Array_Type, Data_Type >, 116, 117	counters-meat.hpp, 297
counters	COUNTERS_TYPE
Model< Array_Type, Data_Counter_Type,	counters-meat.hpp, 297
Data_Rule_Type, Data_Rule_Dyn_Type >,	Counting, 13
179	counts
statscounter-meat.hpp, 373	DEFMRuleDynData, 137
support-meat.hpp, 382	PhyloRuleDynData, 200
Counters< Array_Type, Data_Type >, 116	Counts_type
$\sim$ Counters, 117	typedefs.hpp, 388
add_counter, 117, 118	covar_sort
add_hash, 118	DEFMData, 131
Counters, 116, 117	covar_used
gen_hash, 118	DEFMData, 131
get_descriptions, 118	covariates
get_names, 119	DEFMData, 131
operator=, 119	CSS_APPEND
operator[], 120	network-css.hpp, 314
size, 120	CSS_CASE_ELSE
counters-meat.hpp	network-css.hpp, 314
add_dims, 303	CSS_CASE_PERCEIVED
count_fun, 298	network-css.hpp, 314
count_fun_, 303	CSS_CASE_TRUTH
counter, 304	network-css.hpp, 314
counter_, 304	CSS_CHECK_SIZE
COUNTER_TEMPLATE, 296, 298, 299	network-css.hpp, 314
COUNTER_TEMPLATE_ARGS, 296	CSS_CHECK_SIZE_INIT
COUNTER_TYPE, 297	network-css.hpp, 315
COUNTERS_TEMPLATE, 297, 299–301	CSS_NET_COUNTER_LAMBDA_INIT
COUNTERS_TEMPLATE_ARGS, 297	network-css.hpp, 315
COUNTERS_TYPE, 297	CSS_PERCEIVED_CELLS
data, 301	network-css.hpp, 315
data_, 304	CSS_SIZE

network-css.hpp, 315	likelihood, 124
CSS_TRUE_CELLS	logodds, 124
network-css.hpp, 316	motif_census, 125
cumprob	print, 125
model-meat.hpp, 349	set_names, 125
current col	simulate, 125
ConstBArrayRowIter< Cell_Type, Data_Type >,	defm-formula.hpp
110	defm_motif_parser, 307
current row	defm-meat.hpp
ConstBArrayRowlter< Cell_Type, Data_Type >,	DEFM_LOOP_ARRAYS, 355
110	DEFM_RANGES, 355
current_stats	keygen_defm, 356
statscounter-meat.hpp, 373	defm.hpp
• •	• •
Support< Array_Type, Data_Counter_Type,	DEFMArray, 311
Data_Rule_Type, Data_Rule_Dyn_Type >,	UNI_SUB, 311
228	DEFM_COUNTER
D	Phylo rules, 15
	DEFM_COUNTER_LAMBDA
BArray Cell_Type, Data_Type >, 48	Phylo rules, 15
BArrayDense < Cell_Type, Data_Type >, 67, 68	DEFM_LOOP_ARRAYS
Rule < Array_Type, Data_Type >, 211	defm-meat.hpp, 355
D_ptr	defm_motif_parser
BArray< Cell_Type, Data_Type >, 48	defm-formula.hpp, 307
BArrayDense< Cell_Type, Data_Type >, 68	DEFM RANGES
dat	defm-meat.hpp, 355
Flock, 145	DEFM RULE
data	Phylo rules, 15
barray-meat.hpp, 250	DEFM_RULE_LAMBDA
barraydense-meat.hpp, 274	Phylo rules, 16
Counter< Array_Type, Data_Type >, 114	DEFM_RULEDYN_LAMBDA
counters-meat.hpp, 301	Phylo rules, 16
PowerSet< Array_Type, Data_Rule_Type >, 206	DEFMArray
data	·
counters-meat.hpp, 304	defm.hpp, 311
model-meat.hpp, 349	DEFMArray counters, 19
Data_Counter_Type	counter_absdiff, 21
	counter_ctriads, 22
model-meat.hpp, 349	counter_degree, 22
Data_Rule_Type	counter_density, 22
model-meat.hpp, 350	counter_diff, 23
DEFAULT_DUPLICATION	counter_edges, 23
phylo.hpp, 330	counter_fixed_effect, 23
default_val	counter_idegree, 24
BArray< Cell_Type, Data_Type >, 48	counter_idegree15, 24
BArrayDense< Cell_Type, Data_Type >, 68	counter_isolates, 24, 25
DEFM, 121	counter istar2, 25
DEFM, 122	counter_logit_intercept, 25
get_ID, 122	counter mutual, 25
get_m_order, 122	counter nodecov, 26
get_model, 122	counter nodeicov, 26
get_n_covars, 123	counter_nodematch, 26
get_n_obs, 123	counter_nodeocov, 26
get_n_rows, 123	
get_n_y, 123	counter_odegree, 26, 27
get_X, 123	counter_odegree15, 27
get_X_names, 123	counter_ones, 27
get_Y, 124	counter_ostar2, 28
get_Y_names, 124	counter_transition, 28
· — —	counter_transition_formula, 29
init, 124	counter_ttriads, 29
is_motif, 124	

NETWORK_COUNTER, 29	180
rules_dont_become_zero, 29	Support< Array_Type, Data_Counter_Type,
rules_markov_fixed, 30	Data_Rule_Type, Data_Rule_Dyn_Type >,
DEFMCounter	228
Phylo rules, 17	support-meat.hpp, 383
DEFMCounterData, 126	delete_data
~DEFMCounterData, 127	barray-meat.hpp, 250
DEFMCounterData, 126	barraydense-meat.hpp, 274
idx, 127 indices, 128	delete_data_ barray-meat.hpp, 250
is_motif, 128	barraydense-meat.hpp, 275
is_true, 127	delete_rengine
logical, 128	Geese, 161
num, 127	Model< Array_Type, Data_Counter_Type,
numbers, 128	Data_Rule_Type, Data_Rule_Dyn_Type >,
DEFMCounters	180
Phylo rules, 17	delete_rules
DEFMData, 129	Model< Array_Type, Data_Counter_Type,
∼DEFMData, 130	Data_Rule_Type, Data_Rule_Dyn_Type >,
array, 131	180
at, 131	model-meat.hpp, 350
covar_sort, 131	Support< Array_Type, Data_Counter_Type,
covar_used, 131	Data_Rule_Type, Data_Rule_Dyn_Type >,
covariates, 131	228
DEFMData, 130	support-meat.hpp, 383
obs_start, 131	delete_rules_dyn
X_ncol, 132	Model < Array_Type, Data_Counter_Type,
X_nrow, 132	Data_Rule_Type, Data_Rule_Dyn_Type >,
DEFMModel	180
Phylo rules, 17	model-meat.hpp, 350
DEFMRule	Support< Array_Type, Data_Counter_Type,
Phylo rules, 17	Data_Rule_Type, Data_Rule_Dyn_Type >,
DEFMRuleData, 132	228
DEFMRuleData, 133	support-meat.hpp, 383
idx, 134	delete_support
indices, 134	Geese, 161
init, 134	desc
is_true, 134	Counter< Array_Type, Data_Type >, 115
logical, 135	counters-meat.hpp, 301
num, 134	desc_
numbers, 135	counters-meat.hpp, 304
DEFMRuleDyn	directed
Phylo rules, 17	NetworkData, 188
DEFMRuleDynData, 135	DUPL_DUPL
$\sim$ DEFMRuleDynData, 136	phylo.hpp, 330
counts, 137	DUPL_EITH
DEFMRuleDynData, 136	phylo.hpp, 330
DEFMRules	DUPL_SPEC
Phylo rules, 17	phylo.hpp, 330
DEFMRulesDyn	duplication
Phylo rules, 18	Node, 193
DEFMStatsCounter	NodeData, 196
Phylo rules, 18	PhyloRuleDynData, 200
DEFMSupport	el
Phylo rules, 18	barraydense-meat.hpp, 275
delete_counters	el_colsums
Model Array_Type, Data_Counter_Type,	barraydense-meat.hpp, 275
Data_Rule_Type, Data_Rule_Dyn_Type >,	el_rowsums
	51_15440dillio

