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

6.4 DEFMArray counters . . . 19
6.4.1 Detailed Description . . . 21
6.4.2 Function Documentation . . . 21
6.4.2.1 counter_absdiff() . . . 22
6.4.2.2 counter_ctriads() [1/2] . . . 22
6.4.2.3 counter_ctriads() [2/2] . . . 22
6.4.2.4 counter_degree() . . . 22
6.4.2.5 counter_density() . . . 23
6.4.2.6 counter_diff() . . . 23
6.4.2.7 counter_edges() . . . 23
6.4.2.8 counter_fixed_effect() . . . 23
6.4.2.9 counter_idegree() [1/2] . . . 24
6.4.2.10 counter_idegree() [2/2] . . . 24
6.4.2.11 counter_idegree15() [1/2] . . . 24
6.4.2.12 counter_idegree15() [2/2] . . . 24
6.4.2.13 counter_isolates() [1/2] . . . 25
6.4.2.14 counter_isolates() [2/2] . . . 25
6.4.2.15 counter_istar2() [1/2] . . . 25
6.4.2.16 counter_istar2() [2/2] . . . 25
6.4.2.17 counter_logit_intercept() . . . 25
6.4.2.18 counter_mutual() . . . 26
6.4.2.19 counter_nodecov() . . . 26
6.4.2.20 counter_nodeicov() . . . 26
6.4.2.21 counter_nodematch() . . . 26
6.4.2.22 counter_nodeocov() . . . 26
6.4.2.23 counter_odegree() [1/2] . . . 27
6.4.2.24 counter_odegree() [2/2] . . . 27
6.4.2.25 counter_odegree15() [1/2] . . . 27
6.4.2.26 counter_odegree15() [2/2] . . . 27
6.4.2.27 counter_ones() . . . 27
6.4.2.28 counter_ostar2() [1/2] . . . 28
6.4.2.29 counter_ostar2() [2/2] . . . 28
6.4.2.30 counter_transition() . . . 28
6.4.2.31 counter_transition_formula() . . . 29
6.4.2.32 counter_ttriads() [1/2] . . . 29
6.4.2.33 counter_ttriads() [2/2] . . . 29
6.4.2.34 NETWORK_COUNTER() . . . 29
6.4.2.35 rules_dont_become_zero() . . . 30
6.4.2.36 rules_markov_fixed() . . . 30
6.5 Phylo counters . . . 30
6.5.1 Detailed Description . . . 31
6.5.2 Function Documentation . . . 32

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 . . . 43
8.1.1 Detailed Description . . . 45
8.1.2 Constructor & Destructor Documentation . . . 46
8.1.2.1 BArray() [1/6] . . . 46
8.1.2.2 BArray() [2/6] . . . 46
8.1.2.3 BArray() [3/6] . . . 46
8.1.2.4 BArray() [4/6] . . . 47
8.1.2.5 BArray() [5/6] . . . 47
8.1.2.6 BArray() [6/6] . . . 47
8.1.2.7 ~BArray() . . . 47
8.1.3 Member Function Documentation . . . 47
8.1.3.1 clear() . . . 47
8.1.3.2 col() . . . 48
8.1.3.3 D() [1/2] . . . 48
8.1.3.4 D() [2/2] . . . 48
8.1.3.5 D_ptr() [1/2] . . . 48
8.1.3.6 D_ptr() [2/2] . . . 48
8.1.3.7 default_val() . . . 48
8.1.3.8 flush_data() . . . 48
8.1.3.9 get_cell() . . . 49
8.1.3.10 get_col_vec() [1/2] . . . 49
8.1.3.11 get_col_vec() [2/2] . . . 49
8.1.3.12 get_entries() . . . 49
8.1.3.13 get_row_vec() [1/2] . . . 49
8.1.3.14 get_row_vec() [2/2] . . . 50
8.1.3.15 insert_cell() [1/3] . . . 50
8.1.3.16 insert_cell() [2/3] . . . 50
8.1.3.17 insert_cell() [3/3] . . . 50
8.1.3.18 is_dense() . . . 50
8.1.3.19 is_empty() . . . 51
8.1.3.20 ncol() . . . 51
8.1.3.21 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
8.1.3.26 operator+=() [1/3] . . . 52
8.1.3.27 operator+=() [2/3] . . . 52
8.1.3.28 operator+=() [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
8.4.2.2 BArrayDense() [2/6] . . . 65
8.4.2.3 BArrayDense() [3/6] . . . 66
8.4.2.4 BArrayDense() [4/6] . . . 66
8.4.2.5 BArrayDense() [5/6] . . . 66
8.4.2.6 BArrayDense() [6/6] . . . 66
8.4.2.7 ~BArrayDense() . . . 67
8.4.3 Member Function Documentation . . . 67
8.4.3.1 clear() . . . 67
8.4.3.2 col() [1/2] . . . 67
8.4.3.3 col() [2/2] . . . 67
8.4.3.4 colsum() . . . 67
8.4.3.5 D() [1/2] . . . 68
8.4.3.6 D() [2/2] . . . 68
8.4.3.7 D_ptr() [1/2] . . . 68
8.4.3.8 D_ptr() [2/2] . . . 68
8.4.3.9 default_val() . . . 68
8.4.3.10 get_cell() . . . 68
8.4.3.11 get_col_vec() [1/2] . . . 69
8.4.3.12 get_col_vec() [2/2] . . . 69
8.4.3.13 get_data() . . . 69

8.4.3.14 get_entries() . . . 69
8.4.3.15 get_row_vec() [1/2] . . . 69
8.4.3.16 get_row_vec() [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
8.4.3.25 operator()() [2/2] . . . 71
8.4.3.26 operator∗=() . . . 71
8.4.3.27 operator+=() [1/3] . . . 72
8.4.3.28 operator+=() [2/3] . . . 72
8.4.3.29 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
8.4.3.35 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
8.4.3.42 row() [1/2] . . . 74
8.4.3.43 row() [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
8.4.4 Friends And Related Function Documentation . . . 76
8.4.4.1 BArrayDenseCell< Cell_Type, Data_Type > . . . 77

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

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

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

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

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

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

8.21.4.2 logical . . . 127
8.21.4.3 numbers . . . 127
8.22 DEFMData Class Reference . . . 127
8.22.1 Detailed Description . . . 128
8.22.2 Constructor & Destructor Documentation . . . 128
8.22.2.1 DEFMData() [1/2] . . . 128
8.22.2.2 DEFMData() [2/2] . . . 129
8.22.2.3 ~DEFMData() . . . 130
8.22.3 Member Function Documentation . . . 130
8.22.3.1 at() . . . 130
8.22.4 Member Data Documentation . . . 130
8.22.4.1 array . . . 130
8.22.4.2 covar_sort . . . 130
8.22.4.3 covar_used . . . 131
8.22.4.4 covariates . . . 131
8.22.4.5 obs_start . . . 131
8.22.4.6 X_ncol . . . 131
8.22.4.7 X_nrow . . . 131
8.23 DEFMRuleData Class Reference . . . 132
8.23.1 Detailed Description . . . 132
8.23.2 Constructor & Destructor Documentation . . . 132
8.23.2.1 DEFMRuleData() [1/3] . . . 133
8.23.2.2 DEFMRuleData() [2/3] . . . 133
8.23.2.3 DEFMRuleData() [3/3] . . . 133
8.23.3 Member Function Documentation . . . 133
8.23.3.1 idx() . . . 133
8.23.3.2 is_true() . . . 133
8.23.3.3 num() . . . 134
8.23.4 Member Data Documentation . . . 134
8.23.4.1 indices . . . 134
8.23.4.2 init . . . 134
8.23.4.3 logical . . . 134
8.23.4.4 numbers . . . 134
8.24 DEFMRuleDynData Class Reference . . . 135
8.24.1 Detailed Description . . . 135
8.24.2 Constructor & Destructor Documentation . . . 136
8.24.2.1 DEFMRuleDynData() . . . 136
8.24.2.2 ~DEFMRuleDynData() . . . 136
8.24.3 Member Data Documentation . . . 136
8.24.3.1 counts . . . 136
8.25 Entries< Cell_Type > Class Template Reference . . . 136
8.25.1 Detailed Description . . . 137

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

8.27.1 Detailed Description . . . 146
8.27.2 Constructor & Destructor Documentation . . . 146
8.27.2.1 FreqTable() . . . 146
8.27.2.2 ~FreqTable() . . . 147
8.27.3 Member Function Documentation . . . 147
8.27.3.1 add() . . . 147
8.27.3.2 as_vector() . . . 147
8.27.3.3 clear() . . . 147
8.27.3.4 get_data() . . . 147
8.27.3.5 get_index() . . . 148
8.27.3.6 make_hash() . . . 148
8.27.3.7 print() . . . 148
8.27.3.8 reserve() . . . 148
8.27.3.9 size() . . . 148
8.28 Geese Class Reference . . . 149
8.28.1 Detailed Description . . . 152
8.28.2 Constructor & Destructor Documentation . . . 152
8.28.2.1 Geese() [1/4] . . . 152
8.28.2.2 Geese() [2/4] . . . 153
8.28.2.3 Geese() [3/4] . . . 153
8.28.2.4 Geese() [4/4] . . . 153
8.28.2.5 ~Geese() . . . 153
8.28.3 Member Function Documentation . . . 153
8.28.3.1 calc_reduced_sequence() . . . 153
8.28.3.2 calc_sequence() . . . 154
8.28.3.3 colnames() . . . 154
8.28.3.4 get_annotated_nodes() . . . 154
8.28.3.5 get_counters() . . . 154
8.28.3.6 get_model() . . . 154
8.28.3.7 get_probabilities() . . . 154
8.28.3.8 get_rengine() . . . 155
8.28.3.9 get_states() . . . 155
8.28.3.10 get_support_fun() . . . 155
8.28.3.11 inherit_support() . . . 155
8.28.3.12 init() . . . 155
8.28.3.13 init_node() . . . 156
8.28.3.14 likelihood() . . . 156
8.28.3.15 likelihood_exhaust() . . . 156
8.28.3.16 nannotations() . . . 156
8.28.3.17 nfuns() . . . 156
8.28.3.18 nleafs() . . . 157
8.28.3.19 nnodes() . . . 157

8.28.3.20 nterms() . . . 157
8.28.3.21 observed_counts() . . . 157
8.28.3.22 operator=() [1/2] . . . 157
8.28.3.23 operator=() [2/2] . . . 157
8.28.3.24 parse_polytomies() . . . 158
8.28.3.25 predict() . . . 158
8.28.3.26 predict_backend() . . . 158
8.28.3.27 predict_exhaust() . . . 158
8.28.3.28 predict_exhaust_backend() . . . 159
8.28.3.29 predict_sim() . . . 159
8.28.3.30 print() . . . 159
8.28.3.31 print_observed_counts() . . . 159
8.28.3.32 set_seed() . . . 159
8.28.3.33 simulate() . . . 160
8.28.3.34 support_size() . . . 160
8.28.3.35 update_annotations() . . . 160
8.28.4 Member Data Documentation . . . 160
8.28.4.1 delete_rengine . . . 160
8.28.4.2 delete_support . . . 160
8.28.4.3 initialized . . . 161
8.28.4.4 map_to_nodes . . . 161
8.28.4.5 nfunctions . . . 161
8.28.4.6 nodes . . . 161
8.28.4.7 pset_loc . . . 161
8.28.4.8 reduced_sequence . . . 161
8.28.4.9 sequence . . . 162
8.29 Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference . . . 162
8.29.1 Detailed Description . . . 164
8.29.2 Constructor & Destructor Documentation . . . 165
8.29.2.1 Model() [1/3] . . . 165
8.29.2.2 Model() [2/3] . . . 165
8.29.2.3 Model() [3/3] . . . 165
8.29.2.4 ~Model() . . . 165
8.29.3 Member Function Documentation . . . 166
8.29.3.1 add_array() . . . 166
8.29.3.2 add_counter() [1/2] . . . 166
8.29.3.3 add_counter() [2/2] . . . 166
8.29.3.4 add_hasher() . . . 167
8.29.3.5 add_rule() [1/2] . . . 167
8.29.3.6 add_rule() [2/2] . . . 167
8.29.3.7 add_rule_dyn() [1/2] . . . 167

8.29.3.8 add_rule_dyn() [2/2] . . . 167
8.29.3.9 colnames() . . . 168
8.29.3.10 conditional_prob() . . . 168
8.29.3.11 gen_key() . . . 168
8.29.3.12 get_arrays2support() . . . 169
8.29.3.13 get_counters() . . . 169
8.29.3.14 get_norm_const() . . . 169
8.29.3.15 get_pset() . . . 169
8.29.3.16 get_pset_arrays() . . . 169
8.29.3.17 get_pset_probs() . . . 170
8.29.3.18 get_pset_stats() [1/2] . . . 170
8.29.3.19 get_pset_stats() [2/2] . . . 170
8.29.3.20 get_rengine() . . . 170
8.29.3.21 get_rules() . . . 170
8.29.3.22 get_rules_dyn() . . . 171
8.29.3.23 get_stats_support() . . . 171
8.29.3.24 get_stats_target() . . . 171
8.29.3.25 get_support_fun() . . . 171
8.29.3.26 likelihood() [1/4] . . . 171
8.29.3.27 likelihood() [2/4] . . . 172
8.29.3.28 likelihood() [3/4] . . . 172
8.29.3.29 likelihood() [4/4] . . . 172
8.29.3.30 likelihood_total() . . . 172
8.29.3.31 nrules() . . . 173
8.29.3.32 nrules_dyn() . . . 173
8.29.3.33 nterms() . . . 173
8.29.3.34 operator=() . . . 173
8.29.3.35 print() . . . 173
8.29.3.36 print_stats() . . . 174
8.29.3.37 sample() [1/2] . . . 174
8.29.3.38 sample() [2/2] . . . 174
8.29.3.39 set_counters() . . . 174
8.29.3.40 set_rengine() . . . 174
8.29.3.41 set_rules() . . . 175
8.29.3.42 set_rules_dyn() . . . 175
8.29.3.43 set_seed() . . . 175
8.29.3.44 set_transform_model() . . . 175
8.29.3.45 size() . . . 176
8.29.3.46 size_unique() . . . 176
8.29.3.47 store_psets() . . . 176
8.29.3.48 support_size() . . . 176
8.29.3.49 transform_model() . . . 176

8.30 NetCounterData Class Reference . . . 177
8.30.1 Detailed Description . . . 177
8.30.2 Constructor & Destructor Documentation . . . 177
8.30.2.1 NetCounterData() [1/2] . . . 177
8.30.2.2 NetCounterData() [2/2] . . . 177
8.30.2.3 ~NetCounterData() . . . 178
8.30.3 Member Data Documentation . . . 178
8.30.3.1 indices . . . 178
8.30.3.2 numbers . . . 178
8.31 NetworkData Class Reference . . . 178
8.31.1 Detailed Description . . . 179
8.31.2 Constructor & Destructor Documentation . . . 179
8.31.2.1 NetworkData() [1/3] . . . 179
8.31.2.2 NetworkData() [2/3] . . . 179
8.31.2.3 NetworkData() [3/3] . . . 179
8.31.2.4 ~NetworkData() . . . 180
8.31.3 Member Data Documentation . . . 180
8.31.3.1 directed . . . 180
8.31.3.2 vertex_attr . . . 180
8.32 Node Class Reference . . . 180
8.32.1 Detailed Description . . . 181
8.32.2 Constructor & Destructor Documentation . . . 181
8.32.2.1 Node() [1/5] . . . 182
8.32.2.2 Node() [2/5] . . . 182
8.32.2.3 Node() [3/5] . . . 182
8.32.2.4 Node() [4/5] . . . 182
8.32.2.5 Node() [5/5] . . . 182
8.32.2.6 ~Node() . . . 183
8.32.3 Member Function Documentation . . . 183
8.32.3.1 get_parent() . . . 183
8.32.3.2 is_leaf() . . . 183
8.32.3.3 noffspring() . . . 183
8.32.4 Member Data Documentation . . . 183
8.32.4.1 annotations . . . 183
8.32.4.2 array . . . 184
8.32.4.3 arrays . . . 184
8.32.4.4 duplication . . . 184
8.32.4.5 id . . . 184
8.32.4.6 narray . . . 184
8.32.4.7 offspring . . . 185
8.32.4.8 ord . . . 185
8.32.4.9 parent . . . 185

8.32.4.10 probability . . . 185
8.32.4.11 subtree_prob . . . 185
8.32.4.12 visited . . . 186
8.33 NodeData Class Reference . . . 186
8.33.1 Detailed Description . . . 186
8.33.2 Constructor & Destructor Documentation . . . 186
8.33.2.1 NodeData() . . . 186
8.33.3 Member Data Documentation . . . 187
8.33.3.1 blengths . . . 187
8.33.3.2 duplication . . . 187
8.33.3.3 states . . . 187
8.34 PhyloCounterData Class Reference . . . 187
8.34.1 Detailed Description . . . 188
8.34.2 Constructor & Destructor Documentation . . . 188
8.34.2.1 PhyloCounterData() [1/2] . . . 188
8.34.2.2 PhyloCounterData() [2/2] . . . 188
8.34.3 Member Function Documentation . . . 188
8.34.3.1 at() . . . 188
8.34.3.2 begin() . . . 188
8.34.3.3 empty() . . . 189
8.34.3.4 end() . . . 189
8.34.3.5 get_counters() . . . 189
8.34.3.6 operator()() . . . 189
8.34.3.7 operator[]() . . . 189
8.34.3.8 push_back() . . . 189
8.34.3.9 reserve() . . . 190
8.34.3.10 shrink_to_fit() . . . 190
8.34.3.11 size() . . . 190
8.35 PhyloRuleDynData Class Reference . . . 190
8.35.1 Detailed Description . . . 190
8.35.2 Constructor & Destructor Documentation . . . 191
8.35.2.1 PhyloRuleDynData() . . . 191
8.35.2.2 ~PhyloRuleDynData() . . . 191
8.35.3 Member Data Documentation . . . 191
8.35.3.1 counts . . . 191
8.35.3.2 duplication . . . 191
8.35.3.3 lb . . . 191
8.35.3.4 pos . . . 192
8.35.3.5 ub . . . 192
8.36 PowerSet< Array_Type, Data_Rule_Type > Class Template Reference . . . 192
8.36.1 Detailed Description . . . 193
8.36.2 Constructor & Destructor Documentation . . . 194

8.36.2.1 PowerSet() [1/3] . . . 194
8.36.2.2 PowerSet() [2/3] . . . 194
8.36.2.3 PowerSet() [3/3] . . . 194
8.36.2.4 ∼PowerSet() . . . 194
8.36.3 Member Function Documentation . . . 194
8.36.3.1 add_rule() [1/2] . . . 195
8.36.3.2 add_rule() [2/2] . . . 195
8.36.3.3 begin() . . . 195
8.36.3.4 calc() . . . 195
8.36.3.5 end() . . . 195
8.36.3.6 get_data() . . . 196
8.36.3.7 get_data_ptr() . . . 196
8.36.3.8 init_support() . . . 196
8.36.3.9 operator[]() . . . 196
8.36.3.10 reset() . . . 196
8.36.3.11 size() . . . 197
8.36.4 Member Data Documentation . . . 197
8.36.4.1 coordinates_free . . . 197
8.36.4.2 coordinates_locked . . . 197
8.36.4.3 data . . . 197
8.36.4.4 EmptyArray . . . 197
8.36.4.5 M . . . 198
8.36.4.6 N . . . 198
8.36.4.7 n_free . . . 198
8.36.4.8 n_locked . . . 198
8.36.4.9 rules . . . 198
8.36.4.10 rules_deleted . . . 199
8.37 Progress Class Reference . . . 199
8.37.1 Detailed Description . . . 199
8.37.2 Constructor & Destructor Documentation . . . 199
8.37.2.1 Progress() . . . 199
8.37.2.2 ∼Progress() . . . 200
8.37.3 Member Function Documentation . . . 200
8.37.3.1 end() . . . 200
8.37.3.2 next() . . . 200
8.38 Rule< Array_Type, Data_Type > Class Template Reference . . . 200
8.38.1 Detailed Description . . . 201
8.38.2 Constructor & Destructor Documentation . . . 201
8.38.2.1 Rule() [1/2] . . . 201
8.38.2.2 Rule() [2/2] . . . 201
8.38.2.3 ∼Rule() . . . 202
8.38.3 Member Function Documentation . . . 202

8.38.3.1 D() . . . 202
8.38.3.2 get_description() [1/2] . . . 202
8.38.3.3 get_description() [2/2] . . . 202
8.38.3.4 get_name() [1/2] . . . 202
8.38.3.5 get_name() [2/2] . . . 203
8.38.3.6 operator()() . . . 203
8.39 Rules< Array_Type, Data_Type > Class Template Reference . . . 203
8.39.1 Detailed Description . . . 204
8.39.2 Constructor & Destructor Documentation . . . 204
8.39.2.1 Rules() [1/2] . . . 204
8.39.2.2 Rules() [2/2] . . . 204
8.39.2.3 ~Rules() . . . 204
8.39.3 Member Function Documentation . . . 205
8.39.3.1 add_rule() [1/2] . . . 205
8.39.3.2 add_rule() [2/2] . . . 205
8.39.3.3 get_descriptions() . . . 205
8.39.3.4 get_names() . . . 205
8.39.3.5 get_seq() . . . 205
8.39.3.6 operator()() . . . 206
8.39.3.7 operator=() . . . 206
8.39.3.8 size() . . . 207
8.40 StatsCounter< Array_Type, Data_Type > Class Template Reference . . . 207
8.40.1 Detailed Description . . . 207
8.40.2 Constructor & Destructor Documentation . . . 208
8.40.2.1 StatsCounter() [1/3] . . . 208
8.40.2.2 StatsCounter() [2/3] . . . 208
8.40.2.3 StatsCounter() [3/3] . . . 208
8.40.2.4 ~StatsCounter() . . . 208
8.40.3 Member Function Documentation . . . 209
8.40.3.1 add_counter() . . . 209
8.40.3.2 count_all() . . . 209
8.40.3.3 count_current() . . . 209
8.40.3.4 count_init() . . . 209
8.40.3.5 get_counters() . . . 209
8.40.3.6 get_descriptions() . . . 210
8.40.3.7 get_names() . . . 210
8.40.3.8 reset_array() . . . 210
8.40.3.9 set_counters() . . . 210
8.40.3.10 size() . . . 210
8.41 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference . . . 211
8.41.1 Detailed Description . . . 212

8.41.2 Constructor & Destructor Documentation . . . 213
8.41.2.1 Support() [1/3] . . . 213
8.41.2.2 Support() [2/3] . . . 213
8.41.2.3 Support() [3/3] . . . 213
8.41.2.4 ~Support() . . . 213
8.41.3 Member Function Documentation . . . 214
8.41.3.1 add_counter() . . . 214
8.41.3.2 add_rule() [1/2] . . . 214
8.41.3.3 add_rule() [2/2] . . . 214
8.41.3.4 add_rule_dyn() [1/2] . . . 214
8.41.3.5 add_rule_dyn() [2/2] . . . 214
8.41.3.6 calc() . . . 215
8.41.3.7 eval_rules_dyn() . . . 215
8.41.3.8 get_counters() . . . 215
8.41.3.9 get_counts() . . . 215
8.41.3.10 get_current_stats() . . . 216
8.41.3.11 get_data() . . . 216
8.41.3.12 get_rules() . . . 216
8.41.3.13 get_rules_dyn() . . . 216
8.41.3.14 init_support() . . . 216
8.41.3.15 print() . . . 217
8.41.3.16 reset_array() [1/2] . . . 217
8.41.3.17 reset_array() [2/2] . . . 217
8.41.3.18 set_counters() . . . 217
8.41.3.19 set_rules() . . . 217
8.41.3.20 set_rules_dyn() . . . 218
8.41.4 Member Data Documentation . . . 218
8.41.4.1 change_stats . . . 218
8.41.4.2 coordiantes_n_free . . . 218
8.41.4.3 coordiantes_n_locked . . . 218
8.41.4.4 coordinates_free . . . 219
8.41.4.5 coordinates_locked . . . 219
8.41.4.6 current_stats . . . 219
8.41.4.7 delete_counters . . . 219
8.41.4.8 delete_rules . . . 219
8.41.4.9 delete_rules_dyn . . . 220
8.41.4.10 hashes . . . 220
8.41.4.11 hashes_initialized . . . 220
8.41.4.12 M . . . 220
8.41.4.13 max_num_elements . . . 220
8.41.4.14 N . . . 221
8.41.4.15 n_counters . . . 221

8.42 vecHasher< T > Struct Template Reference . . . 221
8.42.1 Detailed Description . . . 221
8.42.2 Member Function Documentation . . . 221
8.42.2.1 operator()() . . . 221

9 File Documentation **223**
9.1 include/barry/barray-bones.hpp File Reference . . . 223
9.2 include/barry/barray-iterator.hpp File Reference . . . 223
9.3 include/barry/barray-meat-operators.hpp File Reference . . . 224
9.3.1 Macro Definition Documentation . . . 224
9.3.1.1 BARRAY_TEMPLATE . . . 225
9.3.1.2 BARRAY_TEMPLATE_ARGS . . . 225
9.3.1.3 BARRAY_TYPE . . . 225
9.3.1.4 COL . . . 225
9.3.1.5 ROW . . . 225
9.3.2 Function Documentation . . . 225
9.3.2.1 BARRAY_TEMPLATE() [1/6] . . . 226
9.3.2.2 BARRAY_TEMPLATE() [2/6] . . . 226
9.3.2.3 BARRAY_TEMPLATE() [3/6] . . . 226
9.3.2.4 BARRAY_TEMPLATE() [4/6] . . . 226
9.3.2.5 BARRAY_TEMPLATE() [5/6] . . . 226
9.3.2.6 BARRAY_TEMPLATE() [6/6] . . . 227
9.3.2.7 BARRAY_TEMPLATE_ARGS() . . . 227
9.3.2.8 BARRAY_TYPE() . . . 227
9.3.2.9 for() . . . 227
9.3.2.10 operator()() . . . 227
9.3.3 Variable Documentation . . . 227
9.3.3.1 rhs . . . 228
9.3.3.2 this . . . 228
9.4 include/barry/barray-meat.hpp File Reference . . . 228
9.4.1 Macro Definition Documentation . . . 230
9.4.1.1 BARRAY_TEMPLATE . . . 230
9.4.1.2 BARRAY_TEMPLATE_ARGS . . . 230
9.4.1.3 BARRAY_TYPE . . . 231
9.4.1.4 COL . . . 231
9.4.1.5 ROW . . . 231
9.4.2 Function Documentation . . . 231
9.4.2.1 ans() . . . 231
9.4.2.2 BARRAY_TEMPLATE() [1/24] . . . 231
9.4.2.3 BARRAY_TEMPLATE() [2/24] . . . 232
9.4.2.4 BARRAY_TEMPLATE() [3/24] . . . 232
9.4.2.5 BARRAY_TEMPLATE() [4/24] . . . 232

9.4.2.6 BARRAY_TEMPLATE() [5/24] . . . 232
9.4.2.7 BARRAY_TEMPLATE() [6/24] . . . 232
9.4.2.8 BARRAY_TEMPLATE() [7/24] . . . 232
9.4.2.9 BARRAY_TEMPLATE() [8/24] . . . 233
9.4.2.10 BARRAY_TEMPLATE() [9/24] . . . 233
9.4.2.11 BARRAY_TEMPLATE() [10/24] . . . 233
9.4.2.12 BARRAY_TEMPLATE() [11/24] . . . 233
9.4.2.13 BARRAY_TEMPLATE() [12/24] . . . 233
9.4.2.14 BARRAY_TEMPLATE() [13/24] . . . 234
9.4.2.15 BARRAY_TEMPLATE() [14/24] . . . 234
9.4.2.16 BARRAY_TEMPLATE() [15/24] . . . 234
9.4.2.17 BARRAY_TEMPLATE() [16/24] . . . 234
9.4.2.18 BARRAY_TEMPLATE() [17/24] . . . 234
9.4.2.19 BARRAY_TEMPLATE() [18/24] . . . 234
9.4.2.20 BARRAY_TEMPLATE() [19/24] . . . 235
9.4.2.21 BARRAY_TEMPLATE() [20/24] . . . 235
9.4.2.22 BARRAY_TEMPLATE() [21/24] . . . 235
9.4.2.23 BARRAY_TEMPLATE() [22/24] . . . 235
9.4.2.24 BARRAY_TEMPLATE() [23/24] . . . 235
9.4.2.25 BARRAY_TEMPLATE() [24/24] . . . 235
9.4.2.26 COL() . . . 236
9.4.2.27 for() [1/3] . . . 236
9.4.2.28 for() [2/3] . . . 236
9.4.2.29 for() [3/3] . . . 236
9.4.2.30 if() [1/17] . . . 236
9.4.2.31 if() [2/17] . . . 236
9.4.2.32 if() [3/17] . . . 237
9.4.2.33 if() [4/17] . . . 237
9.4.2.34 if() [5/17] . . . 237
9.4.2.35 if() [6/17] . . . 237
9.4.2.36 if() [7/17] . . . 237
9.4.2.37 if() [8/17] . . . 237
9.4.2.38 if() [9/17] . . . 238
9.4.2.39 if() [10/17] . . . 238
9.4.2.40 if() [11/17] . . . 238
9.4.2.41 if() [12/17] . . . 238
9.4.2.42 if() [13/17] . . . 238
9.4.2.43 if() [14/17] . . . 238
9.4.2.44 if() [15/17] . . . 238
9.4.2.45 if() [16/17] . . . 239
9.4.2.46 if() [17/17] . . . 239
9.4.2.47 M() . . . 239

9.4.2.48 resize() [1/2] . . . 239
9.4.2.49 resize() [2/2] . . . 239
9.4.2.50 return() . . . 239
9.4.2.51 ROW() [1/2] . . . 240
9.4.2.52 ROW() [2/2] . . . 240
9.4.3 Variable Documentation . . . 240
9.4.3.1 add . . . 240
9.4.3.2 ans . . . 240
9.4.3.3 Array_ . . . 240
9.4.3.4 check_bounds . . . 241
9.4.3.5 check_exists . . . 241
9.4.3.6 col0 . . . 241
9.4.3.7 const . . . 241
9.4.3.8 copy_data . . . 242
9.4.3.9 data . . . 242
9.4.3.10 delete_data . . . 242
9.4.3.11 delete_data_ . . . 242
9.4.3.12 else . . . 242
9.4.3.13 false . . . 243
9.4.3.14 first . . . 243
9.4.3.15 i1 . . . 243
9.4.3.16 j . . . 243
9.4.3.17 j0 . . . 243
9.4.3.18 j1 . . . 243
9.4.3.19 M . . . 244
9.4.3.20 M_ . . . 244
9.4.3.21 N . . . 244
9.4.3.22 NCells . . . 244
9.4.3.23 report . . . 244
9.4.3.24 return . . . 245
9.4.3.25 row0 . . . 245
9.4.3.26 search . . . 245
9.4.3.27 source . . . 245
9.4.3.28 target . . . 245
9.4.3.29 v . . . 245
9.4.3.30 value . . . 246
9.5 include/barry/barraycell-bones.hpp File Reference . . . 246
9.6 include/barry/barraycell-meat.hpp File Reference . . . 246
9.7 include/barry/barraydense-bones.hpp File Reference . . . 247
9.8 include/barry/barraydense-meat-operators.hpp File Reference . . . 247
9.8.1 Macro Definition Documentation . . . 248
9.8.1.1 BDENSE_TEMPLATE . . . 248

9.8.1.2 BDENSE_TEMPLATE_ARGS . . . 248
9.8.1.3 BDENSE_TYPE . . . 248
9.8.1.4 COL . . . 248
9.8.1.5 POS . . . 249
9.8.1.6 POS_N . . . 249
9.8.1.7 ROW . . . 249
9.8.2 Function Documentation . . . 249
9.8.2.1 BDENSE_TEMPLATE() [1/4] . . . 249
9.8.2.2 BDENSE_TEMPLATE() [2/4] . . . 249
9.8.2.3 BDENSE_TEMPLATE() [3/4] . . . 250
9.8.2.4 BDENSE_TEMPLATE() [4/4] . . . 250
9.8.2.5 BDENSE_TEMPLATE_ARGS() . . . 250
9.8.2.6 BDENSE_TYPE() . . . 250
9.9 include/barry/barraydense-meat.hpp File Reference . . . 250
9.9.1 Macro Definition Documentation . . . 253
9.9.1.1 BDENSE_TEMPLATE . . . 253
9.9.1.2 BDENSE_TEMPLATE_ARGS . . . 253
9.9.1.3 BDENSE_TYPE . . . 253
9.9.1.4 COL . . . 253
9.9.1.5 POS . . . 253
9.9.1.6 POS_N . . . 254
9.9.1.7 ROW . . . 254
9.9.1.8 ZERO_CELL . . . 254
9.9.2 Function Documentation . . . 254
9.9.2.1 ans() . . . 254
9.9.2.2 BDENSE_TEMPLATE() [1/39] . . . 254
9.9.2.3 BDENSE_TEMPLATE() [2/39] . . . 255
9.9.2.4 BDENSE_TEMPLATE() [3/39] . . . 255
9.9.2.5 BDENSE_TEMPLATE() [4/39] . . . 255
9.9.2.6 BDENSE_TEMPLATE() [5/39] . . . 255
9.9.2.7 BDENSE_TEMPLATE() [6/39] . . . 255
9.9.2.8 BDENSE_TEMPLATE() [7/39] . . . 255
9.9.2.9 BDENSE_TEMPLATE() [8/39] . . . 256
9.9.2.10 BDENSE_TEMPLATE() [9/39] . . . 256
9.9.2.11 BDENSE_TEMPLATE() [10/39] . . . 256
9.9.2.12 BDENSE_TEMPLATE() [11/39] . . . 256
9.9.2.13 BDENSE_TEMPLATE() [12/39] . . . 256
9.9.2.14 BDENSE_TEMPLATE() [13/39] . . . 257
9.9.2.15 BDENSE_TEMPLATE() [14/39] . . . 257
9.9.2.16 BDENSE_TEMPLATE() [15/39] . . . 257
9.9.2.17 BDENSE_TEMPLATE() [16/39] . . . 257
9.9.2.18 BDENSE_TEMPLATE() [17/39] . . . 257

9.9.2.19 BDENSE_TEMPLATE() [18/39] . . . 258
9.9.2.20 BDENSE_TEMPLATE() [19/39] . . . 258
9.9.2.21 BDENSE_TEMPLATE() [20/39] . . . 258
9.9.2.22 BDENSE_TEMPLATE() [21/39] . . . 258
9.9.2.23 BDENSE_TEMPLATE() [22/39] . . . 258
9.9.2.24 BDENSE_TEMPLATE() [23/39] . . . 259
9.9.2.25 BDENSE_TEMPLATE() [24/39] . . . 259
9.9.2.26 BDENSE_TEMPLATE() [25/39] . . . 259
9.9.2.27 BDENSE_TEMPLATE() [26/39] . . . 259
9.9.2.28 BDENSE_TEMPLATE() [27/39] . . . 259
9.9.2.29 BDENSE_TEMPLATE() [28/39] . . . 260
9.9.2.30 BDENSE_TEMPLATE() [29/39] . . . 260
9.9.2.31 BDENSE_TEMPLATE() [30/39] . . . 260
9.9.2.32 BDENSE_TEMPLATE() [31/39] . . . 260
9.9.2.33 BDENSE_TEMPLATE() [32/39] . . . 260
9.9.2.34 BDENSE_TEMPLATE() [33/39] . . . 260
9.9.2.35 BDENSE_TEMPLATE() [34/39] . . . 261
9.9.2.36 BDENSE_TEMPLATE() [35/39] . . . 261
9.9.2.37 BDENSE_TEMPLATE() [36/39] . . . 261
9.9.2.38 BDENSE_TEMPLATE() [37/39] . . . 261
9.9.2.39 BDENSE_TEMPLATE() [38/39] . . . 261
9.9.2.40 BDENSE_TEMPLATE() [39/39] . . . 261
9.9.2.41 for() . . . 262
9.9.2.42 if() [1/4] . . . 262
9.9.2.43 if() [2/4] . . . 262
9.9.2.44 if() [3/4] . . . 262
9.9.2.45 if() [4/4] . . . 262
9.9.2.46 insert_cell() [1/2] . . . 262
9.9.2.47 insert_cell() [2/2] . . . 263
9.9.2.48 M() . . . 263
9.9.2.49 resize() [1/6] . . . 263
9.9.2.50 resize() [2/6] . . . 263
9.9.2.51 resize() [3/6] . . . 263
9.9.2.52 resize() [4/6] . . . 263
9.9.2.53 resize() [5/6] . . . 264
9.9.2.54 resize() [6/6] . . . 264
9.9.2.55 rm_cell() [1/3] . . . 264
9.9.2.56 rm_cell() [2/3] . . . 264
9.9.2.57 rm_cell() [3/3] . . . 264
9.9.2.58 va_end() . . . 264
9.9.2.59 va_start() . . . 265
9.9.2.60 vprintf() . . . 265

9.9.3 Variable Documentation . . . 265
9.9.3.1 add . . . 265
9.9.3.2 ans . . . 265
9.9.3.3 check_bounds . . . 265
9.9.3.4 check_exists . . . 266
9.9.3.5 col . . . 266
9.9.3.6 const . . . 266
9.9.3.7 copy_data . . . 266
9.9.3.8 data . . . 266
9.9.3.9 delete_data . . . 267
9.9.3.10 delete_data_ . . . 267
9.9.3.11 el . . . 267
9.9.3.12 el_colsums . . . 267
9.9.3.13 el_rowsums . . . 267
9.9.3.14 else . . . 268
9.9.3.15 false . . . 268
9.9.3.16 i1 . . . 268
9.9.3.17 j . . . 268
9.9.3.18 j0 . . . 268
9.9.3.19 j1 . . . 268
9.9.3.20 M . . . 269
9.9.3.21 M_ . . . 269
9.9.3.22 N . . . 269
9.9.3.23 report . . . 269
9.9.3.24 return . . . 269
9.9.3.25 source . . . 270
9.9.3.26 target . . . 270
9.9.3.27 v . . . 270
9.9.3.28 val0 . . . 270
9.9.3.29 val1 . . . 270
9.9.3.30 value . . . 270
9.10 include/barry/barraydensecell-bones.hpp File Reference . . . 271
9.10.1 Macro Definition Documentation . . . 271
9.10.1.1 POS . . . 271
9.11 include/barry/barraydensecell-meat.hpp File Reference . . . 272
9.11.1 Macro Definition Documentation . . . 272
9.11.1.1 POS . . . 272
9.12 include/barry/barraydensecol-bones.hpp File Reference . . . 272
9.12.1 Macro Definition Documentation . . . 273
9.12.1.1 POS . . . 273
9.12.1.2 POS_N . . . 273
9.12.1.3 ZERO_CELL . . . 273

9.13 include/barry/barraydenserow-bones.hpp File Reference . . . 274
9.13.1 Macro Definition Documentation . . . 274
9.13.1.1 POS . . . 274
9.13.1.2 POS_N . . . 275
9.13.1.3 ZERO_CELL . . . 275
9.14 include/barry/barrayrow-bones.hpp File Reference . . . 275
9.15 include/barry/barrayrow-meat.hpp File Reference . . . 275
9.15.1 Macro Definition Documentation . . . 275
9.15.1.1 BROW_TEMPLATE . . . 276
9.15.1.2 BROW_TEMPLATE_ARGS . . . 276
9.15.1.3 BROW_TYPE . . . 276
9.15.2 Function Documentation . . . 276
9.15.2.1 BROW_TEMPLATE() [1/5] . . . 276
9.15.2.2 BROW_TEMPLATE() [2/5] . . . 276
9.15.2.3 BROW_TEMPLATE() [3/5] . . . 277
9.15.2.4 BROW_TEMPLATE() [4/5] . . . 277
9.15.2.5 BROW_TEMPLATE() [5/5] . . . 277
9.16 include/barry/barrayvector-bones.hpp File Reference . . . 277
9.17 include/barry/barrayvector-meat.hpp File Reference . . . 278
9.18 include/barry/barry-configuration.hpp File Reference . . . 278
9.18.1 Macro Definition Documentation . . . 279
9.18.1.1 BARRY_CHECK_SUPPORT . . . 279
9.18.1.2 BARRY_ISFINITE . . . 279
9.18.1.3 BARRY_MAX_NUM_ELEMENTS . . . 279
9.18.1.4 BARRY_SAFE_EXP . . . 279
9.18.1.5 printf_barry . . . 279
9.18.2 Typedef Documentation . . . 279
9.18.2.1 Map . . . 280
9.19 include/barry/barry-debug.hpp File Reference . . . 280
9.19.1 Macro Definition Documentation . . . 280
9.19.1.1 BARRY_DEBUG_LEVEL . . . 280
9.20 include/barry/barry-macros.hpp File Reference . . . 281
9.20.1 Macro Definition Documentation . . . 281
9.20.1.1 BARRY_ONE . . . 281
9.20.1.2 BARRY_ONE_DENSE . . . 281
9.20.1.3 BARRY_UNUSED . . . 282
9.20.1.4 BARRY_ZERO . . . 282
9.20.1.5 BARRY_ZERO_DENSE . . . 282
9.21 include/barry/barry.hpp File Reference . . . 282
9.21.1 Macro Definition Documentation . . . 283
9.21.1.1 BARRY_HPP . . . 284
9.21.1.2 BARRY_VERSION . . . 284