barraydense-meat.hpp, 275	$\sim$ Flock, 141
else	add_data, 141
barray-meat.hpp, 250	colnames, 141
barraydense-meat.hpp, 275	dat, 145
model-meat.hpp, 350	Flock, 141
support-meat.hpp, 383	get_counters, 142
empty	get_model, 142
PhyloCounterData, 197	get_stats_support, 142
EmptyArray	get_stats_target, 142
PowerSet < Array_Type, Data_Rule_Type >, 206	get_support_fun, 142
statscounter-meat.hpp, 373	init, 142
end	initialized, 145
BArrayDenseCol $<$ Cell_Type, Data_Type $>$ , 83	likelihood_joint, 143
${\sf BArrayDenseCol\_const} < {\sf Cell\_Type},  {\sf Data\_Type} >,$	model, 146
85	nfunctions, 146
BArrayDenseRow $<$ Cell_Type, Data_Type $>$ , 87	nfuns, 143
BArrayDenseRow_const< Cell_Type, Data_Type	nleafs, 143
>, 90	nnodes, 143
BArrayVector< Cell_Type, Data_Type >, 97	nterms, 144
$BArrayVector\_const < Cell\_Type, Data\_Type >$ ,	ntrees, 144
101	operator(), 144
PhyloCounterData, 198	parse_polytomies, 144
PowerSet < Array_Type, Data_Rule_Type >, 204	print, 145
Progress, 209	rengine, 146
Entries	set_seed, 145
Entries< Cell_Type >, 138	support_size, 145
Entries < Cell_Type >, 137	flush_data
$\sim$ Entries, 138	BArray< Cell_Type, Data_Type >, 48
Entries, 138	for
resize, 138	barray-meat-operators.hpp, 235
source, 139	barray-meat.hpp, 244
target, 139	barraydense-meat.hpp, 269
val, 139	counters-meat.hpp, 301
eval_rules_dyn	model-meat.hpp, 339
Support< Array_Type, Data_Counter_Type,	statscounter-meat.hpp, 370
Data_Rule_Type, Data_Rule_Dyn_Type >,	support-meat.hpp, 377
224	force_new
EXISTS, 41	model-meat.hpp, 350
AS_ONE, 41	FreqTable
AS_ZERO, 41	FreqTable < T >, 147
BOTH, 42	FreqTable < T >, 146
NONE, 42	$\sim$ FreqTable, 147
ONE, 42	add, 148
TWO, 42	as_vector, 148
UKNOWN, 42	clear, 148
£	FreqTable, 147
f_	get_data, 148
statscounter-meat.hpp, 373	get_index, 148
support-meat.hpp, 384	make_hash, 149
false	print, 149
barray-meat.hpp, 250	reserve, 149
barraydense-meat.hpp, 276	size, 149
first harray most hap 251	fun
barray-meat.hpp, 251	counters-meat.hpp, 305
first_calc_done  Model	fun_
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >,	counters-meat.hpp, 305
	model-meat.hpp, 351
180 Flock 139	Gaesa 150

$\sim$ Geese, 154	Geese, 155
calc_reduced_sequence, 154	get_arrays2support
calc_sequence, 154	Model< Array_Type, Data_Counter_Type
colnames, 155	Data_Rule_Type, Data_Rule_Dyn_Type $>$
delete_rengine, 161	171
delete_support, 161	get_cell
Geese, 153, 154	BArray< Cell_Type, Data_Type >, 48
get_annotated_nodes, 155	BArrayDense < Cell_Type, Data_Type >, 68
get_counters, 155	get_col_vec
get_model, 155	BArray< Cell_Type, Data_Type >, 49
get_probabilities, 155	BArrayDense < Cell_Type, Data_Type >, 68, 69
get_rengine, 155	get_counters
get_states, 156	Flock, 142
get_support_fun, 156	Geese, 155
inherit_support, 156	Model< Array_Type, Data_Counter_Type
init, 156	Data_Rule_Type, Data_Rule_Dyn_Type >
init_node, 156	171
initialized, 161	PhyloCounterData, 198
likelihood, 157	StatsCounter< Array_Type, Data_Type >, 218
likelihood exhaust, 157	Support< Array_Type, Data_Counter_Type
map_to_nodes, 162	Data_Rule_Type, Data_Rule_Dyn_Type >
nannotations, 157	224
nfunctions, 162	get_counts
nfuns, 157	Support< Array_Type, Data_Counter_Type
nleafs, 157	Data_Rule_Type, Data_Rule_Dyn_Type >
nnodes, 158	224
nodes, 162	get_current_stats
nterms, 158	Support< Array_Type, Data_Counter_Type
observed_counts, 158	Data_Rule_Type, Data_Rule_Dyn_Type >
operator=, 158	224
parse_polytomies, 158	get_data
predict, 159	BArrayDense < Cell_Type, Data_Type >, 69
predict_backend, 159	FreqTable $< T >$ , 148
predict_exhaust, 159	PowerSet < Array_Type, Data_Rule_Type >, 204
predict exhaust backend, 159	Support< Array_Type, Data_Counter_Type
predict_sim, 160	Data_Rule_Type, Data_Rule_Dyn_Type >
print, 160	225
print_observed_counts, 160	get_data_ptr
pset_loc, 162	PowerSet< Array Type, Data Rule Type >, 205
reduced sequence, 162	get_description
sequence, 162	Counter< Array_Type, Data_Type >, 113
set_seed, 160	Rule < Array_Type, Data_Type >, 211
simulate, 160	get_descriptions
support_size, 161	Counters < Array_Type, Data_Type >, 118
update_annotations, 161	Rules < Array Type, Data Type >, 214
geese-bones.hpp	StatsCounter< Array_Type, Data_Type >, 218
INITIALIZED, 358	
	get_entries  BArray< Cell Type, Data Type >, 49
keygen_full, 359	
RULE_FUNCTION, 359	BArrayDense< Cell_Type, Data_Type >, 69
vec_diff, 359	get_hasher
vector_caster, 359	Counter < Array_Type, Data_Type >, 113
gen_hash	get_ID
Counters < Array_Type, Data_Type >, 118	DEFM, 122
gen_key	get_index
Model < Array_Type, Data_Counter_Type,	FreqTable < T >, 148
Data_Rule_Type, Data_Rule_Dyn_Type >,	• — —
171	phylo.hpp, 335
get_annotated_nodes	get_m_order