9.21.1.3 BARRY_VERSION_MAYOR . . . 284
9.21.1.4 BARRY_VERSION_MINOR . . . 284
9.21.1.5 COUNTER_FUNCTION . . . 284
9.21.1.6 COUNTER_LAMBDA . . . 285
9.21.1.7 RULE_FUNCTION . . . 285
9.21.1.8 RULE_LAMBDA . . . 285
9.22 include/barry/cell-bones.hpp File Reference . . . 285
9.23 include/barry/cell-meat.hpp File Reference . . . 286
9.24 include/barry/col-bones.hpp File Reference . . . 286
9.25 include/barry/counters-bones.hpp File Reference . . . 286
9.26 include/barry/counters-meat.hpp File Reference . . . 287
9.26.1 Macro Definition Documentation . . . 288
9.26.1.1 COUNTER_TEMPLATE . . . 288
9.26.1.2 COUNTER_TEMPLATE_ARGS . . . 289
9.26.1.3 COUNTER_TYPE . . . 289
9.26.1.4 COUNTERS_TEMPLATE . . . 289
9.26.1.5 COUNTERS_TEMPLATE_ARGS . . . 289
9.26.1.6 COUNTERS_TYPE . . . 289
9.26.1.7 TMP_HASHER_CALL . . . 289
9.26.2 Function Documentation . . . 290
9.26.2.1 count_fun() . . . 290
9.26.2.2 COUNTER_TEMPLATE() [1/9] . . . 290
9.26.2.3 COUNTER_TEMPLATE() [2/9] . . . 290
9.26.2.4 COUNTER_TEMPLATE() [3/9] . . . 290
9.26.2.5 COUNTER_TEMPLATE() [4/9] . . . 290
9.26.2.6 COUNTER_TEMPLATE() [5/9] . . . 291
9.26.2.7 COUNTER_TEMPLATE() [6/9] . . . 291
9.26.2.8 COUNTER_TEMPLATE() [7/9] . . . 291
9.26.2.9 COUNTER_TEMPLATE() [8/9] . . . 291
9.26.2.10 COUNTER_TEMPLATE() [9/9] . . . 291
9.26.2.11 COUNTERS_TEMPLATE() [1/9] . . . 292
9.26.2.12 COUNTERS_TEMPLATE() [2/9] . . . 292
9.26.2.13 COUNTERS_TEMPLATE() [3/9] . . . 292
9.26.2.14 COUNTERS_TEMPLATE() [4/9] . . . 292
9.26.2.15 COUNTERS_TEMPLATE() [5/9] . . . 292
9.26.2.16 COUNTERS_TEMPLATE() [6/9] . . . 292
9.26.2.17 COUNTERS_TEMPLATE() [7/9] . . . 293
9.26.2.18 COUNTERS_TEMPLATE() [8/9] . . . 293
9.26.2.19 COUNTERS_TEMPLATE() [9/9] . . . 293
9.26.2.20 data() . . . 293
9.26.2.21 desc() . . . 293
9.26.2.22 for() . . . 293

9.26.2.23 hasher() [1/2] . . . 294
9.26.2.24 hasher() [2/2] . . . 294
9.26.2.25 hasher_fun() [1/2] . . . 294
9.26.2.26 hasher_fun() [2/2] . . . 294
9.26.2.27 if() [1/3] . . . 294
9.26.2.28 if() [2/3] . . . 294
9.26.2.29 if() [3/3] . . . 295
9.26.2.30 init_fun() [1/3] . . . 295
9.26.2.31 init_fun() [2/3] . . . 295
9.26.2.32 init_fun() [3/3] . . . 295
9.26.2.33 name() . . . 295
9.26.3 Variable Documentation . . . 295
9.26.3.1 add_dims . . . 295
9.26.3.2 count_fun_ . . . 296
9.26.3.3 counter . . . 296
9.26.3.4 counter_ . . . 296
9.26.3.5 data_ . . . 296
9.26.3.6 desc_ . . . 297
9.26.3.7 fun . . . 297
9.26.3.8 fun_ . . . 297
9.26.3.9 hasher_fun_ . . . 297
9.26.3.10 i . . . 298
9.26.3.11 init_fun_ . . . 298
9.26.3.12 j . . . 298
9.26.3.13 name_ . . . 298
9.26.3.14 noexcept . . . 298
9.26.3.15 res . . . 299
9.26.3.16 return . . . 299
9.27 include/barry/counters/defm-formula.hpp File Reference . . . 299
9.27.1 Function Documentation . . . 299
9.27.1.1 defm_motif_parser() . . . 300
9.28 include/barry/counters/defm.hpp File Reference . . . 301
9.28.1 Macro Definition Documentation . . . 303
9.28.1.1 UNI_SUB . . . 303
9.28.2 Typedef Documentation . . . 303
9.28.2.1 DEFMArray . . . 303
9.29 include/barry/models/defm.hpp File Reference . . . 304
9.30 include/barry/counters/network-css.hpp File Reference . . . 304
9.30.1 Macro Definition Documentation . . . 306
9.30.1.1 CSS_APPEND . . . 306
9.30.1.2 CSS_CASE_ELSE . . . 306
9.30.1.3 CSS_CASE_PERCEIVED . . . 306

9.30.1.4 CSS_CASE_TRUTH . . . 306
9.30.1.5 CSS_CHECK_SIZE . . . 307
9.30.1.6 CSS_CHECK_SIZE_INIT . . . 307
9.30.1.7 CSS_NET_COUNTER_LAMBDA_INIT . . . 307
9.30.1.8 CSS_PERCEIVED_CELLS . . . 307
9.30.1.9 CSS_SIZE . . . 308
9.30.1.10 CSS_TRUE_CELLS . . . 308
9.30.2 Function Documentation . . . 308
9.30.2.1 counter_css_census01() . . . 308
9.30.2.2 counter_css_census02() . . . 308
9.30.2.3 counter_css_census03() . . . 309
9.30.2.4 counter_css_census04() . . . 309
9.30.2.5 counter_css_census05() . . . 309
9.30.2.6 counter_css_census06() . . . 309
9.30.2.7 counter_css_census07() . . . 310
9.30.2.8 counter_css_census08() . . . 310
9.30.2.9 counter_css_census09() . . . 310
9.30.2.10 counter_css_census10() . . . 310
9.30.2.11 counter_css_completely_false_recip_comiss() . . . 311
9.30.2.12 counter_css_completely_false_recip_omiss() . . . 311
9.30.2.13 counter_css_mixed_recip() . . . 311
9.30.2.14 counter_css_partially_false_recip_commi() . . . 311
9.30.2.15 counter_css_partially_false_recip_omiss() . . . 312
9.31 include/barry/counters/network.hpp File Reference . . . 312
9.31.1 Macro Definition Documentation . . . 315
9.31.1.1 BARRY_ZERO_NETWORK . . . 315
9.31.1.2 BARRY_ZERO_NETWORK_DENSE . . . 316
9.31.1.3 NET_C_DATA_IDX . . . 316
9.31.1.4 NET_C_DATA_NUM . . . 316
9.31.1.5 NETWORK_COUNTER . . . 316
9.31.1.6 NETWORK_COUNTER_LAMBDA . . . 316
9.31.1.7 NETWORK_RULE . . . 317
9.31.1.8 NETWORK_RULE_LAMBDA . . . 317
9.31.1.9 NETWORKDENSE_COUNTER_LAMBDA . . . 317
9.31.2 Typedef Documentation . . . 317
9.31.2.1 NetCounter . . . 317
9.31.2.2 NetCounters . . . 318
9.31.2.3 NetModel . . . 318
9.31.2.4 NetRule . . . 318
9.31.2.5 NetRules . . . 318
9.31.2.6 NetStatsCounter . . . 318
9.31.2.7 NetSupport . . . 318

9.31.2.8 Network . . . 319
9.31.2.9 NetworkDense . . . 319
9.31.3 Function Documentation . . . 319
9.31.3.1 rules_zerodiag() . . . 319
9.32 include/barry/counters/phylo.hpp File Reference . . . 319
9.32.1 Macro Definition Documentation . . . 322
9.32.1.1 DEFAULT_DUPLICATION . . . 322
9.32.1.2 DUPL_DUPL . . . 322
9.32.1.3 DUPL_EITH . . . 322
9.32.1.4 DUPL_SPEC . . . 322
9.32.1.5 IF_MATCHES . . . 323
9.32.1.6 IF_NOTMATCHES . . . 323
9.32.1.7 IS_DUPLICATION . . . 323
9.32.1.8 IS_EITHER . . . 323
9.32.1.9 IS_SPECIATION . . . 323
9.32.1.10 MAKE_DUPL_VARS . . . 324
9.32.1.11 PHYLO_CHECK_MISSING . . . 324
9.32.1.12 PHYLO_COUNTER_LAMBDA . . . 324
9.32.1.13 PHYLO_RULE_DYN_LAMBDA . . . 324
9.32.2 Typedef Documentation . . . 325
9.32.2.1 PhyloArray . . . 325
9.32.2.2 PhyloCounter . . . 325
9.32.2.3 PhyloCounters . . . 325
9.32.2.4 PhyloModel . . . 325
9.32.2.5 PhyloPowerSet . . . 325
9.32.2.6 PhyloRule . . . 326
9.32.2.7 PhyloRuleData . . . 326
9.32.2.8 PhyloRuleDyn . . . 326
9.32.2.9 PhyloRules . . . 326
9.32.2.10 PhyloRulesDyn . . . 326
9.32.2.11 PhyloStatsCounter . . . 326
9.32.2.12 PhyloSupport . . . 327
9.32.3 Function Documentation . . . 327
9.32.3.1 get_last_name() . . . 327
9.33 include/barry/freqtable.hpp File Reference . . . 327
9.34 include/barry/model-bones.hpp File Reference . . . 328
9.35 include/barry/model-meat.hpp File Reference . . . 328
9.35.1 Macro Definition Documentation . . . 329
9.35.1.1 MODEL_TEMPLATE . . . 329
9.35.1.2 MODEL_TEMPLATE_ARGS . . . 329
9.35.1.3 MODEL_TYPE . . . 329
9.35.2 Function Documentation . . . 329

9.35.2.1 likelihood_() . . . 330
9.35.2.2 MODEL_TEMPLATE() [1/2] . . . 330
9.35.2.3 MODEL_TEMPLATE() [2/2] . . . 330
9.35.2.4 update_normalizing_constant() . . . 330
9.36 include/barry/models/defm/defm-bones.hpp File Reference . . . 331
9.37 include/barry/models/defm/defm-meat.hpp File Reference . . . 331
9.37.1 Macro Definition Documentation . . . 332
9.37.1.1 DEFM_LOOP_ARRAYS . . . 332
9.37.1.2 DEFM_RANGES . . . 332
9.37.2 Function Documentation . . . 332
9.37.2.1 keygen_defm() . . . 332
9.38 include/barry/models/geese.hpp File Reference . . . 333
9.39 include/barry/models/geese/flock-bones.hpp File Reference . . . 333
9.40 include/barry/models/geese/flock-meat.hpp File Reference . . . 334
9.41 include/barry/models/geese/geese-bones.hpp File Reference . . . 334
9.41.1 Macro Definition Documentation . . . 335
9.41.1.1 INITIALIZED . . . 335
9.41.2 Function Documentation . . . 335
9.41.2.1 keygen_full() . . . 335
9.41.2.2 RULE_FUNCTION() . . . 335
9.41.2.3 vec_diff() . . . 336
9.41.2.4 vector_caster() . . . 336
9.42 include/barry/models/geese/geese-meat-constructors.hpp File Reference . . . 336
9.43 include/barry/models/geese/geese-meat-likelihood.hpp File Reference . . . 337
9.44 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference . . . 338
9.45 include/barry/models/geese/geese-meat-predict.hpp File Reference . . . 338
9.46 include/barry/models/geese/geese-meat-predict_exhaust.hpp File Reference . . . 339
9.47 include/barry/models/geese/geese-meat-predict_sim.hpp File Reference . . . 339
9.48 include/barry/models/geese/geese-meat-simulate.hpp File Reference . . . 340
9.49 include/barry/models/geese/geese-meat.hpp File Reference . . . 340
9.50 include/barry/models/geese/geese-node-bones.hpp File Reference . . . 341
9.51 include/barry/powerset-bones.hpp File Reference . . . 341
9.52 include/barry/powerset-meat.hpp File Reference . . . 342
9.53 include/barry/progress.hpp File Reference . . . 342
9.53.1 Macro Definition Documentation . . . 343
9.53.1.1 BARRY_PROGRESS_BAR_WIDTH . . . 343
9.54 include/barry/rules-bones.hpp File Reference . . . 343
9.54.1 Function Documentation . . . 343
9.54.1.1 rule_fun_default() . . . 344
9.55 include/barry/rules-meat.hpp File Reference . . . 344
9.56 include/barry/statscounter-bones.hpp File Reference . . . 344
9.57 include/barry/statscounter-meat.hpp File Reference . . . 345

9.57.1 Macro Definition Documentation . . . 346
9.57.1.1 STATSCOUNTER_TEMPLATE . . . 346
9.57.1.2 STATSCOUNTER_TEMPLATE_ARGS . . . 346
9.57.1.3 STATSCOUNTER_TYPE . . . 346
9.57.2 Function Documentation . . . 346
9.57.2.1 clear() . . . 346
9.57.2.2 for() . . . 347
9.57.2.3 resize() . . . 347
9.57.2.4 STATSCOUNTER_TEMPLATE() [1/9] . . . 347
9.57.2.5 STATSCOUNTER_TEMPLATE() [2/9] . . . 347
9.57.2.6 STATSCOUNTER_TEMPLATE() [3/9] . . . 347
9.57.2.7 STATSCOUNTER_TEMPLATE() [4/9] . . . 347
9.57.2.8 STATSCOUNTER_TEMPLATE() [5/9] . . . 348
9.57.2.9 STATSCOUNTER_TEMPLATE() [6/9] . . . 348
9.57.2.10 STATSCOUNTER_TEMPLATE() [7/9] . . . 348
9.57.2.11 STATSCOUNTER_TEMPLATE() [8/9] . . . 348
9.57.2.12 STATSCOUNTER_TEMPLATE() [9/9] . . . 348
9.57.3 Variable Documentation . . . 348
9.57.3.1 counter . . . 349
9.57.3.2 counter_deleted . . . 349
9.57.3.3 counters . . . 349
9.57.3.4 counters_ . . . 349
9.57.3.5 current_stats . . . 349
9.57.3.6 EmptyArray . . . 350
9.57.3.7 f_ . . . 350
9.57.3.8 j . . . 350
9.57.3.9 return . . . 350
9.58 include/barry/support-bones.hpp File Reference . . . 350
9.59 include/barry/support-meat.hpp File Reference . . . 351
9.59.1 Macro Definition Documentation . . . 352
9.59.1.1 BARRY_SUPPORT_MEAT_HPP . . . 352
9.59.1.2 SUPPORT_TEMPLATE . . . 352
9.59.1.3 SUPPORT_TEMPLATE_ARGS . . . 353
9.59.1.4 SUPPORT_TYPE . . . 353
9.59.2 Function Documentation . . . 353
9.59.2.1 calc_backend_dense() . . . 353
9.59.2.2 calc_backend_sparse() . . . 353
9.59.2.3 for() . . . 353
9.59.2.4 if() [1/3] . . . 354
9.59.2.5 if() [2/3] . . . 354
9.59.2.6 if() [3/3] . . . 354
9.59.2.7 insert_cell() [1/2] . . . 354

9.59.2.8 insert_cell() [2/2] . . . 354
9.59.2.9 rm_cell() . . . 355
9.59.2.10 SUPPORT_TEMPLATE() [1/17] . . . 355
9.59.2.11 SUPPORT_TEMPLATE() [2/17] . . . 355
9.59.2.12 SUPPORT_TEMPLATE() [3/17] . . . 355
9.59.2.13 SUPPORT_TEMPLATE() [4/17] . . . 355
9.59.2.14 SUPPORT_TEMPLATE() [5/17] . . . 356
9.59.2.15 SUPPORT_TEMPLATE() [6/17] . . . 356
9.59.2.16 SUPPORT_TEMPLATE() [7/17] . . . 356
9.59.2.17 SUPPORT_TEMPLATE() [8/17] . . . 356
9.59.2.18 SUPPORT_TEMPLATE() [9/17] . . . 356
9.59.2.19 SUPPORT_TEMPLATE() [10/17] . . . 356
9.59.2.20 SUPPORT_TEMPLATE() [11/17] . . . 357
9.59.2.21 SUPPORT_TEMPLATE() [12/17] . . . 357
9.59.2.22 SUPPORT_TEMPLATE() [13/17] . . . 357
9.59.2.23 SUPPORT_TEMPLATE() [14/17] . . . 357
9.59.2.24 SUPPORT_TEMPLATE() [15/17] . . . 357
9.59.2.25 SUPPORT_TEMPLATE() [16/17] . . . 358
9.59.2.26 SUPPORT_TEMPLATE() [17/17] . . . 358
9.59.3 Variable Documentation . . . 358
9.59.3.1 array_bank . . . 358
9.59.3.2 change_stats_different . . . 358
9.59.3.3 coord_i . . . 358
9.59.3.4 coord_j . . . 358
9.59.3.5 counters . . . 359
9.59.3.6 counters_ . . . 359
9.59.3.7 delete_counters . . . 359
9.59.3.8 delete_rules . . . 359
9.59.3.9 delete_rules_dyn . . . 359
9.59.3.10 else . . . 360
9.59.3.11 f_ . . . 360
9.59.3.12 hashes . . . 360
9.59.3.13 return . . . 360
9.59.3.14 rules . . . 360
9.59.3.15 rules_ . . . 361
9.59.3.16 rules_dyn . . . 361
9.59.3.17 stats_bank . . . 361
9.59.3.18 tmp_chng . . . 361
9.60 include/barry/typedefs.hpp File Reference . . . 362
9.60.1 Typedef Documentation . . . 364
9.60.1.1 Col_type . . . 364
9.60.1.2 Counter_fun_type . . . 364

9.60.1.3 Counts_type . . . 364
9.60.1.4 Hasher_fun_type . . . 364
9.60.1.5 MapVec_type . . . 365
9.60.1.6 Row_type . . . 365
9.60.1.7 Rule_fun_type . . . 365
9.60.1.8 uint . . . 365
9.60.2 Function Documentation . . . 365
9.60.2.1 sort_array() . . . 365
9.60.2.2 vec_equal() . . . 366
9.60.2.3 vec_equal_approx() . . . 366
9.60.2.4 vec_inner_prod() [1/2] . . . 367
9.60.2.5 vec_inner_prod() [2/2] . . . 367
9.61 README.md File Reference . . . 367

Index **369**

Chapter 1

Main Page

Barry: your to-go motif accountant

This repository contains a C++ template library that essentially counts sufficient statistics on binary arrays. Its primary goal is to provide a general framework for building discrete exponential-family models. A particular example is Exponential Random Graph Models (ERGMs), but we can use barry to deal with non-square arrays.

Among the key features included in barry, we have:

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

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

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

Examples

Counting statistics in a graph

In the following code we create an array of size 5x5 of class `Network` (available in the namespace `netcounters`), add/remove ties, print the graph, and count common statistics used in ERGMs:

```
#include <iostream>
#include <ostream>
#include "../include/barry.hpp"
typedef std::vector< unsigned int > vuint;
int main() {
  // Creating network of size six with five ties
  netcounters::Network net(
      6, 6,
      {0, 0, 4, 4, 2, 0, 1},
      {1, 2, 0, 2, 4, 0, 1}
  );

  // How does this looks like?
  net.print("Current view");

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

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

  // Initializing the data. The program deals with 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();

  std::cout «
    "Edges             : " « counts[0] « std::endl «
    "Transitive triads : " « counts[1] « std::endl «
    "Isolates          : " « counts[2] « std::endl «
    "C triads          : " « counts[3] « std::endl «
    "Mutuals           : " « counts[4] « std::endl;

  return 0;
}
```

Compiling this program using g++

```
g++ -std=c++11 -Wall -pedantic 08-counts.cpp -o counts && ./counts
```

Yields the following output:

```
Current view
[  0,]  1  1  1  .  .  .
[  1,]  .  1  .  .  .  .
[  2,]  .  .  .  .  1  .
[  3,]  .  .  .  .  .  .
[  4,]  1  .  1  .  .  .
[  5,]  .  .  .  .  .  .
New view
[  0,]  .  1  1  .  .  .
[  1,]  1  .  .  .  .  .
[  2,]  1  .  .  .  1  .
[  3,]  .  .  .  .  .  .
[  4,]  1  .  1  .  .  .
[  5,]  .  .  .  .  .  .
Edges             : 7
Transitive triads : 3
Isolates          : 2
C triads          : 1
Mutuals           : 3
```

Features

Efficient memory usage

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

Documentation

More information can be found in the Doxygen website here and in the PDF version of the documentation here.

Code of Conduct

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

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Counting 13
Statistical Models 13
Phylo rules 14
DEFMArray counters 19
Phylo counters 30

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

BArray< Cell_Type, Data_Type > . . . 43
BArray< bool, bool > . . . 43
BArray< Cell_Type, Data_Type > . . . 43
BArrayCell< Cell_Type, Data_Type > . . . 57
BArrayCell_const< Cell_Type, Data_Type > . . . 60
BArrayDense< Cell_Type, Data_Type > . . . 62
BArrayDense< bool, bool > . . . 62
BArrayDenseCell< Cell_Type, Data_Type > . . . 78
BArrayDenseCell_const< Cell_Type, Data_Type > . . . 82
BArrayDenseCol< Cell_Type, Data_Type > . . . 82
BArrayDenseCol_const< Cell_Type, Data_Type > . . . 84
BArrayDenseRow< Cell_Type, Data_Type > . . . 86
BArrayDenseRow_const< Cell_Type, Data_Type > . . . 89
BArrayRow< Cell_Type, Data_Type > . . . 91
BArrayRow_const< Cell_Type, Data_Type > . . . 93
BArrayVector< Cell_Type, Data_Type > . . . 95
BArrayVector_const< Cell_Type, Data_Type > . . . 99
Cell< Cell_Type > . . . 103
Cell< bool > . . . 103
Cell_const< Cell_Type > . . . 108
ConstBArrayRowIter< Cell_Type, Data_Type > . . . 108
Counter< Array_Type, Data_Type > . . . 110
Counters< Array_Type, Data_Type > . . . 116
Counters< Array_Type, Data_Type > . . . 116
Counters< BArray< bool, bool >, bool > . . . 116
Counters< BArray<>, bool > . . . 116
DEFM . . . 120
DEFMCounterData . . . 125
DEFMData . . . 127
DEFMRuleData . . . 132
    DEFMRuleDynData . . . 135
Entries< Cell_Type > . . . 136
Flock . . . 139
FreqTable< T > . . . 145
Geese . . . 149

Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > 162
NetCounterData 177
NetworkData 178
Node 180
NodeData 186
PhyloCounterData 187
PhyloRuleDynData 190
PowerSet< Array_Type, Data_Rule_Type > 192
Progress 199
Rule< Array_Type, Data_Type > 200
Rules< Array_Type, Data_Type > 203
Rules< BArray< bool, bool >, bool > 203
Rules< BArray<>, bool > 203
StatsCounter< Array_Type, Data_Type > 207
StatsCounter< BArray<>, bool > 207
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > 211
Support< BArray<>, bool, bool, bool > 211
vecHasher< T > 221

Chapter 4

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 . . . 43
BArrayCell< Cell_Type, Data_Type > . . . 57
BArrayCell_const< Cell_Type, Data_Type > . . . 60
BArrayDense< Cell_Type, Data_Type >
: Baseline class for binary arrays . . . 62
BArrayDenseCell< Cell_Type, Data_Type > . . . 78
BArrayDenseCell_const< Cell_Type, Data_Type > . . . 82
BArrayDenseCol< Cell_Type, Data_Type > . . . 82
BArrayDenseCol_const< Cell_Type, Data_Type > . . . 84
BArrayDenseRow< Cell_Type, Data_Type > . . . 86
BArrayDenseRow_const< Cell_Type, Data_Type > . . . 89
BArrayRow< Cell_Type, Data_Type > . . . 91
BArrayRow_const< Cell_Type, Data_Type > . . . 93
BArrayVector< Cell_Type, Data_Type >
: Row or column of a `BArray` . . . 95
BArrayVector_const< Cell_Type, Data_Type > . . . 99
Cell< Cell_Type >
: Entries in BArray. For now, it only has two members: . . . 103
Cell_const< Cell_Type > . . . 108
ConstBArrayRowIter< Cell_Type, Data_Type > . . . 108
Counter< Array_Type, Data_Type >
: A counter function based on change statistics . . . 110
Counters< Array_Type, Data_Type >
: Vector of counters . . . 116
DEFM . . . 120
DEFMCounterData
: Data class used to store arbitrary uint or double vectors . . . 125
DEFMData . . . 127
DEFMRuleData . . . 132
DEFMRuleDynData . . . 135
Entries< Cell_Type >
: A wrapper class to store `source`, `target`, `val` from a `BArray` object . . . 136
Flock
: A Flock is a group of Geese . . . 139

FreqTable< T >
Frequency table of vectors . . . 145
Geese
Annotated Phylo Model . . . 149
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: . . . 162
NetCounterData
Data class used to store arbitrary uint or double vectors . . . 177
NetworkData
Data class for Networks . . . 178
Node
A single node for the model . . . 180
NodeData
Data definition for the PhyloArray class . . . 186
PhyloCounterData . . . 187
PhyloRuleDynData . . . 190
PowerSet< Array_Type, Data_Rule_Type >
Powerset of a binary array . . . 192
Progress
A simple progress bar . . . 199
Rule< Array_Type, Data_Type >
Rule for determining if a cell should be included in a sequence . . . 200
Rules< Array_Type, Data_Type >
Vector of objects of class Rule . . . 203
StatsCounter< Array_Type, Data_Type >
Count stats for a single Array . . . 207
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >
Compute the support of sufficient statistics . . . 211
vecHasher< T > . . . 221

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

include/barry/barray-bones.hpp . . . 223
include/barry/barray-iterator.hpp . . . 223
include/barry/barray-meat-operators.hpp . . . 224
include/barry/barray-meat.hpp . . . 228
include/barry/barraycell-bones.hpp . . . 246
include/barry/barraycell-meat.hpp . . . 246
include/barry/barraydense-bones.hpp . . . 247
include/barry/barraydense-meat-operators.hpp . . . 247
include/barry/barraydense-meat.hpp . . . 250
include/barry/barraydensecell-bones.hpp . . . 271
include/barry/barraydensecell-meat.hpp . . . 272
include/barry/barraydensecol-bones.hpp . . . 272
include/barry/barraydenserow-bones.hpp . . . 274
include/barry/barrayrow-bones.hpp . . . 275
include/barry/barrayrow-meat.hpp . . . 275
include/barry/barrayvector-bones.hpp . . . 277
include/barry/barrayvector-meat.hpp . . . 278
include/barry/barry-configuration.hpp . . . 278
include/barry/barry-debug.hpp . . . 280
include/barry/barry-macros.hpp . . . 281
include/barry/barry.hpp . . . 282
include/barry/cell-bones.hpp . . . 285
include/barry/cell-meat.hpp . . . 286
include/barry/col-bones.hpp . . . 286
include/barry/counters-bones.hpp . . . 286
include/barry/counters-meat.hpp . . . 287
include/barry/freqtable.hpp . . . 327
include/barry/model-bones.hpp . . . 328
include/barry/model-meat.hpp . . . 328
include/barry/powerset-bones.hpp . . . 341
include/barry/powerset-meat.hpp . . . 342
include/barry/progress.hpp . . . 342
include/barry/rules-bones.hpp . . . 343
include/barry/rules-meat.hpp . . . 344
include/barry/statscounter-bones.hpp . . . 344

include/barry/statscounter-meat.hpp . . . 345
include/barry/support-bones.hpp . . . 350
include/barry/support-meat.hpp . . . 351
include/barry/typedefs.hpp . . . 362
include/barry/counters/defm-formula.hpp . . . 299
include/barry/counters/defm.hpp . . . 301
include/barry/counters/network-css.hpp . . . 304
include/barry/counters/network.hpp . . . 312
include/barry/counters/phylo.hpp . . . 319
include/barry/models/defm.hpp . . . 304
include/barry/models/geese.hpp . . . 333
include/barry/models/defm/defm-bones.hpp . . . 331
include/barry/models/defm/defm-meat.hpp . . . 331
include/barry/models/geese/flock-bones.hpp . . . 333
include/barry/models/geese/flock-meat.hpp . . . 334
include/barry/models/geese/geese-bones.hpp . . . 334
include/barry/models/geese/geese-meat-constructors.hpp . . . 336
include/barry/models/geese/geese-meat-likelihood.hpp . . . 337
include/barry/models/geese/geese-meat-likelihood_exhaust.hpp . . . 338
include/barry/models/geese/geese-meat-predict.hpp . . . 338
include/barry/models/geese/geese-meat-predict_exhaust.hpp . . . 339
include/barry/models/geese/geese-meat-predict_sim.hpp . . . 339
include/barry/models/geese/geese-meat-simulate.hpp . . . 340
include/barry/models/geese/geese-meat.hpp . . . 340
include/barry/models/geese/geese-node-bones.hpp . . . 341

Chapter 6

Module Documentation

6.1 Counting

Classes

- class NetworkData

  Data class for Networks.
- class NodeData

  Data definition for the `PhyloArray` *class.*
- class Counter< Array_Type, Data_Type >

  A counter function based on change statistics.

6.1.1 Detailed Description

`barry` includes a flexible way to generate counters based on change statistics. Since most of the time we are counting many motifs in a graph, change statistics make a reasonable (and efficient) way to make such counts.

In particular, let the motif be defined as $s(y)$, with y as the binary array. The change statistic when adding cell y_{ij}, i.e. when the cell moves from being 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, uint pos, uint lb, uint ub, unsigned int duplication=DEFAULT_DUPLICATIO

  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

Macros for defining counters

- #define DEFM_COUNTER(a) inline double (a) (const DEFMArray & Array, uint i, uint j, DEFMCounterData & data)
- #define DEFM_COUNTER_LAMBDA(a)

Macros for defining rules

- #define DEFM_RULE(a) inline bool (a) (const DEFMArray & Array, uint i, uint 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

```
#define DEFM_COUNTER(
              a ) inline double (a) (const DEFMArray & Array, uint i, uint j, DEFMCounterData
& data)
```

Function for definition of a network counter function

Definition at line 212 of file defm.hpp.

6.3.2.2 DEFM_COUNTER_LAMBDA

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

Value:

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

Lambda function for definition of a network counter function

Definition at line 216 of file defm.hpp.

6.3.2.3 DEFM_RULE

```
#define DEFM_RULE(
              a ) inline bool (a) (const DEFMArray & Array, uint i, uint j, bool & data)
```

Function for definition of a network counter function

Definition at line 226 of file defm.hpp.

6.3.2.4 DEFM_RULE_LAMBDA

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

Value:

```
Rule_fun_type<DEFMArray, DEFMRuleData> a = \
[](const DEFMArray & Array, uint i, uint j, DEFMRuleData & data) -> bool
```

Lambda function for definition of a network counter function

Definition at line 230 of file defm.hpp.

6.3.2.5 DEFM_RULEDYN_LAMBDA

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

Value:

```
Rule_fun_type<DEFMArray, DEFMRuleDynData> a = \
[](const DEFMArray & Array, uint i, uint j, DEFMRuleDynData & data) -> bool
```

Lambda function for definition of a network counter function

Definition at line 236 of file defm.hpp.

6.3.2.6 MAKE_DEFM_HASHER

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

Value:

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

Definition at line 195 of file defm.hpp.

6.3.3 Typedef Documentation

6.3.3.1 DEFMCounter

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

Definition at line 152 of file defm.hpp.

6.3.3.2 DEFMCounters

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

Definition at line 153 of file defm.hpp.

6.3.3.3 DEFMModel

```
typedef Model<DEFMArray, DEFMCounterData,DEFMRuleData,DEFMRuleDynData> DEFMModel
```

Definition at line 156 of file defm.hpp.

6.3.3.4 DEFMRule

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

Definition at line 159 of file defm.hpp.

6.3.3.5 DEFMRuleDyn

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

Definition at line 161 of file defm.hpp.

6.3.3.6 DEFMRules

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

Definition at line 160 of file defm.hpp.

6.3.3.7 DEFMRulesDyn

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

Definition at line 162 of file defm.hpp.

6.3.3.8 DEFMStatsCounter

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

Definition at line 155 of file defm.hpp.

6.3.3.9 DEFMSupport

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

Definition at line 154 of file defm.hpp.

6.3.4 Function Documentation

6.3.4.1 ncol()

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

Definition at line 173 of file defm.hpp.

6.3.4.2 nrow()

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

Definition at line 177 of file defm.hpp.

6.3.4.3 operator()()

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

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

Parameters

i	
j	

Returns

double

Definition at line 168 of file defm.hpp.

6.3.4.4 print()

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

Definition at line 181 of file defm.hpp.

6.3.4.5 rule_dyn_limit_changes()

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

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, uint 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, uint attr_id, double alpha=1.0, double tail_head=true)

  Sum of attribute difference between ego and alter to pow(alpha)

- NETWORK_COUNTER (init_single_attr)
- template<typename Tnet = Network>
  void counter_nodeicov (NetCounters< Tnet > *counters, uint attr_id)
- template<typename Tnet = Network>
  void counter_nodeocov (NetCounters< Tnet > *counters, uint attr_id)
- template<typename Tnet = Network>
  void counter_nodecov (NetCounters< Tnet > *counters, uint attr_id)
- template<typename Tnet = Network>
  void counter_nodematch (NetCounters< Tnet > *counters, uint attr_id)
- template<typename Tnet = Network>
  void counter_idegree (NetCounters< Tnet > *counters, std::vector< uint > d)

  Counts number of vertices with a given in-degree.
- template<> void counter_idegree (NetCounters< NetworkDense > *counters, std::vector< uint > d)
- template<typename Tnet = Network>
  void counter_odegree (NetCounters< Tnet > *counters, std::vector< uint > d)

  Counts number of vertices with a given out-degree.
- template<> void counter_odegree (NetCounters< NetworkDense > *counters, std::vector< uint > d)
- template<typename Tnet = Network>
  void counter_degree (NetCounters< Tnet > *counters, std::vector< uint > 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()

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

Sum of absolute attribute difference between ego and alter.

Definition at line 910 of file network.hpp.

6.4.2.2 counter_ctriads() [1/2]

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

Definition at line 665 of file network.hpp.

6.4.2.3 counter_ctriads() [2/2]

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

Definition at line 610 of file network.hpp.

6.4.2.4 counter_degree()

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

Counts number of vertices with a given out-degree.

Definition at line 1328 of file network.hpp.

6.4.2.5 counter_density()

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

Definition at line 731 of file network.hpp.

6.4.2.6 counter_diff()

```
template<typename Tnet = Network>
void counter_diff (
            NetCounters< Tnet > * counters,
            uint attr_id,
            double alpha = 1.0,
            double tail_head = true )  [inline]
```

Sum of attribute difference between ego and alter to pow(alpha)

Definition at line 955 of file network.hpp.

6.4.2.7 counter_edges()

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

Number of edges.

Definition at line 152 of file network.hpp.

6.4.2.8 counter_fixed_effect()

```
void counter_fixed_effect (
            DEFMCounters * counters,
            int covar_index,
            double k,
            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 780 of file defm.hpp.

6.4.2.9 counter_idegree() [1/2]

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

Definition at line 1172 of file network.hpp.

6.4.2.10 counter_idegree() [2/2]

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

Counts number of vertices with a given in-degree.

Definition at line 1125 of file network.hpp.

6.4.2.11 counter_idegree15() [1/2]

```
template<>
void counter_idegree15 (
            NetCounters< NetworkDense > * counters )  [inline]
```

Definition at line 787 of file network.hpp.

6.4.2.12 counter_idegree15() [2/2]

```
template<typename Tnet = Network>
void counter_idegree15 (
            NetCounters< Tnet > * counters )  [inline]
```

Definition at line 759 of file network.hpp.

6.4.2.13 counter_isolates() [1/2]

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

Definition at line 215 of file network.hpp.

6.4.2.14 counter_isolates() [2/2]

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

Number of isolated vertices.

Definition at line 175 of file network.hpp.

6.4.2.15 counter_istar2() [1/2]

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

Definition at line 338 of file network.hpp.

6.4.2.16 counter_istar2() [2/2]

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

Definition at line 312 of file network.hpp.

6.4.2.17 counter_logit_intercept()

```
void counter_logit_intercept (
            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 321 of file defm.hpp.

6.4.2.18 counter_mutual()

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

Number of mutual ties.

Definition at line 256 of file network.hpp.

6.4.2.19 counter_nodecov()

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

Definition at line 1068 of file network.hpp.

6.4.2.20 counter_nodeicov()

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

Definition at line 1018 of file network.hpp.

6.4.2.21 counter_nodematch()

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

Definition at line 1093 of file network.hpp.

6.4.2.22 counter_nodeocov()

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

Definition at line 1043 of file network.hpp.

6.4.2.23 counter_odegree() [1/2]

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

Definition at line 1273 of file network.hpp.

6.4.2.24 counter_odegree() [2/2]

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

Counts number of vertices with a given out-degree.

Definition at line 1225 of file network.hpp.

6.4.2.25 counter_odegree15() [1/2]

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

Definition at line 864 of file network.hpp.

6.4.2.26 counter_odegree15() [2/2]

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

Definition at line 836 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 254 of file defm.hpp.

6.4.2.28 counter_ostar2() [1/2]

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

Definition at line 404 of file network.hpp.

6.4.2.29 counter_ostar2() [2/2]

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

Definition at line 376 of file network.hpp.

6.4.2.30 counter_transition()

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

Prevalence of ones.

Parameters

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

Definition at line 440 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 749 of file defm.hpp.

6.4.2.32 counter_ttriads() [1/2]

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

Definition at line 531 of file network.hpp.

6.4.2.33 counter_ttriads() [2/2]

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

Definition at line 441 of file network.hpp.

6.4.2.34 NETWORK_COUNTER()

```
NETWORK_COUNTER (
            init_single_attr  )
```

Definition at line 999 of file network.hpp.

6.4.2.35 rules_dont_become_zero()

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

Blocks switching a one to zero.

Parameters

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

Definition at line 848 of file defm.hpp.

6.4.2.36 rules_markov_fixed()

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

Number of edges.

Definition at line 823 of file defm.hpp.

6.5 Phylo counters

Counters for phylogenetic modeling.

Functions

- void counter_overall_gains (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  Overall functional gains.
- void counter_gains (PhyloCounters *counters, std::vector< uint > nfun, unsigned int duplication=DEFAULT_DUPLICATION)

  Functional gains for a specific function (`nfun`).
- void counter_gains_k_offspring (PhyloCounters *counters, std::vector< uint > nfun, uint k=1u, unsigned int duplication=DEFAULT_DUPLICATION)

  k genes gain function nfun
- void counter_genes_changing (PhyloCounters *counters, unsigned int 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, unsigned int nfunA, unsigned int nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  Keeps track of how many pairs of genes preserve pseudostate.
- void counter_prop_genes_changing (PhyloCounters *counters, unsigned int 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, unsigned int duplication=DEFAULT_DUPLICATION)

  Overall functional loss.

- void counter_maxfuns (PhyloCounters *counters, uint lb, uint ub, unsigned int duplication=DEFAULT_DUPLICATION)

  Cap the number of functions per gene.
- void counter_loss (PhyloCounters *counters, std::vector< uint > nfun, unsigned int duplication=DEFAULT_DUPLICATION)

  Total count of losses for an specific function.
- void counter_overall_changes (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  Total number of changes. Use this statistic to account for "preservation".
- void counter_subfun (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  Total count of Sub-functionalization events.
- void counter_cogain (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  Co-evolution (joint gain or loss)
- void counter_longest (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  Longest branch mutates (either by gain or by loss)
- void counter_neofun (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  Total number of neofunctionalization events.
- void counter_pairwise_neofun_singlefun (PhyloCounters *counters, uint nfunA, unsigned int duplication=DEFAULT_DUPLICATI

  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, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  Total number of neofunctionalization events.
- void counter_co_opt (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  Function co-opting.
- void counter_k_genes_changing (PhyloCounters *counters, unsigned int k, unsigned int 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, unsigned int duplication=DEFAULT_DUPLICATION)

  Indicator function. Equals to one if k genes changed and zero otherwise.
- void counter_gains_from_0 (PhyloCounters *counters, std::vector< uint > nfun, unsigned int duplication=DEFAULT_DUPLICAT

  Used when all the functions are in 0 (like the root node prob.)
- void counter_overall_gains_from_0 (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  Used when all the functions are in 0 (like the root node prob.)
- void counter_pairwise_overall_change (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  Used when all the functions are in 0 (like the root node prob.)
- void counter_pairwise_preserving (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  Used when all the functions are in 0 (like the root node prob.)
- void counter_pairwise_first_gain (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int 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

6.5.2.1 counter_co_opt()

```
void counter_co_opt (
            PhyloCounters * counters,
            uint nfunA,
            uint nfunB,
            unsigned int duplication = DEFAULT_DUPLICATION )  [inline]
```

Function co-opting.

Function co-opting of functions A and B happens when, for example, function B is gained as a new featured leveraging what function A already does; without losing function A. The sufficient statistic is defined as follows:

$$x_{pa}(1-x_{pb})\sum_{i<j}\left[x^p_{ia}(1-x^p_{ib})x^p_{ja}x^p_{jb}+x^p_{ja}(1-x^p_{jb})x^p_{ia}x^p_{ib}\right]$$

This algorithm implements the change statistic.

Definition at line 1393 of file phylo.hpp.

6.5.2.2 counter_cogain()

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

Co-evolution (joint gain or loss)

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

Definition at line 888 of file phylo.hpp.

6.5.2.3 counter_gains()

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

Functional gains for a specific function (`nfun`).

Definition at line 193 of file phylo.hpp.

6.5.2.4 counter_gains_from_0()

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

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

Needs to specify function a.

Definition at line 1727 of file phylo.hpp.

6.5.2.5 counter_gains_k_offspring()

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

k genes gain function nfun

Definition at line 253 of file phylo.hpp.

6.5.2.6 counter_genes_changing()

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

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

Definition at line 325 of file phylo.hpp.

6.5.2.7 counter_k_genes_changing()

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

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

Definition at line 1491 of file phylo.hpp.

6.5.2.8 counter_less_than_p_prop_genes_changing()

```
void counter_less_than_p_prop_genes_changing (
            PhyloCounters * counters,
            double p,
            unsigned int duplication = DEFAULT_DUPLICATION )  [inline]
```

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

$<$ How many genes diverge the parent

Definition at line 1611 of file phylo.hpp.

6.5.2.9 counter_longest()

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

Longest branch mutates (either by gain or by loss)

Definition at line 945 of file phylo.hpp.

6.5.2.10 counter_loss()

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

Total count of losses for an specific function.

Definition at line 688 of file phylo.hpp.

6.5.2.11 counter_maxfuns()

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

Cap the number of functions per gene.

Definition at line 626 of file phylo.hpp.

6.5.2.12 counter_neofun()

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

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 1115 of file phylo.hpp.

6.5.2.13 counter_neofun_a2b()

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

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 1260 of file phylo.hpp.

6.5.2.14 counter_overall_changes()

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

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

Definition at line 740 of file phylo.hpp.

6.5.2.15 counter_overall_gains()

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

Overall functional gains.

Total number of gains (irrespective of the function).

Definition at line 155 of file phylo.hpp.

6.5.2.16 counter_overall_gains_from_0()

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

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

Needs to specify function a.

Definition at line 1793 of file phylo.hpp.

6.5.2.17 counter_overall_loss()

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

Overall functional loss.

Definition at line 578 of file phylo.hpp.

6.5.2.18 counter_pairwise_first_gain()

```
void counter_pairwise_first_gain (
            PhyloCounters * counters,
            uint nfunA,
            uint nfunB,
            unsigned int duplication = DEFAULT_DUPLICATION )  [inline]
```

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

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

Definition at line 2045 of file phylo.hpp.

6.5.2.19 counter_pairwise_neofun_singlefun()

```
void counter_pairwise_neofun_singlefun (
            PhyloCounters * counters,
            uint nfunA,
            unsigned int duplication = DEFAULT_DUPLICATION )  [inline]
```

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

Definition at line 1196 of file phylo.hpp.

6.5.2.20 counter_pairwise_overall_change()

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

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

Needs to specify function a.

Definition at line 1841 of file phylo.hpp.

6.5.2.21 counter_pairwise_preserving()

```
void counter_pairwise_preserving (
            PhyloCounters * counters,
            uint nfunA,
            uint nfunB,
            unsigned int duplication = DEFAULT_DUPLICATION )  [inline]
```

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

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

Definition at line 1906 of file phylo.hpp.

6.5.2.22 counter_preserve_pseudogene()

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

Keeps track of how many pairs of genes preserve pseudostate.

Definition at line 394 of file phylo.hpp.

6.5.2.23 counter_prop_genes_changing()

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

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

Definition at line 476 of file phylo.hpp.

6.5.2.24 counter_subfun()

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

Total count of Sub-functionalization events.

It requires to specify data = {funA, funB}

Definition at line 799 of file phylo.hpp.

Chapter 7

Namespace Documentation

7.1 barry Namespace Reference

barry: Your go-to motif accountant

Namespaces

- counters

  Tree class and TreeIterator class.

7.1.1 Detailed Description

barry: Your go-to motif accountant

7.2 barry::counters Namespace Reference

Tree class and TreeIterator class.

Namespaces

- defm
- network
- phylo

7.2.1 Detailed Description

Tree class and TreeIterator 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 28 of file typedefs.hpp.

7.6.2.2 NONE

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

Definition at line 29 of file typedefs.hpp.

7.6.2.3 ONE

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

Definition at line 30 of file typedefs.hpp.

7.6.2.4 TWO

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

Definition at line 31 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 46 of file typedefs.hpp.

7.7.2.2 AS_ZERO

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

Definition at line 45 of file typedefs.hpp.

7.7.2.3 BOTH

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

Definition at line 39 of file typedefs.hpp.

7.7.2.4 NONE

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

Definition at line 40 of file typedefs.hpp.

7.7.2.5 ONE

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

Definition at line 41 of file typedefs.hpp.

7.7.2.6 TWO

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

Definition at line 42 of file typedefs.hpp.

7.7.2.7 UKNOWN

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

Definition at line 44 of file typedefs.hpp.

Chapter 8

Class Documentation

8.1 BArray< Cell_Type, Data_Type > Class Template Reference

Baseline class for binary arrays.

`#include <barray-bones.hpp>`

Public Member Functions

- bool operator== (const BArray< Cell_Type, Data_Type > &Array_)
- ~BArray ()
- void out_of_range (uint i, uint j) const
- Cell_Type get_cell (uint i, uint j, bool check_bounds=true) const
- std::vector< Cell_Type > get_col_vec (uint i, bool check_bounds=true) const
- std::vector< Cell_Type > get_row_vec (uint i, bool check_bounds=true) const
- void get_col_vec (std::vector< Cell_Type > *x, uint i, bool check_bounds=true) const
- void get_row_vec (std::vector< Cell_Type > *x, uint i, bool check_bounds=true) const
- const Row_type< Cell_Type > & row (uint i, bool check_bounds=true) const
- const Col_type< Cell_Type > & col (uint i, bool check_bounds=true) const
- Entries< Cell_Type > get_entries () const

  Get the edgelist.
- void transpose ()
- void clear (bool hard=true)
- void resize (uint N_, uint M_)
- void reserve ()
- void print (const char *fmt=nullptr,...) const
- 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 unsigned int vector ranging from 0 to M_*
target	*When `true` tries to add repeated observations.*

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

- void set_data (Data_Type *data_, bool delete_data_=false)
  Set the data object.
- Data_Type * D_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 (uint i, uint j, bool check_bounds=true) const
- uint nrow () const noexcept
- uint ncol () const noexcept
- uint nnozero () const noexcept
- Cell< Cell_Type > default_val () const

Cell-wise insertion/deletion

Parameters

i,j	*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.*

- BArray< Cell_Type, Data_Type > & operator+= (const std::pair< uint, uint > &coords)
- BArray< Cell_Type, Data_Type > & operator-= (const std::pair< uint, uint > &coords)
- BArrayCell< Cell_Type, Data_Type > operator() (uint i, uint j, bool check_bounds=true)
- const Cell_Type operator() (uint i, uint j, bool check_bounds=true) const
- void rm_cell (uint i, uint j, bool check_bounds=true, bool check_exists=true)
- void insert_cell (uint i, uint j, const Cell< Cell_Type > &v, bool check_bounds, bool check_exists)
- void insert_cell (uint i, uint j, Cell< Cell_Type > &&v, bool check_bounds, bool check_exists)
- void insert_cell (uint i, uint j, Cell_Type v, bool check_bounds, bool check_exists)
- void swap_cells (uint i0, uint j0, uint i1, uint j1, bool check_bounds=true, int check_exists=CHECK::BOTH, int *report=nullptr)
- void toggle_cell (uint i, uint j, bool check_bounds=true, int check_exists=EXISTS::UKNOWN)
- void toggle_lock (uint i, uint j, bool check_bounds=true)

Column/row wise interchange

- void swap_rows (uint i0, uint i1, bool check_bounds=true)
- void swap_cols (uint j0, uint j1, bool check_bounds=true)
- void zero_row (uint i, bool check_bounds=true)
- void zero_col (uint j, bool check_bounds=true)

Arithmetic operators

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

Public Attributes

- bool visited = false

Friends

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

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_map`s, i.e. `std::vector< std::unordered_map<unsigned int,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]

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

Empty array.

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

8.1.2.3 BArray() [3/6]

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

Edgelist with data.

8.1.2.4 BArray() [4/6]

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

Edgelist with no data (simpler)

8.1.2.5 BArray() [5/6]

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

Copy constructor.

8.1.2.6 BArray() [6/6]

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

Move operator.

8.1.2.7 ~BArray()

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

8.1.3 Member Function Documentation

8.1.3.1 clear()

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

8.1.3.2 col()

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

8.1.3.3 D() [1/2]

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

8.1.3.4 D() [2/2]

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

8.1.3.5 D_ptr() [1/2]

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

8.1.3.6 D_ptr() [2/2]

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

8.1.3.7 default_val()

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

8.1.3.8 flush_data()

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

8.1.3.9 get_cell()

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

8.1.3.10 get_col_vec() [1/2]

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

8.1.3.11 get_col_vec() [2/2]

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

8.1.3.12 get_entries()

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

Get the edgelist.

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

Returns

Entries<Cell_Type>

8.1.3.13 get_row_vec() [1/2]

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

8.1.3.14 get_row_vec() [2/2]

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

8.1.3.15 insert_cell() [1/3]

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

8.1.3.16 insert_cell() [2/3]

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

8.1.3.17 insert_cell() [3/3]

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

8.1.3.18 is_dense()

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

Definition at line 240 of file barray-bones.hpp.

8.1.3.19 is_empty()

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

8.1.3.20 ncol()

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

8.1.3.21 nnozero()

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

8.1.3.22 nrow()

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

8.1.3.23 operator()() [1/2]

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

8.1.3.24 operator()() [2/2]

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

8.1.3.25 operator*=()

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

8.1.3.26 operator+=() [1/3]

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

8.1.3.27 operator+=() [2/3]

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

8.1.3.28 operator+=() [3/3]

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

8.1.3.29 operator-=() [1/3]

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

8.1.3.30 operator-=() [2/3]

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

8.1.3.31 operator-=() [3/3]

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

8.1.3.32 operator/=()

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

8.1.3.33 operator=() [1/2]

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

Move assignment.

8.1.3.34 operator=() [2/2]

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

Assignment constructor.

8.1.3.35 operator==()

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

8.1.3.36 out_of_range()

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

8.1.3.37 print()

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

8.1.3.38 reserve()

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

8.1.3.39 resize()

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

8.1.3.40 rm_cell()

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

8.1.3.41 row()

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

8.1.3.42 set_data()

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

Set the data object.

Parameters

*data_*	
*delete_data_*	

8.1.3.43 swap_cells()

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

8.1.3.44 swap_cols()

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

8.1.3.45 swap_rows()

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

8.1.3.46 toggle_cell()

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

8.1.3.47 toggle_lock()

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

8.1.3.48 transpose()

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

8.1.3.49 zero_col()

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

8.1.3.50 zero_row()

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

8.1.4 Friends And Related Function Documentation

8.1.4.1 BArrayCell< Cell_Type, Data_Type >

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

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

8.1.4.2 BArrayCell_const< Cell_Type, Data_Type >

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

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

8.1.5 Member Data Documentation

8.1.5.1 visited

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

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

Definition at line 54 of file barray-bones.hpp.

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

- include/barry/barray-bones.hpp

8.2 BArrayCell< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

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

8.2.1 Detailed Description

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

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

8.2.2 Constructor & Destructor Documentation

8.2.2.1 BArrayCell()

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

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

8.2.2.2 ~BArrayCell()

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

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

8.2.3 Member Function Documentation

8.2.3.1 operator Cell_Type()

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

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

8.2.3.2 operator*=()

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

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

8.2.3.3 operator+=()

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

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

8.2.3.4 operator-=()

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

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

8.2.3.5 operator/=()

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

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

8.2.3.6 operator=()

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

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

8.2.3.7 operator==()

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

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

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

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

8.3 BArrayCell_const< Cell_Type, Data_Type > Class Template Reference

`#include <barraycell-bones.hpp>`

Public Member Functions

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

8.3.1 Detailed Description

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

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

8.3.2 Constructor & Destructor Documentation

8.3.2.1 BArrayCell_const()

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

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

8.3.2.2 ~BArrayCell_const()

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

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

8.3.3 Member Function Documentation

8.3.3.1 operator Cell_Type()

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

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

8.3.3.2 operator"!=()

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

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

8.3.3.3 operator<()

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

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

8.3.3.4 operator<=()

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

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

8.3.3.5 operator==()

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

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

8.3.3.6 operator>()

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

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

8.3.3.7 operator>=()

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

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

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

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

8.4 BArrayDense< Cell_Type, Data_Type > Class Template Reference

Baseline class for binary arrays.

`#include <barraydense-bones.hpp>`

Public Member Functions

- bool operator== (const BArrayDense< Cell_Type, Data_Type > &Array_)
- ~BArrayDense ()
- void out_of_range (uint i, uint j) const
- Cell_Type get_cell (uint i, uint j, bool check_bounds=true) const
- std::vector< Cell_Type > get_col_vec (uint i, bool check_bounds=true) const
- std::vector< Cell_Type > get_row_vec (uint i, bool check_bounds=true) const
- void get_col_vec (std::vector< Cell_Type > *x, uint i, bool check_bounds=true) const
- void get_row_vec (std::vector< Cell_Type > *x, uint i, bool check_bounds=true) const
- BArrayDenseRow< Cell_Type, Data_Type > & row (uint i, bool check_bounds=true)
- const BArrayDenseRow_const< Cell_Type, Data_Type > row (uint i, bool check_bounds=true) const
- BArrayDenseCol< Cell_Type, Data_Type > & col (uint j, bool check_bounds=true)
- const BArrayDenseCol_const< Cell_Type, Data_Type > col (uint j, bool check_bounds=true) const
- Entries< Cell_Type > get_entries () const

  Get the edgelist.
- void transpose ()
- void clear (bool hard=true)
- void resize (uint N_, uint M_)
- void reserve ()
- void print (const char *fmt=nullptr,...) const
- bool is_dense () const noexcept
- const std::vector< Cell_Type > & get_data () const
- const Cell_Type rowsum (unsigned int i) const
- const Cell_Type colsum (unsigned int i) const

Constructors

Parameters

N_	*Number of rows*
M_	*Number of columns*
source	*An unsigned vector ranging from 0 to N_*
target	*An unsigned int vector ranging from 0 to M_*
target	*When `true` tries to add repeated observations.*
value	*Cell_Type defaul fill-in value (zero, by default.)*

- BArrayDense ()
  Zero-size array.
- BArrayDense (uint N_, uint M_, Cell_Type value=static_cast< Cell_Type >(0))
  Empty array.
- BArrayDense (uint N_, uint M_, const std::vector< uint > &source, const std::vector< uint > &target, const std::vector< Cell_Type > &value, bool add=true)
  Edgelist with data.
- BArrayDense (uint N_, uint M_, const std::vector< uint > &source, const std::vector< uint > &target, bool add=true)
  Edgelist with no data (simpler)
- BArrayDense (const BArrayDense< Cell_Type, Data_Type > &Array_, bool copy_data=false)
  Copy constructor.
- BArrayDense< Cell_Type, Data_Type > & operator= (const BArrayDense< Cell_Type, Data_Type > &Array_)
  Assignment constructor.
- BArrayDense (BArrayDense< Cell_Type, Data_Type > &&x) noexcept
  Move operator.
- BArrayDense< Cell_Type, Data_Type > & operator= (BArrayDense< Cell_Type, Data_Type > &&x) noexcept
  Move assignment.

- void set_data (Data_Type *data_, bool delete_data_=false)
  Set the data object.
- Data_Type * D_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 (uint i, uint j, bool check_bounds=true) const
- uint nrow () const noexcept
- uint ncol () const noexcept
- uint nnozero () const noexcept

- Cell< Cell_Type > default_val () const

Cell-wise insertion/deletion

Parameters

i,j	*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< uint, uint > &coords)
- BArrayDense< Cell_Type, Data_Type > & operator-= (const std::pair< uint, uint > &coords)
- BArrayDenseCell< Cell_Type, Data_Type > operator() (uint i, uint j, bool check_bounds=true)
- const Cell_Type operator() (uint i, uint j, bool check_bounds=true) const
- void rm_cell (uint i, uint j, bool check_bounds=true, bool check_exists=true)
- void insert_cell (uint i, uint j, const Cell< Cell_Type > &v, bool check_bounds, bool check_exists)
- void insert_cell (uint i, uint j, Cell_Type v, bool check_bounds, bool check_exists)
- void swap_cells (uint i0, uint j0, uint i1, uint j1, bool check_bounds=true, int check_exists=CHECK::BOTH, int *report=nullptr)
- void toggle_cell (uint i, uint j, bool check_bounds=true, int check_exists=EXISTS::UKNOWN)
- void toggle_lock (uint i, uint j, bool check_bounds=true)

Column/row wise interchange

- void swap_rows (uint i0, uint i1, bool check_bounds=true)
- void swap_cols (uint j0, uint j1, bool check_bounds=true)
- void zero_row (uint i, bool check_bounds=true)
- void zero_col (uint j, bool check_bounds=true)

Arithmetic operators

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

Public Attributes

- bool visited = false

Friends

- class 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]

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

Empty array.

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

8.4.2.3 BArrayDense() [3/6]

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

Edgelist with data.

8.4.2.4 BArrayDense() [4/6]

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

Edgelist with no data (simpler)

8.4.2.5 BArrayDense() [5/6]

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

Copy constructor.

8.4.2.6 BArrayDense() [6/6]

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

Move operator.

8.4.2.7 ~BArrayDense()

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

8.4.3 Member Function Documentation

8.4.3.1 clear()

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

8.4.3.2 col() [1/2]

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

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

8.4.3.3 col() [2/2]

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

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

8.4.3.4 colsum()

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

8.4.3.5 D() [1/2]

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

8.4.3.6 D() [2/2]

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

8.4.3.7 D_ptr() [1/2]

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

8.4.3.8 D_ptr() [2/2]

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

8.4.3.9 default_val()

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

8.4.3.10 get_cell()

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

8.4.3.11 get_col_vec() [1/2]

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

8.4.3.12 get_col_vec() [2/2]

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

8.4.3.13 get_data()

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

8.4.3.14 get_entries()

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

Get the edgelist.

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

Returns

Entries<Cell_Type>

8.4.3.15 get_row_vec() [1/2]

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

8.4.3.16 get_row_vec() [2/2]

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

8.4.3.17 insert_cell() [1/2]

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

8.4.3.18 insert_cell() [2/2]

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

8.4.3.19 is_dense()

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

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

8.4.3.20 is_empty()

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

8.4.3.21 ncol()

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

8.4.3.22 nnozero()

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

8.4.3.23 nrow()

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

8.4.3.24 operator()() [1/2]

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

8.4.3.25 operator()() [2/2]

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

8.4.3.26 operator*=()

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

8.4.3.27 operator+=() [1/3]

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

8.4.3.28 operator+=() [2/3]

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

8.4.3.29 operator+=() [3/3]

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

8.4.3.30 operator-=() [1/3]

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

8.4.3.31 operator-=() [2/3]

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

8.4.3.32 operator-=() [3/3]

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

8.4.3.33 operator/=()

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

8.4.3.34 operator=() [1/2]

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

Move assignment.

8.4.3.35 operator=() [2/2]

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

Assignment constructor.

8.4.3.36 operator==()

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

8.4.3.37 out_of_range()

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

8.4.3.38 print()

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

8.4.3.39 reserve()

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

8.4.3.40 resize()

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

8.4.3.41 rm_cell()

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

8.4.3.42 row() [1/2]

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

8.4.3.43 row() [2/2]

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

8.4.3.44 rowsum()

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

8.4.3.45 set_data()

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

Set the data object.

Parameters

*data_*	
*delete_↩ data_*	

8.4.3.46 swap_cells()

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

8.4.3.47 swap_cols()

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

8.4.3.48 swap_rows()

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

8.4.3.49 toggle_cell()

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

8.4.3.50 toggle_lock()

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

8.4.3.51 transpose()

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

8.4.3.52 zero_col()

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

8.4.3.53 zero_row()

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

8.4.4 Friends And Related Function Documentation

8.4.4.1 BArrayDenseCell< Cell_Type, Data_Type >

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

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

8.4.4.2 BArrayDenseCol< Cell_Type, Data_Type >

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

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

8.4.4.3 BArrayDenseCol_const< Cell_Type, Data_Type >

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

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

8.4.4.4 BArrayDenseRow< Cell_Type, Data_Type >

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

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

8.4.4.5 BArrayDenseRow_const< Cell_Type, Data_Type >

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

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

8.4.5 Member Data Documentation

8.4.5.1 visited

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

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

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

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

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

8.5 BArrayDenseCell< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

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

Friends

- class BArrayDense< Cell_Type, Data_Type >
- class BArrayDenseCol< Cell_Type, Data_Type >
- class BArrayDenseCol_const< Cell_Type, Data_Type >

8.5.1 Detailed Description

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

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

8.5.2 Constructor & Destructor Documentation

8.5.2.1 BArrayDenseCell()

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

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

8.5.2.2 ~BArrayDenseCell()

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

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

8.5.3 Member Function Documentation

8.5.3.1 operator Cell_Type()

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

Definition at line 72 of file barraydensecell-meat.hpp.

8.5.3.2 operator*=()

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

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

8.5.3.3 operator+=()

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

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

8.5.3.4 operator-=()

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

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

8.5.3.5 operator/=()

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

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

8.5.3.6 operator=() [1/2]

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

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

8.5.3.7 operator=() [2/2]

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

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

8.5.3.8 operator==()

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

Definition at line 77 of file barraydensecell-meat.hpp.

8.5.4 Friends And Related Function Documentation

8.5.4.1 BArrayDense< Cell_Type, Data_Type >

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

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

8.5.4.2 BArrayDenseCol< Cell_Type, Data_Type >

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

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

8.5.4.3 BArrayDenseCol_const< Cell_Type, Data_Type >

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

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

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

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

8.6 BArrayDenseCell_const< Cell_Type, Data_Type > Class Template Reference

8.6.1 Detailed Description

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

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

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

- include/barry/barraydense-bones.hpp

8.7 BArrayDenseCol< Cell_Type, Data_Type > Class Template Reference

`#include <barraydensecol-bones.hpp>`

Public Member Functions

- BArrayDenseCol (BArrayDense< Cell_Type, Data_Type > &array_, unsigned int j)
- Col_type< Cell_Type >::iterator & begin ()
- Col_type< Cell_Type >::iterator & end ()
- size_t size () const noexcept
- std::pair< unsigned int, Cell_Type * > & operator() (unsigned int i)

Friends

- class BArrayDense< Cell_Type, Data_Type >
- class BArrayDenseCell< Cell_Type, Data_Type >
- class BArrayDenseCell_const< Cell_Type, Data_Type >

8.7.1 Detailed Description

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

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

8.7.2 Constructor & Destructor Documentation

8.7.2.1 BArrayDenseCol()

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

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

8.7.3 Member Function Documentation

8.7.3.1 begin()

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

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

8.7.3.2 end()

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

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

8.7.3.3 operator()()

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

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

8.7.3.4 size()

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

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

8.7.4 Friends And Related Function Documentation

8.7.4.1 BArrayDense< Cell_Type, Data_Type >

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

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

8.7.4.2 BArrayDenseCell< Cell_Type, Data_Type >

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

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

8.7.4.3 BArrayDenseCell_const< Cell_Type, Data_Type >

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

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

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

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

8.8 BArrayDenseCol_const< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

- BArrayDenseCol_const (const BArrayDense< Cell_Type, Data_Type > &array_, unsigned int j)
- Col_type< Cell_Type >::iterator begin ()
- Col_type< Cell_Type >::iterator end ()
- size_t size () const noexcept
- const std::pair< unsigned int, Cell_Type * > operator() (unsigned int i) const

Friends

- class BArrayDenseCell< Cell_Type, Data_Type >
- class BArrayDenseCell_const< Cell_Type, Data_Type >

8.8.1 Detailed Description

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

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

8.8.2 Constructor & Destructor Documentation

8.8.2.1 BArrayDenseCol_const()

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

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

8.8.3 Member Function Documentation

8.8.3.1 begin()

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

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

8.8.3.2 end()

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

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

8.8.3.3 operator()()

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

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

8.8.3.4 size()

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

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

8.8.4 Friends And Related Function Documentation

8.8.4.1 BArrayDenseCell< Cell_Type, Data_Type >

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

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

8.8.4.2 BArrayDenseCell_const< Cell_Type, Data_Type >

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

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

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

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

8.9 BArrayDenseRow< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

- BArrayDenseRow (BArrayDense< Cell_Type, Data_Type > &array_, unsigned int i)
- Row_type< Cell_Type >::iterator & begin ()
- Row_type< Cell_Type >::iterator & end ()
- size_t size () const noexcept
- std::pair< unsigned int, Cell< Cell_Type > > & operator() (unsigned int i)

Friends

- class BArrayDense< Cell_Type, Data_Type >
- class BArrayDenseCell< Cell_Type, Data_Type >
- class BArrayDenseCell_const< Cell_Type, Data_Type >

8.9.1 Detailed Description

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

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

8.9.2 Constructor & Destructor Documentation

8.9.2.1 BArrayDenseRow()

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

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

8.9.3 Member Function Documentation

8.9.3.1 begin()

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

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

8.9.3.2 end()

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

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

8.9.3.3 operator()()

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

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

8.9.3.4 size()

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

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

8.9.4 Friends And Related Function Documentation

8.9.4.1 BArrayDense< Cell_Type, Data_Type >

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

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

8.9.4.2 BArrayDenseCell< Cell_Type, Data_Type >

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

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

8.9.4.3 BArrayDenseCell_const< Cell_Type, Data_Type >

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

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

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

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

8.10 BArrayDenseRow_const< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

- BArrayDenseRow_const (const BArrayDense< Cell_Type, Data_Type > &array_, unsigned int 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< unsigned int, Cell< Cell_Type > > operator() (unsigned int i) const

Friends

- class BArrayDenseCell< Cell_Type, Data_Type >
- class BArrayDenseCell_const< Cell_Type, Data_Type >

8.10.1 Detailed Description

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

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

8.10.2 Constructor & Destructor Documentation

8.10.2.1 BArrayDenseRow_const()

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

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

8.10.3 Member Function Documentation

8.10.3.1 begin()

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

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

8.10.3.2 end()

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

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

8.10.3.3 operator()()

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

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

8.10.3.4 size()

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

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

8.10.4 Friends And Related Function Documentation

8.10.4.1 BArrayDenseCell< Cell_Type, Data_Type >

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

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

8.10.4.2 BArrayDenseCell_const< Cell_Type, Data_Type >

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

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

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

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

8.11 BArrayRow< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

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

8.11.1 Detailed Description

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

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

8.11.2 Constructor & Destructor Documentation

8.11.2.1 BArrayRow()

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

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

8.11.2.2 ~BArrayRow()

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

Definition at line 26 of file 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*=()

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

8.11.3.3 operator+=()

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

8.11.3.4 operator-=()

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

8.11.3.5 operator/=()

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

8.11.3.6 operator=()

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

8.11.3.7 operator==()

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

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

- include/barry/barrayrow-bones.hpp

8.12 BArrayRow_const< Cell_Type, Data_Type > Class Template Reference

`#include <barrayrow-bones.hpp>`

Public Member Functions

- BArrayRow_const (const BArray< Cell_Type, Data_Type > *Array_, uint 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()

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

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"!=()

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

8.12.3.3 operator<()

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

8.12.3.4 operator<=()

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

8.12.3.5 operator==()

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

8.12.3.6 operator>()

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

8.12.3.7 operator>=()

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

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

- include/barry/barrayrow-bones.hpp

8.13 BArrayVector< Cell_Type, Data_Type > Class Template Reference

Row or column of a BArray

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

Public Member Functions

- BArrayVector (BArray< Cell_Type, Data_Type > *Array_, uint &dim_ uint &i_, bool check_bounds=true)
  Construct a new BArrayVector object.
- ~BArrayVector ()
- bool is_row () const noexcept
- bool is_col () const noexcept
- uint size () const noexcept
- std::vector< Cell_Type >::const_iterator begin () noexcept
- std::vector< Cell_Type >::const_iterator end () noexcept
- void operator= (const Cell_Type &val)
- void operator+= (const Cell_Type &val)
- void operator-= (const Cell_Type &val)
- void operator*= (const Cell_Type &val)
- void operator/= (const Cell_Type &val)
- operator std::vector< Cell_Type > () const
- bool operator== (const Cell_Type &val) const

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()

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

Construct a new BArrayVector object.

Parameters

*Array_*	Pointer to a `BArray` object
*dim_*	Dimension. 0 means row and 1 means column.
*i_*	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*=()

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

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

8.13.3.7 operator+=()

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

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

8.13.3.8 operator-=()

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

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

8.13.3.9 operator/=()

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

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

8.13.3.10 operator=()

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

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

8.13.3.11 operator==()

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

Definition at line 185 of file barrayvector-meat.hpp.

8.13.3.12 size()

```
template<typename Cell_Type , typename Data_Type >
uint 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_, uint &dim_ uint &i_, bool check_bounds=true)
- ~BArrayVector_const ()
- bool is_row () const noexcept
- bool is_col () const noexcept
- uint size () const noexcept
- std::vector< Cell_Type >::const_iterator begin () noexcept
- std::vector< Cell_Type >::const_iterator end () noexcept
- operator std::vector< Cell_Type > () const
- bool operator== (const Cell_Type &val) const
- bool operator!= (const Cell_Type &val) const
- bool operator< (const Cell_Type &val) const
- bool operator> (const Cell_Type &val) const
- bool operator<= (const Cell_Type &val) const
- bool operator>= (const Cell_Type &val) const

8.14.1 Detailed Description

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

Definition at line 73 of file barrayvector-bones.hpp.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 BArrayVector_const()

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

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

8.14.2.2 ~BArrayVector_const()

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

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

8.14.3 Member Function Documentation

8.14.3.1 begin()

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

8.14.3.2 end()

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

8.14.3.3 is_col()

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

8.14.3.4 is_row()

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

8.14.3.5 operator std::vector< Cell_Type >()

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

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

8.14.3.6 operator"!=()

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

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

8.14.3.7 operator<()

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

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

8.14.3.8 operator<=()

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

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

8.14.3.9 operator==()

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

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

8.14.3.10 operator>()

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

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

8.14.3.11 operator>=()

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

Definition at line 315 of file barrayvector-meat.hpp.

8.14.3.12 size()

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

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

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

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 (unsigned int x)
- void add (int x)
- Cell ()
- Cell ()
- Cell ()

Public Attributes

- Cell_Type value
- bool visited
- bool active

8.15.1 Detailed Description

template<class Cell_Type>
class Cell< Cell_Type >

Entries in BArray. For now, it only has two members:

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

Definition at line 10 of file cell-bones.hpp.

8.15.2 Constructor & Destructor Documentation

8.15.2.1 Cell() [1/7]

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

8.15.2.2 Cell() [2/7]

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

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

8.15.2.3 ~Cell()

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

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

8.15.2.4 Cell() [3/7]

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

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

8.15.2.5 Cell() [4/7]

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

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

8.15.2.6 Cell() [5/7]

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

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

8.15.2.7 Cell() [6/7]

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

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

8.15.2.8 Cell() [7/7]

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

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

8.15.3 Member Function Documentation

8.15.3.1 add() [1/4]

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

8.15.3.2 add() [2/4]

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

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

8.15.3.3 add() [3/4]

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

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

8.15.3.4 add() [4/4]

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

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

8.15.3.5 operator Cell_Type()

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

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

8.15.3.6 operator"!=()

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

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

8.15.3.7 operator=() [1/2]

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

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

8.15.3.8 operator=() [2/2]

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

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

8.15.3.9 operator==()

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

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

8.15.4 Member Data Documentation

8.15.4.1 active

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

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

8.15.4.2 value

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

Definition at line 12 of file cell-bones.hpp.

8.15.4.3 visited

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

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

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

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

8.16 Cell_const< Cell_Type > Class Template Reference

8.16.1 Detailed Description

template<typename Cell_Type>
class Cell_const< Cell_Type >

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

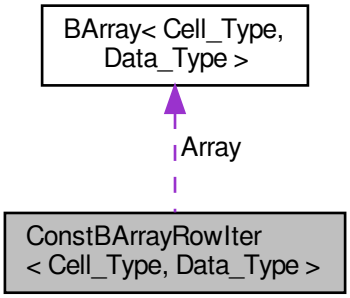
The documentation for this class was generated from the following file:

- include/barry/barray-meat.hpp

8.17 ConstBArrayRowIter< Cell_Type, Data_Type > Class Template Reference

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

Collaboration diagram for ConstBArrayRowIter< Cell_Type, Data_Type >:



Public Member Functions

- ConstBArrayRowIter (const BArray< Cell_Type, Data_Type > *Array_)
- ~ConstBArrayRowIter ()

Public Attributes

- uint current_row
- uint current_col
- Row_type< Cell_Type >::const_iterator iter
- const BArray< Cell_Type, Data_Type > * Array

8.17.1 Detailed Description

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

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

8.17.2 Constructor & Destructor Documentation

8.17.2.1 ConstBArrayRowIter()

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

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

8.17.2.2 ~ConstBArrayRowIter()

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

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

8.17.3 Member Data Documentation

8.17.3.1 Array

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

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

8.17.3.2 current_col

```
template<typename Cell_Type , typename Data_Type >
uint 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 >
uint 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, uint i, uint j)
- double init (Array_Type &Array, uint i, uint 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 (Counter_fun_type< Array_Type, Data_Type > count_fun_, Counter_fun_type< Array_Type, Data_Type > init_fun_, Hasher_fun_type< Array_Type, Data_Type > hasher_fun_, Data_Type data_, std::string name_="", std::string desc_="")
- Counter (const Counter< Array_Type, Data_Type > &counter_)

  Copy constructor.
- Counter (Counter< Array_Type, Data_Type > &&counter_) noexcept

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

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

  Move assignment.

- void set_hasher (Hasher_fun_type< Array_Type, Data_Type > fun)

  Get and set the hasher function.
- Hasher_fun_type< Array_Type, Data_Type > get_hasher ()

Public Attributes

- Counter_fun_type< Array_Type, Data_Type > count_fun
- Counter_fun_type< Array_Type, Data_Type > init_fun
- Hasher_fun_type< Array_Type, Data_Type > hasher_fun
- Data_Type data
- std::string name = ""
- std::string desc = ""

8.18.1 Detailed Description

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

A counter function based on change statistics.

This class is used by `CountStats` and `StatsCounter` as a way to count statistics using change statistics.

Definition at line 35 of file counters-bones.hpp.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 Counter() [1/4]

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

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

8.18.2.2 Counter() [2/4]

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

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

8.18.2.3 Counter() [3/4]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Counter< Array_Type, Data_Type >::Counter (
            const Counter< Array_Type, Data_Type > & counter_ )
```

Copy constructor.

8.18.2.4 Counter() [4/4]

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

Move constructor.

8.18.2.5 ~Counter()

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

Definition at line 75 of file counters-bones.hpp.

8.18.3 Member Function Documentation

8.18.3.1 count()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
double Counter< Array_Type, Data_Type >::count (
            Array_Type & Array,
            uint i,
            uint j )
```

8.18.3.2 get_description()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::string Counter< Array_Type, Data_Type >::get_description ( ) const
```

8.18.3.3 get_hasher()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Hasher_fun_type<Array_Type,Data_Type> Counter< Array_Type, Data_Type >::get_hasher ( )
```

8.18.3.4 get_name()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::string Counter< Array_Type, Data_Type >::get_name ( ) const
```

8.18.3.5 init()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
double Counter< Array_Type, Data_Type >::init (
            Array_Type & Array,
            uint i,
            uint j )
```

8.18.3.6 operator=() [1/2]

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

Copy assignment.

8.18.3.7 operator=() [2/2]

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

Move assignment.

8.18.3.8 set_hasher()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
void Counter< Array_Type, Data_Type >::set_hasher (
            Hasher_fun_type< Array_Type, Data_Type > fun )
```

Get and set the hasher function.

The hasher function is used to characterize the support of the array. This way, if possible, the support enumeration is recycled.

Parameters

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[ ] (uint idx)

  Returns a pointer to a particular counter.
- std::size_t size () const noexcept

  Number of counters in the set.
- void add_counter (Counter< Array_Type, Data_Type > counter)
- void add_counter (Counter_fun_type< Array_Type, Data_Type > count_fun_, Counter_fun_type< Array_Type, Data_Type > init_fun_, Hasher_fun_type< Array_Type, Data_Type > hasher_fun_, Data_Type data_, std::string name_="", std::string desc_="")
- std::vector< std::string > get_names () const
- std::vector< std::string > get_descriptions () const
- std::vector< double > gen_hash (const Array_Type &array, bool add_dims=true)

  Generates a hash for the given array according to the counters.
- void add_hash (Hasher_fun_type< Array_Type, Data_Type > fun_)

8.19.1 Detailed Description

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

Vector of counters.

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

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

8.19.2 Constructor & Destructor Documentation

8.19.2.1 Counters() [1/3]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Counters< Array_Type, Data_Type >::Counters ( )
```

8.19.2.2 ~Counters()

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

Definition at line 120 of file counters-bones.hpp.

8.19.2.3 Counters() [2/3]

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

Copy constructor.

Parameters

*counter_*	

8.19.2.4 Counters() [3/3]

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

Move constructor.

Parameters

*counters_*	

8.19.3 Member Function Documentation

8.19.3.1 add_counter() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
void Counters< Array_Type, Data_Type >::add_counter (
            Counter< Array_Type, Data_Type > counter )
```

8.19.3.2 add_counter() [2/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
void Counters< Array_Type, Data_Type >::add_counter (
            Counter_fun_type< Array_Type, Data_Type > count_fun_,
            Counter_fun_type< Array_Type, Data_Type > init_fun_,
            Hasher_fun_type< Array_Type, Data_Type > hasher_fun_,
            Data_Type data_,
            std::string name_ = "",
            std::string desc_ = "" )
```

8.19.3.3 add_hash()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
void Counters< Array_Type, Data_Type >::add_hash (
            Hasher_fun_type< Array_Type, Data_Type > fun_ )
```

8.19.3.4 gen_hash()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::vector< double > Counters< Array_Type, Data_Type >::gen_hash (
            const Array_Type & array,
            bool add_dims = true )
```

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

Parameters

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]

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

Copy assignment constructor.

Parameters

*counter_*	

Returns

Counters<Array_Type,Data_Type>

8.19.3.8 operator=() [2/2]

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

Move assignment constructor.

Parameters

*counter_*	

Returns

Counters<Array_Type,Data_Type>&

8.19.3.9 operator[]()

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

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

uint

Definition at line 164 of file counters-bones.hpp.

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

- include/barry/counters-bones.hpp

8.20 DEFM Class Reference

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

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)
- ~DEFM ()
- 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 std::vector< std::string > & get_X_names ()

8.20.1 Detailed Description

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

8.20.2 Constructor & Destructor Documentation

8.20.2.1 DEFM()

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

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

8.20.2.2 ~DEFM()

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

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

8.20.3 Member Function Documentation

8.20.3.1 get_ID()

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

Definition at line 260 of file defm-meat.hpp.

8.20.3.2 get_m_order()

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

Definition at line 245 of file defm-meat.hpp.

8.20.3.3 get_model()

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

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

8.20.3.4 get_n_covars()

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

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

8.20.3.5 get_n_obs()

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

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

8.20.3.6 get_n_rows()

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

Definition at line 250 of file defm-meat.hpp.

8.20.3.7 get_n_y()

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

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

8.20.3.8 get_X()

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

Definition at line 265 of file defm-meat.hpp.

8.20.3.9 get_X_names()

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

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

8.20.3.10 get_Y()

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

Definition at line 255 of file defm-meat.hpp.