DEEM 100	005
DEFM, 122	225
get_model	get_rules_dyn
DEFM, 122	Model < Array_Type, Data_Counter_Type,
Flock, 142	Data_Rule_Type, Data_Rule_Dyn_Type >, 173
Geese, 155	
get_n_covars	Support< Array_Type, Data_Counter_Type,
DEFM, 123	Data_Rule_Type, Data_Rule_Dyn_Type >, 225
get_n_obs DEFM, 123	
,	get_seq
get_n_rows DEFM, 123	Rules < Array_Type, Data_Type >, 214
_	get_states
get_n_y DEFM, 123	Geese, 156 get_stats_support
	Flock, 142
get_name Counter< Array_Type, Data_Type >, 113	Model < Array_Type, Data_Counter_Type,
Rule < Array_Type, Data_Type >, 211	Data_Rule_Type, Data_Rule_Dyn_Type >,
	173
get_names Counters< Array Type, Data Type >, 119	get_stats_target
Rules < Array Type, Data Type >, 214	Flock, 142
StatsCounter< Array Type, Data Type >, 219	Model < Array Type, Data Counter Type,
get norm const	Data_Rule_Type, Data_Rule_Dyn_Type >,
Model Array_Type, Data_Counter_Type,	173
Data_Rule_Type, Data_Rule_Dyn_Type >,	
171	Flock, 142
get_parent	Geese, 156
Node, 191	Model< Array_Type, Data_Counter_Type,
get_probabilities	Data_Rule_Type, Data_Rule_Dyn_Type >,
Geese, 155	173
get_pset	get_X
Model	DEFM, 123
Data_Rule_Type, Data_Rule_Dyn_Type >,	get_X_names
171	DEFM, 123
get_pset_arrays	get_Y
Model< Array_Type, Data_Counter_Type,	DEFM, 124
Data_Rule_Type, Data_Rule_Dyn_Type >,	get_Y_names
172	DEFM, 124
get_pset_probs	,
Model< Array_Type, Data_Counter_Type,	hasher
Data_Rule_Type, Data_Rule_Dyn_Type >,	counters-meat.hpp, 301, 302
172	hasher_fun
get_pset_stats	Counter< Array_Type, Data_Type >, 115
Model < Array_Type, Data_Counter_Type,	counters-meat.hpp, 302
Data_Rule_Type, Data_Rule_Dyn_Type >,	hasher_fun_
172	counters-meat.hpp, 305
get_rengine	Hasher_fun_type
Geese, 155	typedefs.hpp, 388
Model< Array_Type, Data_Counter_Type,	hashes
Data_Rule_Type, Data_Rule_Dyn_Type >,	Support< Array_Type, Data_Counter_Type,
172	Data_Rule_Type, Data_Rule_Dyn_Type >,
get_row_vec	229
BArray< Cell_Type, Data_Type >, 49	support-meat.hpp, 384
BArrayDense < Cell_Type, Data_Type >, 69	hashes_initialized
get_rules	Support< Array_Type, Data_Counter_Type,
Model< Array_Type, Data_Counter_Type,	Data_Rule_Type, Data_Rule_Dyn_Type >,
Data_Rule_Type, Data_Rule_Dyn_Type >,	229
173	İ
Support	counters-meat.hpp, 305
Data_Rule_Type, Data_Rule_Dyn_Type >,	model-meat.hpp, 351

i1		include/barry/models/geese/flock-bones.hpp, 357
	barray-meat.hpp, 251	include/barry/models/geese/flock-meat.hpp, 357
	barraydense-meat.hpp, 276	include/barry/models/geese/geese-bones.hpp, 358
i_ma	atches	include/barry/models/geese/geese-meat-constructors.hpp,
	model-meat.hpp, 351	360
id		include/barry/models/geese/geese-meat-likelihood.hpp,
	Node, 193	360
idx		include/barry/models/geese/geese-meat-likelihood_exhaust.hpp,
	DEFMCounterData, 127	361
	DEFMRuleData, 134	include/barry/models/geese/geese-meat-predict.hpp,
if		362
	barray-meat.hpp, 244–247	include/barry/models/geese/geese-meat-predict_exhaust.hpp,
	barraydense-meat.hpp, 270	362
	counters-meat.hpp, 302	include/barry/models/geese/geese-meat-predict_sim.hpp,
	model-meat.hpp, 339, 340	363
	support-meat.hpp, 377, 378	include/barry/models/geese/geese-meat-simulate.hpp,
IF N	MATCHES	363
_	phylo.hpp, 330	include/barry/models/geese/geese-meat.hpp, 364
IF N	NOTMATCHES	include/barry/models/geese/geese-node-bones.hpp,
_	phylo.hpp, 331	364
inclu	ude/barry/barray-bones.hpp, 231	include/barry/powerset-bones.hpp, 365
	ude/barry/barray-iterator.hpp, 231	include/barry/powerset-meat.hpp, 365
	ude/barry/barray-meat-operators.hpp, 232	include/barry/progress.hpp, 366
	ude/barry/barray-meat.hpp, 236	include/barry/rules-bones.hpp, 367
	ude/barry/barraycell-bones.hpp, 254	include/barry/rules-meat.hpp, 368
	ude/barry/barraycell-meat.hpp, 254	include/barry/statscounter-bones.hpp, 368
	ude/barry/barraydense-bones.hpp, 255	include/barry/statscounter-meat.hpp, 369
	ude/barry/barraydense-meat-operators.hpp, 255	include/barry/support-bones.hpp, 374
	ude/barry/barraydense-meat.hpp, 258	include/barry/support-meat.hpp, 375
	ude/barry/barraydensecell-bones.hpp, 279	include/barry/typedefs.hpp, 386
	ude/barry/barraydensecell-meat.hpp, 280	indices
	ude/barry/barraydensecol-bones.hpp, 280	DEFMCounterData, 128
	ude/barry/barraydenserow-bones.hpp, 282	DEFMRuleData, 134
	ude/barry/barrayrow-bones.hpp, 283	NetCounterData, 186
	ude/barry/barrayrow-meat.hpp, 283	inherit_support
	ude/barry/barrayvector-bones.hpp, 285	Geese, 156
	ude/barry/barrayvector-meat.hpp, 286	init
	ude/barry/barry-configuration.hpp, 286	Counter< Array_Type, Data_Type >, 113
	ude/barry/barry-debug.hpp, 288	DEFM, 124
	ude/barry/barry-macros.hpp, 289	DEFMRuleData, 134
	ude/barry/barry.hpp, 290	Flock, 142
	ude/barry/cell-bones.hpp, 293	Geese, 156
	ude/barry/cell-meat.hpp, 294	init_fun
	ude/barry/col-bones.hpp, 294	Counter< Array_Type, Data_Type >, 115
	ude/barry/counters-bones.hpp, 294	counters-meat.hpp, 303
	ude/barry/counters-meat.hpp, 295	init_fun_
	ude/barry/counters/defm-formula.hpp, 307	counters-meat.hpp, 306
	ide/barry/counters/defm.hpp, 309	model-meat.hpp, 351
	ide/barry/counters/network-css.hpp, 312	init node
	ude/barry/counters/network-css.hpp, 312	Geese, 156
	ude/barry/counters/phylo.hpp, 327	init_support
	ide/barry/freqtable.hpp, 335	PowerSet < Array_Type, Data_Rule_Type >, 205
	ide/barry/model-bones.hpp, 336	Support< Array_Type, Data_Counter_Type,  Support< Array_Type, Data_Counter_Type,
	ide/barry/model-bories.hpp, 336	Data_Rule_Type, Data_Counter_Type,  Data_Rule_Type, Data_Rule_Dyn_Type >,
	ide/barry/models/defm.hpp, 312	225
	ide/barry/models/defm.hpp, 312 ide/barry/models/defm/defm-bones.hpp, 354	INITIALIZED
	ide/barry/models/defm/defm-bories.hpp, 354	geese-bones.hpp, 358
	ide/barry/models/deim/deim-meat.npp, 355	initialized
HILL	adorban yrnnodeiorgeese.npp, sou	mitalized