8.20.3.11 get_Y_names()

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

Definition at line 368 of file defm-meat.hpp.

8.20.3.12 init()

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

Definition at line 189 of file defm-meat.hpp.

8.20.3.13 likelihood()

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

8.20.3.14 logodds()

```
std::vector< double > DEFM::logodds (
            const std::vector< double > & par,
            size_t i,
            size_t j )  [inline]
```

Definition at line 309 of file defm-meat.hpp.

8.20.3.15 motif_census()

```
barry::FreqTable< int > DEFM::motif_census (
            std::vector< size_t > idx )  [inline]
```

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

8.20.3.16 set_names()

```
void DEFM::set_names (
            std::vector< std::string > Y_names_,
            std::vector< std::string > X_names_ )  [inline]
```

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

8.20.3.17 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 uint 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_)
- size_t idx (size_t i) const
- double num (size_t i) const
- bool is_true (size_t i) const
- ~DEFMCounterData ()

Public Attributes

- std::vector< size_t > indices
- std::vector< double > numbers
- std::vector< bool > logical

8.21.1 Detailed Description

Data class used to store arbitrary uint 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 82 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_ )  [inline]
```

Definition at line 83 of file defm.hpp.

8.21.2.3 ~DEFMCounterData()

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

Definition at line 94 of file defm.hpp.

8.21.3 Member Function Documentation

8.21.3.1 idx()

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

Definition at line 90 of file defm.hpp.

8.21.3.2 is_true()

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

Definition at line 92 of file defm.hpp.

8.21.3.3 num()

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

Definition at line 91 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 logical

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

Definition at line 80 of file defm.hpp.

8.21.4.3 numbers

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

Definition at line 79 of file defm.hpp.

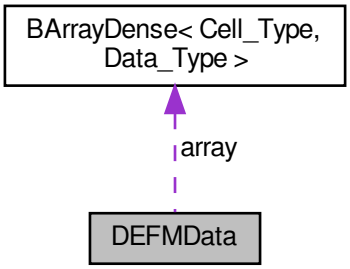
The documentation for this class was generated from the following file:

- 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]

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

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 ~DEFMData()

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

Definition at line 69 of file defm.hpp.

8.22.3 Member Function Documentation

8.22.3.1 at()

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

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.

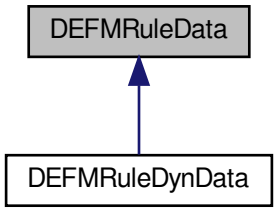
The documentation for this class was generated from the following file:

- 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_t i) const
- DEFMRuleData ()
- DEFMRuleData (std::vector< double > numbers_, std::vector< size_t > indices_, std::vector< bool > logical_)
- DEFMRuleData (std::vector< double > numbers_, std::vector< size_t > indices_)

Public Attributes

- std::vector< double > numbers
- std::vector< size_t > indices
- std::vector< bool > logical
- bool init = false

8.23.1 Detailed Description

Definition at line 98 of file defm.hpp.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 DEFMRuleData() [1/3]

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

Definition at line 111 of file defm.hpp.

8.23.2.2 DEFMRuleData() [2/3]

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

Definition at line 113 of file defm.hpp.

8.23.2.3 DEFMRuleData() [3/3]

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

Definition at line 119 of file defm.hpp.

8.23.3 Member Function Documentation

8.23.3.1 idx()

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

Definition at line 108 of file defm.hpp.

8.23.3.2 is_true()

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

Definition at line 109 of file defm.hpp.

8.23.3.3 num()

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

Definition at line 107 of file defm.hpp.

8.23.4 Member Data Documentation

8.23.4.1 indices

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

Definition at line 102 of file defm.hpp.

8.23.4.2 init

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

Definition at line 105 of file defm.hpp.

8.23.4.3 logical

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

Definition at line 103 of file defm.hpp.

8.23.4.4 numbers

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

Definition at line 101 of file defm.hpp.

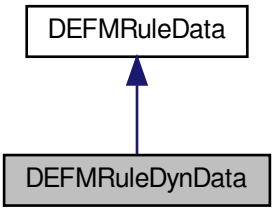
The documentation for this class was generated from the following file:

- include/barry/counters/defm.hpp

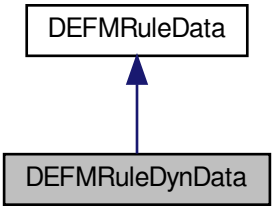
8.24 DEFMRuleDynData Class Reference

`#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 133 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 137 of file defm.hpp.

8.24.2.2 ~DEFMRuleDynData()

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

Definition at line 144 of file defm.hpp.

8.24.3 Member Data Documentation

8.24.3.1 counts

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

Definition at line 135 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 (uint n)
- ~Entries ()
- void resize (uint n)

Public Attributes

- std::vector< uint > source
- std::vector< uint > target
- std::vector< Cell_Type > val

8.25.1 Detailed Description

template<typename Cell_Type>
class Entries< Cell_Type >

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

Template Parameters

Cell_Type	Any type

Definition at line 79 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 85 of file typedefs.hpp.

8.25.2.2 Entries() [2/2]

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

Definition at line 86 of file typedefs.hpp.

8.25.2.3 ~Entries()

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

Definition at line 93 of file typedefs.hpp.

8.25.3 Member Function Documentation

8.25.3.1 resize()

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

Definition at line 95 of file typedefs.hpp.

8.25.4 Member Data Documentation

8.25.4.1 source

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

Definition at line 81 of file typedefs.hpp.

8.25.4.2 target

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

Definition at line 82 of file typedefs.hpp.

8.25.4.3 val

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

Definition at line 83 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 ()
- unsigned int add_data (std::vector< std::vector< unsigned int > > &annotations, std::vector< unsigned int > &geneid, std::vector< int > &parent, std::vector< bool > &duplication)

  Add a tree to the flock.
- void set_seed (const unsigned int &s)

  Set the seed of the model.
- void init (unsigned int bar_width=BARRY_PROGRESS_BAR_WIDTH)
- phylocounters::PhyloCounters * get_counters ()
- phylocounters::PhyloSupport * get_support_fun ()
- std::vector< std::vector< double > > * get_stats_support ()
- std::vector< std::vector< double > > * get_stats_target ()
- phylocounters::PhyloModel * get_model ()
- double likelihood_joint (const std::vector< double > &par, bool as_log=false, bool use_reduced_sequence=true)

  Returns the joint likelihood of the model.
- Geese * operator() (unsigned int i, bool check_bounds=true)

  Access the i-th geese element.

Information about the model

- unsigned int nfuns () const noexcept
- unsigned int ntrees () const noexcept
- std::vector< unsigned int > nnodes () const noexcept
- std::vector< unsigned int > nleafs () const noexcept
- unsigned int nterms () const
- unsigned int support_size () const noexcept
- std::vector< std::string > colnames () const
- unsigned int parse_polytomies (bool verb=true, std::vector< size_t > *dist=nullptr) const noexcept

  Check polytomies and return the largest.
- void print () const

Public Attributes

- std::vector< Geese > dat
- unsigned int 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 builds a model with multiple trees (Geese objects), with all of these using the same PhyloModel object. Available counters (terms) can be found in counter-phylo.

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

8.26.2 Constructor & Destructor Documentation

8.26.2.1 Flock()

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

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

8.26.2.2 ~Flock()

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

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

8.26.3 Member Function Documentation

8.26.3.1 add_data()

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

Add a tree to the flock.

Parameters

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

Returns

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

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

8.26.3.2 colnames()

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

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

8.26.3.3 get_counters()

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

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

8.26.3.4 get_model()

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

Definition at line 131 of file flock-meat.hpp.

8.26.3.5 get_stats_support()

```
std::vector< std::vector< double > > * Flock::get_stats_support ( )  [inline]
```

Definition at line 117 of file flock-meat.hpp.

8.26.3.6 get_stats_target()

```
std::vector< std::vector< double > > * Flock::get_stats_target ( )  [inline]
```

Definition at line 124 of file flock-meat.hpp.

8.26.3.7 get_support_fun()

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

Definition at line 110 of file flock-meat.hpp.

8.26.3.8 init()

```
void Flock::init (
            unsigned int bar_width = BARRY_PROGRESS_BAR_WIDTH )  [inline]
```

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

8.26.3.9 likelihood_joint()

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

Returns the joint likelihood of the model.

Parameters

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

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

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

8.26.3.11 nleafs()

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

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

8.26.3.12 nnodes()

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

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

8.26.3.13 nterms()

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

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

8.26.3.14 ntrees()

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

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

8.26.3.15 operator()()

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

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()

```
unsigned int Flock::parse_polytomies (
            bool verb = true,
            std::vector< size_t > * dist = nullptr ) const  [inline], [noexcept]
```

Check polytomies and return the largest.

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

8.26.3.17 print()

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

Definition at line 258 of file flock-meat.hpp.

8.26.3.18 set_seed()

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

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()

```
unsigned int 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

```
unsigned int 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()

```
template<typename T >
size_t FreqTable< T >::add (
            const std::vector< T > & x,
            size_t * h_precomp )  [inline]
```

Definition at line 59 of file freqtable.hpp.

8.27.3.2 as_vector()

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

Definition at line 139 of file freqtable.hpp.

8.27.3.3 clear()

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

Definition at line 168 of file freqtable.hpp.

8.27.3.4 get_data()

```
template<typename T = double>
const std::vector< double >& FreqTable< T >::get_data ( ) const  [inline]
```

Definition at line 40 of file freqtable.hpp.

8.27.3.5 get_index()

```
template<typename T = double>
const std::unordered_map<size_t,size_t>& FreqTable< T >::get_index ( ) const  [inline]
```

Definition at line 41 of file freqtable.hpp.

8.27.3.6 make_hash()

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

Definition at line 239 of file freqtable.hpp.

8.27.3.7 print()

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

Definition at line 204 of file freqtable.hpp.

8.27.3.8 reserve()

```
template<typename T >
void FreqTable< T >::reserve (
            size_t n,
            size_t k )  [inline]
```

Definition at line 182 of file freqtable.hpp.

8.27.3.9 size()

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

Number of unique elements in the table. (.

Returns

size_t

Definition at line 231 of file freqtable.hpp.

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

- include/barry/freqtable.hpp

8.28 Geese Class Reference

Annotated Phylo Model.

`#include <geese-bones.hpp>`

Public Member Functions

- ~Geese ()
- void init (unsigned int 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 unsigned int &s)
- std::vector< std::vector< unsigned int > > simulate (const std::vector< double > &par)
- std::vector< std::vector< double > > observed_counts ()
- void print_observed_counts ()
- void print () const

  Prints information about the GEESE.
- void init_node (Node &n)
- void update_annotations (unsigned int nodeid, std::vector< unsigned int > newann)
- std::vector< std::vector< bool > > get_states () const

  Powerset of a gene's possible states.
- std::vector< unsigned int > get_annotated_nodes () const

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

Construct a new Geese object

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

Parameters

annotations	*A vector of vectors with annotations. It should be of length k (number of functions). Each vector should be of length N (equal to the number of nodes, including interior). Possible values are 0, 1, and 9.*
geneid	*Id of the gene. It should be of length N.*
parent	*Id of the parent gene. Also of length N*
duplication	*Logical scalar indicating the type of event (true: duplication, false: speciation.)*

The ordering of the entries does not matter. Passing the nodes in post order or not makes no difference to the constructor.

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

Information about the model

Parameters

verb	*When `true` it will print out information about the encountered polytomies.*

- unsigned int nfuns () const noexcept
  Number of functions analyzed.
- unsigned int nnodes () const noexcept
  Number of nodes (interior + leaf)
- unsigned int nleafs () const noexcept
  Number of leaf.
- unsigned int nterms () const
  Number of terms included.
- unsigned int support_size () const noexcept
  Number of unique sets of sufficient stats.
- std::vector< unsigned int > nannotations () const noexcept
  Number of annotations.
- std::vector< std::string > colnames () const
  Names of the terms in the model.
- unsigned int 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 `Node`s objects. This contains the probability that the ith node has either of the possible states.

Returns

std::vector< double > Returns the posterior probability

- std::vector< std::vector< double > > predict (const std::vector< double > &par, std::vector< std::vector< double > > *res_prob=nullptr, bool leave_one_out=false, bool only_annotated=false, bool use_reduced↩_sequence=true)
- std::vector< std::vector< double > > predict_backend (const std::vector< double > &par, bool use_↩reduced_sequence, const std::vector< uint > &preorder)
- std::vector< std::vector< double > > predict_exhaust_backend (const std::vector< double > &par, const std::vector< uint > &preorder)
- std::vector< std::vector< double > > predict_exhaust (const std::vector< double > &par)
- std::vector< std::vector< double > > predict_sim (const std::vector< double > &par, bool only_↩annotated=false, unsigned int nsims=10000u)

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

These functions provide direct access to some member objects that are shared by the nodes within `Geese`.

Returns

`get_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

- unsigned int nfunctions
- std::map< unsigned int, Node > nodes
- barry::MapVec_type< unsigned int > map_to_nodes
- std::vector< std::vector< std::vector< size_t > > > pset_loc

  Locations of columns.
- std::vector< unsigned int > sequence
- std::vector< unsigned int > 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< unsigned int > > & annotations,
            std::vector< unsigned int > & geneid,
            std::vector< int > & parent,
            std::vector< bool > & duplication )  [inline]
```

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

8.28.2.3 Geese() [3/4]

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

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

8.28.2.4 Geese() [4/4]

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

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

8.28.2.5 ~Geese()

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

Definition at line 91 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 330 of file geese-meat.hpp.

8.28.3.2 calc_sequence()

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

Definition at line 286 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 452 of file geese-meat.hpp.

8.28.3.4 get_annotated_nodes()

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

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

Definition at line 668 of file geese-meat.hpp.

8.28.3.5 get_counters()

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

Definition at line 651 of file geese-meat.hpp.

8.28.3.6 get_model()

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

Definition at line 656 of file geese-meat.hpp.

8.28.3.7 get_probabilities()

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

Definition at line 378 of file geese-meat.hpp.

8.28.3.8 get_rengine()

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

Definition at line 646 of file geese-meat.hpp.

8.28.3.9 get_states()

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

Powerset of a gene's possible states.

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

Returns

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

Definition at line 664 of file geese-meat.hpp.

8.28.3.10 get_support_fun()

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

Definition at line 660 of file geese-meat.hpp.

8.28.3.11 inherit_support()

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

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

8.28.3.12 init()

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

Definition at line 103 of file geese-meat.hpp.

8.28.3.13 init_node()

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

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

8.28.3.14 likelihood()

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

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

8.28.3.15 likelihood_exhaust()

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

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

8.28.3.16 nannotations()

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

Number of annotations.

Definition at line 443 of file geese-meat.hpp.

8.28.3.17 nfuns()

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

Number of functions analyzed.

Definition at line 399 of file geese-meat.hpp.

8.28.3.18 nleafs()

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

Number of leaf.

Definition at line 413 of file geese-meat.hpp.

8.28.3.19 nnodes()

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

Number of nodes (interior + leaf)

Definition at line 406 of file geese-meat.hpp.

8.28.3.20 nterms()

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

Number of terms included.

Definition at line 425 of file geese-meat.hpp.

8.28.3.21 observed_counts()

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

Definition at line 494 of file geese-meat.hpp.

8.28.3.22 operator=() [1/2]

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

8.28.3.23 operator=() [2/2]

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

8.28.3.24 parse_polytomies()

```
unsigned int Geese::parse_polytomies (
            bool verb = true,
            std::vector< size_t > * dist = nullptr ) const  [inline], [noexcept]
```

Check polytomies and return the largest.

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

8.28.3.25 predict()

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

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

8.28.3.26 predict_backend()

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

< True if the array belongs to the set

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

8.28.3.27 predict_exhaust()

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

Definition at line 5 of file geese-meat-predict_exhaust.hpp.

8.28.3.28 predict_exhaust_backend()

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

Definition at line 47 of file geese-meat-predict_exhaust.hpp.

8.28.3.29 predict_sim()

```
std::vector< std::vector< double > > Geese::predict_sim (
            const std::vector< double > & par,
            bool only_annotated = false,
            unsigned int 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 628 of file geese-meat.hpp.

8.28.3.31 print_observed_counts()

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

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

8.28.3.32 set_seed()

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

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

8.28.3.33 simulate()

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

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

8.28.3.34 support_size()

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

Number of unique sets of sufficient stats.

Definition at line 433 of file geese-meat.hpp.

8.28.3.35 update_annotations()

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

Definition at line 257 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< unsigned int > Geese::map_to_nodes
```

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

8.28.4.5 nfunctions

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

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

8.28.4.6 nodes

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

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

8.28.4.7 pset_loc

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

Locations of columns.

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

8.28.4.8 reduced_sequence

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

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

8.28.4.9 sequence

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

Public Member Functions

- void set_rengine (std::mt19937 *rengine_, bool delete_=false)
- void set_seed (unsigned int s)
- Model ()
- Model (uint size_)
- Model (const Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > &Model_)
- Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > & operator= (const Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > &Model_)
- ~Model ()
- void store_psets () noexcept
- std::vector< double > gen_key (const Array_Type &Array_)
- uint add_array (const Array_Type &Array_, bool force_new=false)

  Adds an array to the support of not already included.
- void print_stats (uint i) const
- void print () const

  Prints information about the model.
- Array_Type sample (const Array_Type &Array_, const std::vector< double > &params={})
- Array_Type sample (const uint &i, const std::vector< double > &params)
- double conditional_prob (const Array_Type &Array_, const std::vector< double > &params, unsigned int i, unsigned int 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</tt> member.

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

- void add_counter (Counter< Array_Type, Data_Counter_Type > &counter)
- void add_counter (Counter_fun_type< Array_Type, Data_Counter_Type > count_fun_, Counter_fun_type< Array_Type, Data_Counter_Type > init_fun_=nullptr, Data_Counter_Type data_=nullptr)
- void set_counters (Counters< Array_Type, Data_Counter_Type > *counters_)
- void add_hasher (Hasher_fun_type< Array_Type, Data_Counter_Type > fun_)

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

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

- void add_rule (Rule< Array_Type, Data_Rule_Type > &rule)
- void add_rule (Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_, Data_Rule_Type data_)
- void set_rules (Rules< Array_Type, Data_Rule_Type > *rules_)
- void add_rule_dyn (Rule< Array_Type, Data_Rule_Dyn_Type > &rule)
- void add_rule_dyn (Rule_fun_type< Array_Type, Data_Rule_Dyn_Type > count_fun_, Data_Rule_Dyn↩ _Type data_)
- void set_rules_dyn (Rules< Array_Type, Data_Rule_Dyn_Type > *rules_)

Likelihood functions.

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

Parameters

params	*Vector of parameters*
as_log	*When `true`, the function returns the log-likelihood.*

- double likelihood (const std::vector< double > &params, const uint &i, bool as_log=false)
- double likelihood (const std::vector< double > &params, const Array_Type &Array_, int i=-1, bool as_↩ log=false)
- double likelihood (const std::vector< double > &params, const std::vector< double > &target_, const uint &i, bool as_log=false)
- double likelihood (const std::vector< double > &params, const double *target_, const uint &i, bool as_↩ log=false)
- double likelihood_total (const std::vector< double > &params, bool as_log=false)

Extract elements by index

Parameters

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

- double get_norm_const (const std::vector< double > &params, const uint &i, bool as_log=false)

- const std::vector< Array_Type > * get_pset (const uint &i)
- const std::vector< double > * get_pset_stats (const uint &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.*

- unsigned int size () const noexcept
- unsigned int size_unique () const noexcept
- unsigned int nterms () const noexcept
- unsigned int nrules () const noexcept
- unsigned int nrules_dyn () const noexcept
- unsigned int 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< unsigned int > * 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 *, unsigned int)> fun, std::vector< std::string > names)

  Set the transform_model_fun object.
- std::vector< double > transform_model (double *data, unsigned int k)

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:

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

This implementation aims to reduce the number of times that the support needs to be computed. Models included here use more than a single array, and thus allow the function to recycle support sets as needed. For example, if we are looking at directed graphs all of the same size and without vertex level features, i.e. a model that only counts edges, triangles, etc. then the support needs to be fully computed only once.

Template Parameters

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 = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Model ( )
```

8.29.2.2 Model() [2/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Model (
            uint size_ )
```

8.29.2.3 Model() [3/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Model (
            const Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >
& Model_ )
```

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>
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::~Model ( )  [inline]
```

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

8.29.3 Member Function Documentation

8.29.3.1 add_array()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_array (
            const Array_Type & Array_,
            bool force_new = false )
```

Adds an array to the support of not already included.

Parameters

*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]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_counter (
            Counter< Array_Type, Data_Counter_Type > & counter )
```

8.29.3.3 add_counter() [2/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_counter (
            Counter_fun_type< Array_Type, Data_Counter_Type > count_fun_,
            Counter_fun_type< Array_Type, Data_Counter_Type > init_fun_ = nullptr,
            Data_Counter_Type data_ = nullptr )
```

8.29.3.4 add_hasher()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_hasher (
            Hasher_fun_type< Array_Type, Data_Counter_Type > fun_ )
```

8.29.3.5 add_rule() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > & rule )
```

8.29.3.6 add_rule() [2/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
            Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_,
            Data_Rule_Type data_ )
```

8.29.3.7 add_rule_dyn() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_dyn
(
            Rule< Array_Type, Data_Rule_Dyn_Type > & rule )
```

8.29.3.8 add_rule_dyn() [2/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_dyn
(
            Rule_fun_type< Array_Type, Data_Rule_Dyn_Type > count_fun_,
            Data_Rule_Dyn_Type data_ )
```

8.29.3.9 colnames()

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

8.29.3.10 conditional_prob()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::conditional↩
_prob (
            const Array_Type & Array_,
            const std::vector< double > & params,
            unsigned int i,
            unsigned int j )
```

Conditional probability ("Gibbs sampler")

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

Parameters

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()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< double > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_↩
Type >::gen_key (
            const Array_Type & Array_ )
```

8.29.3.12 get_arrays2support()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< unsigned int >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_↩
Rule_Dyn_Type >::get_arrays2support ( )
```

8.29.3.13 get_counters()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Counters<Array_Type,Data_Counter_Type>* Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_counters ( )
```

8.29.3.14 get_norm_const()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_norm_↩
const (
            const std::vector< double > & params,
            const uint & i,
            bool as_log = false )
```

8.29.3.15 get_pset()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
const std::vector< Array_Type >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data↩
_Rule_Dyn_Type >::get_pset (
            const uint & i )
```

8.29.3.16 get_pset_arrays()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector< Array_Type > >* Model< Array_Type, Data_Counter_Type, Data_Rule_↩
Type, Data_Rule_Dyn_Type >::get_pset_arrays ( )
```

8.29.3.17 get_pset_probs()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector<double> >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_pset_probs ( )
```

8.29.3.18 get_pset_stats() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector<double> >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_pset_stats ( )
```

Statistics of the support(s)

8.29.3.19 get_pset_stats() [2/2]

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

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) x n unique elements`, with the data stored by-row. The last element of each entry corresponds to the weights, i.e., the frequency with which such sufficient statistics are observed in the support.

8.29.3.25 get_support_fun()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support<Array_Type,Data_Counter_Type,Data_Rule_Type,Data_Rule_Dyn_Type>* Model< Array_Type,
Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_support_fun ( )
```

8.29.3.26 likelihood() [1/4]

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

8.29.3.27 likelihood() [2/4]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::likelihood
(
            const std::vector< double > & params,
            const double * target_,
            const uint & i,
            bool as_log = false )
```

8.29.3.28 likelihood() [3/4]

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

8.29.3.29 likelihood() [4/4]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::likelihood
(
            const std::vector< double > & params,
            const uint & i,
            bool as_log = false )
```

8.29.3.30 likelihood_total()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::likelihood↩
_total (
            const std::vector< double > & params,
            bool as_log = false )
```

8.29.3.31 nrules()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
unsigned int Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >↩
::nrules ( ) const  [noexcept]
```

8.29.3.32 nrules_dyn()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
unsigned int 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>
unsigned int Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >↩
::nterms ( ) const  [noexcept]
```

8.29.3.34 operator=()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Model<Array_Type,Data_Counter_Type,Data_Rule_Type,Data_Rule_Dyn_Type>& Model< 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_ )
```

8.29.3.35 print()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::print ( )
const
```

Prints information about the model.

8.29.3.36 print_stats()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::print_stats (
            uint i ) const
```

8.29.3.37 sample() [1/2]

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

8.29.3.38 sample() [2/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Array_Type Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::sample
(
            const uint & i,
            const std::vector< double > & params )
```

8.29.3.39 set_counters()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_counters
(
            Counters< Array_Type, Data_Counter_Type > * counters_ )
```

8.29.3.40 set_rengine()

```
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_rengine (
            std::mt19937 * rengine_,
            bool delete_ = false )  [inline]
```

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

8.29.3.41 set_rules()

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

8.29.3.42 set_rules_dyn()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_rules_dyn
(
            Rules< Array_Type, Data_Rule_Dyn_Type > * rules_ )
```

8.29.3.43 set_seed()

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

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

8.29.3.44 set_transform_model()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_transform↩
_model (
            std::function< std::vector< double >(double *, unsigned int)> fun,
            std::vector< std::string > names )
```

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>
unsigned int 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>
unsigned int 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>
unsigned int Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >↩
::support_size ( ) const  [noexcept]
```

8.29.3.49 transform_model()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector<double> Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type
>::transform_model (
            double * data,
            unsigned int k )
```

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

- include/barry/model-bones.hpp

8.30 NetCounterData Class Reference

Data class used to store arbitrary uint or double vectors.

`#include <network.hpp>`

Public Member Functions

- NetCounterData ()
- NetCounterData (const std::vector< uint > indices_, const std::vector< double > numbers_)
- ~NetCounterData ()

Public Attributes

- std::vector< uint > indices
- std::vector< double > numbers

8.30.1 Detailed Description

Data class used to store arbitrary uint or double vectors.

Definition at line 56 of file network.hpp.

8.30.2 Constructor & Destructor Documentation

8.30.2.1 NetCounterData() [1/2]

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

Definition at line 62 of file network.hpp.

8.30.2.2 NetCounterData() [2/2]

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

Definition at line 63 of file network.hpp.

8.30.2.3 ~NetCounterData()

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

Definition at line 68 of file network.hpp.

8.30.3 Member Data Documentation

8.30.3.1 indices

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

Definition at line 59 of file network.hpp.

8.30.3.2 numbers

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

Definition at line 60 of file network.hpp.

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

- include/barry/counters/network.hpp

8.31 NetworkData Class Reference

Data class for Networks.

```
#include <network.hpp>
```

Public Member Functions

- NetworkData ()
- NetworkData (std::vector< double > vertex_attr_, bool directed_=true)

  Constructor using a single attribute.
- NetworkData (std::vector< std::vector< double > > vertex_attr_, bool directed_=true)

  Constructor using multiple attributes.
- ~NetworkData ()

Public Attributes

- bool directed = true
- std::vector< std::vector< double > > vertex_attr

8.31.1 Detailed Description

Data class for Networks.

Details on the available counters for `NetworkData` can be found in the DEFMArray counters section.

This holds information about whether the graph is directed or not, and, if defined, vectors of node (vertex) attributes (`vertex_attr`).

Definition at line 19 of file network.hpp.

8.31.2 Constructor & Destructor Documentation

8.31.2.1 NetworkData() [1/3]

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

Definition at line 25 of file network.hpp.

8.31.2.2 NetworkData() [2/3]

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

Constructor using a single attribute.

Parameters

*vertex_attr_*	Double vector of length equal to the number of vertices in the data.
*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_↩ attr_*	Vector of double vectors. The size equals to the number of attributes to be created. Each individual vector should be of length equal to the number of vertices.
*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.

8.31.3.2 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:

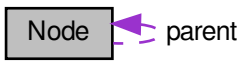
- include/barry/counters/network.hpp

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
- unsigned int noffspring () const noexcept
- bool is_leaf () const noexcept

Construct a new Node object

- Node ()
- Node (unsigned int id_, unsigned int ord_, bool duplication_)
- Node (unsigned int id_, unsigned int ord_, std::vector< unsigned int > annotations_, bool duplication_)
- Node (Node &&x) noexcept
- Node (const Node &x)

Public Attributes

- unsigned int id

  Id of the node (as specified in the input)
- unsigned int ord

  Order in which the node was created.
- phylocounters::PhyloArray array
- std::vector< unsigned int > annotations

  Observed annotations (only defined for Geese)
- bool duplication
- std::vector< phylocounters::PhyloArray > arrays = {}

  Arrays given all possible states.
- Node * parent = nullptr

  Parent node.
- std::vector< Node * > offspring = {}

  Offspring nodes.
- std::vector< unsigned int > narray = {}

  ID of the array in the model.
- bool visited = false
- std::vector< double > subtree_prob

  Induced subtree probabilities.
- std::vector< double > probability

  The probability of observing each state.

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 36 of file geese-node-bones.hpp.

8.32.2.2 Node() [2/5]

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

Definition at line 56 of file geese-node-bones.hpp.

8.32.2.3 Node() [3/5]

```
Node::Node (
            unsigned int id_,
            unsigned int ord_,
            std::vector< unsigned int > annotations_,
            bool duplication_ )  [inline]
```

Definition at line 62 of file geese-node-bones.hpp.

8.32.2.4 Node() [4/5]

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

Definition at line 69 of file geese-node-bones.hpp.

8.32.2.5 Node() [5/5]

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

Definition at line 83 of file geese-node-bones.hpp.

8.32.2.6 ~Node()

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

Definition at line 47 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 97 of file geese-node-bones.hpp.

8.32.3.2 is_leaf()

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

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

8.32.3.3 noffspring()

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

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

8.32.4 Member Data Documentation

8.32.4.1 annotations

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

Observed annotations (only defined for Geese)

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

8.32.4.2 array

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

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

8.32.4.3 arrays

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

Arrays given all possible states.

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

8.32.4.4 duplication

```
bool Node::duplication
```

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

8.32.4.5 id

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

Id of the node (as specified in the input)

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

8.32.4.6 narray

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

ID of the array in the model.

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

8.32.4.7 offspring

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

Offspring nodes.

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

8.32.4.8 ord

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

Order in which the node was created.

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

8.32.4.9 parent

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

Parent node.

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

8.32.4.10 probability

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

The probability of observing each state.

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

8.32.4.11 subtree_prob

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

Induced subtree probabilities.

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

8.32.4.12 visited

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

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

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

- include/barry/models/geese/geese-node-bones.hpp

8.33 NodeData Class Reference

Data definition for the `PhyloArray` class.

`#include <phylo.hpp>`

Public Member Functions

- NodeData (const std::vector< double > &blengths_, const std::vector< bool > &states_, bool duplication_=true)

Public Attributes

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

8.33.1 Detailed Description

Data definition for the `PhyloArray` class.

Details about the available counters for `PhyloArray` objects can be found in the Phylo counters section.

This holds basic information about a given node.

Definition at line 38 of file phylo.hpp.

8.33.2 Constructor & Destructor Documentation

8.33.2.1 NodeData()

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

Definition at line 58 of file phylo.hpp.

8.33.3 Member Data Documentation

8.33.3.1 blengths

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

Branch length.

Definition at line 44 of file phylo.hpp.

8.33.3.2 duplication

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

Definition at line 54 of file phylo.hpp.

8.33.3.3 states

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

State of the parent node.

Definition at line 49 of file phylo.hpp.

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

- include/barry/counters/phylo.hpp

8.34 PhyloCounterData Class Reference

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

Public Member Functions

- PhyloCounterData (std::vector< uint > data_, std::vector< double > *counters_=nullptr)
- PhyloCounterData ()
- uint at (uint d)
- uint operator() (uint d)
- uint operator[ ] (uint d)
- void reserve (uint x)
- void push_back (uint x)
- void shrink_to_fit ()
- uint size ()
- std::vector< uint >::iterator begin ()
- std::vector< uint >::iterator end ()
- bool empty ()
- std::vector< double > * get_counters ()

8.34.1 Detailed Description

Definition at line 69 of file phylo.hpp.

8.34.2 Constructor & Destructor Documentation

8.34.2.1 PhyloCounterData() [1/2]

```
PhyloCounterData::PhyloCounterData (
            std::vector< uint > data_,
            std::vector< double > * counters_ = nullptr )  [inline]
```

Definition at line 75 of file phylo.hpp.

8.34.2.2 PhyloCounterData() [2/2]

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

Definition at line 80 of file phylo.hpp.

8.34.3 Member Function Documentation

8.34.3.1 at()

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

Definition at line 82 of file phylo.hpp.

8.34.3.2 begin()

```
std::vector< uint >::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< uint >::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()()

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

Definition at line 83 of file phylo.hpp.

8.34.3.7 operator[]()

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

Definition at line 84 of file phylo.hpp.

8.34.3.8 push_back()

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

Definition at line 86 of file phylo.hpp.

8.34.3.9 reserve()

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

Definition at line 85 of file phylo.hpp.

8.34.3.10 shrink_to_fit()

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

Definition at line 87 of file phylo.hpp.

8.34.3.11 size()

```
uint 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_, uint pos_, uint lb_, uint ub_, uint duplication_)
- ~PhyloRuleDynData ()

Public Attributes

- const std::vector< double > * counts
- uint pos
- uint lb
- uint ub
- uint duplication

8.35.1 Detailed Description

Definition at line 2147 of file phylo.hpp.

8.35.2 Constructor & Destructor Documentation

8.35.2.1 PhyloRuleDynData()

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

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

```
uint PhyloRuleDynData::duplication
```

Definition at line 2153 of file phylo.hpp.

8.35.3.3 lb

```
uint PhyloRuleDynData::lb
```

Definition at line 2151 of file phylo.hpp.

8.35.3.4 pos

```
uint PhyloRuleDynData::pos
```

Definition at line 2150 of file phylo.hpp.

8.35.3.5 ub

```
uint PhyloRuleDynData::ub
```

Definition at line 2152 of file phylo.hpp.

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

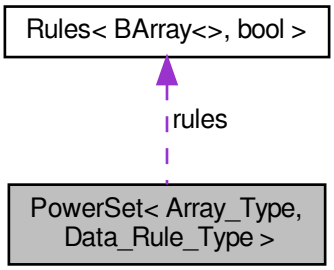
- include/barry/counters/phylo.hpp

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 (uint N_, uint M_)

Construct and destroy a PowerSet object

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

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

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

- void add_rule (Rule< Array_Type, Data_Rule_Type > rule)
- void add_rule (Rule_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 unsigned int &i) const

Public Attributes

- Array_Type EmptyArray
- std::vector< Array_Type > data
- Rules< Array_Type, Data_Rule_Type > * rules
- uint N
- uint M
- bool rules_deleted = false
- std::vector< 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]

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
PowerSet< Array_Type, Data_Rule_Type >::PowerSet (
            uint N_,
            uint M_ )  [inline]
```

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

8.36.2.3 PowerSet() [3/3]

```
template<typename Array_Type , typename Data_Rule_Type >
PowerSet< Array_Type, Data_Rule_Type >::PowerSet (
            const Array_Type & array )  [inline]
```

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

8.36.2.4 ~PowerSet()

```
template<typename Array_Type , typename Data_Rule_Type >
PowerSet< Array_Type, Data_Rule_Type >::~PowerSet  [inline]
```

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

8.36.3 Member Function Documentation

8.36.3.1 add_rule() [1/2]

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > rule )  [inline]
```

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

8.36.3.2 add_rule() [2/2]

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::add_rule (
            Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_,
            Data_Rule_Type data_ )  [inline]
```

Definition at line 182 of file powerset-meat.hpp.

8.36.3.3 begin()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type >::iterator PowerSet< Array_Type, Data_Rule_Type >::begin ( )  [inline]
```

Definition at line 68 of file powerset-bones.hpp.

8.36.3.4 calc()

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::calc  [inline]
```

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

8.36.3.5 end()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type >::iterator PowerSet< Array_Type, Data_Rule_Type >::end ( )  [inline]
```

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

8.36.3.6 get_data()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type > PowerSet< Array_Type, Data_Rule_Type >::get_data ( ) const  [inline]
```

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

8.36.3.7 get_data_ptr()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
const std::vector< Array_Type >* PowerSet< Array_Type, Data_Rule_Type >::get_data_ptr ( )
const  [inline]
```

Definition at line 66 of file powerset-bones.hpp.

8.36.3.8 init_support()

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::init_support  [inline]
```

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

8.36.3.9 operator[]()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
const Array_Type& PowerSet< Array_Type, Data_Rule_Type >::operator[] (
            const unsigned int & i ) const  [inline]
```

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

8.36.3.10 reset()

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::reset (
            uint N_,
            uint M_ )  [inline]
```

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

8.36.3.11 size()

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

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

8.36.4 Member Data Documentation

8.36.4.1 coordinates_free

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< size_t > PowerSet< Array_Type, Data_Rule_Type >::coordinates_free
```

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

8.36.4.2 coordinates_locked

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< size_t > PowerSet< Array_Type, Data_Rule_Type >::coordinates_locked
```

Definition at line 27 of file powerset-bones.hpp.

8.36.4.3 data

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type > PowerSet< Array_Type, Data_Rule_Type >::data
```

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

8.36.4.4 EmptyArray

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
Array_Type PowerSet< Array_Type, Data_Rule_Type >::EmptyArray
```

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

8.36.4.5 M

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
uint 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>
uint 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, uint i, uint 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_↩ dat_	*When* `true`, *the* `Rule` *destructor will delete the pointer, if defined.*

- Rule ()
- Rule (Rule_fun_type< Array_Type, Data_Type > fun_, Data_Type dat_, std::string name_="", std::string desc_="")

8.38.1 Detailed Description

template<typename Array_Type = BArray<>, typename Data_Type = bool>
class Rule< Array_Type, Data_Type >

Rule for determining if a cell should be included in a sequence.

Rules can be used together with `Support` and `PowerSet` to determine which cells should be included when enumerating all possible realizations of a binary array.

Template Parameters

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]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Rule< Array_Type, Data_Type >::Rule (
            Rule_fun_type< Array_Type, Data_Type > fun_,
            Data_Type dat_,
            std::string name_ = "",
            std::string desc_ = "" )  [inline]
```

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

8.38.2.3 ~Rule()

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

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

8.38.3 Member Function Documentation

8.38.3.1 D()

```
template<typename Array_Type = 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()()

```
template<typename Array_Type , typename Data_Type >
bool Rule< Array_Type, Data_Type >::operator() (
            const Array_Type & a,
            uint i,
            uint j )  [inline]
```

Definition at line 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 ()
- uint size () const noexcept
- bool operator() (const Array_Type &a, uint i, uint 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]

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::Rules (
            const Rules< Array_Type, Data_Type > & rules_ )  [inline]
```

Definition at line 5 of file rules-meat.hpp.

8.39.2.3 ~Rules()

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::~Rules ( )  [inline]
```

Definition at line 82 of file rules-bones.hpp.

8.39.3 Member Function Documentation

8.39.3.1 add_rule() [1/2]

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::add_rule (
            Rule< Array_Type, Data_Type > rule )  [inline]
```

Definition at line 66 of file rules-meat.hpp.

8.39.3.2 add_rule() [2/2]

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::add_rule (
            Rule_fun_type< Array_Type, Data_Type > rule_,
            Data_Type data_,
            std::string name_ = "",
            std::string description_ = "" )  [inline]
```

Definition at line 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()

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::get_seq (
            const Array_Type & a,
            std::vector< size_t > * free,
            std::vector< size_t > * locked = nullptr )  [inline]
```

Computes the sequence of free and locked cells in an BArray.

Parameters

a	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()()

```
template<typename Array_Type , typename Data_Type >
bool Rules< Array_Type, Data_Type >::operator() (
            const Array_Type & a,
            uint i,
            uint j )  [inline]
```

Check whether a given cell is free or locked.

Parameters

a	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=()

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type > Rules< Array_Type, Data_Type >::operator= (
            const Rules< Array_Type, Data_Type > & rules_ )
```

Definition at line 19 of file rules-meat.hpp.

8.39.3.8 size()

```
template<typename Array_Type , typename Data_Type >
uint 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 (uint i, uint j)
  Counter functions This function recurses through the entries of `Array` and at each step of adding a new cell it uses the functions to list the statistics.
- void count_current (uint i, uint j)
- std::vector< double > count_all ()
- Counters< Array_Type, Data_Type > * get_counters ()
- std::vector< std::string > get_names () const
- std::vector< std::string > get_descriptions () const
- size_t size () const

8.40.1 Detailed Description

template<typename Array_Type, typename Data_Type>
class StatsCounter< Array_Type, Data_Type >

Count stats for a single Array.

Users can a list of functions that can be used with this. The baseline set of arguments is a pointer to a binary array and a dataset to add the counts to.

Definition at line 14 of file statscounter-bones.hpp.

8.40.2 Constructor & Destructor Documentation

8.40.2.1 StatsCounter() [1/3]

```
template<typename Array_Type , typename Data_Type >
StatsCounter< Array_Type, Data_Type >::StatsCounter (
            const Array_Type * Array_ )  [inline]
```

Creator of a StatsCounter

Parameters

*Array_*	A const pointer to a BArray.