Flock, 145 Geese, 161	keygen_full geese-bones.hpp, 359
insert_cell BArray< Cell_Type, Data_Type >, 50 BArrayDense< Cell_Type, Data_Type >, 70 barraydense-meat.hpp, 270 model-meat.hpp, 340	keys2support  Model < Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 181
support-meat.hpp, 378	lb PhyloRuleDynData, 200
is_col BArrayVector< Cell_Type, Data_Type >, 97	likelihood
BArrayVector_const< Cell_Type, Data_Type >, 37  BArrayVector_const< Cell_Type, Data_Type >,  101	DEFM, 124 Geese, 157
is_dense	Model < Array_Type, Data_Counter_Type,
BArray< Cell_Type, Data_Type >, 50 BArrayDense< Cell_Type, Data_Type >, 70	Data_Rule_Type, Data_Rule_Dyn_Type >, 174
IS_DUPLICATION	likelihood_
phylo.hpp, 331	model-meat.hpp, 340
IS_EITHER	likelihood_exhaust Geese, 157
phylo.hpp, 331	likelihood_joint
is_empty BArray< Cell_Type, Data_Type >, 50	Flock, 143
BArrayDense< Cell Type, Data Type >, 70	likelihood_total
is leaf	Model < Array_Type, Data_Counter_Type,
Node, 192	Data_Rule_Type, Data_Rule_Dyn_Type >,
is_motif	175
DEFM, 124	locator
DEFMCounterData, 128	model-meat.hpp, 352
is_row	logical
BArrayVector< Cell_Type, Data_Type >, 97	DEFMCounterData, 128
BArrayVector_const< Cell_Type, Data_Type >,	DEFMRuleData, 135
101	logodds DEFM, 124
IS_SPECIATION phylo.hpp, 331	DET WI, TET
is_true	M
DEFMCounterData, 127	barray-meat.hpp, 247, 251
DEFMRuleData, 134	barraydense-meat.hpp, 271, 276
iter	PowerSet < Array_Type, Data_Rule_Type >, 206
ConstBArrayRowlter< Cell_Type, Data_Type >, 110	Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 229
j	$M_{\_}$
barray-meat.hpp, 251	barray-meat.hpp, 252
barraydense-meat.hpp, 276	barraydense-meat.hpp, 277
counters-meat.hpp, 306	MAKE_DEFM_HASHER
model-meat.hpp, 351	Phylo rules, 16 MAKE DUPL VARS
statscounter-meat.hpp, 374	phylo.hpp, 331
j0	make_hash
barray-meat.hpp, 251 barraydense-meat.hpp, 276	FreqTable $<$ T $>$ , 149
j1	Map
barray-meat.hpp, 251	barry-configuration.hpp, 288
barraydense-meat.hpp, 276	map_to_nodes
	Geese, 162
k	MapVec_type
model-meat.hpp, 352	typedefs.hpp, 389
key	max_num_elements
model-meat.hpp, 352	Support Array_Type, Data_Counter_Type,
keygen_defm defm-meat.hpp, 356	Data_Rule_Type, Data_Rule_Dyn_Type >, 229

Model	set_rengine, 177
Model< Array_Type, Data_Counter_Type,	set_rules, 177
Data_Rule_Type, Data_Rule_Dyn_Type >,	set_rules_dyn, 177
167, 168	set_seed, 177
model	set transform model, 178
Flock, 146	size, 178
Model < Array_Type, Data_Counter_Type, Data_Rule_Type,	size_unique, 178
Data_Rule_Dyn_Type >, 163	stats_support, 183
~Model, 168	stats_support_n_arrays, 183
add array, 168	stats_target, 183
— ··	
add_counter, 169	store_psets, 178
add_hasher, 169	support_fun, 184
add_rule, 169	support_size, 179
add_rule_dyn, 170	transform_model, 179
arrays2support, 179	transform_model_fun, 184
colnames, 170	transform_model_term_names, 184
conditional_prob, 170	with_pset, 185
counter_fun, 179 mo	del-meat.hpp
counters, 179	a, 348
delete_counters, 180	count_fun_, 348
delete_rengine, 180	counter, 349
delete_rules, 180	counters_, 349
delete_rules_dyn, 180	cumprob, 349
first_calc_done, 180	data , 349
gen_key, 171	Data_Counter_Type, 349
get_arrays2support, 171	Data_Rule_Type, 350
get_counters, 171	delete_rules, 350
get_counters, 171 get_norm_const, 171	delete_rules_dyn, 350
<del>-</del>	·
get_pset, 171	else, 350
get_pset_arrays, 172	for, 339
get_pset_probs, 172	force_new, 350
get_pset_stats, 172	fun_, 351
get_rengine, 172	i, 351
get_rules, 173	i_matches, 351
get_rules_dyn, 173	if, 339, 340
get_stats_support, 173	init_fun_, <mark>351</mark>
get_stats_target, 173	insert_cell, 340
get_support_fun, 173	j, 351
keys2support, 181	k, 352
likelihood, 174	key, 352
likelihood_total, 175	likelihood_, 340
Model, 167, 168	locator, 352
normalizing_constants, 181	MODEL_TEMPLATE, 338, 340-346
nrules, 175	MODEL_TEMPLATE_ARGS, 338
nrules_dyn, 175	MODEL_TYPE, 339
nterms, 175	params, 352
operator=, 175	probs, 352
params last, 181	•
• – •	pset_arrays, 352
print, 176	push_back, 346, 347
print_stats, 176	r, 353
pset_arrays, 181	return, 347, 353
pset_probs, 182	rule_fun_, 353
pset_stats, 182	rules, 353
rengine, 182	rules_, 353
rules, 182	rules_dyn, 354
rules_dyn, 183	set_counters, 347
sample, 176	set_rules, 347
set_counters, 177	set_rules_dyn, 347
	·