Definition at line 37 of file statscounter-bones.hpp.

8.40.2.2 StatsCounter() [2/3]

```
template<typename Array_Type , typename Data_Type >
StatsCounter< Array_Type, Data_Type >::StatsCounter (
            const StatsCounter< Array_Type, Data_Type > & counter )
```

Copy constructor.

Parameters

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()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::add_counter (
            Counter< Array_Type, Data_Type > f_ )
```

8.40.3.2 count_all()

```
template<typename Array_Type , typename Data_Type >
std::vector< double > StatsCounter< Array_Type, Data_Type >::count_all  [inline]
```

Definition at line 99 of file statscounter-meat.hpp.

8.40.3.3 count_current()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::count_current (
            uint i,
            uint j )
```

8.40.3.4 count_init()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::count_init (
            uint i,
            uint j )
```

Counter functions This function recurses through the entries of `Array` and at each step of adding a new cell it uses the functions to list the statistics.

8.40.3.5 get_counters()

```
template<typename Array_Type , typename Data_Type >
Counters<Array_Type,Data_Type>* StatsCounter< Array_Type, Data_Type >::get_counters ( )
```

8.40.3.6 get_descriptions()

```
template<typename Array_Type , typename Data_Type >
std::vector< std::string > StatsCounter< Array_Type, Data_Type >::get_descriptions ( ) const
```

8.40.3.7 get_names()

```
template<typename Array_Type , typename Data_Type >
std::vector< std::string > StatsCounter< Array_Type, Data_Type >::get_names ( ) const
```

8.40.3.8 reset_array()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::reset_array (
            const Array_Type * Array_ )
```

Changes the reference array for the counting.

Parameters

*Array_*	A pointer to an array of class `Array_Type`.

8.40.3.9 set_counters()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::set_counters (
            Counters< Array_Type, Data_Type > * counters_ )
```

8.40.3.10 size()

```
template<typename Array_Type , typename Data_Type >
size_t StatsCounter< Array_Type, Data_Type >::size ( ) const  [inline]
```

Definition at line 86 of file statscounter-bones.hpp.

The documentation for this class was generated from the following files:

- include/barry/statscounter-bones.hpp
- include/barry/statscounter-meat.hpp

8.41 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference

Compute the support of sufficient statistics.

`#include <support-bones.hpp>`

Public Member Functions

- Support (const Array_Type &Array_)

  Constructor passing a reference Array.
- Support (uint N_, uint M_)

  Constructor specifying the dimensions of the array (empty).
- Support ()
- ~Support ()
- void init_support (std::vector< Array_Type > *array_bank=nullptr, std::vector< double > *stats_bank=nullptr)
- void calc (std::vector< Array_Type > *array_bank=nullptr, std::vector< double > *stats_bank=nullptr, unsigned int 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 uint &i, const uint &j)

Public Attributes

- uint N
- uint M
- bool delete_counters = true
- bool delete_rules = true
- bool delete_rules_dyn = true
- uint max_num_elements = BARRY_MAX_NUM_ELEMENTS
- std::vector< double > current_stats
- std::vector< 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]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Support (
            const Array_Type & Array_ )  [inline]
```

Constructor passing a reference Array.

Definition at line 87 of file support-bones.hpp.

8.41.2.2 Support() [2/3]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Support (
            uint N_,
            uint M_ )  [inline]
```

Constructor specifying the dimensions of the array (empty).

Definition at line 96 of file support-bones.hpp.

8.41.2.3 Support() [3/3]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Support ( )
[inline]
```

Definition at line 103 of file support-bones.hpp.

8.41.2.4 ~Support()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::~Support ( )
[inline]
```

Definition at line 110 of file support-bones.hpp.

8.41.3 Member Function Documentation

8.41.3.1 add_counter()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_counter
(
            Counter< Array_Type, Data_Counter_Type > f_ )
```

8.41.3.2 add_rule() [1/2]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > * f_ )
```

8.41.3.3 add_rule() [2/2]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > f_ )
```

8.41.3.4 add_rule_dyn() [1/2]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_↩
dyn (
            Rule< Array_Type, Data_Rule_Dyn_Type > * f_ )
```

8.41.3.5 add_rule_dyn() [2/2]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_↩
dyn (
            Rule< Array_Type, Data_Rule_Dyn_Type > f_ )
```

8.41.3.6 calc()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::calc (
            std::vector< Array_Type > * array_bank = nullptr,
            std::vector< double > * stats_bank = nullptr,
            unsigned int max_num_elements_ = 0u )
```

Computes the entire support.

Not to be used by the user. Sets the starting point in the array (column-major).

Parameters

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 uint & i,
            const uint & j )
```

8.41.3.8 get_counters()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Counters<Array_Type,Data_Counter_Type>* Support< Array_Type, Data_Counter_Type, Data_Rule_↩
Type, Data_Rule_Dyn_Type >::get_counters ( )
```

Vector of 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()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::init_↩
support (
            std::vector< Array_Type > * array_bank = nullptr,
            std::vector< double > * stats_bank = nullptr )
```

8.41.3.15 print()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::print ( )
const
```

8.41.3.16 reset_array() [1/2]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::reset_array
( )
```

8.41.3.17 reset_array() [2/2]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::reset_array
(
            const Array_Type & Array_ )
```

8.41.3.18 set_counters()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_↩
counters (
            Counters< Array_Type, Data_Counter_Type > * counters_ )
```

8.41.3.19 set_rules()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_rules (
            Rules< Array_Type, Data_Rule_Type > * rules_ )
```

8.41.3.20 set_rules_dyn()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_rules↩
_dyn (
            Rules< Array_Type, Data_Rule_Dyn_Type > * rules_ )
```

8.41.4 Member Data Documentation

8.41.4.1 change_stats

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< double > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn↩
_Type >::change_stats
```

Definition at line 80 of file support-bones.hpp.

8.41.4.2 coordiantes_n_free

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::coordiantes↩
_n_free
```

Definition at line 78 of file support-bones.hpp.

8.41.4.3 coordiantes_n_locked

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::coordiantes↩
_n_locked
```

Definition at line 79 of file support-bones.hpp.

8.41.4.4 coordinates_free

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< size_t > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn↩
_Type >::coordinates_free
```

Definition at line 76 of file support-bones.hpp.

8.41.4.5 coordinates_locked

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< size_t > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn↩
_Type >::coordinates_locked
```

Definition at line 77 of file support-bones.hpp.

8.41.4.6 current_stats

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< double > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn↩
_Type >::current_stats
```

Definition at line 75 of file support-bones.hpp.

8.41.4.7 delete_counters

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↩
counters = true
```

Definition at line 69 of file support-bones.hpp.

8.41.4.8 delete_rules

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↩
rules = true
```

Definition at line 70 of file support-bones.hpp.

8.41.4.9 delete_rules_dyn

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↩
rules_dyn = true
```

Definition at line 71 of file support-bones.hpp.

8.41.4.10 hashes

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< size_t > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn↩
_Type >::hashes
```

Definition at line 81 of file support-bones.hpp.

8.41.4.11 hashes_initialized

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< bool > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_↩
Type >::hashes_initialized
```

Definition at line 82 of file support-bones.hpp.

8.41.4.12 M

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint 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>
uint 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>
uint 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 106 of file typedefs.hpp.

8.42.2 Member Function Documentation

8.42.2.1 operator()()

```
template<typename T >
std::size_t vecHasher< T >::operator() (
            std::vector< T > const & dat ) const  [inline], [noexcept]
```

Definition at line 109 of file typedefs.hpp.

The documentation for this struct was generated from the following file:

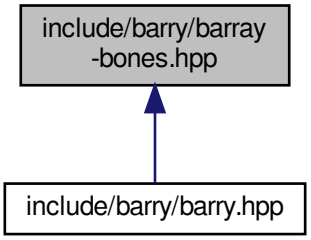
- include/barry/typedefs.hpp

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.

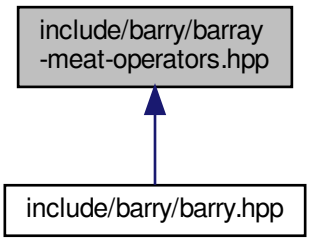
9.2 include/barry/barray-iterator.hpp File Reference

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 (uint i=0u;i< nrow();++i) for(uint j=0u = el[POS(i, j)]
- j< ncol();++j) this-> operator() (i, j)+
- BARRAY_TEMPLATE (BARRAY_TYPE()&, operator+=)(const Cell_Type &rhs)
- BARRAY_TEMPLATE (BARRAY_TYPE()&, operator-=)(const BArray< Cell_Type
- BARRAY_TEMPLATE (BARRAY_TYPE()&, operator-=)(const Cell_Type &rhs)
- BARRAY_TEMPLATE (BARRAY_TYPE()&, operator*=)(const Cell_Type &rhs)
- BARRAY_TEMPLATE (BARRAY_TYPE()&, operator/=)(const Cell_Type &rhs)

Variables

- Data_Type & rhs
- return * this

9.3.1 Macro Definition Documentation

9.3.1.1 BARRAY_TEMPLATE

```
#define BARRAY_TEMPLATE(
              a,
              b )  template BARRAY_TEMPLATE_ARGS() inline a BARRAY_TYPE()::b
```

Definition at line 11 of file barray-meat-operators.hpp.

9.3.1.2 BARRAY_TEMPLATE_ARGS

```
template BARRAY_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

Definition at line 9 of file barray-meat-operators.hpp.

9.3.1.3 BARRAY_TYPE

```
template Data_Type BARRAY_TYPE( ) BArray<Cell_Type, Data_Type>
```

Definition at line 7 of file barray-meat-operators.hpp.

9.3.1.4 COL

```
#define COL(
              a ) this->el_ji[a]
```

Definition at line 15 of file barray-meat-operators.hpp.

9.3.1.5 ROW

```
#define ROW(
              a ) this->el_ij[a]
```

Definition at line 14 of file barray-meat-operators.hpp.

9.3.2 Function Documentation

9.3.2.1 BARRAY_TEMPLATE() [1/6]

```
BARRAY_TEMPLATE (
            BARRAY_TYPE()& ,
            operator*  ) const &
```

Definition at line 88 of file barray-meat-operators.hpp.

9.3.2.2 BARRAY_TEMPLATE() [2/6]

```
BARRAY_TEMPLATE (
            BARRAY_TYPE()& ,
            operator+  ) const
```

9.3.2.3 BARRAY_TEMPLATE() [3/6]

```
BARRAY_TEMPLATE (
            BARRAY_TYPE()& ,
            operator+  ) const &
```

Definition at line 46 of file barray-meat-operators.hpp.

9.3.2.4 BARRAY_TEMPLATE() [4/6]

```
BARRAY_TEMPLATE (
            BARRAY_TYPE()& ,
            operator-  ) const
```

9.3.2.5 BARRAY_TEMPLATE() [5/6]

```
BARRAY_TEMPLATE (
            BARRAY_TYPE()& ,
            operator-  ) const &
```

Definition at line 75 of file barray-meat-operators.hpp.

9.3.2.6 BARRAY_TEMPLATE() [6/6]

```
BARRAY_TEMPLATE (
              BARRAY_TYPE()& ,
              operator/  ) const &
```

Definition at line 105 of file barray-meat-operators.hpp.

9.3.2.7 BARRAY_TEMPLATE_ARGS()

```
template BARRAY_TEMPLATE_ARGS ( ) const &
```

9.3.2.8 BARRAY_TYPE()

```
template const BARRAY_TYPE ( ) &
```

Definition at line 20 of file barray-meat-operators.hpp.

9.3.2.9 for()

```
for ( ) = el[POS(i, j)]  [pure virtual]
```

Definition at line 66 of file barray-meat-operators.hpp.

9.3.2.10 operator()()

```
j< ncol(); ++j) this-> operator() (
              i ,
              j  )
```

9.3.3 Variable Documentation

9.3.3.1 rhs

```
Data_Type & rhs
```

Initial value:

```
{
    checkdim_(*this, rhs)
```

Definition at line 33 of file barray-meat-operators.hpp.

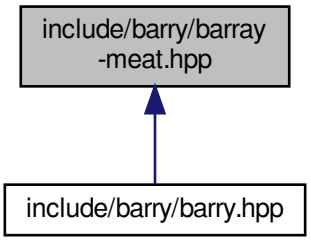
9.3.3.2 this

```
return * this
```

Definition at line 43 of file barray-meat-operators.hpp.

9.4 include/barry/barray-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define BARRAY_TYPE() BArray<Cell_Type, Data_Type>
- #define BARRAY_TEMPLATE_ARGS() <typename Cell_Type, typename Data_Type>
- #define BARRAY_TEMPLATE(a, b) template BARRAY_TEMPLATE_ARGS() inline a BARRAY_TYPE()::b
- #define ROW(a) this->el_ij[a]
- #define COL(a) this->el_ji[a]

Functions

- BARRAY_TEMPLATE (, BArray)(uint N_
- el_ij resize (N)
- el_ji resize (M)
- for (uint i=0u;i< source.size();++i)
- Data_Type bool M (Array_.M)
- BARRAY_TEMPLATE (BARRAY_TYPE() &, operator=)(const BArray< Cell_Type
- BARRAY_TEMPLATE (, BArray)(BARRAY_TYPE() &&x) noexcept
- BARRAY_TEMPLATE (BARRAY_TYPE() &, operator=)(BARRAY_TYPE() &&x) noexcept
- BARRAY_TEMPLATE (bool, operator==)(const BARRAY_TYPE() &Array_)
- BARRAY_TEMPLATE (,~BArray)()
- BARRAY_TEMPLATE (void, set_data)(Data_Type *data_
- BARRAY_TEMPLATE (Data_Type *, D_ptr)()
- BARRAY_TEMPLATE (Data_Type &, D)()
- BARRAY_TEMPLATE (void, out_of_range)(uint i
- BARRAY_TEMPLATE (Cell_Type, get_cell)(uint i
- if (ROW(i).size()==0u) return(Cell_Type) 0.0
- if (search !=ROW(i).end()) return search -> second.value
- return (Cell_Type) 0.0
- BARRAY_TEMPLATE (std::vector< Cell_Type >, get_row_vec)(uint 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)(uint i
- if (check_exists)
- COL (j).emplace(i
- & ROW (i)[j])
- BARRAY_TEMPLATE (void, swap_cells)(uint i0
- if (report !=nullptr)(*report)
- if (check0 &check1)
- else if (!check0 &check1)
- else if (check0 &!check1)
- BARRAY_TEMPLATE (void, toggle_cell)(uint i
- BARRAY_TEMPLATE (void, swap_rows)(uint 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)(uint 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)(uint i
- for (auto row=row0.begin();row !=row0.end();++row) rm_cell(i
- BARRAY_TEMPLATE (void, zero_col)(uint j
- if (COL(j).size()==0u) return
- BARRAY_TEMPLATE (void, transpose)()
- BARRAY_TEMPLATE (void, clear)(bool hard)
- BARRAY_TEMPLATE (void, resize)(uint N_
- if (M_< M) for(uint j = N_

Variables

- uint M_
- uint const std::vector< uint > & source
- uint const std::vector< uint > const std::vector< uint > & target
- uint const std::vector< uint > const std::vector< uint > const std::vector< Cell_Type > & value
- uint const std::vector< uint > const std::vector< uint > 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_
- uint j const
- uint j
- auto search = ROW(i).find(j)
- return ans
- uint const Cell< Cell_Type > & v
- uint const Cell< Cell_Type > bool check_bounds
- uint const Cell< Cell_Type > bool bool check_exists
- else
- NCells
- uint j0
- uint uint i1
- uint uint uint j1
- uint uint uint bool int int * report
- auto row0 = ROW(i)
- row first
- row false
- auto col0 = COL(j)

9.4.1 Macro Definition Documentation

9.4.1.1 BARRAY_TEMPLATE

```
#define BARRAY_TEMPLATE(
              a,
              b )  template BARRAY_TEMPLATE_ARGS() inline a BARRAY_TYPE()::b
```

Definition at line 17 of file barray-meat.hpp.

9.4.1.2 BARRAY_TEMPLATE_ARGS

```
#define BARRAY_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

Definition at line 15 of file barray-meat.hpp.

9.4.1.3 BARRAY_TYPE

```
#define BARRAY_TYPE( ) BArray<Cell_Type, Data_Type>
```

Definition at line 13 of file barray-meat.hpp.

9.4.1.4 COL

```
#define COL(
              a ) this->el_ji[a]
```

Definition at line 21 of file barray-meat.hpp.

9.4.1.5 ROW

```
#define ROW(
              a ) this->el_ij[a]
```

Definition at line 20 of file barray-meat.hpp.

9.4.2 Function Documentation

9.4.2.1 ans()

```
std::vector< Cell_Type > ans (
             ncol() ,
             (Cell_Type) false )
```

9.4.2.2 BARRAY_TEMPLATE() [1/24]

```
BARRAY_TEMPLATE (
             BArray  ) &&  [noexcept]
```

Definition at line 230 of file barray-meat.hpp.

9.4.2.3 BARRAY_TEMPLATE() [2/24]

```
BARRAY_TEMPLATE (
            BArray  )
```

9.4.2.4 BARRAY_TEMPLATE() [3/24]

```
BARRAY_TEMPLATE (
            ~ BArray )
```

Definition at line 339 of file barray-meat.hpp.

9.4.2.5 BARRAY_TEMPLATE() [4/24]

```
BARRAY_TEMPLATE (
            BARRAY_TYPE() & ,
            operator-  ) const
```

Definition at line 597 of file barray-meat.hpp.

9.4.2.6 BARRAY_TEMPLATE() [5/24]

```
BARRAY_TEMPLATE (
            BARRAY_TYPE() & ,
            operator  ) &&  [noexcept]
```

Definition at line 272 of file barray-meat.hpp.

9.4.2.7 BARRAY_TEMPLATE() [6/24]

```
BARRAY_TEMPLATE (
            BARRAY_TYPE() & ,
            operator  ) const
```

9.4.2.8 BARRAY_TEMPLATE() [7/24]

```
BARRAY_TEMPLATE (
            bool ,
            operator  = = ) const &
```

Definition at line 321 of file barray-meat.hpp.

9.4.2.9 BARRAY_TEMPLATE() [8/24]

```
BARRAY_TEMPLATE (
            Cell_Type ,
            get_cell  )
```

9.4.2.10 BARRAY_TEMPLATE() [9/24]

```
BARRAY_TEMPLATE (
            Data_Type & ,
            D  )
```

Definition at line 372 of file barray-meat.hpp.

9.4.2.11 BARRAY_TEMPLATE() [10/24]

```
BARRAY_TEMPLATE (
            Data_Type * ,
            D_ptr  )
```

Definition at line 361 of file barray-meat.hpp.

9.4.2.12 BARRAY_TEMPLATE() [11/24]

```
BARRAY_TEMPLATE (
            std::vector< Cell_Type > ,
            get_row_vec  )
```

9.4.2.13 BARRAY_TEMPLATE() [12/24]

```
BARRAY_TEMPLATE (
            void ,
            clear  )
```

Definition at line 1130 of file barray-meat.hpp.

9.4.2.14 BARRAY_TEMPLATE() [13/24]

```
BARRAY_TEMPLATE (
            void ,
            get_row_vec  )
```

Definition at line 452 of file barray-meat.hpp.

9.4.2.15 BARRAY_TEMPLATE() [14/24]

```
BARRAY_TEMPLATE (
            void ,
            insert_cell  )
```

9.4.2.16 BARRAY_TEMPLATE() [15/24]

```
BARRAY_TEMPLATE (
            void ,
            out_of_range  )
```

9.4.2.17 BARRAY_TEMPLATE() [16/24]

```
BARRAY_TEMPLATE (
            void ,
            resize  )
```

9.4.2.18 BARRAY_TEMPLATE() [17/24]

```
BARRAY_TEMPLATE (
            void ,
            set_data  )
```

9.4.2.19 BARRAY_TEMPLATE() [18/24]

```
BARRAY_TEMPLATE (
            void ,
            swap_cells  )
```

9.4.2.20 BARRAY_TEMPLATE() [19/24]

```
BARRAY_TEMPLATE (
            void ,
            swap_cols  )
```

9.4.2.21 BARRAY_TEMPLATE() [20/24]

```
BARRAY_TEMPLATE (
            void ,
            swap_rows  )
```

9.4.2.22 BARRAY_TEMPLATE() [21/24]

```
BARRAY_TEMPLATE (
            void ,
            toggle_cell  )
```

9.4.2.23 BARRAY_TEMPLATE() [22/24]

```
BARRAY_TEMPLATE (
            void ,
            transpose  )
```

Definition at line 1069 of file barray-meat.hpp.

9.4.2.24 BARRAY_TEMPLATE() [23/24]

```
BARRAY_TEMPLATE (
            void ,
            zero_col  )
```

9.4.2.25 BARRAY_TEMPLATE() [24/24]

```
BARRAY_TEMPLATE (
            void ,
            zero_row  )
```

9.4.2.26 COL()

```
COL (
              j  )
```

9.4.2.27 for() [1/3]

```
for (
              auto row = row0.begin();row !=row0.end();++row )
```

9.4.2.28 for() [2/3]

```
for (
              const auto &iter  :rowi, false )
```

9.4.2.29 for() [3/3]

```
for ( )
```

Definition at line 51 of file barray-meat.hpp.

9.4.2.30 if() [1/17]

```
else if (
              !check0 && check1 )
```

Definition at line 1008 of file barray-meat.hpp.

9.4.2.31 if() [2/17]

```
else if (
              !check0 & check1 )
```

Definition at line 856 of file barray-meat.hpp.

9.4.2.32 if() [3/17]

```
if (
              !move0 &&!  move1 )
```

9.4.2.33 if() [4/17]

```
else if (
              check0 &!  check1 )
```

Definition at line 864 of file barray-meat.hpp.

9.4.2.34 if() [5/17]

```
else if (
              check0 &&!  check1 )
```

Definition at line 999 of file barray-meat.hpp.

9.4.2.35 if() [6/17]

```
if (
              check0 && check1 )
```

Definition at line 972 of file barray-meat.hpp.

9.4.2.36 if() [7/17]

```
if (
              check0 & check1 )
```

Definition at line 838 of file barray-meat.hpp.

9.4.2.37 if() [8/17]

```
else if (
              check_exists  = = CHECK::BOTH )
```

Definition at line 679 of file barray-meat.hpp.

9.4.2.38 if() [9/17]

```
if (
             COL(j).size()  = =0u )
```

9.4.2.39 if() [10/17]

```
if (
             COL(j0).size()  = =0u )
```

9.4.2.40 if() [11/17]

```
if (
             COL(j1).size()  = =0u )
```

9.4.2.41 if() [12/17]

```
else if ( ) = N_
```

Definition at line 86 of file barray-meat.hpp.

9.4.2.42 if() [13/17]

```
if (
             report !    = nullptr )
```

9.4.2.43 if() [14/17]

```
if (
             ROW(i).size()  = =0u )
```

9.4.2.44 if() [15/17]

```
if (
             ROW(i0).size()  = =0u )
```

9.4.2.45 if() [16/17]

```
if (
             ROW(i1).size()   = =0u )
```

9.4.2.46 if() [17/17]

```
if (
             search !     = ROW(i).end() ) -> second.value
```

9.4.2.47 M()

```
Data_Type bool M (
             Array_.  M )
```

Definition at line 136 of file barray-meat.hpp.

9.4.2.48 resize() [1/2]

```
el_ji resize (
             M  )
```

9.4.2.49 resize() [2/2]

```
el_ij resize (
             N  )
```

9.4.2.50 return()

```
return (
             Cell_Type  )
```

9.4.2.51 ROW() [1/2]

```
& ROW (
              i  )
```

9.4.2.52 ROW() [2/2]

```
ROW (
              i0  )
```

9.4.3 Variable Documentation

9.4.3.1 add

```
uint const std::vector< uint > const std::vector< uint > 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

```
uint 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

```
uint bool check_bounds const
```

Initial value:

```
{
    if (i >= N)
        throw std::range_error("The row is out of range.")
```

Definition at line 402 of file barray-meat.hpp.

9.4.3.8 copy_data

```
Data_Type bool copy_data
```

Definition at line 135 of file barray-meat.hpp.

9.4.3.9 data

```
data = data_
```

Definition at line 354 of file barray-meat.hpp.

9.4.3.10 delete_data

```
delete_data = delete_data_
```

Definition at line 355 of file barray-meat.hpp.

9.4.3.11 delete_data_

```
bool delete_data_
```

Initial value:

```
{
    if ((data != nullptr) && delete_data)
        delete data
```

Definition at line 348 of file barray-meat.hpp.

9.4.3.12 else

```
else (
             void  )
```

Initial value:

```
{
        ROW(i).insert(std::pair< uint, Cell<Cell_Type>>(j, v))
```

Definition at line 703 of file barray-meat.hpp.

9.4.3.13 false

```
row false
```

Definition at line 1042 of file barray-meat.hpp.

9.4.3.14 first

```
row first
```

Definition at line 1042 of file barray-meat.hpp.

9.4.3.15 i1

```
uint i1
```

Definition at line 776 of file barray-meat.hpp.

9.4.3.16 j

```
uint j
```

Initial value:

```
{
    if (init_fun == nullptr)
        return 0.0
```

Definition at line 414 of file barray-meat.hpp.

9.4.3.17 j0

```
uint j0
```

Definition at line 775 of file barray-meat.hpp.

9.4.3.18 j1

```
uint j1
```

Definition at line 776 of file barray-meat.hpp.

9.4.3.19 M

```
M = M_
```

Definition at line 44 of file barray-meat.hpp.

9.4.3.20 M_

```
uint M_
```

Initial value:

```
{

    if (N_ < N)
        for (uint i = N_; i < N; ++i)
            zero_row(i, false)
```

Definition at line 30 of file barray-meat.hpp.

9.4.3.21 N

```
if (source.size() != target.size()) throw std if (source.size() != value.size()) throw std N =
N_
```

Definition at line 43 of file barray-meat.hpp.

9.4.3.22 NCells

```
NCells
```

Definition at line 707 of file barray-meat.hpp.

9.4.3.23 report

```
uint uint uint 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

```
uint const std::vector< uint > & source
```

Definition at line 31 of file barray-meat.hpp.

9.4.3.28 target

```
uint const std::vector< uint > const std::vector< uint > & target
```

Definition at line 32 of file barray-meat.hpp.

9.4.3.29 v

```
uint Cell_Type v
```

Definition at line 671 of file barray-meat.hpp.

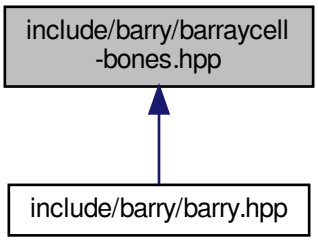
9.4.3.30 value

```
uint const std::vector< uint > const std::vector< uint > const std::vector< Cell_Type >&
value
```

Definition at line 33 of file barray-meat.hpp.

9.5 include/barry/barraycell-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:

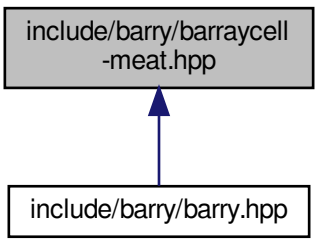


Classes

- class BArrayCell< Cell_Type, Data_Type >
- class BArrayCell_const< Cell_Type, Data_Type >

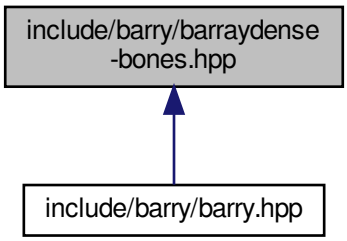
9.6 include/barry/barraycell-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



9.7 include/barry/barraydense-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



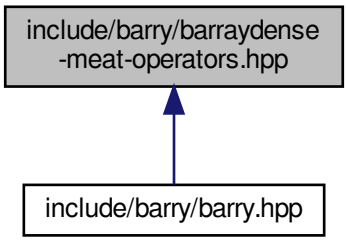
Classes

- class BArrayDense< Cell_Type, Data_Type >

  Baseline class for binary arrays.

9.8 include/barry/barraydense-meat-operators.hpp File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define BDENSE_TYPE() BArrayDense<Cell_Type, Data_Type>
- #define BDENSE_TEMPLATE_ARGS() <typename Cell_Type, typename Data_Type>
- #define BDENSE_TEMPLATE(a, b) template BDENSE_TEMPLATE_ARGS() inline a BDENSE_TYPE()::b
- #define ROW(a) this->el_ij[a]
- #define COL(a) this->el_ji[a]
- #define POS(a, b) (b)*N + (a)
- #define POS_N(a, b, c) (b)*(c) + (a)

Functions

- template BDENSE_TEMPLATE_ARGS () inline void checkdim_(const BDENSE_TYPE() &lhs
- template const BDENSE_TYPE () &rhs)
- BDENSE_TEMPLATE (BDENSE_TYPE()&, operator+=)(const BDENSE_TYPE() &rhs)
- BDENSE_TEMPLATE (BDENSE_TYPE()&, operator-=)(const BDENSE_TYPE() &rhs)
- BDENSE_TEMPLATE (BDENSE_TYPE()&, operator∗=)(const Cell_Type &rhs)
- BDENSE_TEMPLATE (BDENSE_TYPE()&, operator/=)(const Cell_Type &rhs)

9.8.1 Macro Definition Documentation

9.8.1.1 BDENSE_TEMPLATE

```
#define BDENSE_TEMPLATE(
              a,
              b )  template BDENSE_TEMPLATE_ARGS() inline a BDENSE_TYPE()::b
```

Definition at line 11 of file barraydense-meat-operators.hpp.

9.8.1.2 BDENSE_TEMPLATE_ARGS

```
template BDENSE_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

Definition at line 9 of file barraydense-meat-operators.hpp.

9.8.1.3 BDENSE_TYPE

```
template Data_Type BDENSE_TYPE( ) BArrayDense<Cell_Type, Data_Type>
```

Definition at line 7 of file barraydense-meat-operators.hpp.

9.8.1.4 COL

```
#define COL(
              a ) this->el_ji[a]
```

Definition at line 15 of file barraydense-meat-operators.hpp.

9.8.1.5 POS

```
#define POS(
              a,
              b ) (b)*N + (a)
```

Definition at line 16 of file barraydense-meat-operators.hpp.

9.8.1.6 POS_N

```
#define POS_N(
              a,
              b,
              c ) (b)*(c) + (a)
```

Definition at line 17 of file barraydense-meat-operators.hpp.

9.8.1.7 ROW

```
#define ROW(
              a ) this->el_ij[a]
```

Definition at line 14 of file barraydense-meat-operators.hpp.

9.8.2 Function Documentation

9.8.2.1 BDENSE_TEMPLATE() [1/4]

```
BDENSE_TEMPLATE (
             BDENSE_TYPE()& ,
             operator*  ) const &
```

Definition at line 90 of file barraydense-meat-operators.hpp.

9.8.2.2 BDENSE_TEMPLATE() [2/4]

```
BDENSE_TEMPLATE (
             BDENSE_TYPE()& ,
             operator+  ) const &
```

Definition at line 34 of file barraydense-meat-operators.hpp.

9.8.2.3 BDENSE_TEMPLATE() [3/4]

```
BDENSE_TEMPLATE (
            BDENSE_TYPE()& ,
            operator-  ) const &
```

Definition at line 61 of file barraydense-meat-operators.hpp.

9.8.2.4 BDENSE_TEMPLATE() [4/4]

```
BDENSE_TEMPLATE (
            BDENSE_TYPE()& ,
            operator/  ) const &
```

Definition at line 101 of file barraydense-meat-operators.hpp.

9.8.2.5 BDENSE_TEMPLATE_ARGS()

```
template BDENSE_TEMPLATE_ARGS ( ) const &
```

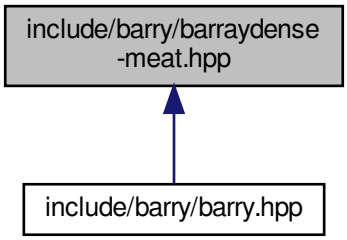
9.8.2.6 BDENSE_TYPE()

```
template const BDENSE_TYPE ( ) &
```

Definition at line 22 of file 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)(uint N_
- el resize (N *M, ZERO_CELL)
- el_rowsums resize (N, ZERO_CELL)
- el_colsums resize (M, ZERO_CELL)
- for (uint i=0u;i< source.size();++i)
- BDENSE_TEMPLATE (, BArrayDense)(const BDENSE_TYPE() &Array_
- bool M (Array_.M)
- BDENSE_TEMPLATE (BDENSE_TYPE() &, operator=)(const BDENSE_TYPE() &Array_)
- BDENSE_TEMPLATE (, BArrayDense)(BDENSE_TYPE() &&x) noexcept
- BDENSE_TEMPLATE (BDENSE_TYPE() &, operator=)(BDENSE_TYPE() &&x) noexcept
- BDENSE_TEMPLATE (bool, operator==)(const BDENSE_TYPE() &Array_)
- BDENSE_TEMPLATE (, ~BArrayDense)()
- BDENSE_TEMPLATE (void, set_data)(Data_Type *data_
- BDENSE_TEMPLATE (Data_Type *, D_ptr)()
- BDENSE_TEMPLATE (const Data_Type *, D_ptr)() const
- BDENSE_TEMPLATE (Data_Type &, D)()
- BDENSE_TEMPLATE (const Data_Type &, D)() const
- BDENSE_TEMPLATE (void, out_of_range)(uint i
- BDENSE_TEMPLATE (Cell_Type, get_cell)(uint i
- BDENSE_TEMPLATE (std::vector< Cell_Type >, get_row_vec)(uint 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)(uint i
- BDENSE_TEMPLATE (unsigned int, nrow)() const noexcept
- BDENSE_TEMPLATE (unsigned int, ncol)() const noexcept
- BDENSE_TEMPLATE (unsigned int, nnozero)() const noexcept
- BDENSE_TEMPLATE (Cell< Cell_Type >, default_val)() const
- BDENSE_TEMPLATE (BDENSE_TYPE() &, operator+=)(const std
- BDENSE_TEMPLATE (BDENSE_TYPE() &, operator-=)(const std
- BDENSE_TEMPLATE (void, insert_cell)(uint i
- if (el[POS(i, j)]==BARRY_ZERO_DENSE)
- BDENSE_TEMPLATE (void, swap_cells)(uint 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)(uint i
- else rm_cell (i, j, false, false)
- BDENSE_TEMPLATE (void, swap_rows)(uint i0

- BDENSE_TEMPLATE (void, swap_cols)(uint j0
- BDENSE_TEMPLATE (void, zero_row)(uint i
- if (el_rowsums[i]==ZERO_CELL) return
- BDENSE_TEMPLATE (void, zero_col)(uint j
- if (el_colsums[j]==ZERO_CELL) return
- BDENSE_TEMPLATE (void, transpose)()
- BDENSE_TEMPLATE (void, clear)(bool hard)
- BDENSE_TEMPLATE (void, resize)(uint 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)
- vprintf (fmt, args)
- va_end (args)
- BDENSE_TEMPLATE (const std::vector< Cell_Type > &, get_data)() const
- BDENSE_TEMPLATE (const Cell_Type, rowsum)(unsigned int i) const
- BDENSE_TEMPLATE (const Cell_Type, colsum)(unsigned int j) const

Variables

- uint M_
- uint const std::vector< uint > & source
- uint const std::vector< uint > const std::vector< uint > & target
- uint const std::vector< uint > const std::vector< uint > const std::vector< Cell_Type > & value
- uint const std::vector< uint > const std::vector< uint > 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_
- uint j const
- uint j
- return el [POS(i, j)] == ZERO_CELL
- return ans
- uint const Cell< Cell_Type > & v
- uint const Cell< Cell_Type > bool check_bounds
- uint const Cell< Cell_Type > bool bool check_exists
- else
- el_rowsums [i] = (v.value - old)
- el_colsums [j] = (v.value - old)
- uint j0
- uint uint i1
- uint uint uint j1
- uint uint uint bool int int ∗ report
- Cell_Type val0 = el[POS(i0,j0)]
- Cell_Type val1 = el[POS(i1,j1)]
- false
- col

9.9.1 Macro Definition Documentation

9.9.1.1 BDENSE_TEMPLATE

```
#define BDENSE_TEMPLATE(
            a,
            b )  template BDENSE_TEMPLATE_ARGS() inline a BDENSE_TYPE()::b
```

Definition at line 27 of file barraydense-meat.hpp.

9.9.1.2 BDENSE_TEMPLATE_ARGS

```
#define BDENSE_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

Definition at line 25 of file barraydense-meat.hpp.

9.9.1.3 BDENSE_TYPE

```
#define BDENSE_TYPE( ) BArrayDense<Cell_Type, Data_Type>
```

Definition at line 23 of file barraydense-meat.hpp.

9.9.1.4 COL

```
#define COL(
            a ) this->el_ji[a]
```

Definition at line 31 of file barraydense-meat.hpp.

9.9.1.5 POS

```
#define POS(
            a,
            b ) (b)*N + (a)
```

Definition at line 32 of file barraydense-meat.hpp.

9.9.1.6 POS_N

```
#define POS_N(
              a,
              b,
              c ) (b)*(c) + (a)
```

Definition at line 33 of file barraydense-meat.hpp.

9.9.1.7 ROW

```
#define ROW(
              a ) this->el_ij[a]
```

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()

```
std::vector< Cell_Type > ans (
             ncol() ,
             static_cast< Cell_Type > false )
```

9.9.2.2 BDENSE_TEMPLATE() [1/39]

```
BDENSE_TEMPLATE (
             BArrayDense  ) &&  [noexcept]
```

Definition at line 240 of file barraydense-meat.hpp.

9.9.2.3 BDENSE_TEMPLATE() [2/39]

```
BDENSE_TEMPLATE (
            BArrayDense  ) const &
```

9.9.2.4 BDENSE_TEMPLATE() [3/39]

```
BDENSE_TEMPLATE (
            BArrayDense  )
```

9.9.2.5 BDENSE_TEMPLATE() [4/39]

```
BDENSE_TEMPLATE (
            ~ BArrayDense )
```

Definition at line 318 of file barraydense-meat.hpp.

9.9.2.6 BDENSE_TEMPLATE() [5/39]

```
BDENSE_TEMPLATE (
            BDENSE_TYPE() & ,
            operator+  ) const
```

Definition at line 566 of file barraydense-meat.hpp.

9.9.2.7 BDENSE_TEMPLATE() [6/39]

```
BDENSE_TEMPLATE (
            BDENSE_TYPE() & ,
            operator-  ) const
```

Definition at line 584 of file barraydense-meat.hpp.

9.9.2.8 BDENSE_TEMPLATE() [7/39]

```
BDENSE_TEMPLATE (
            BDENSE_TYPE() & ,
            operator  ) &&  [noexcept]
```

Definition at line 257 of file barraydense-meat.hpp.

9.9.2.9 BDENSE_TEMPLATE() [8/39]

```
BDENSE_TEMPLATE (
            BDENSE_TYPE() & ,
            operator  ) const &
```

Definition at line 194 of file 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]

```
BDENSE_TEMPLATE (
            Cell< Cell_Type > ,
            default_val  ) const
```

Definition at line 562 of file barraydense-meat.hpp.

9.9.2.13 BDENSE_TEMPLATE() [12/39]

```
BDENSE_TEMPLATE (
            Cell_Type ,
            get_cell  )
```

9.9.2.14 BDENSE_TEMPLATE() [13/39]

```
BDENSE_TEMPLATE (
            const Cell_Type,
            colsum  ) const
```

Definition at line 999 of file barraydense-meat.hpp.

9.9.2.15 BDENSE_TEMPLATE() [14/39]

```
BDENSE_TEMPLATE (
            const Cell_Type,
            rowsum  ) const
```

Definition at line 994 of file barraydense-meat.hpp.

9.9.2.16 BDENSE_TEMPLATE() [15/39]

```
BDENSE_TEMPLATE (
            const Data_Type & ,
            D  ) const
```

Definition at line 353 of file barraydense-meat.hpp.

9.9.2.17 BDENSE_TEMPLATE() [16/39]

```
BDENSE_TEMPLATE (
            const Data_Type * ,
            D_ptr  ) const
```

Definition at line 345 of file barraydense-meat.hpp.

9.9.2.18 BDENSE_TEMPLATE() [17/39]

```
BDENSE_TEMPLATE (
            const std::vector< Cell_Type > & ,
            get_data  ) const
```

Definition at line 989 of file barraydense-meat.hpp.

9.9.2.19 BDENSE_TEMPLATE() [18/39]

```
BDENSE_TEMPLATE (
            Data_Type & ,
            D  )
```

Definition at line 349 of file barraydense-meat.hpp.

9.9.2.20 BDENSE_TEMPLATE() [19/39]

```
BDENSE_TEMPLATE (
            Data_Type * ,
            D_ptr  )
```

Definition at line 341 of file barraydense-meat.hpp.

9.9.2.21 BDENSE_TEMPLATE() [20/39]

```
BDENSE_TEMPLATE (
            Entries< Cell_Type > ,
            get_entries  ) const
```

Definition at line 502 of file barraydense-meat.hpp.