size, 347	NetModel
stats, 354	network.hpp, 326
stats_support_n_arrays, 354	NetRule
temp_stats, 347	network.hpp, 326
tmp_counts, 348	NetRules
update_normalizing_constant, 348	network.hpp, 326
urand, 348	NetStatsCounter
MODEL_TEMPLATE	network.hpp, 326
model-meat.hpp, 338, 340-346	NetSupport
MODEL_TEMPLATE_ARGS	network.hpp, 326
model-meat.hpp, 338	Network
MODEL_TYPE	network.hpp, 326
model-meat.hpp, 339	network-css.hpp
motif_census	counter_css_census01, 316
DEFM, 125	counter_css_census02, 316
N	counter_css_census03, 316
N	counter_css_census04, 317
barray-meat.hpp, 252	counter_css_census05, 317
barraydense-meat.hpp, 277	counter_css_census06, 317
PowerSet< Array_Type, Data_Rule_Type >, 207	counter_css_census07, 317
Support< Array_Type, Data_Counter_Type,	counter_css_census08, 318
Data_Rule_Type, Data_Rule_Dyn_Type >,	counter_css_census09, 318
229	counter_css_census10, 318
n_counters	counter_css_completely_false_recip_comiss, 318
Support< Array_Type, Data_Counter_Type,	counter_css_completely_false_recip_omiss, 319
Data_Rule_Type, Data_Rule_Dyn_Type >,	counter_css_mixed_recip, 319
230	counter_css_partially_false_recip_commi, 319
n_free	counter_css_partially_false_recip_omiss, 320
PowerSet < Array_Type, Data_Rule_Type >, 207	CSS_APPEND, 314
n_locked	CSS_CASE_ELSE, 314
PowerSet< Array_Type, Data_Rule_Type >, 207	CSS_CASE_PERCEIVED, 314
name	CSS_CASE_TRUTH, 314
Counter< Array_Type, Data_Type >, 115	CSS_CHECK_SIZE, 314
counters-meat.hpp, 303	CSS CHECK SIZE INIT, 315
name_	CSS_NET_COUNTER_LAMBDA_INIT, 315
counters-meat.hpp, 306	CSS_PERCEIVED_CELLS, 315
nannotations	CSS_SIZE, 315
Geese, 157	CSS TRUE CELLS, 316
narray	network.hpp
Node, 193	BARRY_ZERO_NETWORK, 323
NCells	BARRY_ZERO_NETWORK_DENSE, 323
barray-meat.hpp, 252	NET_C_DATA_IDX, 324
ncol	NET_C_DATA_NUM, 324
BArray< Cell_Type, Data_Type >, 51	NetCounter, 325
BArrayDense < Cell_Type, Data_Type >, 70	NetCounters, 325
Phylo rules, 18	NetModel, 326
NET_C_DATA_IDX	NetRule, 326
network.hpp, 324	NetRules, 326
NET_C_DATA_NUM	NetStatsCounter, 326
network.hpp, 324	NetSupport, 326
NetCounter	Network, 326
network.hpp, 325	NETWORK COUNTER, 324
NetCounterData, 185	NETWORK_COUNTER_LAMBDA, 324
∼NetCounterData, 186	NETWORK RULE, 324
indices, 186	NETWORK_RULE_LAMBDA, 325
NetCounterData, 186	NetworkDense, 327
numbers, 186	
NetCounters	NETWORKDENSE_COUNTER_LAMBDA, 325
network.hpp, 325	rules_zerodiag, 327
ee e prove <del>e</del>	

NETWORK_COUNTER	states, 196
DEFMArray counters, 29	nodes
network.hpp, 324	Geese, 162
NETWORK_COUNTER_LAMBDA	noexcept
network.hpp, 324	counters-meat.hpp, 306
NETWORK RULE	noffspring
network.hpp, 324	Node, 192
NETWORK_RULE_LAMBDA	NONE
network.hpp, 325	CHECK, 40
NetworkData, 187	EXISTS, 42
~NetworkData, 188	normalizing_constants
directed, 188	Model Array_Type, Data_Counter_Type,
	7= 71 / = = 71 /
NetworkData, 187, 188	Data_Rule_Type, Data_Rule_Dyn_Type >,
vertex_attr, 188	181
NetworkDense	nrow
network.hpp, 327	BArray< Cell_Type, Data_Type >, 51
NETWORKDENSE_COUNTER_LAMBDA	BArrayDense< Cell_Type, Data_Type >, 71
network.hpp, 325	Phylo rules, 18
next	nrules
Progress, 209	Model < Array_Type, Data_Counter_Type,
nfunctions	Data_Rule_Type, Data_Rule_Dyn_Type >,
Flock, 146	175
Geese, 162	nrules_dyn
nfuns	Model< Array_Type, Data_Counter_Type,
Flock, 143	Data_Rule_Type, Data_Rule_Dyn_Type >,
Geese, 157	175
nleafs	nterms
Flock, 143	Flock, 144
Geese, 157	Geese, 158
nnodes	•
	7= 71 / = = 71 /
Flock, 143	Data_Rule_Type, Data_Rule_Dyn_Type >,
Geese, 158	175
nnozero	ntrees
BArray< Cell_Type, Data_Type >, 51	Flock, 144
BArrayDense < Cell_Type, Data_Type >, 71	num
Node, 189	DEFMCounterData, 127
$\sim$ Node, 191	DEFMRuleData, 134
annotations, 192	numbers
array, 192	DEFMCounterData, 128
arrays, 192	DEFMRuleData, 135
arrays_valid, 193	NetCounterData, 186
duplication, 193	
get_parent, 191	obs_start
id, 193	DEFMData, 131
is_leaf, 192	observed_counts
narray, 193	Geese, 158
Node, 190, 191	offspring
noffspring, 192	Node, 193
offspring, 193	ONE
• •	CHECK, 40
ord, 194	EXISTS, 42
parent, 194	operator BArrayRow< Cell_Type, Data_Type >
probability, 194	BArrayRow< Cell_Type, Data_Type >, 92
subtree_prob, 194	operator BArrayRow_const< Cell_Type, Data_Type >
visited, 194	
NodeData, 195	BArrayRow_const< Cell_Type, Data_Type >, 94
blengths, 196	operator Cell_Type
duplication, 196	BArrayCell Cell_Type, Data_Type >, 58
NodeData, 195	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 >,	BArrayDenseCell< Cell_Type, Data_Type >, 79 BArrayRow< Cell_Type, Data_Type >, 92 BArrayVector< Cell_Type, Data_Type >, 98 operator-=
operator!=  BArrayCell_const< Cell_Type, Data_Type >, 61  BArrayRow_const< Cell_Type, Data_Type >, 94  BArrayVector_const< Cell_Type, Data_Type >,  101	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
Cell< Cell_Type >, 106 operator< BArrayCell_const< Cell_Type, Data_Type >, 61 BArrayRow_const< Cell_Type, Data_Type >, 94	operator/=  BArray< Cell_Type, Data_Type >, 53  BArrayCell< Cell_Type, Data_Type >, 59  BArrayDense< Cell_Type, Data_Type >, 72
BArrayVector_const< Cell_Type, Data_Type >, 102 operator<=	BArrayDenseCell< Cell_Type, Data_Type >, 80 BArrayRow< Cell_Type, Data_Type >, 93 BArrayVector< Cell_Type, Data_Type >, 98
BArrayCell_const< Cell_Type, Data_Type >, 61 BArrayRow_const< Cell_Type, Data_Type >, 95 BArrayVector_const< Cell_Type, Data_Type >,	operator=  BArray< Cell_Type, Data_Type >, 53  BArrayCell< Cell_Type, Data_Type >, 59  BArrayDepage < Cell_Type, Data_Type > 73
operator> BArrayCell_const< Cell_Type, Data_Type >, 61 BArrayRow_const< Cell_Type, Data_Type >, 95	BArrayDense < Cell_Type, Data_Type >, 73 BArrayDenseCell < Cell_Type, Data_Type >, 80 BArrayRow < Cell_Type, Data_Type >, 93 BArrayVector < Cell_Type, Data_Type >, 99
BArrayVector_const< Cell_Type, Data_Type >, 102 operator>=	Cell< Cell_Type >, 106, 107 Counter< Array_Type, Data_Type >, 113, 114 Counters< Array_Type, Data_Type >, 119
BArrayCell_const< Cell_Type, Data_Type >, 62 BArrayRow_const< Cell_Type, Data_Type >, 95 BArrayVector_const< Cell_Type, Data_Type >, 102	Geese, 158  Model < Array_Type, Data_Counter_Type,  Data_Rule_Type, Data_Rule_Dyn_Type >,  175
operator*=	Rules < Array_Type, Data_Type >, 215
BArray< Cell_Type, Data_Type >, 51	operator==
BArrayCell< Cell_Type, Data_Type >, 58	BArray< Cell_Type, Data_Type >, 53
BArrayDense< Cell_Type, Data_Type >, 71 BArrayDenseCell< Cell_Type, Data_Type >, 79	BArrayCell< Cell_Type, Data_Type >, 59 BArrayCell_const< Cell_Type, Data_Type >, 61
BArrayRow< Cell_Type, Data_Type >, 92 BArrayVector< Cell_Type, Data_Type >, 98 operator()	BArrayDense < Cell_Type, Data_Type >, 73 BArrayDenseCell < Cell_Type, Data_Type >, 80 BArrayRow < Cell_Type, Data_Type >, 93
BArray< Cell_Type, Data_Type >, 51 barray-meat-operators.hpp, 235	BArrayRow_const< Cell_Type, Data_Type >, 95 BArrayVector< Cell_Type, Data_Type >, 99
BArrayDense< Cell_Type, Data_Type >, 71 BArrayDenseCol< Cell_Type, Data_Type >, 83 BArrayDenseCol_const< Cell_Type, Data_Type >,	BArrayVector_const< Cell_Type, Data_Type >, 102 Cell< Cell_Type >, 107
85	operator[]
BArrayDenseRow< Cell_Type, Data_Type >, 88	Counters< Array_Type, Data_Type >, 120
BArrayDenseRow_const< Cell_Type, Data_Type >, 90	PhyloCounterData, 198 PowerSet < Array_Type, Data_Rule_Type >, 205
Flock, 144	ord
Phylo rules, 18	Node, 194
PhyloCounterData, 198	out_of_range
Rule< Array_Type, Data_Type >, 212	BArray< Cell_Type, Data_Type >, 53
Rules < Array_Type, Data_Type >, 215	BArrayDense < Cell_Type, Data_Type >, 73
vecHasher< T >, 230	
operator+=	params
BArray< Cell_Type, Data_Type >, 52	model-meat.hpp, 352
BArrayCell< Cell_Type, Data_Type >, 58 BArrayDense< Cell_Type, Data_Type >, 71, 72	params_last

Model < Array_Type, Data_Counter_Type,	DUPL_SPEC, 330
Data_Rule_Type, Data_Rule_Dyn_Type >,	get_last_name, 335
181	IF_MATCHES, 330
parent	IF NOTMATCHES, 331
Node, 194	IS DUPLICATION, 331
parse_polytomies	IS EITHER, 331
Flock, 144	IS SPECIATION, 331
Geese, 158	MAKE_DUPL_VARS, 331
Phylo counters, 30	PHYLO_CHECK_MISSING, 332
counter_co_opt, 31	PHYLO_COUNTER_LAMBDA, 332
counter_cogain, 32	PHYLO_RULE_DYN_LAMBDA, 332
counter_gains, 32	PhyloArray, 333
counter_gains_from_0, 32	PhyloCounter, 333
counter_gains_k_offspring, 33	PhyloCounters, 333
counter_genes_changing, 33	PhyloModel, 333
counter_k_genes_changing, 33	PhyloPowerSet, 333
counter_less_than_p_prop_genes_changing, 33	PhyloRule, 333
counter_longest, 34	PhyloRuleData, 334
	-
counter_loss, 34	PhyloRuleDyn, 334
counter_maxfuns, 34	PhyloRules, 334
counter_neofun, 34	PhyloRulesDyn, 334
counter_neofun_a2b, 35	PhyloStatsCounter, 334
counter_overall_changes, 35	PhyloSupport, 334
counter_overall_gains, 35	PHYLO_CHECK_MISSING
counter_overall_gains_from_0, 35	phylo.hpp, 332
counter_overall_loss, 36	PHYLO_COUNTER_LAMBDA
counter_pairwise_first_gain, 36	phylo.hpp, 332
counter_pairwise_neofun_singlefun, 36	PHYLO_RULE_DYN_LAMBDA
counter_pairwise_overall_change, 36	phylo.hpp, 332
counter_pairwise_preserving, 37	PhyloArray
counter_preserve_pseudogene, 37	phylo.hpp, 333
	PhyloCounter
counter_prop_genes_changing, 37	•
counter_subfun, 37	phylo.hpp, 333
Phylo rules, 14	PhyloCounterData, 196
DEFM_COUNTER, 15	at, 197
DEFM_COUNTER_LAMBDA, 15	begin, 197
DEFM_RULE, 15	empty, 197
DEFM_RULE_LAMBDA, 16	end, 198
DEFM_RULEDYN_LAMBDA, 16	get_counters, 198
DEFMCounter, 17	operator(), 198
DEFMCounters, 17	operator[], 198
DEFMModel, 17	PhyloCounterData, 197
DEFMRule, 17	push_back, 198
DEFMRuleDyn, 17	reserve, 198
DEFMRules, 17	shrink_to_fit, 199
DEFMRulesDyn, 18	size, 199
DEFMStatsCounter, 18	PhyloCounters
DEFMSupport, 18	phylo.hpp, 333
MAKE_DEFM_HASHER, 16	PhyloModel
ncol, 18	phylo.hpp, 333
nrow, 18	PhyloPowerSet
operator(), 18	phylo.hpp, 333
print, 19	PhyloRule
rule_dyn_limit_changes, 19	phylo.hpp, 333
phylo.hpp	PhyloRuleData
DEFAULT_DUPLICATION, 330	phylo.hpp, 334
DUPL_DUPL, 330	PhyloRuleDyn
DUPL EITH, 330	phylo.hpp, 334
DOI L_LITTI, 000	phylolipp, 304

PhyloRuleDynData, 199  ~PhyloRuleDynData, 200 counts, 200 duplication, 200 lb, 200 PhyloRuleDynData, 200 pos, 201 ub, 201 PhyloRules	Geese, 159 predict_exhaust     Geese, 159 predict_exhaust_backend     Geese, 159 predict_sim     Geese, 160 print     BArray < Cell_Type, Data_Type >, 53
phylo.hpp, 334 PhyloRulesDyn	BArrayDense < Cell_Type, Data_Type >, 73 DEFM, 125
phylo.hpp, 334	Flock, 145
PhyloStatsCounter	FreqTable < T >, 149
phylo.hpp, 334	Geese, 160
PhyloSupport	Model< Array_Type, Data_Counter_Type,
phylo.hpp, 334	Data_Rule_Type, Data_Rule_Dyn_Type >,
POS barraydense-meat-operators.hpp, 256	176 Phylo rules, 19
barraydense-meat-operators.npp, 250 barraydense-meat.hpp, 261	Support< Array_Type, Data_Counter_Type,
barraydensecell-bones.hpp, 279	Data_Rule_Type, Data_Rule_Dyn_Type >,
barraydensecell-meat.hpp, 280	225
barraydensecol-bones.hpp, 281	print_observed_counts
barraydenserow-bones.hpp, 282	Geese, 160
pos	print_stats
PhyloRuleDynData, 201	Model< Array_Type, Data_Counter_Type,
POS_N	Data_Rule_Type, Data_Rule_Dyn_Type >,
barraydense-meat-operators.hpp, 257	176
barraydense-meat.hpp, 261 barraydensecol-bones.hpp, 281	printf_barry barraydense-meat.hpp, 271
barraydenserow-bones.hpp, 282	barry-configuration.hpp, 287
PowerSet	probability
PowerSet< Array_Type, Data_Rule_Type >, 203	Node, 194
PowerSet< Array_Type, Data_Rule_Type >, 201	probs
$\sim$ PowerSet, 203	model-meat.hpp, 352
add_rule, 203, 204	Progress, 208
begin, 204	∼Progress, 208
calc, 204	end, 209
coordinates_free, 206	next, 209
coordinates_locked, 206	Progress, 208
data, 206 EmptyArray, 206	progress.hpp BARRY_PROGRESS_BAR_WIDTH, 366
end, 204	pset_arrays
get_data, 204	Model< Array_Type, Data_Counter_Type,
get_data_ptr, 205	Data_Rule_Type, Data_Rule_Dyn_Type >,
init_support, 205	181
M, 206	model-meat.hpp, 352
N, 207	pset_loc
n_free, 207	Geese, 162
n_locked, 207	pset_probs
operator[], 205	Model < Array_Type, Data_Counter_Type,
PowerSet, 203	Data_Rule_Type, Data_Rule_Dyn_Type >, 182
reset, 205 rules, 207	pset stats
rules_deleted, 207	Model Array_Type, Data_Counter_Type,
size, 205	Data_Rule_Type, Data_Rule_Dyn_Type >,
predict	182
Geese, 159	push_back
predict_backend	model-meat.hpp, 346, 347