9.9.2.22 BDENSE_TEMPLATE() [21/39]

```
BDENSE_TEMPLATE (
            std::vector< Cell_Type > ,
            get_row_vec  )
```

9.9.2.23 BDENSE_TEMPLATE() [22/39]

```
BDENSE_TEMPLATE (
            unsigned int ,
            ncol  ) const  [noexcept]
```

Definition at line 548 of file barraydense-meat.hpp.

9.9.2.24 BDENSE_TEMPLATE() [23/39]

```
BDENSE_TEMPLATE (
            unsigned int ,
            nnozero  ) const  [noexcept]
```

Definition at line 552 of file barraydense-meat.hpp.

9.9.2.25 BDENSE_TEMPLATE() [24/39]

```
BDENSE_TEMPLATE (
            unsigned int ,
            nrow  ) const  [noexcept]
```

Definition at line 544 of file barraydense-meat.hpp.

9.9.2.26 BDENSE_TEMPLATE() [25/39]

```
BDENSE_TEMPLATE (
            void ,
            clear  )
```

Definition at line 896 of file barraydense-meat.hpp.

9.9.2.27 BDENSE_TEMPLATE() [26/39]

```
BDENSE_TEMPLATE (
            void ,
            get_row_vec  )
```

Definition at line 402 of file barraydense-meat.hpp.

9.9.2.28 BDENSE_TEMPLATE() [27/39]

```
BDENSE_TEMPLATE (
            void ,
            insert_cell  )
```

9.9.2.29 BDENSE_TEMPLATE() [28/39]

```
BDENSE_TEMPLATE (
             void ,
             out_of_range  )
```

9.9.2.30 BDENSE_TEMPLATE() [29/39]

```
BDENSE_TEMPLATE (
             void ,
             print  ) const
```

9.9.2.31 BDENSE_TEMPLATE() [30/39]

```
BDENSE_TEMPLATE (
             void ,
             reserve  )
```

Definition at line 946 of file barraydense-meat.hpp.

9.9.2.32 BDENSE_TEMPLATE() [31/39]

```
BDENSE_TEMPLATE (
             void ,
             resize  )
```

9.9.2.33 BDENSE_TEMPLATE() [32/39]

```
BDENSE_TEMPLATE (
             void ,
             set_data  )
```

9.9.2.34 BDENSE_TEMPLATE() [33/39]

```
BDENSE_TEMPLATE (
             void ,
             swap_cells  )
```

9.9.2.35 BDENSE_TEMPLATE() [34/39]

```
BDENSE_TEMPLATE (
            void ,
            swap_cols  )
```

9.9.2.36 BDENSE_TEMPLATE() [35/39]

```
BDENSE_TEMPLATE (
            void ,
            swap_rows  )
```

9.9.2.37 BDENSE_TEMPLATE() [36/39]

```
BDENSE_TEMPLATE (
            void ,
            toggle_cell  )
```

9.9.2.38 BDENSE_TEMPLATE() [37/39]

```
BDENSE_TEMPLATE (
            void ,
            transpose  )
```

Definition at line 868 of file barraydense-meat.hpp.

9.9.2.39 BDENSE_TEMPLATE() [38/39]

```
BDENSE_TEMPLATE (
            void ,
            zero_col  )
```

9.9.2.40 BDENSE_TEMPLATE() [39/39]

```
BDENSE_TEMPLATE (
            void ,
            zero_row  )
```

9.9.2.41 for()

```
for ( )
```

Definition at line 64 of file barraydense-meat.hpp.

9.9.2.42 if() [1/4]

```
if (
              (i0==i1) &&(j0==j1)  )
```

9.9.2.43 if() [2/4]

```
if (
              el [POS(i, j)] = = BARRY_ZERO_DENSE )
```

Definition at line 663 of file barraydense-meat.hpp.

9.9.2.44 if() [3/4]

```
if (
              el_colsums [j] = =ZERO_CELL )
```

9.9.2.45 if() [4/4]

```
if (
              el_rowsums [i] = =ZERO_CELL )
```

9.9.2.46 insert_cell() [1/2]

```
insert_cell (
              i0 ,
              j0 ,
              val1 ,
              false ,
              false  )
```

9.9.2.47 insert_cell() [2/2]

```
insert_cell (
            i1 ,
            j1 ,
            val0 ,
            false ,
            false  )
```

9.9.2.48 M()

```
bool M (
            Array_.  M )
```

Definition at line 157 of file barraydense-meat.hpp.

9.9.2.49 resize() [1/6]

```
el_colsums resize (
            M ,
            ZERO_CELL  )
```

9.9.2.50 resize() [2/6]

```
el_colsums resize (
            M_ ,
            ZERO_CELL  )
```

9.9.2.51 resize() [3/6]

```
el resize (
            N * M,
            ZERO_CELL  )
```

9.9.2.52 resize() [4/6]

```
el_rowsums resize (
            N ,
            ZERO_CELL  )
```

9.9.2.53 resize() [5/6]

```
el resize (
             N_ * M_,
             ZERO_CELL  )
```

9.9.2.54 resize() [6/6]

```
el_rowsums resize (
             N_ ,
             ZERO_CELL  )
```

9.9.2.55 rm_cell() [1/3]

```
else rm_cell (
             i ,
             j ,
             false ,
             false  )
```

9.9.2.56 rm_cell() [2/3]

```
rm_cell (
             i0 ,
             j0 ,
             false ,
             false  )
```

9.9.2.57 rm_cell() [3/3]

```
rm_cell (
             i1 ,
             j1 ,
             false ,
             false  )
```

9.9.2.58 va_end()

```
va_end (
             args  )
```

9.9.2.59 va_start()

```
va_start (
             args ,
             fmt  )
```

9.9.2.60 vprintf()

```
vprintf (
             fmt ,
             args  )
```

9.9.3 Variable Documentation

9.9.3.1 add

```
uint const std::vector< uint > const std::vector< uint > 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 (uint j = 0u; j < M; ++j)
        std::swap(el[POS(i0, j)], el[POS(i1, j)])
```

Definition at line 654 of file barraydense-meat.hpp.

9.9.3.4 check_exists

```
uint 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:

```
{
    if ((data != nullptr) && delete_data)
        delete data
```

Definition at line 328 of file barraydense-meat.hpp.

9.9.3.11 el

```
return el == ZERO_CELL
```

Definition at line 381 of file barraydense-meat.hpp.

9.9.3.12 el_colsums

```
el_colsums[j] = (v.value - old)
```

Definition at line 675 of file barraydense-meat.hpp.

9.9.3.13 el_rowsums

```
el_rowsums[i] = (v.value - old)
```

Definition at line 674 of file barraydense-meat.hpp.

9.9.3.14 else

```
else (
             void  )
```

Initial value:

```
{
        Cell_Type old = el[POS(i,j)]
```

Definition at line 670 of file barraydense-meat.hpp.

9.9.3.15 false

```
false
```

Definition at line 767 of file barraydense-meat.hpp.

9.9.3.16 i1

```
uint 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

```
uint j0
```

Definition at line 720 of file barraydense-meat.hpp.

9.9.3.19 j1

```
uint j1
```

Definition at line 721 of file barraydense-meat.hpp.

9.9.3.20 M

```
M = M_
```

Definition at line 57 of file barraydense-meat.hpp.

9.9.3.21 M_

```
uint 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

```
uint uint uint 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

```
uint const std::vector< uint >& source
```

Definition at line 44 of file barraydense-meat.hpp.

9.9.3.26 target

```
uint const std::vector< uint > const std::vector< uint >& target
```

Definition at line 45 of file barraydense-meat.hpp.

9.9.3.27 v

```
uint 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.

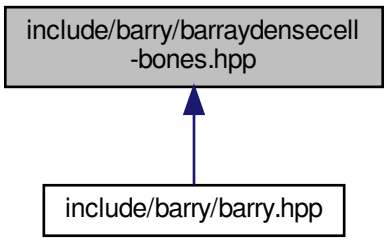
9.9.3.30 value

```
uint const std::vector< uint > const std::vector< uint > const std::vector< Cell_Type >&
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

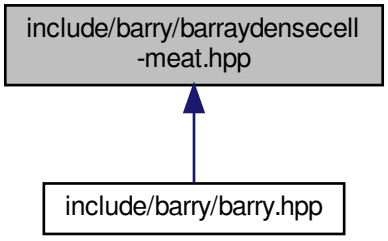
9.10.1.1 POS

```
#define POS(
              a,
              b ) (a) + (b) * N
```

Definition at line 6 of file barraydensecell-bones.hpp.

9.11 include/barry/barraydensecell-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define POS(a, b) (a) + (b) ∗ dat->N

9.11.1 Macro Definition Documentation

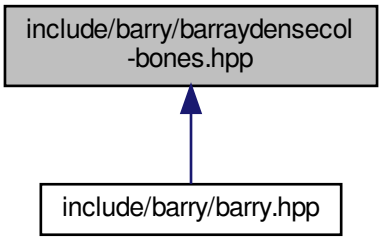
9.11.1.1 POS

```
#define POS(
              a,
              b ) (a) + (b) * dat->N
```

Definition at line 6 of file barraydensecell-meat.hpp.

9.12 include/barry/barraydensecol-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class BArrayDenseCol< Cell_Type, Data_Type >
- class BArrayDenseCol_const< Cell_Type, Data_Type >

Macros

- #define POS(a, b) (b)*N + (a)
- #define POS_N(a, b, c) (b)*(c) + (a)
- #define ZERO_CELL static_cast<Cell_Type>(0.0)

9.12.1 Macro Definition Documentation

9.12.1.1 POS

```
#define POS(
              a,
              b ) (b)*N + (a)
```

Definition at line 4 of file barraydensecol-bones.hpp.

9.12.1.2 POS_N

```
#define POS_N(
              a,
              b,
              c ) (b)*(c) + (a)
```

Definition at line 5 of file barraydensecol-bones.hpp.

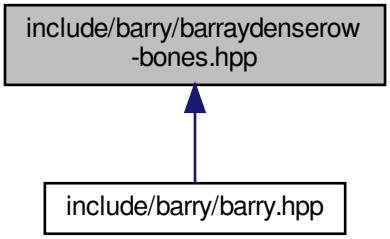
9.12.1.3 ZERO_CELL

```
#define ZERO_CELL static_cast<Cell_Type>(0.0)
```

Definition at line 6 of file barraydensecol-bones.hpp.

9.13 include/barry/barraydenserow-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class BArrayDenseRow< Cell_Type, Data_Type >
- class BArrayDenseRow_const< Cell_Type, Data_Type >

Macros

- #define POS(a, b) (b) * N + (a)
- #define POS_N(a, b, c) (b)*(c) + (a)
- #define ZERO_CELL static_cast< Cell_Type >(0.0)

9.13.1 Macro Definition Documentation

9.13.1.1 POS

```
#define POS(
              a,
              b ) (b) * N + (a)
```

Definition at line 4 of file barraydenserow-bones.hpp.

9.13.1.2 POS_N

```
#define POS_N(
              a,
              b,
              c ) (b)*(c) + (a)
```

Definition at line 5 of file barraydenserow-bones.hpp.

9.13.1.3 ZERO_CELL

```
#define ZERO_CELL static_cast< Cell_Type >(0.0)
```

Definition at line 6 of file barraydenserow-bones.hpp.

9.14 include/barry/barrayrow-bones.hpp File Reference

Classes

- class BArrayRow< Cell_Type, Data_Type >
- class BArrayRow_const< Cell_Type, Data_Type >

9.15 include/barry/barrayrow-meat.hpp File Reference

Macros

- #define BROW_TYPE() BArrayRow<Cell_Type, Data_Type>
- #define BROW_TEMPLATE_ARGS() <typename Cell_Type, typename Data_Type>
- #define BROW_TEMPLATE(a, b) template BROW_TEMPLATE_ARGS() inline a BROW_TYPE()::b

Functions

- BROW_TEMPLATE (void, operator=)(const BROW_TYPE() &val)
- BROW_TEMPLATE (void, operator+=)(const BROW_TYPE() &val)
- BROW_TEMPLATE (void, operator-=)(const BROW_TYPE() &val)
- BROW_TEMPLATE (void, operator*=)(const BROW_TYPE() &val)
- BROW_TEMPLATE (void, operator/=)(const BROW_TYPE() &val)

9.15.1 Macro Definition Documentation

9.15.1.1 BROW_TEMPLATE

```
#define BROW_TEMPLATE(
              a,
              b )  template BROW_TEMPLATE_ARGS() inline a BROW_TYPE()::b
```

Definition at line 8 of file barrayrow-meat.hpp.

9.15.1.2 BROW_TEMPLATE_ARGS

```
#define BROW_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

Definition at line 6 of file barrayrow-meat.hpp.

9.15.1.3 BROW_TYPE

```
#define BROW_TYPE( ) BArrayRow<Cell_Type, Data_Type>
```

Definition at line 4 of file barrayrow-meat.hpp.

9.15.2 Function Documentation

9.15.2.1 BROW_TEMPLATE() [1/5]

```
BROW_TEMPLATE (
             void ,
             operator*  ) const &
```

Definition at line 45 of file barrayrow-meat.hpp.

9.15.2.2 BROW_TEMPLATE() [2/5]

```
BROW_TEMPLATE (
             void ,
             operator+  ) const &
```

Definition at line 25 of file barrayrow-meat.hpp.

9.15.2.3 BROW_TEMPLATE() [3/5]

```
BROW_TEMPLATE (
            void ,
            operator-  ) const &
```

Definition at line 34 of file barrayrow-meat.hpp.

9.15.2.4 BROW_TEMPLATE() [4/5]

```
BROW_TEMPLATE (
            void ,
            operator/  ) const &
```

Definition at line 55 of file barrayrow-meat.hpp.

9.15.2.5 BROW_TEMPLATE() [5/5]

```
BROW_TEMPLATE (
            void ,
            operator  ) const &
```

Definition at line 11 of file barrayrow-meat.hpp.

9.16 include/barry/barrayvector-bones.hpp File Reference

Classes

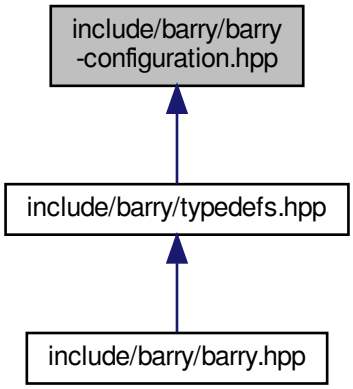
- class BArrayVector< Cell_Type, Data_Type >

  Row or column of a BArray
- class BArrayVector_const< Cell_Type, Data_Type >

9.17 include/barry/barrayvector-meat.hpp File Reference

9.18 include/barry/barry-configuration.hpp File Reference

This graph shows which files directly or indirectly include this file:



Configuration MACROS

These are mostly related to performance. The definitions follow:

- `BARRY_USE_UNORDERED_MAP` If specified, then barry is compiled using `std::unordered_map`. Otherwise it will use `std::map` for the arrays.
- `BARRY_USE_SAFE_EXP` When specified, it will multiply all likelihoods in `Model` by (1/-100)/(1/-100) so that numerical overflows are avoided.
- `BARRY_USE_ISFINITE` When specified, it will introduce a macro that checks whether the likelihood is finite or not.
- `printf_barry` If not specified, will be defined as `printf`.
- `BARRY_DEBUG_LEVEL`, when defined, will make things verbose.

- #define BARRY_SAFE_EXP -100.0
- #define BARRY_ISFINITE(a)
- #define BARRY_CHECK_SUPPORT(x, maxs)
- #define printf_barry printf
- #define BARRY_MAX_NUM_ELEMENTS static_cast< size_t >(UINT_MAX/2u)
- template<typename Ta , typename Tb >
  using Map = std::map< Ta, Tb >

9.18.1 Macro Definition Documentation

9.18.1.1 BARRY_CHECK_SUPPORT

```
#define BARRY_CHECK_SUPPORT(
              x,
              maxs )
```

Definition at line 47 of file barry-configuration.hpp.

9.18.1.2 BARRY_ISFINITE

```
#define BARRY_ISFINITE(
              a )
```

Definition at line 40 of file barry-configuration.hpp.

9.18.1.3 BARRY_MAX_NUM_ELEMENTS

```
#define BARRY_MAX_NUM_ELEMENTS static_cast< size_t >(UINT_MAX/2u)
```

Definition at line 55 of file barry-configuration.hpp.

9.18.1.4 BARRY_SAFE_EXP

```
#define BARRY_SAFE_EXP -100.0
```

Definition at line 33 of file barry-configuration.hpp.

9.18.1.5 printf_barry

```
#define printf_barry printf
```

Definition at line 51 of file barry-configuration.hpp.

9.18.2 Typedef Documentation

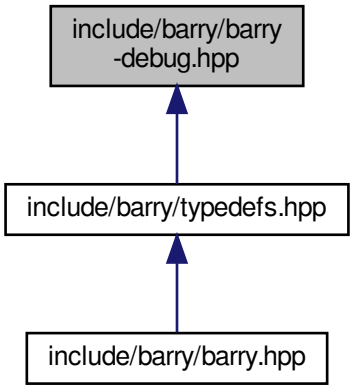
9.18.2.1 Map

```
template<typename Ta , typename Tb >
using Map = std::map<Ta,Tb>
```

Definition at line 27 of file barry-configuration.hpp.

9.19 include/barry/barry-debug.hpp File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define BARRY_DEBUG_LEVEL 0

9.19.1 Macro Definition Documentation

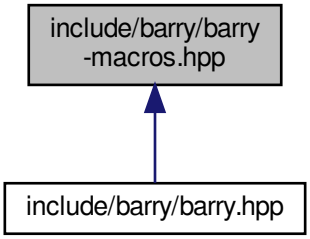
9.19.1.1 BARRY_DEBUG_LEVEL

```
#define BARRY_DEBUG_LEVEL 0
```

Definition at line 5 of file barry-debug.hpp.

9.20 include/barry/barry-macros.hpp File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define BARRY_ZERO Cell<Cell_Type>(0.0)
- #define BARRY_ZERO_DENSE static_cast<Cell_Type>(0.0)
- #define BARRY_ONE Cell<Cell_Type>(1.0)
- #define BARRY_ONE_DENSE static_cast<Cell_Type>(1.0)
- #define BARRY_UNUSED(expr) do { (void)(expr); } while (0);

9.20.1 Macro Definition Documentation

9.20.1.1 BARRY_ONE

```
#define BARRY_ONE Cell<Cell_Type>(1.0)
```

Definition at line 7 of file barry-macros.hpp.

9.20.1.2 BARRY_ONE_DENSE

```
#define BARRY_ONE_DENSE static_cast<Cell_Type>(1.0)
```

Definition at line 8 of file barry-macros.hpp.

9.20.1.3 BARRY_UNUSED

```
#define BARRY_UNUSED(
             expr ) do { (void)(expr); } while (0);
```

Definition at line 10 of file barry-macros.hpp.

9.20.1.4 BARRY_ZERO

```
#define BARRY_ZERO Cell<Cell_Type>(0.0)
```

Definition at line 4 of file barry-macros.hpp.

9.20.1.5 BARRY_ZERO_DENSE

```
#define BARRY_ZERO_DENSE static_cast<Cell_Type>(0.0)
```

Definition at line 5 of file barry-macros.hpp.

9.21 include/barry/barry.hpp File Reference

```
#include <iostream>
#include <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 TreeIterator 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, uint i, uint j, Data_Type & data)\
```

Definition at line 96 of file barry.hpp.

9.21.1.6 COUNTER_LAMBDA

```
#define COUNTER_LAMBDA(
                a )
```

Value:

```
    template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
    Counter_fun_type<Array_Type, Data_Type> a = \
    [](const Array_Type & Array, uint i, uint j, Data_Type & data)
```

Definition at line 99 of file barry.hpp.

9.21.1.7 RULE_FUNCTION

```
#define RULE_FUNCTION(
                a )
```

Value:

```
    template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
    inline bool (a) (const Array_Type & Array, uint i, uint j, Data_Type & data)\
```

Definition at line 103 of file barry.hpp.

9.21.1.8 RULE_LAMBDA

```
#define RULE_LAMBDA(
                a )
```

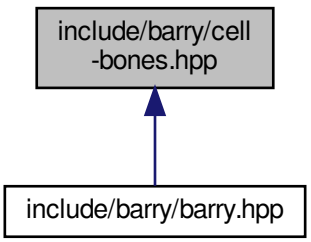
Value:

```
    template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
    Rule_fun_type<Array_Type, Data_Type> a = \
    [](const Array_Type & Array, uint i, uint j, Data_Type & data)
```

Definition at line 106 of file barry.hpp.

9.22 include/barry/cell-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



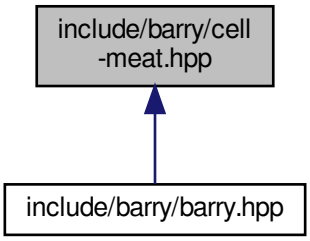
Classes

- class Cell< Cell_Type >

  Entries in BArray. For now, it only has two members:

9.23 include/barry/cell-meat.hpp File Reference

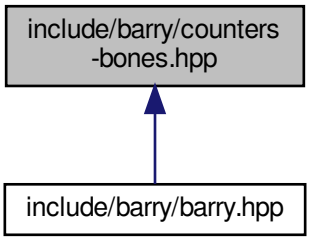
This graph shows which files directly or indirectly include this file:



9.24 include/barry/col-bones.hpp File Reference

9.25 include/barry/counters-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

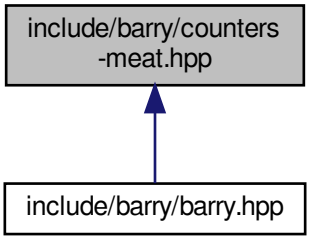
- class Counter< Array_Type, Data_Type >

  A counter function based on change statistics.
- class Counters< Array_Type, Data_Type >

  Vector of counters.

9.26 include/barry/counters-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define COUNTER_TYPE() Counter<Array_Type,Data_Type>
- #define COUNTER_TEMPLATE_ARGS() <typename Array_Type, typename Data_Type>
- #define COUNTER_TEMPLATE(a, b) template COUNTER_TEMPLATE_ARGS() inline a COUNTER_TYPE()↩ ::b
- #define TMP_HASHER_CALL Hasher_fun_type<Array_Type,Data_Type>
- #define COUNTERS_TYPE() Counters<Array_Type,Data_Type>
- #define COUNTERS_TEMPLATE_ARGS() <typename Array_Type, typename Data_Type>
- #define COUNTERS_TEMPLATE(a, b) template COUNTERS_TEMPLATE_ARGS() inline a COUNTERS_TYPE()↩ ::b

Functions

- COUNTER_TEMPLATE (, Counter)(const Counter< Array_Type
- Data_Type init_fun (counter_.init_fun)
- Data_Type hasher_fun (counter_.hasher_fun)
- Data_Type &&counter_ init_fun (std::move(counter_.init_fun))
- Data_Type &&counter_ hasher_fun (std::move(counter_.hasher_fun))
- Data_Type &&counter_ data (std::move(counter_.data))
- Data_Type &&counter_ name (std::move(counter_.name))
- Data_Type &&counter_ desc (std::move(counter_.desc))

  Move constructor.
- COUNTER_TEMPLATE (COUNTER_TYPE(), operator=)(const Counter< Array_Type
- COUNTER_TEMPLATE (COUNTER_TYPE() &, operator=)(Counter< Array_Type
- COUNTER_TEMPLATE (double, count)(Array_Type &Array

  < *Move assignment*
- return count_fun (Array, i, j, data)
- COUNTER_TEMPLATE (double, init)(Array_Type &Array
- return init_fun (Array, i, j, data)
- COUNTER_TEMPLATE (std::string, get_name)() const
- COUNTER_TEMPLATE (std::string, get_description)() const
- COUNTER_TEMPLATE (void, set_hasher)(Hasher_fun_type< Array_Type

- COUNTER_TEMPLATE (TMP_HASHER_CALL, get_hasher)()
- COUNTERS_TEMPLATE (, Counters)()
- COUNTERS_TEMPLATE (COUNTER_TYPE() &, operator[ ])(uint idx)
- Data_Type hasher (counter_.hasher)
- Data_Type &&counters_ hasher (std::move(counters_.hasher))
- COUNTERS_TEMPLATE (COUNTERS_TYPE(), operator=)(const Counters< Array_Type
- COUNTERS_TEMPLATE (COUNTERS_TYPE() &, operator=)(Counters< Array_Type
- COUNTERS_TEMPLATE (void, add_counter)(Counter< Array_Type
- COUNTERS_TEMPLATE (std::vector< std::string >, get_names)() const
- COUNTERS_TEMPLATE (std::vector< std::string >, get_descriptions)() const
- COUNTERS_TEMPLATE (std::vector< double >, gen_hash)(const Array_Type &array
- for (auto &c:data)
- if (add_dims)
- if (hasher)
- if (res.size()==0u) res.push_back(0.0)
- COUNTERS_TEMPLATE (void, add_hash)(Hasher_fun_type< Array_Type

Variables

- Data_Type & counter_
- Data_Type &&counter_ noexcept
- uint i
- uint uint j
- Data_Type fun
- Data_Type counter
- return
- Data_Type count_fun_
- Data_Type Counter_fun_type< Array_Type, Data_Type > init_fun_
- Data_Type Counter_fun_type< Array_Type, Data_Type > Hasher_fun_type< Array_Type, Data_Type > hasher_fun_
- Data_Type Counter_fun_type< Array_Type, Data_Type > Hasher_fun_type< Array_Type, Data_Type > Data_Type data_
- Data_Type Counter_fun_type< Array_Type, Data_Type > Hasher_fun_type< Array_Type, Data_Type > Data_Type std::string name_
- Data_Type Counter_fun_type< Array_Type, Data_Type > Hasher_fun_type< Array_Type, Data_Type > Data_Type std::string std::string desc_
- bool add_dims
- return res
- Data_Type fun_

9.26.1 Macro Definition Documentation

9.26.1.1 COUNTER_TEMPLATE

```
#define COUNTER_TEMPLATE(
            a,
            b )  template COUNTER_TEMPLATE_ARGS() inline a COUNTER_TYPE()::b
```

Definition at line 8 of file counters-meat.hpp.

9.26.1.2 COUNTER_TEMPLATE_ARGS

```
#define COUNTER_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

Definition at line 6 of file counters-meat.hpp.

9.26.1.3 COUNTER_TYPE

```
#define COUNTER_TYPE( ) Counter<Array_Type,Data_Type>
```

Definition at line 4 of file counters-meat.hpp.

9.26.1.4 COUNTERS_TEMPLATE

```
#define COUNTERS_TEMPLATE(
              a,
              b )  template COUNTERS_TEMPLATE_ARGS() inline a COUNTERS_TYPE()::b
```

Definition at line 129 of file counters-meat.hpp.

9.26.1.5 COUNTERS_TEMPLATE_ARGS

```
#define COUNTERS_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

Definition at line 127 of file counters-meat.hpp.

9.26.1.6 COUNTERS_TYPE

```
#define COUNTERS_TYPE( ) Counters<Array_Type,Data_Type>
```

Definition at line 125 of file counters-meat.hpp.

9.26.1.7 TMP_HASHER_CALL

```
#define TMP_HASHER_CALL Hasher_fun_type<Array_Type,Data_Type>
```

Definition at line 115 of file counters-meat.hpp.

9.26.2 Function Documentation

9.26.2.1 count_fun()

```
return count_fun (
             Array ,
             i ,
             j ,
             data  )
```

9.26.2.2 COUNTER_TEMPLATE() [1/9]

```
COUNTER_TEMPLATE (
             Counter  ) const
```

9.26.2.3 COUNTER_TEMPLATE() [2/9]

```
COUNTER_TEMPLATE (
             COUNTER_TYPE() & ,
             operator  )
```

9.26.2.4 COUNTER_TEMPLATE() [3/9]

```
COUNTER_TEMPLATE (
             COUNTER_TYPE() ,
             operator  ) const
```

9.26.2.5 COUNTER_TEMPLATE() [4/9]

```
COUNTER_TEMPLATE (
             double ,
             count  ) &
```

< Move assignment

9.26.2.6 COUNTER_TEMPLATE() [5/9]

```
COUNTER_TEMPLATE (
            double ,
            init  ) &
```

9.26.2.7 COUNTER_TEMPLATE() [6/9]

```
COUNTER_TEMPLATE (
            std::string ,
            get_description  ) const
```

Definition at line 107 of file counters-meat.hpp.

9.26.2.8 COUNTER_TEMPLATE() [7/9]

```
COUNTER_TEMPLATE (
            std::string ,
            get_name  ) const
```

Definition at line 103 of file counters-meat.hpp.

9.26.2.9 COUNTER_TEMPLATE() [8/9]

```
COUNTER_TEMPLATE (
            TMP_HASHER_CALL ,
            get_hasher  )
```

Definition at line 116 of file counters-meat.hpp.

9.26.2.10 COUNTER_TEMPLATE() [9/9]

```
COUNTER_TEMPLATE (
            void ,
            set_hasher  )
```

9.26.2.11 COUNTERS_TEMPLATE() [1/9]

```
COUNTERS_TEMPLATE (
            Counters  )
```

Definition at line 132 of file counters-meat.hpp.

9.26.2.12 COUNTERS_TEMPLATE() [2/9]

```
COUNTERS_TEMPLATE (
            COUNTER_TYPE() & ,
            operator [] )
```

Definition at line 134 of file counters-meat.hpp.

9.26.2.13 COUNTERS_TEMPLATE() [3/9]

```
COUNTERS_TEMPLATE (
            COUNTERS_TYPE() & ,
            operator  )
```

9.26.2.14 COUNTERS_TEMPLATE() [4/9]

```
COUNTERS_TEMPLATE (
            COUNTERS_TYPE() ,
            operator  ) const
```

9.26.2.15 COUNTERS_TEMPLATE() [5/9]

```
COUNTERS_TEMPLATE (
            std::vector< double > ,
            gen_hash  ) const &
```

9.26.2.16 COUNTERS_TEMPLATE() [6/9]

```
COUNTERS_TEMPLATE (
            std::vector< std::string > ,
            get_descriptions  ) const
```

Definition at line 212 of file counters-meat.hpp.

9.26.2.17 COUNTERS_TEMPLATE() [7/9]

```
COUNTERS_TEMPLATE (
            std::vector< std::string > ,
            get_names  ) const
```

Definition at line 201 of file counters-meat.hpp.

9.26.2.18 COUNTERS_TEMPLATE() [8/9]

```
COUNTERS_TEMPLATE (
            void ,
            add_counter  )
```

9.26.2.19 COUNTERS_TEMPLATE() [9/9]

```
COUNTERS_TEMPLATE (
            void ,
            add_hash  )
```

9.26.2.20 data()

```
Data_Type&& counter_ data (
            std::move(counter_.data)  )
```

9.26.2.21 desc()

```
Data_Type&& counter_ desc (
            std::move(counter_.desc)  )
```

Move constructor.

Definition at line 32 of file counters-meat.hpp.

9.26.2.22 for()

```
for (
            auto &c:data  )
```

Definition at line 231 of file counters-meat.hpp.

9.26.2.23 hasher() [1/2]

```
Data_Type hasher (
            counter_.  hasher )
```

Definition at line 141 of file counters-meat.hpp.

9.26.2.24 hasher() [2/2]

```
Data_Type&& counters_ hasher (
            std::move(counters_.hasher)  )
```

Definition at line 144 of file counters-meat.hpp.

9.26.2.25 hasher_fun() [1/2]

```
Data_Type hasher_fun (
            counter_.  hasher_fun )
```

Definition at line 13 of file counters-meat.hpp.

9.26.2.26 hasher_fun() [2/2]

```
Data_Type&& counter_ hasher_fun (
            std::move(counter_.hasher_fun)  )
```

9.26.2.27 if() [1/3]

```
if (
            add_dims  )
```

Definition at line 246 of file counters-meat.hpp.

9.26.2.28 if() [2/3]

```
if (
            hasher  )
```

Definition at line 253 of file counters-meat.hpp.

9.26.2.29 if() [3/3]

```
if (
             res.  size() = =0u )
```

9.26.2.30 init_fun() [1/3]

```
return init_fun (
             Array ,
             i ,
             j ,
             data  )
```

9.26.2.31 init_fun() [2/3]

```
Data_Type init_fun (
             counter_.  init_fun )
```

9.26.2.32 init_fun() [3/3]

```
Data_Type&& counter_ init_fun (
             std::move(counter_.init_fun)  )
```

9.26.2.33 name()

```
Data_Type&& counter_ name (
             std::move(counter_.name)  )
```

9.26.3 Variable Documentation

9.26.3.1 add_dims

```
bool add_dims
```

Initial value:

```
{
    std::vector<double> res
```

Definition at line 225 of file counters-meat.hpp.

9.26.3.2 count_fun_

```
Data_Type count_fun_
```

Definition at line 179 of file counters-meat.hpp.

9.26.3.3 counter

```
Data_Type counter
```

Initial value:

```
{

    data.push_back(counter)
```

Definition at line 170 of file counters-meat.hpp.

9.26.3.4 counter_

```
Data_Type & counter_
```

Initial value:

```
{
    if (this != &counter_) {
        this->count_fun = counter_.count_fun;
        this->init_fun = counter_.init_fun;
        this->hasher_fun = counter_.hasher_fun;

        this->data = counter_.data;
        this->name = counter_.name;
        this->desc = counter_.desc;
    }
    return *this
```

Definition at line 12 of file counters-meat.hpp.

9.26.3.5 data_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> Hasher_fun_type<Array_Type,Data_Type> Data↩
_Type data_
```

Definition at line 182 of file counters-meat.hpp.

9.26.3.6 desc_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> Hasher_fun_type<Array_Type,Data_Type> Data↩
_Type std::string std::string desc_
```

Initial value:

```
{

    data.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

```
uint i
```

Definition at line 83 of file counters-meat.hpp.

9.26.3.11 init_fun_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> init_fun_
```

Definition at line 180 of file counters-meat.hpp.

9.26.3.12 j

```
uint uint j
```

Initial value:

```
{
    if (count_fun == nullptr)
        return 0.0
```

Definition at line 83 of file counters-meat.hpp.

9.26.3.13 name_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> Hasher_fun_type<Array_Type,Data_Type> Data↩
_Type std::string name_
```

Definition at line 183 of file counters-meat.hpp.

9.26.3.14 noexcept

```
Data_Type &&counters_ noexcept
```

Initial value:

```
{
    if (this != &counter_)
    {
        this->data = std::move(counter_.data);

        this->count_fun = std::move(counter_.count_fun);
        this->init_fun = std::move(counter_.init_fun);
        this->hasher_fun = std::move(counter_.hasher_fun);

        this->name = std::move(counter_.name);
        this->desc = std::move(counter_.desc);
    }
    return *this
```

Definition at line 26 of file counters-meat.hpp.

9.26.3.15 res

```
return res
```

Definition at line 263 of file counters-meat.hpp.

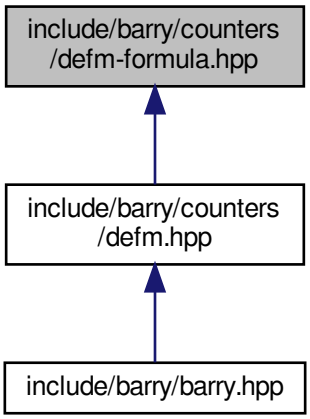
9.26.3.16 return

```
return
```

Definition at line 175 of file counters-meat.hpp.

9.27 include/barry/counters/defm-formula.hpp File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void defm_motif_parser (std::string formula, std::vector< size_t > &locations, std::vector< bool > &signs, size_t m_order, size_t y_ncol)

  Parses a motif formula.

9.27.1 Function Documentation

9.27.1.1 defm_motif_parser()

```
void defm_motif_parser (
            std::string formula,
            std::vector< size_t > & locations,
            std::vector< bool > & signs,
            size_t m_order,
            size_t y_ncol )  [inline]
```

Parses a motif formula.

This function will take the formula and generate the corresponding input for defm::counter_transition(). Formulas can be specified in the following ways:

- Intercept effect: {...} No transition, only including the current state.
- Transition effect: {...} > {...} Includes current and previous states.

The general notation is `[0]y[column id]_[row id]`. A preceeding zero means that the value of the cell is considered to be zero. The column id goes between 0 and the number of columns in the array - 1 (so it is indexed from 0,) and the row id goes from 0 to m_order.

Intercept effects

Intercept effects only involve a single set of curly brackets. Using the 'greater-than' symbol (i.e., '<') is only for transition effects. When specifying intercept effects, users can skip the `row_id`, e.g., `y0_0` is equivalent to `y0`. If the passed `row id` is different from the Markov order, i.e., `row_id != m_order`, then the function returns with an error.

Examples:

- "`{y0, 0y1}`" is equivalent to set a motif with the first element equal to one and the second to zero.

Transition effects

Transition effects can be specified using two sets of curly brackets and an greater-than symbol, i.e., `{...} > {...}`. The first set of brackets, which we call LHS, can only hold `row id` that are less than `m_order`.

Parameters

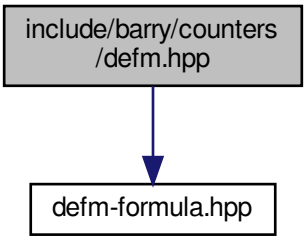
formula	
locations	
signs	
m_order	
y_ncol	

Definition at line 46 of file defm-formula.hpp.