PhyloCounterData, 198	barray-meat.hpp, 253
r	Row_type
model-meat.hpp, 353	typedefs.hpp, 389
README.md, 391	rowsum
reduced_sequence	BArrayDense < Cell_Type, Data_Type >, 74
Geese, 162	Rule
rengine	Rule < Array_Type, Data_Type >, 210
Flock, 146	Rule < Array_Type, Data_Type >, 209  ~Rule, 210
Model < Array_Type, Data_Counter_Type,	D, 211
Data_Rule_Type, Data_Rule_Dyn_Type >,	get_description, 211
182	get_name, 211
report	operator(), 212
barray-meat.hpp, 252	Rule, 210
barraydense-meat.hpp, 277	rule_dyn_limit_changes
res	Phylo rules, 19
counters-meat.hpp, 306	rule_fun_
reserve	model-meat.hpp, 353
BArray< Cell_Type, Data_Type >, 54	rule_fun_default
BArrayDense < Cell_Type, Data_Type >, 73	rules-bones.hpp, 367
FreqTable $<$ T $>$ , 149	Rule_fun_type
PhyloCounterData, 198	typedefs.hpp, 389
reset	RULE FUNCTION
PowerSet< Array_Type, Data_Rule_Type >, 205	barry.hpp, 293
reset_array	geese-bones.hpp, 359
StatsCounter< Array_Type, Data_Type >, 219	RULE LAMBDA
Support< Array_Type, Data_Counter_Type,	barry.hpp, 293
Data_Rule_Type, Data_Rule_Dyn_Type >,	Rules
226	Rules< Array_Type, Data_Type >, 213
resize	rules
BArray< Cell_Type, Data_Type >, 54	Model< Array_Type, Data_Counter_Type,
barray-meat.hpp, 247	Data_Rule_Type, Data_Rule_Dyn_Type >,
BArrayDense< Cell_Type, Data_Type >, 74	182
barraydense-meat.hpp, 271, 272	model-meat.hpp, 353
Entries < Cell_Type >, 138	PowerSet < Array_Type, Data_Rule_Type >, 207
statscounter-meat.hpp, 370	support-meat.hpp, 384
return	Rules < Array_Type, Data_Type >, 212
barray-meat.hpp, 247, 252	$\sim$ Rules, 213
barraydense-meat.hpp, 277	add_rule, 214
counters-meat.hpp, 307	get_descriptions, 214
model-meat.hpp, 347, 353	get_names, 214
statscounter-meat.hpp, 374	get_seq, 214
support-meat.hpp, 384	operator(), 215
rhs	operator=, 215
barray-meat-operators.hpp, 235	Rules, 213
rm_cell	size, 215
BArray< Cell_Type, Data_Type >, 54	rules-bones.hpp
BArrayDense < Cell_Type, Data_Type >, 74	rule_fun_default, 367
barraydense-meat.hpp, 272	rules_
support-meat.hpp, 378	model-meat.hpp, 353
ROW	support-meat.hpp, 384
barray-meat-operators.hpp, 233	rules_deleted
barray-meat.hpp, 239, 247, 248	PowerSet < Array_Type, Data_Rule_Type >, 207
barraydense-meat-operators.hpp, 257	rules_dont_become_zero
barraydense-meat.hpp, 262	DEFMArray counters, 29
row	rules_dyn
BArray< Cell_Type, Data_Type >, 54	
DA	Model< Array_Type, Data_Counter_Type,
BArrayDense < Cell_Type, Data_Type >, 74 row0	Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >,

183	Model < Array_Type, Data_Counter_Type,
model-meat.hpp, 354	Data_Rule_Type, Data_Rule_Dyn_Type >,
support-meat.hpp, 385	178
rules_markov_fixed	shrink_to_fit
DEFMArray counters, 30	PhyloCounterData, 199
rules_zerodiag	simulate
network.hpp, 327	DEFM, 125
	Geese, 160
sample  Model< Array Type, Data Counter Type,	size
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >,	BArrayDenseCol< Cell_Type, Data_Type >, 83
176	BArrayDenseCol_const< Cell_Type, Data_Type >,
search	86
barray-meat.hpp, 253	BArrayDenseRow< Cell_Type, Data_Type >, 88
sequence	BArrayDenseRow_const< Cell_Type, Data_Type
Geese, 162	>, 90
set_counters	BArrayVector< Cell_Type, Data_Type >, 99
Model< Array_Type, Data_Counter_Type,	BArrayVector_const< Cell_Type, Data_Type >,
Data_Rule_Type, Data_Rule_Dyn_Type >,	103
177	Counters < Array_Type, Data_Type >, 120
model-meat.hpp, 347	FreqTable < T >, 149  Model < Array Type, Data Counter Type.
StatsCounter< Array_Type, Data_Type >, 219	Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >,
Support< Array_Type, Data_Counter_Type,	178
Data_Rule_Type, Data_Rule_Dyn_Type >,	model-meat.hpp, 347
226	PhyloCounterData, 199
set_data	PowerSet < Array_Type, Data_Rule_Type >, 205
BArray< Cell_Type, Data_Type >, 54	Rules < Array_Type, Data_Type >, 215
BArrayDense < Cell_Type, Data_Type >, 74	StatsCounter< Array_Type, Data_Type >, 219
set_hasher	size_unique
Counter< Array_Type, Data_Type >, 114	 Model< Array_Type, Data_Counter_Type,
set_names	Data_Rule_Type, Data_Rule_Dyn_Type >,
DEFM, 125	178
set_rengine	sort_array
Model< Array_Type, Data_Counter_Type,	typedefs.hpp, 389
Data_Rule_Type, Data_Rule_Dyn_Type >,	source
177	barray-meat.hpp, 253
set_rules  Model< Array_Type, Data_Counter_Type,	barraydense-meat.hpp, 277
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >,	Entries < Cell_Type >, 139
177	states
model-meat.hpp, 347	NodeData, 196
Support< Array_Type, Data_Counter_Type,	Statistical Models, 13
Data_Rule_Type, Data_Rule_Dyn_Type >,	stats model-meat.hpp, 354
226	stats_bank
set_rules_dyn	support-meat.hpp, 385
Model< Array_Type, Data_Counter_Type,	stats_support
Data_Rule_Type, Data_Rule_Dyn_Type >,	Model < Array_Type, Data_Counter_Type,
177	Data_Rule_Type, Data_Rule_Dyn_Type >,
model-meat.hpp, 347	183
Support< Array_Type, Data_Counter_Type,	stats_support_n_arrays
Data_Rule_Type, Data_Rule_Dyn_Type >,	Model< Array_Type, Data_Counter_Type,
226	Data_Rule_Type, Data_Rule_Dyn_Type >,
set_seed	183
Flock, 145	model-meat.hpp, 354
Geese, 160	stats_target
Model Array_Type, Data_Counter_Type,	Model < Array_Type, Data_Counter_Type,
Data_Rule_Type, Data_Rule_Dyn_Type >,	Data_Rule_Type, Data_Rule_Dyn_Type >,
177	183
set_transform_model	

StatsCounter	coordinates_locked, 228
StatsCounter < Array_Type, Data_Type >, 217	current_stats, 228
StatsCounter< Array_Type, Data_Type >, 216	delete_counters, 228
~StatsCounter, 217	delete_rules, 228
add_counter, 218	delete rules dyn, 228
count_all, 218	eval_rules_dyn, 224
count_current, 218	get_counters, 224
count_init, 218	get_counts, 224
get_counters, 218	get_current_stats, 224
get_descriptions, 218	get_data, 225
get_names, 219	get_rules, 225
reset_array, 219	get_rules_dyn, 225
set_counters, 219	hashes, 229
size, 219	hashes_initialized, 229
StatsCounter, 217	init_support, 225
statscounter-meat.hpp	M, 229
clear, 370	max_num_elements, 229
counter, 372	N, 229
counter deleted, 372	n_counters, 230
counters, 373	
	print, 225
counters_, 373	reset_array, 226
current_stats, 373	set_counters, 226
EmptyArray, 373	set_rules, 226
f_, 373	set_rules_dyn, 226
for, 370	Support, 222
j, 374	support-meat.hpp
resize, 370	array_bank, 382
return, 374	BARRY_SUPPORT_MEAT_HPP, 376
STATSCOUNTER_TEMPLATE, 370-372	calc_backend_dense, 377
STATSCOUNTER_TEMPLATE_ARGS, 370	calc_backend_sparse, 377
STATSCOUNTER_TYPE, 370	change_stats_different, 382
STATSCOUNTER_TEMPLATE	coord_i, 382
statscounter-meat.hpp, 370–372	coord_j, 382
STATSCOUNTER_TEMPLATE_ARGS	counters, 382
statscounter-meat.hpp, 370	counters_, 383
STATSCOUNTER_TYPE	delete_counters, 383
statscounter-meat.hpp, 370	delete_rules, 383
store_psets	delete_rules_dyn, 383
Model< Array_Type, Data_Counter_Type,	else, 383
Data_Rule_Type, Data_Rule_Dyn_Type >,	f_, 384
178	for, 377
subtree_prob	hashes, 384
Node, 194	if, 377, 378
Support	insert_cell, 378
Support< Array_Type, Data_Counter_Type,	return, 384
Data_Rule_Type, Data_Rule_Dyn_Type >,	rm_cell, 378
222	rules, 384
Support < Array_Type, Data_Counter_Type, Data_Rule_	
Data_Rule_Dyn_Type >, 220	rules_dyn, 385
~Support, 222	stats_bank, 385
add_counter, 223	SUPPORT_TEMPLATE, 376, 379–382
add_rule, 223	SUPPORT_TEMPLATE_ARGS, 376
add_rule_dyn, 223	SUPPORT_TYPE, 377
calc, 223	tmp_chng, 385
change_stats, 227	support_fun
coordiantes_n_free, 227	Model < Array_Type, Data_Counter_Type,
coordiantes_n_locked, 227	Data_Rule_Type, Data_Rule_Dyn_Type >,
coordinates_free, 227	184

support_size Flock, 145	EXISTS, 42 typedefs.hpp
Geese, 161	Col_type, 388
Model< Array_Type, Data_Counter_Type,	Counter_fun_type, 388
Data Rule Type, Data Rule Dyn Type >,	
	Counts_type, 388
179	Hasher_fun_type, 388
SUPPORT_TEMPLATE	MapVec_type, 389
support-meat.hpp, 376, 379–382	Row_type, 389
SUPPORT_TEMPLATE_ARGS	Rule_fun_type, 389
support-meat.hpp, 376	sort_array, 389
SUPPORT_TYPE	vec_equal, 390
support-meat.hpp, 377	vec_equal_approx, 390
swap_cells	vec_inner_prod, 390, 391
BArray< Cell_Type, Data_Type >, 55	
BArrayDense < Cell_Type, Data_Type >, 75	ub
swap_cols	PhyloRuleDynData, 201
BArray< Cell_Type, Data_Type >, 55	UKNOWN
BArrayDense < Cell_Type, Data_Type >, 75	EXISTS, 42
swap rows	UNI_SUB
BArray< Cell_Type, Data_Type >, 55	defm.hpp, 311
BArrayDense< Cell_Type, Data_Type >, 75	update_annotations
ay a sa cas <u>— ypay am — ypas y</u>	Geese, 161
target	update_normalizing_constant
barray-meat.hpp, 253	model-meat.hpp, 348
barraydense-meat.hpp, 278	urand
Entries < Cell_Type >, 139	model-meat.hpp, 348
temp_stats	14,7
model-meat.hpp, 347	V
this	barray-meat.hpp, 253
barray-meat-operators.hpp, 236	barraydense-meat.hpp, 278
tmp_chng	va_end
support-meat.hpp, 385	barraydense-meat.hpp, 272
tmp_counts	va_start
model-meat.hpp, 348	barraydense-meat.hpp, 273
TMP_HASHER_CALL	val
counters-meat.hpp, 297	Entries < Cell_Type >, 139
toggle cell	val0
	barraydense-meat.hpp, 278
BArray Dense Coll Time Data Type > . 75	val1
BArrayDense < Cell_Type, Data_Type >, 75	
toggle_lock	barraydense-meat.hpp, 278
BArray< Cell_Type, Data_Type >, 55	value
BArrayDense < Cell_Type, Data_Type >, 76	barray-meat.hpp, 253
transform_model	barraydense-meat.hpp, 278
Model< Array_Type, Data_Counter_Type,	Cell< Cell_Type >, 107
Data_Rule_Type, Data_Rule_Dyn_Type >,	vec_diff
179	geese-bones.hpp, 359
transform_model_fun	vec_equal
Model< Array_Type, Data_Counter_Type,	typedefs.hpp, 390
Data_Rule_Type, Data_Rule_Dyn_Type >,	vec_equal_approx
184	typedefs.hpp, 390
transform_model_term_names	vec_inner_prod
Model< Array_Type, Data_Counter_Type,	typedefs.hpp, 390, 391
Data_Rule_Type, Data_Rule_Dyn_Type >,	vecHasher $<$ T $>$ , 230
184	operator(), 230
transpose	vector_caster
BArray< Cell_Type, Data_Type >, 56	geese-bones.hpp, 359
BArrayDense< Cell_Type, Data_Type >, 76	vertex_attr
TWO	NetworkData, 188
CHECK, 41	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, 262
     barraydensecol-bones.hpp, 281
     barraydenserow-bones.hpp, 283
     BArray< Cell_Type, Data_Type >, 56
     {\tt BArrayDense}{<}~{\tt Cell\_Type},~{\tt Data\_Type}>, {\tt 76}
zero_row
     {\tt BArray}{<}~{\tt Cell\_Type},~{\tt Data\_Type}>, {\tt \bf 56}
     BArrayDense < Cell_Type, Data_Type >, 76
```