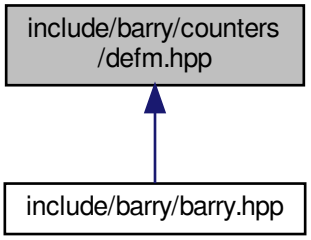
9.28 include/barry/counters/defm.hpp File Reference

```
#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 uint 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, uint i, uint j, DEFMCounterData & data)
- #define DEFM_COUNTER_LAMBDA(a)

Macros for defining rules

- #define DEFM_RULE(a) inline bool (a) (const DEFMArray & Array, uint i, uint j, bool & data)
- #define DEFM_RULE_LAMBDA(a)

Typedefs

- typedef BArrayDense< int, DEFMData > DEFMArray

Convenient typedefs for network objects.

- typedef Counter< DEFMArray, DEFMCounterData > DEFMCounter
- typedef Counters< DEFMArray, DEFMCounterData > DEFMCounters
- typedef Support< DEFMArray, DEFMCounterData, DEFMRuleData, DEFMRuleDynData > DEFMSupport
- typedef StatsCounter< DEFMArray, DEFMCounterData > DEFMStatsCounter
- typedef Model< DEFMArray, DEFMCounterData, DEFMRuleData, DEFMRuleDynData > DEFMModel
- typedef Rule< DEFMArray, DEFMRuleData > DEFMRule
- typedef Rules< DEFMArray, DEFMRuleData > DEFMRules
- typedef Rule< DEFMArray, DEFMRuleDynData > DEFMRuleDyn
- typedef Rules< DEFMArray, DEFMRuleDynData > DEFMRulesDyn

Functions

- void counter_ones (DEFMCounters *counters, int covar_index=-1, std::string vname="", const std::vector< std::string > *x_names=nullptr)

  Prevalence of ones.
- void counter_logit_intercept (DEFMCounters *counters, size_t n_y, std::vector< size_t > which={}, int covar_index=-1, std::string vname="", const std::vector< std::string > *x_names=nullptr, const std::vector< std::string > *y_names=nullptr)
- void counter_transition (DEFMCounters *counters, std::vector< size_t > coords, std::vector< bool > signs, size_t m_order, size_t n_y, int covar_index=-1, std::string vname="", const std::vector< std::string > *x_↩ names=nullptr, const std::vector< std::string > *y_names=nullptr)

  Prevalence of ones.
- void counter_transition_formula (DEFMCounters *counters, std::string formula, size_t m_order, size_t n_y, int covar_index=-1, std::string vname="", const std::vector< std::string > *x_names=nullptr, const std::vector< std::string > *y_names=nullptr)

  Prevalence of ones.
- void counter_fixed_effect (DEFMCounters *counters, int covar_index, double k, std::string vname="", const std::vector< std::string > *x_names=nullptr)

  Prevalence of ones.

Returns true if the cell is free

Parameters

rules	*A pointer to a* `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

```
#define UNI_SUB(
                a )
```

Value:

```
(\
    ((a) == 0) ? "\u2080" : (\
    ((a) == 1) ? "\u2081" : (\
    ((a) == 2) ? "\u2082" : (\
    ((a) == 3) ? "\u2083" : (\
    ((a) == 4) ? "\u2084" : (\
    ((a) == 5) ? "\u2085" : (\
    ((a) == 6) ? "\u2086" : (\
    ((a) == 7) ? "\u2087" : (\
    ((a) == 8) ? "\u2088" : \
    "\u2089"))))))))\
)
```

9.28.2 Typedef Documentation

9.28.2.1 DEFMArray

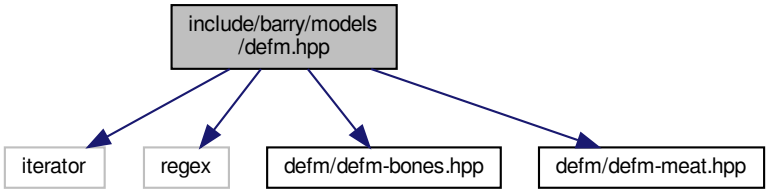
```
typedef BArrayDense<int, DEFMData> DEFMArray
```

Definition at line 25 of file defm.hpp.

9.29 include/barry/models/defm.hpp File Reference

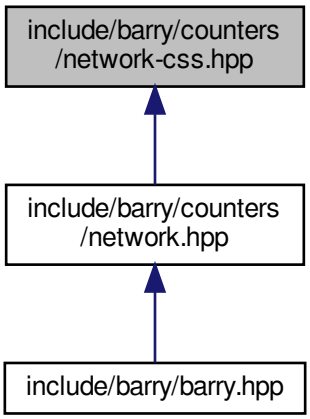
```
#include <iterator>
#include <regex>
#include "defm/defm-bones.hpp"
#include "defm/defm-meat.hpp"
```

Include dependency graph for defm.hpp:



9.30 include/barry/counters/network-css.hpp File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define CSS_SIZE()
- #define CSS_CASE_TRUTH() if ((i < n) && (j < n))
- #define CSS_TRUE_CELLS()
- #define CSS_CASE_PERCEIVED() else if (((i >= s) && (i < e)) & ((j >= s) && (j < e)))

- #define CSS_PERCEIVED_CELLS()
- #define CSS_CASE_ELSE()
- #define CSS_CHECK_SIZE_INIT()
- #define CSS_CHECK_SIZE()
- #define CSS_APPEND(name)
- #define CSS_NET_COUNTER_LAMBDA_INIT()

Functions

- template<typename Tnet = Network>
  void counter_css_partially_false_recip_commi (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)

  Counts errors of commission.
- template<typename Tnet = Network>
  void counter_css_partially_false_recip_omiss (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)

  Counts errors of omission.
- template<typename Tnet = Network>
  void counter_css_completely_false_recip_comiss (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)

  Counts completely false reciprocity (comission)
- template<typename Tnet = Network>
  void counter_css_completely_false_recip_omiss (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)

  Counts completely false reciprocity (omission)
- template<typename Tnet = Network>
  void counter_css_mixed_recip (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)

  Counts mixed reciprocity errors.
- template<typename Tnet = Network>
  void counter_css_census01 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)
- template<typename Tnet = Network>
  void counter_css_census02 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)
- template<typename Tnet = Network>
  void counter_css_census03 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)
- template<typename Tnet = Network>
  void counter_css_census04 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)
- template<typename Tnet = Network>
  void counter_css_census05 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)
- template<typename Tnet = Network>
  void counter_css_census06 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)
- template<typename Tnet = Network>
  void counter_css_census07 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)
- template<typename Tnet = Network>
  void counter_css_census08 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)

- template<typename Tnet = Network>
  void counter_css_census09 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end↩
  _)
- template<typename Tnet = Network>
  void counter_css_census10 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end↩
  _)

9.30.1 Macro Definition Documentation

9.30.1.1 CSS_APPEND

```
#define CSS_APPEND(
              name )
```

Value:

```
    std::string name_ = (name);\
    for (uint i = 0u; i < end_.size(); ++i) { \
    std::string tmpname = name_ + " (" + std::to_string(i) + ")";\
    counters->add_counter(tmp_count, tmp_init, nullptr, \
            NetCounterData({netsize, i == 0u ? netsize : end_[i-1], end_[i]}, {}),\
            tmpname);}
```

Definition at line 42 of file network-css.hpp.

9.30.1.2 CSS_CASE_ELSE

```
#define CSS_CASE_ELSE( )
```

Definition at line 27 of file network-css.hpp.

9.30.1.3 CSS_CASE_PERCEIVED

```
#define CSS_CASE_PERCEIVED( ) else if (((i >= s) && (i < e)) & ((j >= s) && (j < e)))
```

Definition at line 20 of file network-css.hpp.

9.30.1.4 CSS_CASE_TRUTH

```
#define CSS_CASE_TRUTH( ) if ((i < n) && (j < n))
```

Definition at line 13 of file network-css.hpp.

9.30.1.5 CSS_CHECK_SIZE

```
#define CSS_CHECK_SIZE( )
```

Value:

```
    for (uint i = 0u; i < end_.size(); ++i) {\
    if (i == 0u) continue; \
    else if (end_[i] < end_[i-1u]) \
        throw std::logic_error("Endpoints should be specified in order.");}
```

Definition at line 37 of file network-css.hpp.

9.30.1.6 CSS_CHECK_SIZE_INIT

```
#define CSS_CHECK_SIZE_INIT( )
```

Value:

```
    /* The indices fall within the network */ \
    if ((data.indices.at(0) > Array.ncol()) \
    | (data.indices.at(2) > Array.ncol())) \
        throw std::range_error("The network does not match the prescribed size.");
```

Definition at line 31 of file network-css.hpp.

9.30.1.7 CSS_NET_COUNTER_LAMBDA_INIT

```
#define CSS_NET_COUNTER_LAMBDA_INIT( )
```

Value:

```
        NETWORK_COUNTER_LAMBDA(tmp_init) {\
        CSS_CHECK_SIZE_INIT() \
        return 0.0; \
    };
```

Definition at line 49 of file network-css.hpp.

9.30.1.8 CSS_PERCEIVED_CELLS

```
#define CSS_PERCEIVED_CELLS( )
```

Value:

```
    double tji = static_cast<double>(Array(j - s, i - s, false)); \
    double pji = static_cast<double>(Array(j, i, false)); \
    double tij = static_cast<double>(Array(i - s, j - s, false));
```

Definition at line 21 of file network-css.hpp.

9.30.1.9 CSS_SIZE

```
#define CSS_SIZE( )
```

Value:

```
    uint n = data.indices[0u]; \
    uint s = data.indices[1u]; \
    uint e = data.indices[2u];
```

Definition at line 7 of file network-css.hpp.

9.30.1.10 CSS_TRUE_CELLS

```
#define CSS_TRUE_CELLS( )
```

Value:

```
    double tji = static_cast<double>(Array(j, i, false)); \
    double pij = static_cast<double>(Array(i + s, j + s, false)); \
    double pji = static_cast<double>(Array(j + s, i + s, false));
```

Definition at line 14 of file network-css.hpp.

9.30.2 Function Documentation

9.30.2.1 counter_css_census01()

```
template<typename Tnet = Network>
void counter_css_census01 (
             NetCounters< Tnet > * counters,
             uint netsize,
             const std::vector< uint > & end_ )  [inline]
```

Definition at line 275 of file network-css.hpp.

9.30.2.2 counter_css_census02()

```
template<typename Tnet = Network>
void counter_css_census02 (
             NetCounters< Tnet > * counters,
             uint netsize,
             const std::vector< uint > & end_ )  [inline]
```

Definition at line 325 of file network-css.hpp.

9.30.2.3 counter_css_census03()

```
template<typename Tnet = Network>
void counter_css_census03 (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Definition at line 364 of file network-css.hpp.

9.30.2.4 counter_css_census04()

```
template<typename Tnet = Network>
void counter_css_census04 (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Definition at line 403 of file network-css.hpp.

9.30.2.5 counter_css_census05()

```
template<typename Tnet = Network>
void counter_css_census05 (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Definition at line 442 of file network-css.hpp.

9.30.2.6 counter_css_census06()

```
template<typename Tnet = Network>
void counter_css_census06 (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Definition at line 481 of file network-css.hpp.

9.30.2.7 counter_css_census07()

```
template<typename Tnet = Network>
void counter_css_census07 (
             NetCounters< Tnet > * counters,
             uint netsize,
             const std::vector< uint > & end_ )  [inline]
```

Definition at line 520 of file network-css.hpp.

9.30.2.8 counter_css_census08()

```
template<typename Tnet = Network>
void counter_css_census08 (
             NetCounters< Tnet > * counters,
             uint netsize,
             const std::vector< uint > & end_ )  [inline]
```

Definition at line 559 of file network-css.hpp.

9.30.2.9 counter_css_census09()

```
template<typename Tnet = Network>
void counter_css_census09 (
             NetCounters< Tnet > * counters,
             uint netsize,
             const std::vector< uint > & end_ )  [inline]
```

Definition at line 598 of file network-css.hpp.

9.30.2.10 counter_css_census10()

```
template<typename Tnet = Network>
void counter_css_census10 (
             NetCounters< Tnet > * counters,
             uint netsize,
             const std::vector< uint > & end_ )  [inline]
```

Definition at line 637 of file network-css.hpp.

9.30.2.11 counter_css_completely_false_recip_comiss()

```
template<typename Tnet = Network>
void counter_css_completely_false_recip_comiss (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Counts completely false reciprocity (comission)

Definition at line 154 of file network-css.hpp.

9.30.2.12 counter_css_completely_false_recip_omiss()

```
template<typename Tnet = Network>
void counter_css_completely_false_recip_omiss (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Counts completely false reciprocity (omission)

Definition at line 194 of file network-css.hpp.

9.30.2.13 counter_css_mixed_recip()

```
template<typename Tnet = Network>
void counter_css_mixed_recip (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Counts mixed reciprocity errors.

Definition at line 234 of file network-css.hpp.

9.30.2.14 counter_css_partially_false_recip_commi()

```
template<typename Tnet = Network>
void counter_css_partially_false_recip_commi (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

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()

```
template<typename Tnet = Network>
void counter_css_partially_false_recip_omiss (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

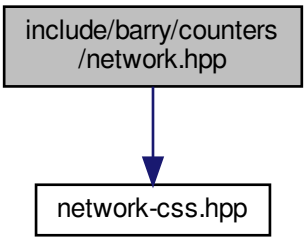
Counts errors of omission.

Definition at line 110 of file network-css.hpp.

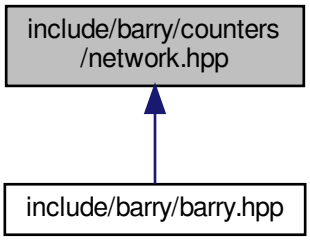
9.31 include/barry/counters/network.hpp File Reference

```
#include "network-css.hpp"
```

Include dependency graph for network.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class NetworkData

  Data class for Networks.
- class NetCounterData

  Data class used to store arbitrary uint or double vectors.

Macros

- #define NET_C_DATA_IDX(i) (data.indices[i])
- #define NET_C_DATA_NUM(i) (data.numbers[i])

Macros for defining counters

- #define NETWORK_COUNTER(a)
- #define NETWORK_COUNTER_LAMBDA(a)
- #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, uint 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, uint attr_id, double alpha=1.0, double tail_head=true)

  Sum of attribute difference between ego and alter to pow(alpha)
- NETWORK_COUNTER (init_single_attr)
- template<typename Tnet = Network>
  void counter_nodeicov (NetCounters< Tnet > *counters, uint attr_id)
- template<typename Tnet = Network>
  void counter_nodeocov (NetCounters< Tnet > *counters, uint attr_id)
- template<typename Tnet = Network>
  void counter_nodecov (NetCounters< Tnet > *counters, uint attr_id)
- template<typename Tnet = Network>
  void counter_nodematch (NetCounters< Tnet > *counters, uint attr_id)
- template<typename Tnet = Network>
  void counter_idegree (NetCounters< Tnet > *counters, std::vector< uint > d)

  Counts number of vertices with a given in-degree.
- template<> void counter_idegree (NetCounters< NetworkDense > *counters, std::vector< uint > d)
- template<typename Tnet = Network>
  void counter_odegree (NetCounters< Tnet > *counters, std::vector< uint > d)

  Counts number of vertices with a given out-degree.
- template<> void counter_odegree (NetCounters< NetworkDense > *counters, std::vector< uint > d)

- template<typename Tnet = Network>
  void counter_degree (NetCounters< Tnet > *counters, std::vector< uint > 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

```
#define NET_C_DATA_IDX(
              i ) (data.indices[i])
```

Definition at line 74 of file network.hpp.

9.31.1.4 NET_C_DATA_NUM

```
#define NET_C_DATA_NUM(
              i ) (data.numbers[i])
```

Definition at line 75 of file network.hpp.

9.31.1.5 NETWORK_COUNTER

```
#define NETWORK_COUNTER(
              a )
```

Value:

```
template<typename Tnet = Network>\
inline double (a) (const Tnet & Array, uint i, uint j, NetCounterData & data)
```

Function for definition of a network counter function

Definition at line 114 of file network.hpp.

9.31.1.6 NETWORK_COUNTER_LAMBDA

```
#define NETWORK_COUNTER_LAMBDA(
              a )
```

Value:

```
Counter_fun_type<Tnet, NetCounterData> a = \
    [](const Tnet & Array, uint i, uint j, NetCounterData & data)
```

Lambda function for definition of a network counter function

Definition at line 119 of file network.hpp.

9.31.1.7 NETWORK_RULE

```
#define NETWORK_RULE(
                a )
```

Value:
```
template<typename Tnet = Network>\
inline bool (a) (const Tnet & Array, uint i, uint j, bool & data)
```

Function for definition of a network counter function

Definition at line 133 of file network.hpp.

9.31.1.8 NETWORK_RULE_LAMBDA

```
#define NETWORK_RULE_LAMBDA(
                a )
```

Value:
```
Rule_fun_type<Tnet, bool> a = \
[](const Tnet & Array, uint i, uint j, bool & data)
```

Lambda function for definition of a network counter function

Definition at line 138 of file network.hpp.

9.31.1.9 NETWORKDENSE_COUNTER_LAMBDA

```
#define NETWORKDENSE_COUNTER_LAMBDA(
                a )
```

Value:
```
Counter_fun_type<NetworkDense, NetCounterData> a = \
    [](const NetworkDense & Array, uint i, uint j, NetCounterData & data)
```

Definition at line 123 of file network.hpp.

9.31.2 Typedef Documentation

9.31.2.1 NetCounter

```
template<typename Tnet = Network>
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()

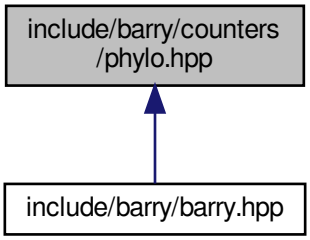
```
template<typename Tnet = Network>
void rules_zerodiag (
            NetRules< Tnet > * rules )  [inline]
```

Number of edges.

Definition at line 1383 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< uint, uint > > PhyloRuleData

Convenient typedefs for Node objects.

- typedef BArrayDense< uint, NodeData > PhyloArray
- typedef Counter< PhyloArray, PhyloCounterData > PhyloCounter
- typedef Counters< PhyloArray, PhyloCounterData > PhyloCounters
- typedef Rule< PhyloArray, PhyloRuleData > PhyloRule
- typedef Rules< PhyloArray, PhyloRuleData > PhyloRules
- typedef Rule< PhyloArray, PhyloRuleDynData > PhyloRuleDyn
- typedef Rules< PhyloArray, PhyloRuleDynData > PhyloRulesDyn
- typedef Support< PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport
- typedef StatsCounter< PhyloArray, PhyloCounterData > PhyloStatsCounter
- typedef Model< PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel
- typedef PowerSet< PhyloArray, PhyloRuleData > PhyloPowerSet

Functions

- std::string get_last_name (unsigned int d)
- void counter_overall_gains (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  Overall functional gains.
- void counter_gains (PhyloCounters *counters, std::vector< uint > nfun, unsigned int duplication=DEFAULT_DUPLICATION)

  Functional gains for a specific function (`nfun`).
- void counter_gains_k_offspring (PhyloCounters *counters, std::vector< uint > nfun, uint k=1u, unsigned int duplication=DEFAULT_DUPLICATION)

  k genes gain function nfun
- void counter_genes_changing (PhyloCounters *counters, unsigned int 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, unsigned int nfunA, unsigned int nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  Keeps track of how many pairs of genes preserve pseudostate.
- void counter_prop_genes_changing (PhyloCounters *counters, unsigned int 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, unsigned int duplication=DEFAULT_DUPLICATION)

  Overall functional loss.
- void counter_maxfuns (PhyloCounters *counters, uint lb, uint ub, unsigned int duplication=DEFAULT_DUPLICATION)

  Cap the number of functions per gene.
- void counter_loss (PhyloCounters *counters, std::vector< uint > nfun, unsigned int duplication=DEFAULT_DUPLICATION)

  Total count of losses for an specific function.
- void counter_overall_changes (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  Total number of changes. Use this statistic to account for "preservation".
- void counter_subfun (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  Total count of Sub-functionalization events.
- void counter_cogain (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  Co-evolution (joint gain or loss)
- void counter_longest (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  Longest branch mutates (either by gain or by loss)
- void counter_neofun (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  Total number of neofunctionalization events.
- void counter_pairwise_neofun_singlefun (PhyloCounters *counters, uint nfunA, unsigned int duplication=DEFAULT_DUPLICATI

  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, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  Total number of neofunctionalization events.
- void counter_co_opt (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  Function co-opting.
- void counter_k_genes_changing (PhyloCounters *counters, unsigned int k, unsigned int 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, unsigned int duplication=DEFAULT_DUPLICATION)

  Indicator function. Equals to one if k genes changed and zero otherwise.
- void counter_gains_from_0 (PhyloCounters *counters, std::vector< uint > nfun, unsigned int duplication=DEFAULT_DUPLICAT

  Used when all the functions are in 0 (like the root node prob.)
- void counter_overall_gains_from_0 (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  Used when all the functions are in 0 (like the root node prob.)
- void counter_pairwise_overall_change (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  Used when all the functions are in 0 (like the root node prob.)

- void counter_pairwise_preserving (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  Used when all the functions are in 0 (like the root node prob.)
- void counter_pairwise_first_gain (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  Used when all the functions are in 0 (like the root node prob.)
- void rule_dyn_limit_changes (PhyloSupport *support, uint pos, uint lb, uint ub, unsigned int duplication=DEFAULT_DUPLICATIO

  Overall functional gains.

9.32.1 Macro Definition Documentation

9.32.1.1 DEFAULT_DUPLICATION

```
#define DEFAULT_DUPLICATION 1u
```

Definition at line 5 of file phylo.hpp.

9.32.1.2 DUPL_DUPL

```
#define DUPL_DUPL 1u
```

Definition at line 7 of file phylo.hpp.

9.32.1.3 DUPL_EITH

```
#define DUPL_EITH 2u
```

Definition at line 8 of file phylo.hpp.

9.32.1.4 DUPL_SPEC

```
#define DUPL_SPEC 0u
```

Definition at line 6 of file phylo.hpp.

9.32.1.5 IF_MATCHES

```
#define IF_MATCHES( )
```

Value:

```
    MAKE_DUPL_VARS() \
    if (IS_EITHER() | IS_DUPLICATION() | IS_SPECIATION())
```

Definition at line 19 of file phylo.hpp.

9.32.1.6 IF_NOTMATCHES

```
#define IF_NOTMATCHES( )
```

Value:

```
    MAKE_DUPL_VARS() \
    if (!IS_EITHER() & !IS_DUPLICATION() & !IS_SPECIATION())
```

Definition at line 21 of file phylo.hpp.

9.32.1.7 IS_DUPLICATION

```
#define IS_DUPLICATION( ) ((DATA_AT == DUPL_DUPL) & (DPL))
```

Definition at line 16 of file phylo.hpp.

9.32.1.8 IS_EITHER

```
#define IS_EITHER( ) (DATA_AT == DUPL_EITH)
```

Definition at line 15 of file phylo.hpp.

9.32.1.9 IS_SPECIATION

```
#define IS_SPECIATION( ) ((DATA_AT == DUPL_SPEC) & (!DPL))
```

Definition at line 17 of file phylo.hpp.

9.32.1.10 MAKE_DUPL_VARS

```
#define MAKE_DUPL_VARS( )
```

Value:

```
    bool DPL = Array.D_ptr()->duplication; \
    unsigned int DATA_AT = data[0u];
```

Definition at line 11 of file phylo.hpp.

9.32.1.11 PHYLO_CHECK_MISSING

```
#define PHYLO_CHECK_MISSING( )
```

Value:

```
    if (Array.D_ptr() == nullptr) \
    throw std::logic_error("The array data is nullptr."); \
```

Definition at line 139 of file phylo.hpp.

9.32.1.12 PHYLO_COUNTER_LAMBDA

```
#define PHYLO_COUNTER_LAMBDA(
                a )
```

Value:

```
    Counter_fun_type<PhyloArray, PhyloCounterData> a = \
    [](const PhyloArray & Array, uint i, uint j, PhyloCounterData & data)
```

Extension of a simple counter.

It allows specifying extra arguments, in particular, the corresponding sets of rows to which this statistic may be relevant. This could be important in the case of, for example, counting correlation type statistics between function 1 and 2, and between function 1 and 3.

Definition at line 133 of file phylo.hpp.

9.32.1.13 PHYLO_RULE_DYN_LAMBDA

```
#define PHYLO_RULE_DYN_LAMBDA(
                a )
```

Value:

```
    Rule_fun_type<PhyloArray, PhyloRuleDynData> a = \
    [](const PhyloArray & Array, uint i, uint j, PhyloRuleDynData & data)
```

Definition at line 136 of file phylo.hpp.

9.32.2 Typedef Documentation

9.32.2.1 PhyloArray

```
typedef BArrayDense<uint, 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< uint, uint > > PhyloRuleData
```

Definition at line 99 of file phylo.hpp.

9.32.2.8 PhyloRuleDyn

```
typedef Rule<PhyloArray,PhyloRuleDynData> PhyloRuleDyn
```

Definition at line 113 of file phylo.hpp.

9.32.2.9 PhyloRules

```
typedef Rules<PhyloArray,PhyloRuleData> PhyloRules
```

Definition at line 111 of file phylo.hpp.

9.32.2.10 PhyloRulesDyn

```
typedef Rules<PhyloArray,PhyloRuleDynData> PhyloRulesDyn
```

Definition at line 114 of file phylo.hpp.

9.32.2.11 PhyloStatsCounter

```
typedef StatsCounter<PhyloArray, PhyloCounterData> PhyloStatsCounter
```

Definition at line 117 of file phylo.hpp.

9.32.2.12 PhyloSupport

```
typedef Support<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport
```

Definition at line 116 of file phylo.hpp.

9.32.3 Function Documentation

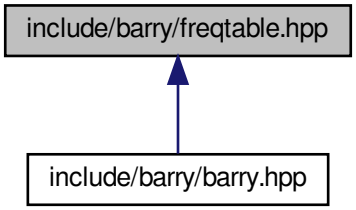
9.32.3.1 get_last_name()

```
std::string get_last_name (
            unsigned int d )  [inline]
```

Definition at line 142 of file phylo.hpp.

9.33 include/barry/freqtable.hpp File Reference

This graph shows which files directly or indirectly include this file:



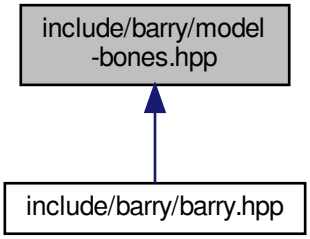
Classes

- class FreqTable< T >

  Frequency table of vectors.

9.34 include/barry/model-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



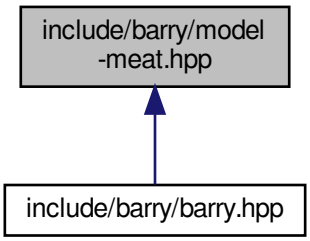
Classes

- class Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >

  General framework for discrete exponential models. This class allows generating discrete exponential models in the form of a linear exponential model:

9.35 include/barry/model-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define MODEL_TYPE()
- #define MODEL_TEMPLATE_ARGS()
- #define MODEL_TEMPLATE(a, b) template MODEL_TEMPLATE_ARGS() inline a MODEL_TYPE()::b

Functions

- double update_normalizing_constant (const double *params, const double *support, size_t k, size_t n)
- double likelihood_ (const double *stats_target, const std::vector< double > &params, const double normalizing_constant, size_t n_params, bool log_=false)
- MODEL_TEMPLATE (, Model)()
- MODEL_TEMPLATE (, Model)(const MODEL_TYPE() &Model_)

9.35.1 Macro Definition Documentation

9.35.1.1 MODEL_TEMPLATE

```
#define MODEL_TEMPLATE(
              a,
              b )  template MODEL_TEMPLATE_ARGS() inline a MODEL_TYPE()::b
```

Definition at line 87 of file model-meat.hpp.

9.35.1.2 MODEL_TEMPLATE_ARGS

```
#define MODEL_TEMPLATE_ARGS( )
```

Value:

```
<typename Array_Type, typename Data_Counter_Type,\
typename Data_Rule_Type, typename Data_Rule_Dyn_Type>
```

Definition at line 84 of file model-meat.hpp.

9.35.1.3 MODEL_TYPE

```
#define MODEL_TYPE( )
```

Value:

```
Model<Array_Type,Data_Counter_Type,Data_Rule_Type,\
Data_Rule_Dyn_Type>
```

Definition at line 81 of file model-meat.hpp.

9.35.2 Function Documentation

9.35.2.1 likelihood_()

```
double likelihood_ (
            const double * stats_target,
            const std::vector< double > & params,
            const double normalizing_constant,
            size_t n_params,
            bool log_ = false )  [inline]
```

Definition at line 45 of file model-meat.hpp.

9.35.2.2 MODEL_TEMPLATE() [1/2]

```
MODEL_TEMPLATE (
            Model  )
```

Definition at line 91 of file model-meat.hpp.

9.35.2.3 MODEL_TEMPLATE() [2/2]

```
MODEL_TEMPLATE (
            Model  ) const &
```

Definition at line 149 of file model-meat.hpp.

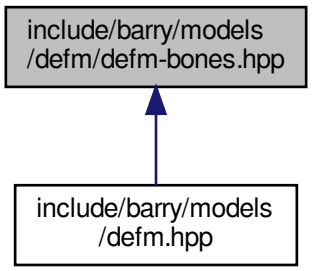
9.35.2.4 update_normalizing_constant()

```
double update_normalizing_constant (
            const double * params,
            const double * support,
            size_t k,
            size_t n )  [inline]
```

Definition at line 9 of file model-meat.hpp.

9.36 include/barry/models/defm/defm-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:

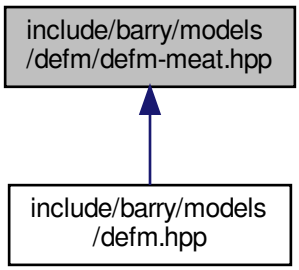


Classes

- class DEFM

9.37 include/barry/models/defm/defm-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define DEFM_RANGES(a)
- #define DEFM_LOOP_ARRAYS(a) for (size_t a = 0u; a < (nobs_i - M_order); ++a)

Functions

- std::vector< double > keygen_defm (const defmcounters::DEFMArray &Array_, defmcounters::↩ DEFMCounterData *data)

9.37.1 Macro Definition Documentation

9.37.1.1 DEFM_LOOP_ARRAYS

```
#define DEFM_LOOP_ARRAYS(
              a )  for (size_t a = 0u; a < (nobs_i - M_order); ++a)
```

Definition at line 35 of file defm-meat.hpp.

9.37.1.2 DEFM_RANGES

```
#define DEFM_RANGES(
              a )
```

Value:

```
    size_t __CONCAT(start_,a) = start_end[a * 2u];\
    size_t __CONCAT(end_,a)   = start_end[a * 2u + 1u];\
    size_t __CONCAT(nobs_,a)  = __CONCAT(end_,i) - __CONCAT(start_,i) + 1u;
```

Definition at line 30 of file defm-meat.hpp.

9.37.2 Function Documentation

9.37.2.1 keygen_defm()

```
std::vector< double > keygen_defm (
             const defmcounters::DEFMArray & Array_,
             defmcounters::DEFMCounterData * data )  [inline]
```

Definition at line 4 of file defm-meat.hpp.

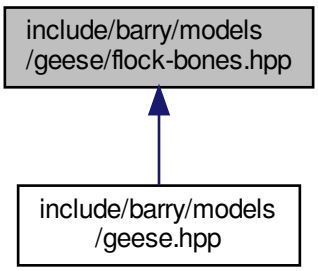
9.38 include/barry/models/geese.hpp File Reference

```
#include "geese/geese-node-bones.hpp"
#include "geese/geese-bones.hpp"
#include "geese/geese-meat.hpp"
#include "geese/geese-meat-constructors.hpp"
#include "geese/geese-meat-likelihood.hpp"
#include "geese/geese-meat-likelihood_exhaust.hpp"
#include "geese/geese-meat-simulate.hpp"
#include "geese/geese-meat-predict.hpp"
#include "geese/geese-meat-predict_exhaust.hpp"
#include "geese/geese-meat-predict_sim.hpp"
#include "geese/flock-bones.hpp"
#include "geese/flock-meat.hpp"
```

Include dependency graph for geese.hpp:

9.39 include/barry/models/geese/flock-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



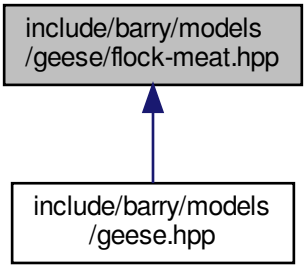
Classes

- class Flock

  A Flock is a group of Geese.

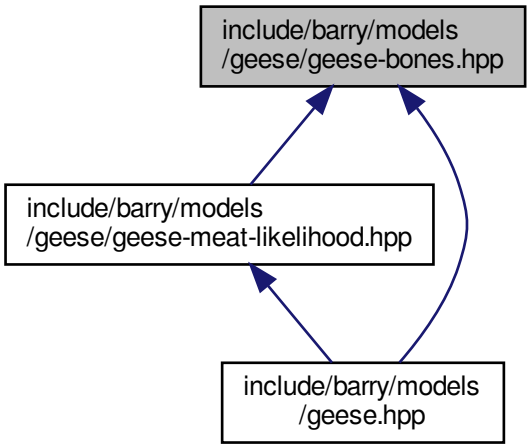
9.40 include/barry/models/geese/flock-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



9.41 include/barry/models/geese/geese-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class Geese

  Annotated Phylo Model.

Macros

- #define INITIALIZED()

Functions

- template<typename Ta , typename Tb >
  std::vector< Ta > vector_caster (const std::vector< Tb > &x)
- RULE_FUNCTION (rule_empty_free)
- std::vector< double > keygen_full (const phylocounters::PhyloArray &array, const phylocounters::Phylo↩
  CounterData *d)
- bool vec_diff (const std::vector< unsigned int > &s, const std::vector< unsigned int > &a)

9.41.1 Macro Definition Documentation

9.41.1.1 INITIALIZED

```
#define INITIALIZED( )
```

Value:

```
    if (!this->initialized) \
    throw std::logic_error("The model has not been initialized yet.");
```

Definition at line 22 of file geese-bones.hpp.

9.41.2 Function Documentation

9.41.2.1 keygen_full()

```
std::vector< double > keygen_full (
            const phylocounters::PhyloArray & array,
            const phylocounters::PhyloCounterData * d )  [inline]
```

Definition at line 36 of file geese-bones.hpp.

9.41.2.2 RULE_FUNCTION()

```
RULE_FUNCTION (
            rule_empty_free  )
```

Definition at line 26 of file geese-bones.hpp.

9.41.2.3 vec_diff()

```
bool vec_diff (
            const std::vector< unsigned int > & s,
            const std::vector< unsigned int > & a )  [inline]
```

Definition at line 61 of file geese-bones.hpp.

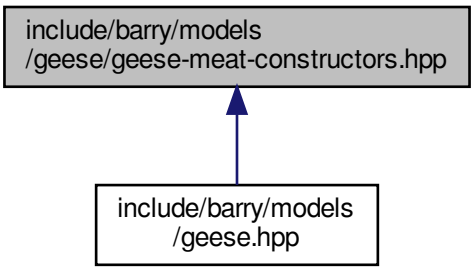
9.41.2.4 vector_caster()

```
template<typename Ta , typename Tb >
std::vector< Ta > vector_caster (
            const std::vector< Tb > & x )  [inline]
```

Definition at line 10 of file geese-bones.hpp.

9.42 include/barry/models/geese/geese-meat-constructors.hpp File Reference

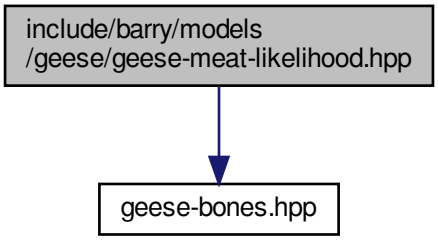
This graph shows which files directly or indirectly include this file:



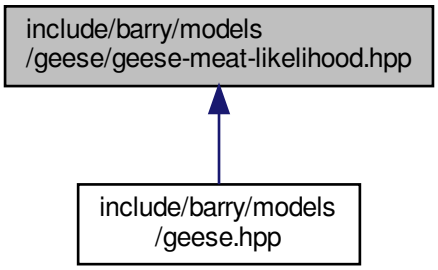
9.43 include/barry/models/geese/geese-meat-likelihood.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat-likelihood.hpp:

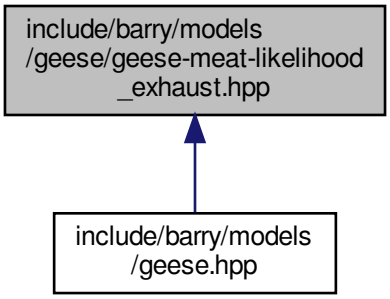


This graph shows which files directly or indirectly include this file:



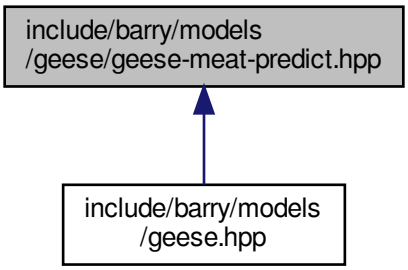
9.44 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference

This graph shows which files directly or indirectly include this file:



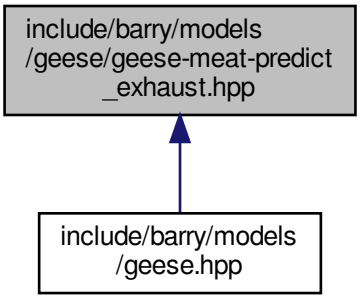
9.45 include/barry/models/geese/geese-meat-predict.hpp File Reference

This graph shows which files directly or indirectly include this file:



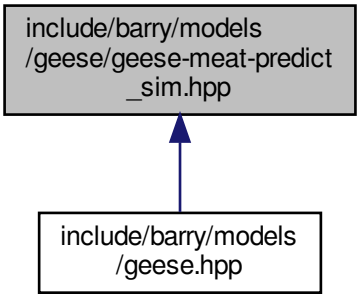
9.46 include/barry/models/geese/geese-meat-predict_exhaust.hpp File Reference

This graph shows which files directly or indirectly include this file:



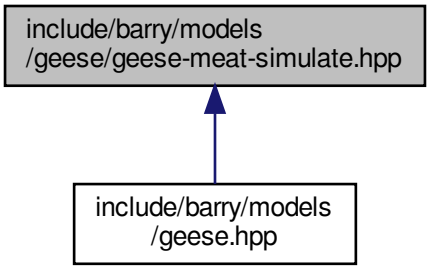
9.47 include/barry/models/geese/geese-meat-predict_sim.hpp File Reference

This graph shows which files directly or indirectly include this file:



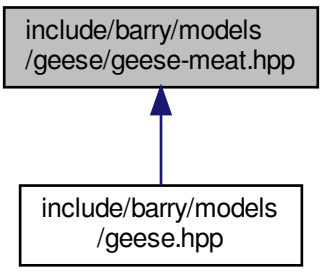
9.48 include/barry/models/geese/geese-meat-simulate.hpp File Reference

This graph shows which files directly or indirectly include this file:



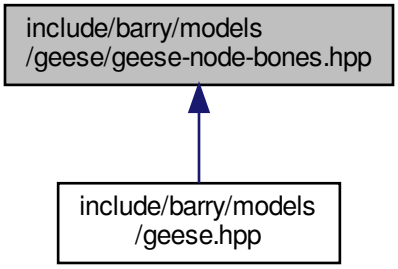
9.49 include/barry/models/geese/geese-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



9.50 include/barry/models/geese/geese-node-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



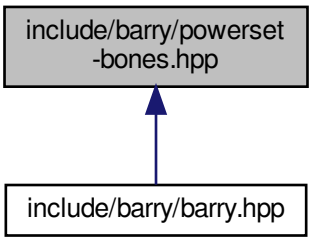
Classes

- class Node

  A single node for the model.

9.51 include/barry/powerset-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



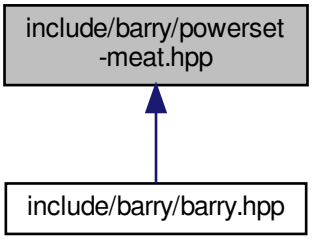
Classes

- class PowerSet< Array_Type, Data_Rule_Type >

  Powerset of a binary array.

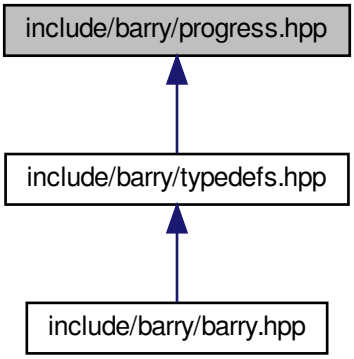
9.52 include/barry/powerset-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



9.53 include/barry/progress.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class Progress

  A simple progress bar.

Macros

- #define BARRY_PROGRESS_BAR_WIDTH 80

9.53.1 Macro Definition Documentation

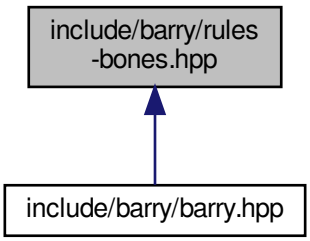
9.53.1.1 BARRY_PROGRESS_BAR_WIDTH

```
#define BARRY_PROGRESS_BAR_WIDTH 80
```

Definition at line 5 of file progress.hpp.

9.54 include/barry/rules-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class Rule< Array_Type, Data_Type >

  Rule for determining if a cell should be included in a sequence.
- class Rules< Array_Type, Data_Type >

  Vector of objects of class Rule.

Functions

- template<typename Array_Type , typename Data_Type >
  bool rule_fun_default (const Array_Type *array, uint i, uint j, Data_Type *dat)

9.54.1 Function Documentation

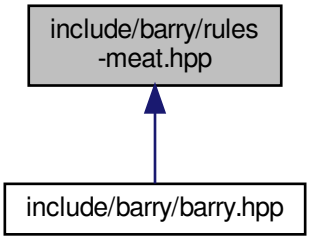
9.54.1.1 rule_fun_default()

```
template<typename Array_Type , typename Data_Type >
bool rule_fun_default (
              const Array_Type * array,
              uint i,
              uint j,
              Data_Type * dat )
```

Definition at line 5 of file rules-bones.hpp.

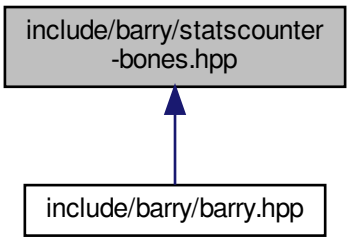
9.55 include/barry/rules-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



9.56 include/barry/statscounter-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



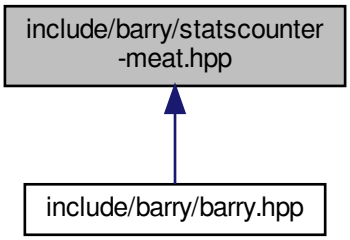
Classes

- class StatsCounter< Array_Type, Data_Type >

  Count stats for a single Array.

9.57 include/barry/statscounter-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define STATSCOUNTER_TYPE() StatsCounter<Array_Type,Data_Type>
- #define STATSCOUNTER_TEMPLATE_ARGS() <typename Array_Type, typename Data_Type>
- #define STATSCOUNTER_TEMPLATE(a, b) template STATSCOUNTER_TEMPLATE_ARGS() inline a STATSCOUNTER_TYPE()::b

Functions

- STATSCOUNTER_TEMPLATE (, StatsCounter)(const StatsCounter< Array_Type
- EmptyArray clear ()
- STATSCOUNTER_TEMPLATE (,~StatsCounter)()
- STATSCOUNTER_TEMPLATE (void, reset_array)(const Array_Type *Array_)
- STATSCOUNTER_TEMPLATE (void, add_counter)(Counter< Array_Type
- STATSCOUNTER_TEMPLATE (void, set_counters)(Counters< Array_Type
- STATSCOUNTER_TEMPLATE (void, count_init)(uint i
- current_stats resize (counters->size(), 0.0)
- for (uint n=0u;n< counters->size();++n) current_stats[n]
- STATSCOUNTER_TEMPLATE (void, count_current)(uint i
- STATSCOUNTER_TEMPLATE (std::vector< std::string >, get_names)() const
- STATSCOUNTER_TEMPLATE (std::vector< std::string >, get_descriptions)() const

Variables

- Data_Type & counter
- EmptyArray = ∗Array
- current_stats = counter.current_stats
- counters = new Counters<Array_Type,Data_Type>((∗counter.counters))
- counter_deleted = false
- Data_Type f_
- return
- Data_Type ∗ counters_
- uint j

9.57.1 Macro Definition Documentation

9.57.1.1 STATSCOUNTER_TEMPLATE

```
#define STATSCOUNTER_TEMPLATE(
              a,
              b )  template STATSCOUNTER_TEMPLATE_ARGS() inline a STATSCOUNTER_TYPE()::b
```

Definition at line 8 of file statscounter-meat.hpp.

9.57.1.2 STATSCOUNTER_TEMPLATE_ARGS

```
template STATSCOUNTER_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

Definition at line 6 of file statscounter-meat.hpp.

9.57.1.3 STATSCOUNTER_TYPE

```
template Data_Type * STATSCOUNTER_TYPE( ) StatsCounter<Array_Type,Data_Type>
```

Definition at line 4 of file statscounter-meat.hpp.

9.57.2 Function Documentation

9.57.2.1 clear()

```
EmptyArray clear ( )
```

9.57.2.2 for()

```
for (
            uint n = 0u;n< counters->size();++n )
```

9.57.2.3 resize()

```
current_stats resize (
            counters-> size(),
            0.  0 )
```

9.57.2.4 STATSCOUNTER_TEMPLATE() [1/9]

```
STATSCOUNTER_TEMPLATE (
            StatsCounter  ) const
```

9.57.2.5 STATSCOUNTER_TEMPLATE() [2/9]

```
STATSCOUNTER_TEMPLATE (
            ~ StatsCounter )
```

Definition at line 27 of file statscounter-meat.hpp.

9.57.2.6 STATSCOUNTER_TEMPLATE() [3/9]

```
STATSCOUNTER_TEMPLATE (
            std::vector< std::string > ,
            get_descriptions  ) const
```

Definition at line 256 of file statscounter-meat.hpp.

9.57.2.7 STATSCOUNTER_TEMPLATE() [4/9]

```
STATSCOUNTER_TEMPLATE (
            std::vector< std::string > ,
            get_names  ) const
```

Definition at line 251 of file statscounter-meat.hpp.

9.57.2.8 STATSCOUNTER_TEMPLATE() [5/9]

```
STATSCOUNTER_TEMPLATE (
            void ,
            add_counter  )
```

9.57.2.9 STATSCOUNTER_TEMPLATE() [6/9]

```
STATSCOUNTER_TEMPLATE (
            void ,
            count_current  )
```

9.57.2.10 STATSCOUNTER_TEMPLATE() [7/9]

```
STATSCOUNTER_TEMPLATE (
            void ,
            count_init  )
```

9.57.2.11 STATSCOUNTER_TEMPLATE() [8/9]

```
STATSCOUNTER_TEMPLATE (
            void ,
            reset_array  ) const
```

Definition at line 34 of file statscounter-meat.hpp.

9.57.2.12 STATSCOUNTER_TEMPLATE() [9/9]

```
STATSCOUNTER_TEMPLATE (
            void ,
            set_counters  )
```

9.57.3 Variable Documentation

9.57.3.1 counter

```
Data_Type& counter
```

Initial value:

```
{
    Array      = counter.Array
```

Definition at line 12 of file statscounter-meat.hpp.

9.57.3.2 counter_deleted

```
counter_deleted = false
```

Definition at line 23 of file statscounter-meat.hpp.

9.57.3.3 counters

```
counters = new Counters<Array_Type,Data_Type>((*counter.counters))
```

Definition at line 22 of file statscounter-meat.hpp.

9.57.3.4 counters_

```
Data_Type* counters_
```

Initial value:

```
{

    if (!counter_deleted)
        delete counters
```

Definition at line 53 of file statscounter-meat.hpp.

9.57.3.5 current_stats

```
current_stats = counter.current_stats
```

Definition at line 19 of file statscounter-meat.hpp.

9.57.3.6 EmptyArray

```
EmptyArray = *Array
```

Definition at line 17 of file statscounter-meat.hpp.

9.57.3.7 f_

```
Data_Rule_Dyn_Type f_
```

Initial value:

```
{
    counters->add_counter(f_)
```

Definition at line 44 of file statscounter-meat.hpp.

9.57.3.8 j

```
uint 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.

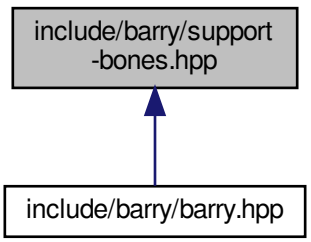
9.57.3.9 return

```
return
```

Definition at line 49 of file statscounter-meat.hpp.

9.58 include/barry/support-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



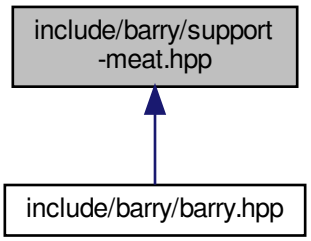
Classes

- class Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >

  Compute the support of sufficient statistics.

9.59 include/barry/support-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define BARRY_SUPPORT_MEAT_HPP 1
- #define SUPPORT_TEMPLATE_ARGS()
- #define SUPPORT_TYPE()
- #define SUPPORT_TEMPLATE(a, b)

Functions

- SUPPORT_TEMPLATE (void, init_support)(std
- SUPPORT_TEMPLATE (void, reset_array)()
- SUPPORT_TEMPLATE (void, reset_array)(const Array_Type &Array_)
- SUPPORT_TEMPLATE (void, calc_backend_sparse)(uint pos
- calc_backend_sparse (pos+1u, array_bank, stats_bank)
- EmptyArray insert_cell (coord_i, coord_j, EmptyArray.default_val().value, false, false)
- for (uint n=0u;n< n_counters;++n)
- if (rules_dyn->size() > 0u)
- if (array_bank !=nullptr) array_bank -> push_back(EmptyArray)
- EmptyArray rm_cell (coord_i, coord_j, false, false)
- if (change_stats_different > 0u)
- SUPPORT_TEMPLATE (void, calc_backend_dense)(uint pos
- calc_backend_dense (pos+1u, array_bank, stats_bank)
- EmptyArray insert_cell (coord_i, coord_j, 1, false, false)
- SUPPORT_TEMPLATE (void, calc)(std
- SUPPORT_TEMPLATE (void, add_counter)(Counter< Array_Type
- SUPPORT_TEMPLATE (void, set_counters)(Counters< Array_Type

- SUPPORT_TEMPLATE (void, add_rule)(Rule< Array_Type
- SUPPORT_TEMPLATE (void, set_rules)(Rules< Array_Type
- SUPPORT_TEMPLATE (void, add_rule_dyn)(Rule< Array_Type
- SUPPORT_TEMPLATE (void, set_rules_dyn)(Rules< Array_Type
- SUPPORT_TEMPLATE (bool, eval_rules_dyn)(const std
- SUPPORT_TEMPLATE (std::vector< double >, get_counts)() const
- SUPPORT_TEMPLATE (std::vector< double > *, get_current_stats)()
- SUPPORT_TEMPLATE (void, print)() const
- SUPPORT_TEMPLATE (const FreqTable< double > &, get_data)() const

Variables

- std::vector< Array_Type > * array_bank
- std::vector< Array_Type > std::vector< double > * stats_bank
- const size_t & coord_i = coordinates_free[pos * 2u]
- const size_t & coord_j = coordinates_free[pos * 2u + 1u]
- double tmp_chng
- unsigned int change_stats_different = hashes_initialized[pos] ? 0u : 1u
- else
- & hashes [pos]
- return
- Data_Counter_Type f_
- Data_Counter_Type * counters_
- delete_counters = false
- counters = counters_
- Data_Rule_Type * rules_
- delete_rules = false
- rules = rules_
- delete_rules_dyn = false
- rules_dyn = rules_

9.59.1 Macro Definition Documentation

9.59.1.1 BARRY_SUPPORT_MEAT_HPP

```
#define BARRY_SUPPORT_MEAT_HPP 1
```

Definition at line 2 of file support-meat.hpp.

9.59.1.2 SUPPORT_TEMPLATE

```
#define SUPPORT_TEMPLATE(
              a,
              b )
```

Value:

```
template SUPPORT_TEMPLATE_ARGS() \
inline a SUPPORT_TYPE()::b
```

Definition at line 10 of file support-meat.hpp.

9.59.1.3 SUPPORT_TEMPLATE_ARGS

```
template SUPPORT_TEMPLATE_ARGS( )
```

Value:

```
    <typename Array_Type, typename \
    Data_Counter_Type, typename Data_Rule_Type, typename Data_Rule_Dyn_Type>
```

Definition at line 4 of file support-meat.hpp.

9.59.1.4 SUPPORT_TYPE

```
template Data_Rule_Dyn_Type * SUPPORT_TYPE( )
```

Value:

```
    Support<Array_Type,Data_Counter_Type,Data_Rule_Type,\
    Data_Rule_Dyn_Type>
```

Definition at line 7 of file support-meat.hpp.

9.59.2 Function Documentation

9.59.2.1 calc_backend_dense()

```
calc_backend_dense (
            pos+ 1u,
            array_bank ,
            stats_bank  )
```

9.59.2.2 calc_backend_sparse()

```
calc_backend_sparse (
            pos+ 1u,
            array_bank ,
            stats_bank  )
```

9.59.2.3 for()

```
for ( )
```

Definition at line 159 of file support-meat.hpp.

9.59.2.4 if() [1/3]

```
if (
             array_bank !     = nullptr ) -> push_back(EmptyArray)
```

9.59.2.5 if() [2/3]

```
if (
             change_stats_different ,
             0u  )
```

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]

```
EmptyArray insert_cell (
             coord_i ,
             coord_j ,
             1 ,
             false ,
             false  )
```

9.59.2.8 insert_cell() [2/2]

```
EmptyArray insert_cell (
             coord_i ,
             coord_j ,
             EmptyArray.default_val().  value,
             false ,
             false  )
```

9.59.2.9 rm_cell()

```
EmptyArray rm_cell (
            coord_i ,
            coord_j ,
            false ,
            false  )
```

9.59.2.10 SUPPORT_TEMPLATE() [1/17]

```
SUPPORT_TEMPLATE (
            bool ,
            eval_rules_dyn  ) const
```

Definition at line 493 of file support-meat.hpp.

9.59.2.11 SUPPORT_TEMPLATE() [2/17]

```
SUPPORT_TEMPLATE (
            const FreqTable< double > & ,
            get_data  ) const
```

Definition at line 562 of file support-meat.hpp.

9.59.2.12 SUPPORT_TEMPLATE() [3/17]

```
SUPPORT_TEMPLATE (
            std::vector< double > * ,
            get_current_stats  )
```

Definition at line 547 of file support-meat.hpp.

9.59.2.13 SUPPORT_TEMPLATE() [4/17]

```
SUPPORT_TEMPLATE (
            std::vector< double > ,
            get_counts  ) const
```

Definition at line 535 of file support-meat.hpp.

9.59.2.14 SUPPORT_TEMPLATE() [5/17]

```
SUPPORT_TEMPLATE (
            void ,
            add_counter  )
```

9.59.2.15 SUPPORT_TEMPLATE() [6/17]

```
SUPPORT_TEMPLATE (
            void ,
            add_rule  )
```

9.59.2.16 SUPPORT_TEMPLATE() [7/17]

```
SUPPORT_TEMPLATE (
            void ,
            add_rule_dyn  )
```

9.59.2.17 SUPPORT_TEMPLATE() [8/17]

```
SUPPORT_TEMPLATE (
            void ,
            calc  )
```

Definition at line 371 of file support-meat.hpp.

9.59.2.18 SUPPORT_TEMPLATE() [9/17]

```
SUPPORT_TEMPLATE (
            void ,
            calc_backend_dense  )
```

9.59.2.19 SUPPORT_TEMPLATE() [10/17]

```
SUPPORT_TEMPLATE (
            void ,
            calc_backend_sparse  )
```

9.59.2.20 SUPPORT_TEMPLATE() [11/17]

```
SUPPORT_TEMPLATE (
            void ,
            init_support  )
```

Definition at line 13 of file support-meat.hpp.

9.59.2.21 SUPPORT_TEMPLATE() [12/17]

```
SUPPORT_TEMPLATE (
            void ,
            print  ) const
```

Definition at line 551 of file support-meat.hpp.

9.59.2.22 SUPPORT_TEMPLATE() [13/17]

```
SUPPORT_TEMPLATE (
            void ,
            reset_array  )
```

Definition at line 114 of file support-meat.hpp.

9.59.2.23 SUPPORT_TEMPLATE() [14/17]

```
SUPPORT_TEMPLATE (
            void ,
            reset_array  ) const &
```

Definition at line 120 of file support-meat.hpp.

9.59.2.24 SUPPORT_TEMPLATE() [15/17]

```
SUPPORT_TEMPLATE (
            void ,
            set_counters  )
```

9.59.2.25 SUPPORT_TEMPLATE() [16/17]

```
SUPPORT_TEMPLATE (
            void ,
            set_rules  )
```

9.59.2.26 SUPPORT_TEMPLATE() [17/17]

```
SUPPORT_TEMPLATE (
            void ,
            set_rules_dyn  )
```

9.59.3 Variable Documentation

9.59.3.1 array_bank

```
std::vector< Array_Type > * array_bank
```

Definition at line 131 of file support-meat.hpp.

9.59.3.2 change_stats_different

```
unsigned int change_stats_different = hashes_initialized[pos] ?  0u :  1u
```

Definition at line 158 of file support-meat.hpp.

9.59.3.3 coord_i

```
const size_t & coord_i = coordinates_free[pos * 2u]
```

Definition at line 144 of file support-meat.hpp.

9.59.3.4 coord_j

```
const size_t & coord_j = coordinates_free[pos * 2u + 1u]
```

Definition at line 145 of file support-meat.hpp.

9.59.3.5 counters

```
counters = counters_
```

Definition at line 421 of file support-meat.hpp.

9.59.3.6 counters_

```
Data_Counter_Type* counters_
```

Initial value:

```
{

    if (delete_counters)
        delete counters
```

Definition at line 414 of file support-meat.hpp.

9.59.3.7 delete_counters

```
delete_counters = false
```

Definition at line 420 of file support-meat.hpp.

9.59.3.8 delete_rules

```
delete_rules = false
```

Definition at line 454 of file support-meat.hpp.

9.59.3.9 delete_rules_dyn

```
delete_rules_dyn = false
```

Definition at line 486 of file support-meat.hpp.

9.59.3.10 else

```
else (
              void  )
```

Initial value:

```
{
        if (change_stats_different > 0u)
            hashes[pos] = data.add(current_stats, nullptr)
```

Definition at line 212 of file support-meat.hpp.

9.59.3.11 f_

```
Data_Rule_Dyn_Type f_
```

Initial value:

```
{

    counters->add_counter(f_)
```

Definition at line 405 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 251 of file support-meat.hpp.

9.59.3.14 rules

```
rules = rules_
```

Definition at line 455 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 448 of file support-meat.hpp.

9.59.3.16 rules_dyn

```
rules_dyn = rules_
```

Definition at line 487 of file support-meat.hpp.

9.59.3.17 stats_bank

```
std::vector< Array_Type > std::vector< double > * stats_bank
```

Initial value:
```
{

    if (pos >= coordiantes_n_free)
        return
```

Definition at line 132 of file support-meat.hpp.

9.59.3.18 tmp_chng

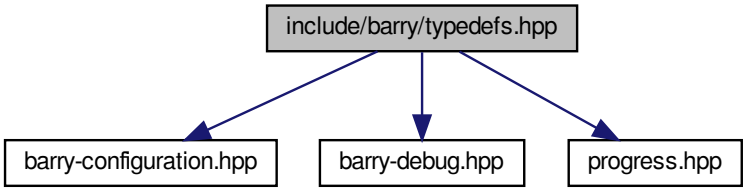
```
double tmp_chng
```

Definition at line 157 of file support-meat.hpp.

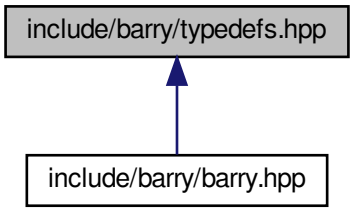
9.60 include/barry/typedefs.hpp File Reference

```
#include "barry-configuration.hpp"
#include "barry-debug.hpp"
#include "progress.hpp"
```

Include dependency graph for typedefs.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Entries< Cell_Type >

  A wrapper class to store `source`, `target`, `val` from a `BArray` object.
- struct vecHasher< T >

Namespaces

- CHECK

  Integer constants used to specify which cell should be check.
- EXISTS

  Integer constants used to specify which cell should be check to exist or not.

Typedefs

- typedef unsigned int uint
- typedef std::vector< std::pair< std::vector< double >, uint > > Counts_type
- template<typename Cell_Type >
  using Row_type = Map< uint, Cell< Cell_Type > >
- template<typename Cell_Type >
  using Col_type = Map< uint, Cell< Cell_Type > * >
- template<typename Ta = double, typename Tb = uint>
  using MapVec_type = std::unordered_map< std::vector< Ta >, Tb, vecHasher< Ta > >
- template<typename Array_Type , typename Data_Type >
  using 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 &, uint, uint, Data_Type &)>

  Counter and rule functions.
- template<typename Array_Type , typename Data_Type >
  using Rule_fun_type = std::function< bool(const Array_Type &, uint, uint, Data_Type &)>

Functions

- 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< uint, Cell<Cell_Type>* >
```

Definition at line 71 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 &, uint, uint, Data_Type &)>
```

Counter and rule functions.

Parameters

Array_Type	a BArray
unit,uint	Focal cell
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 188 of file typedefs.hpp.

9.60.1.3 Counts_type

```
typedef std::vector< std::pair< std::vector<double>, uint > > Counts_type
```

Definition at line 52 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 201 of file typedefs.hpp.

9.60.1.5 MapVec_type

```
template<typename Ta = double, typename Tb = uint>
using MapVec_type = std::unordered_map< std::vector< Ta >, Tb, vecHasher<Ta> >
```

Definition at line 129 of file typedefs.hpp.

9.60.1.6 Row_type

```
template<typename Cell_Type >
using Row_type = Map< uint, Cell<Cell_Type> >
```

Definition at line 68 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 &, uint, uint, Data_Type &)>
```

Definition at line 191 of file typedefs.hpp.

9.60.1.8 uint

```
typedef unsigned int uint
```

Definition at line 18 of file typedefs.hpp.

9.60.2 Function Documentation

9.60.2.1 sort_array()

```
std::vector< size_t > sort_array (
            const double * v,
            size_t start,
            size_t ncols,
            size_t nrows )  [inline]
```

Ascending sorting an array.

It will sort an array solving ties using the next column. Data is stored column-wise.

Template Parameters

T	

Parameters

v	
nrows	

Returns

std::vector<size_t> The sorting index.

Definition at line 142 of file typedefs.hpp.

9.60.2.2 vec_equal()

```
template<typename T >
bool vec_equal (
            const std::vector< T > & a,
            const std::vector< T > & b )  [inline]
```

Compares if -a- and -b- are equal.

Parameters

a,b	Two vectors of the same length

Returns

`true` if all elements are equal.

Definition at line 211 of file typedefs.hpp.

9.60.2.3 vec_equal_approx()

```
template<typename T >
bool vec_equal_approx (
            const std::vector< T > & a,
            const std::vector< T > & b,
            double eps = 1e-100 )  [inline]
```

Definition at line 229 of file typedefs.hpp.

9.60.2.4 vec_inner_prod() [1/2]

```
template<>
double vec_inner_prod (
             const double * a,
             const double * b,
             size_t n )  [inline]
```

Definition at line 275 of file typedefs.hpp.

9.60.2.5 vec_inner_prod() [2/2]

```
template<typename T >
T vec_inner_prod (
             const T * a,
             const T * b,
             size_t n )  [inline]
```

Definition at line 252 of file typedefs.hpp.

9.61 README.md File Reference

Index

~BArray
BArray< Cell_Type, Data_Type >, 47
~BArrayCell
BArrayCell< Cell_Type, Data_Type >, 58
~BArrayCell_const
BArrayCell_const< Cell_Type, Data_Type >, 60
~BArrayDense
BArrayDense< Cell_Type, Data_Type >, 66
~BArrayDenseCell
BArrayDenseCell< Cell_Type, Data_Type >, 79
~BArrayRow
BArrayRow< Cell_Type, Data_Type >, 92
~BArrayRow_const
BArrayRow_const< Cell_Type, Data_Type >, 94
~BArrayVector
BArrayVector< Cell_Type, Data_Type >, 97
~BArrayVector_const
BArrayVector_const< Cell_Type, Data_Type >, 100
~Cell
Cell< Cell_Type >, 104
~ConstBArrayRowIter
ConstBArrayRowIter< Cell_Type, Data_Type >, 109
~Counter
Counter< Array_Type, Data_Type >, 112
~Counters
Counters< Array_Type, Data_Type >, 117
~DEFM
DEFM, 121
~DEFMCounterData
DEFMCounterData, 126
~DEFMData
DEFMData, 130
~DEFMRuleDynData
DEFMRuleDynData, 136
~Entries
Entries< Cell_Type >, 137
~Flock
Flock, 140
~FreqTable
FreqTable< T >, 146
~Geese
Geese, 153
~Model
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 165
~NetCounterData
NetCounterData, 177
~NetworkData
NetworkData, 180
~Node
Node, 182
~PhyloRuleDynData
PhyloRuleDynData, 191
~PowerSet
PowerSet< Array_Type, Data_Rule_Type >, 194
~Progress
Progress, 199
~Rule
Rule< Array_Type, Data_Type >, 201
~Rules
Rules< Array_Type, Data_Type >, 204
~StatsCounter
StatsCounter< Array_Type, Data_Type >, 208
~Support
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 213

active
Cell< Cell_Type >, 107
add
barray-meat.hpp, 240
barraydense-meat.hpp, 265
Cell< Cell_Type >, 105, 106
FreqTable< T >, 147
add_array
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 166
add_counter
Counters< Array_Type, Data_Type >, 117, 118
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 166
StatsCounter< Array_Type, Data_Type >, 209
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 214
add_data
Flock, 140
add_dims
counters-meat.hpp, 295
add_hash
Counters< Array_Type, Data_Type >, 118
add_hasher

Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 166
add_rule
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 167
PowerSet< Array_Type, Data_Rule_Type >, 194, 195
Rules< Array_Type, Data_Type >, 205
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 214
add_rule_dyn
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 167
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 214
annotations
Node, 183
ans
barray-meat.hpp, 231, 240
barraydense-meat.hpp, 254, 265
Array
ConstBArrayRowIter< Cell_Type, Data_Type >, 109
array
DEFMData, 130
Node, 183
Array_
barray-meat.hpp, 240
array_bank
support-meat.hpp, 358
arrays
Node, 184
AS_ONE
EXISTS, 41
as_vector
FreqTable< T >, 147
AS_ZERO
EXISTS, 41
at
DEFMData, 130
PhyloCounterData, 188

BArray
BArray< Cell_Type, Data_Type >, 46, 47
BArray< Cell_Type, Data_Type >, 43
~BArray, 47
BArray, 46, 47
BArrayCell< Cell_Type, Data_Type >, 56
BArrayCell_const< Cell_Type, Data_Type >, 56
clear, 47
col, 47
D, 48
D_ptr, 48
default_val, 48
flush_data, 48
get_cell, 48
get_col_vec, 49
get_entries, 49
get_row_vec, 49
insert_cell, 50
is_dense, 50
is_empty, 50
ncol, 51
nnozero, 51
nrow, 51
operator*=, 51
operator(), 51
operator+=, 52
operator-=, 52
operator/=, 53
operator=, 53
operator==, 53
out_of_range, 53
print, 53
reserve, 54
resize, 54
rm_cell, 54
row, 54
set_data, 54
swap_cells, 55
swap_cols, 55
swap_rows, 55
toggle_cell, 55
toggle_lock, 55
transpose, 56
visited, 57
zero_col, 56
zero_row, 56
barray-meat-operators.hpp
BARRAY_TEMPLATE, 224–226
BARRAY_TEMPLATE_ARGS, 225, 227
BARRAY_TYPE, 225, 227
COL, 225
for, 227
operator(), 227
rhs, 227
ROW, 225
this, 228
barray-meat.hpp
add, 240
ans, 231, 240
Array_, 240
BARRAY_TEMPLATE, 230–235
BARRAY_TEMPLATE_ARGS, 230
BARRAY_TYPE, 230
check_bounds, 240
check_exists, 241
COL, 231, 235
col0, 241
const, 241
copy_data, 241
data, 242

delete_data, 242
delete_data_, 242
else, 242
false, 242
first, 243
for, 236
i1, 243
if, 236–239
j, 243
j0, 243
j1, 243
M, 239, 243
M_, 244
N, 244
NCells, 244
report, 244
resize, 239
return, 239, 244
ROW, 231, 239, 240
row0, 245
search, 245
source, 245
target, 245
v, 245
value, 245
BARRAY_TEMPLATE
barray-meat-operators.hpp, 224–226
barray-meat.hpp, 230–235
BARRAY_TEMPLATE_ARGS
barray-meat-operators.hpp, 225, 227
barray-meat.hpp, 230
BARRAY_TYPE
barray-meat-operators.hpp, 225, 227
barray-meat.hpp, 230
BArrayCell
BArrayCell< Cell_Type, Data_Type >, 58
BArrayCell< Cell_Type, Data_Type >, 57
~BArrayCell, 58
BArray< Cell_Type, Data_Type >, 56
BArrayCell, 58
operator Cell_Type, 58
operator*=, 58
operator+=, 58
operator-=, 59
operator/=, 59
operator=, 59
operator==, 59
BArrayCell_const
BArrayCell_const< Cell_Type, Data_Type >, 60
BArrayCell_const< Cell_Type, Data_Type >, 60
~BArrayCell_const, 60
BArray< Cell_Type, Data_Type >, 56
BArrayCell_const, 60
operator Cell_Type, 61
operator!=, 61
operator<, 61
operator<=, 61
operator>, 61
operator>=, 62
operator==, 61
BArrayDense
BArrayDense< Cell_Type, Data_Type >, 65, 66
BArrayDense< Cell_Type, Data_Type >, 62
~BArrayDense, 66
BArrayDense, 65, 66
BArrayDenseCell< Cell_Type, Data_Type >, 76, 81
BArrayDenseCol< Cell_Type, Data_Type >, 77, 84
BArrayDenseCol_const< Cell_Type, Data_Type >, 77
BArrayDenseRow< Cell_Type, Data_Type >, 77, 88
BArrayDenseRow_const< Cell_Type, Data_Type >, 77
clear, 67
col, 67
colsum, 67
D, 67, 68
D_ptr, 68
default_val, 68
get_cell, 68
get_col_vec, 68, 69
get_data, 69
get_entries, 69
get_row_vec, 69
insert_cell, 70
is_dense, 70
is_empty, 70
ncol, 70
nnozero, 71
nrow, 71
operator*=, 71
operator(), 71
operator+=, 71, 72
operator-=, 72
operator/=, 72
operator=, 73
operator==, 73
out_of_range, 73
print, 73
reserve, 73
resize, 74
rm_cell, 74
row, 74
rowsum, 74
set_data, 74
swap_cells, 75
swap_cols, 75
swap_rows, 75
toggle_cell, 75
toggle_lock, 76
transpose, 76
visited, 77
zero_col, 76
zero_row, 76
barraydense-meat-operators.hpp

BDENSE_TEMPLATE, 248–250
BDENSE_TEMPLATE_ARGS, 248, 250
BDENSE_TYPE, 248, 250
COL, 248
POS, 248
POS_N, 249
ROW, 249
barraydense-meat.hpp
add, 265
ans, 254, 265
BDENSE_TEMPLATE, 253–261
BDENSE_TEMPLATE_ARGS, 253
BDENSE_TYPE, 253
check_bounds, 265
check_exists, 265
COL, 253
col, 266
const, 266
copy_data, 266
data, 266
delete_data, 266
delete_data_, 267
el, 267
el_colsums, 267
el_rowsums, 267
else, 267
false, 268
for, 261
i1, 268
if, 262
insert_cell, 262
j, 268
j0, 268
j1, 268
M, 263, 268
M_, 269
N, 269
POS, 253
POS_N, 253
report, 269
resize, 263, 264
return, 269
rm_cell, 264
ROW, 254
source, 269
target, 270
v, 270
va_end, 264
va_start, 264
val0, 270
val1, 270
value, 270
vprintf, 265
ZERO_CELL, 254
BArrayDenseCell
BArrayDenseCell< Cell_Type, Data_Type >, 79
BArrayDenseCell< Cell_Type, Data_Type >, 78
∼BArrayDenseCell, 79
BArrayDense< Cell_Type, Data_Type >, 76, 81
BArrayDenseCell, 79
BArrayDenseCol< Cell_Type, Data_Type >, 81, 84
BArrayDenseCol_const< Cell_Type, Data_Type >, 81, 86
BArrayDenseRow< Cell_Type, Data_Type >, 88
BArrayDenseRow_const< Cell_Type, Data_Type >, 91
operator Cell_Type, 79
operator∗=, 79
operator+=, 79
operator-=, 80
operator/=, 80
operator=, 80
operator==, 80
barraydensecell-bones.hpp
POS, 271
barraydensecell-meat.hpp
POS, 272
BArrayDenseCell_const< Cell_Type, Data_Type >, 82
BArrayDenseCol< Cell_Type, Data_Type >, 84
BArrayDenseCol_const< Cell_Type, Data_Type >, 86
BArrayDenseRow< Cell_Type, Data_Type >, 88
BArrayDenseRow_const< Cell_Type, Data_Type >, 91
BArrayDenseCol
BArrayDenseCol< Cell_Type, Data_Type >, 82
BArrayDenseCol< Cell_Type, Data_Type >, 82
BArrayDense< Cell_Type, Data_Type >, 77, 84
BArrayDenseCell< Cell_Type, Data_Type >, 81, 84
BArrayDenseCell_const< Cell_Type, Data_Type >, 84
BArrayDenseCol, 82
begin, 83
end, 83
operator(), 83
size, 83
barraydensecol-bones.hpp
POS, 273
POS_N, 273
ZERO_CELL, 273
BArrayDenseCol_const
BArrayDenseCol_const< Cell_Type, Data_Type >, 85
BArrayDenseCol_const< Cell_Type, Data_Type >, 84
BArrayDense< Cell_Type, Data_Type >, 77
BArrayDenseCell< Cell_Type, Data_Type >, 81, 86
BArrayDenseCell_const< Cell_Type, Data_Type >, 86
BArrayDenseCol_const, 85
begin, 85
end, 85
operator(), 85
size, 86
BArrayDenseRow

BArrayDenseRow< Cell_Type, Data_Type >, 87
BArrayDenseRow< Cell_Type, Data_Type >, 86
BArrayDense< Cell_Type, Data_Type >, 77, 88
BArrayDenseCell< Cell_Type, Data_Type >, 88
BArrayDenseCell_const< Cell_Type, Data_Type >, 88
BArrayDenseRow, 87
begin, 87
end, 87
operator(), 88
size, 88
barraydenserow-bones.hpp
POS, 274
POS_N, 274
ZERO_CELL, 275
BArrayDenseRow_const
BArrayDenseRow_const< Cell_Type, Data_Type >, 89
BArrayDenseRow_const< Cell_Type, Data_Type >, 89
BArrayDense< Cell_Type, Data_Type >, 77
BArrayDenseCell< Cell_Type, Data_Type >, 91
BArrayDenseCell_const< Cell_Type, Data_Type >, 91
BArrayDenseRow_const, 89
begin, 90
end, 90
operator(), 90
size, 90
BArrayRow
BArrayRow< Cell_Type, Data_Type >, 92
BArrayRow< Cell_Type, Data_Type >, 91
~BArrayRow, 92
BArrayRow, 92
operator BArrayRow< Cell_Type, Data_Type >, 92
operator*=, 92
operator+=, 92
operator-=, 92
operator/=, 93
operator=, 93
operator==, 93
barrayrow-meat.hpp
BROW_TEMPLATE, 275–277
BROW_TEMPLATE_ARGS, 276
BROW_TYPE, 276
BArrayRow_const
BArrayRow_const< Cell_Type, Data_Type >, 94
BArrayRow_const< Cell_Type, Data_Type >, 93
~BArrayRow_const, 94
BArrayRow_const, 94
operator BArrayRow_const< Cell_Type, Data_Type >, 94
operator!=, 94
operator<, 94
operator<=, 95
operator>, 95
operator>=, 95
operator==, 95
BArrayVector
BArrayVector< Cell_Type, Data_Type >, 96
BArrayVector< Cell_Type, Data_Type >, 95
~BArrayVector, 97
BArrayVector, 96
begin, 97
end, 97
is_col, 97
is_row, 97
operator std::vector< Cell_Type >, 98
operator*=, 98
operator+=, 98
operator-=, 98
operator/=, 98
operator=, 99
operator==, 99
size, 99
BArrayVector_const
BArrayVector_const< Cell_Type, Data_Type >, 100
BArrayVector_const< Cell_Type, Data_Type >, 99
~BArrayVector_const, 100
BArrayVector_const, 100
begin, 101
end, 101
is_col, 101
is_row, 101
operator std::vector< Cell_Type >, 101
operator!=, 101
operator<, 102
operator<=, 102
operator>, 102
operator>=, 102
operator==, 102
size, 103
barry, 39
barry-configuration.hpp
BARRY_CHECK_SUPPORT, 279
BARRY_ISFINITE, 279
BARRY_MAX_NUM_ELEMENTS, 279
BARRY_SAFE_EXP, 279
Map, 279
printf_barry, 279
barry-debug.hpp
BARRY_DEBUG_LEVEL, 280
barry-macros.hpp
BARRY_ONE, 281
BARRY_ONE_DENSE, 281
BARRY_UNUSED, 281
BARRY_ZERO, 282
BARRY_ZERO_DENSE, 282
barry.hpp
BARRY_HPP, 283
BARRY_VERSION, 284
BARRY_VERSION_MAYOR, 284
BARRY_VERSION_MINOR, 284
COUNTER_FUNCTION, 284
COUNTER_LAMBDA, 284
RULE_FUNCTION, 285

RULE_LAMBDA, 285
barry::counters, 39
barry::counters::defm, 40
barry::counters::network, 40
barry::counters::phylo, 40
BARRY_CHECK_SUPPORT
barry-configuration.hpp, 279
BARRY_DEBUG_LEVEL
barry-debug.hpp, 280
BARRY_HPP
barry.hpp, 283
BARRY_ISFINITE
barry-configuration.hpp, 279
BARRY_MAX_NUM_ELEMENTS
barry-configuration.hpp, 279
BARRY_ONE
barry-macros.hpp, 281
BARRY_ONE_DENSE
barry-macros.hpp, 281
BARRY_PROGRESS_BAR_WIDTH
progress.hpp, 343
BARRY_SAFE_EXP
barry-configuration.hpp, 279
BARRY_SUPPORT_MEAT_HPP
support-meat.hpp, 352
BARRY_UNUSED
barry-macros.hpp, 281
BARRY_VERSION
barry.hpp, 284
BARRY_VERSION_MAYOR
barry.hpp, 284
BARRY_VERSION_MINOR
barry.hpp, 284
BARRY_ZERO
barry-macros.hpp, 282
BARRY_ZERO_DENSE
barry-macros.hpp, 282
BARRY_ZERO_NETWORK
network.hpp, 315
BARRY_ZERO_NETWORK_DENSE
network.hpp, 315
BDENSE_TEMPLATE
barraydense-meat-operators.hpp, 248–250
barraydense-meat.hpp, 253–261
BDENSE_TEMPLATE_ARGS
barraydense-meat-operators.hpp, 248, 250
barraydense-meat.hpp, 253
BDENSE_TYPE
barraydense-meat-operators.hpp, 248, 250
barraydense-meat.hpp, 253
begin
BArrayDenseCol< Cell_Type, Data_Type >, 83
BArrayDenseCol_const< Cell_Type, Data_Type >, 85
BArrayDenseRow< Cell_Type, Data_Type >, 87
BArrayDenseRow_const< Cell_Type, Data_Type >, 90
BArrayVector< Cell_Type, Data_Type >, 97
BArrayVector_const< Cell_Type, Data_Type >, 101
PhyloCounterData, 188
PowerSet< Array_Type, Data_Rule_Type >, 195
blengths
NodeData, 187
BOTH
CHECK, 40
EXISTS, 42
BROW_TEMPLATE
barrayrow-meat.hpp, 275–277
BROW_TEMPLATE_ARGS
barrayrow-meat.hpp, 276
BROW_TYPE
barrayrow-meat.hpp, 276

calc
PowerSet< Array_Type, Data_Rule_Type >, 195
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 214
calc_backend_dense
support-meat.hpp, 353
calc_backend_sparse
support-meat.hpp, 353
calc_reduced_sequence
Geese, 153
calc_sequence
Geese, 153
Cell
Cell< Cell_Type >, 104, 105
Cell< Cell_Type >, 103
~Cell, 104
active, 107
add, 105, 106
Cell, 104, 105
operator Cell_Type, 106
operator!=, 106
operator=, 106, 107
operator==, 107
value, 107
visited, 107
Cell_const< Cell_Type >, 108
change_stats
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 218
change_stats_different
support-meat.hpp, 358
CHECK, 40
BOTH, 40
NONE, 40
ONE, 40
TWO, 41
check_bounds
barray-meat.hpp, 240
barraydense-meat.hpp, 265
check_exists
barray-meat.hpp, 241

barraydense-meat.hpp, 265
clear
BArray< Cell_Type, Data_Type >, 47
BArrayDense< Cell_Type, Data_Type >, 67
FreqTable< T >, 147
statscounter-meat.hpp, 346
COL
barray-meat-operators.hpp, 225
barray-meat.hpp, 231, 235
barraydense-meat-operators.hpp, 248
barraydense-meat.hpp, 253
col
BArray< Cell_Type, Data_Type >, 47
BArrayDense< Cell_Type, Data_Type >, 67
barraydense-meat.hpp, 266
col0
barray-meat.hpp, 241
Col_type
typedefs.hpp, 364
colnames
Flock, 141
Geese, 154
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 167
colsum
BArrayDense< Cell_Type, Data_Type >, 67
conditional_prob
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 168
const
barray-meat.hpp, 241
barraydense-meat.hpp, 266
ConstBArrayRowIter
ConstBArrayRowIter< Cell_Type, Data_Type >, 109
ConstBArrayRowIter< Cell_Type, Data_Type >, 108
~ConstBArrayRowIter, 109
Array, 109
ConstBArrayRowIter, 109
current_col, 110
current_row, 110
iter, 110
coord_i
support-meat.hpp, 358
coord_j
support-meat.hpp, 358
coordiantes_n_free
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 218
coordiantes_n_locked
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 218
coordinates_free
PowerSet< Array_Type, Data_Rule_Type >, 197
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 218
coordinates_locked
PowerSet< Array_Type, Data_Rule_Type >, 197
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 219
copy_data
barray-meat.hpp, 241
barraydense-meat.hpp, 266
count
Counter< Array_Type, Data_Type >, 113
count_all
StatsCounter< Array_Type, Data_Type >, 209
count_current
StatsCounter< Array_Type, Data_Type >, 209
count_fun
Counter< Array_Type, Data_Type >, 114
counters-meat.hpp, 290
count_fun_
counters-meat.hpp, 295
count_init
StatsCounter< Array_Type, Data_Type >, 209
Counter
Counter< Array_Type, Data_Type >, 112
counter
counters-meat.hpp, 296
statscounter-meat.hpp, 348
Counter< Array_Type, Data_Type >, 110
~Counter, 112
count, 113
count_fun, 114
Counter, 112
data, 114
desc, 115
get_description, 113
get_hasher, 113
get_name, 113
hasher_fun, 115
init, 113
init_fun, 115
name, 115
operator=, 113, 114
set_hasher, 114
counter_
counters-meat.hpp, 296
counter_absdiff
DEFMArray counters, 21
counter_co_opt
Phylo counters, 32
counter_cogain
Phylo counters, 32
counter_css_census01
network-css.hpp, 308
counter_css_census02
network-css.hpp, 308
counter_css_census03

network-css.hpp, 308
counter_css_census04
network-css.hpp, 309
counter_css_census05
network-css.hpp, 309
counter_css_census06
network-css.hpp, 309
counter_css_census07
network-css.hpp, 309
counter_css_census08
network-css.hpp, 310
counter_css_census09
network-css.hpp, 310
counter_css_census10
network-css.hpp, 310
counter_css_completely_false_recip_comiss
network-css.hpp, 310
counter_css_completely_false_recip_omiss
network-css.hpp, 311
counter_css_mixed_recip
network-css.hpp, 311
counter_css_partially_false_recip_commi
network-css.hpp, 311
counter_css_partially_false_recip_omiss
network-css.hpp, 312
counter_ctriads
DEFMArray counters, 22
counter_degree
DEFMArray counters, 22
counter_deleted
statscounter-meat.hpp, 349
counter_density
DEFMArray counters, 22
counter_diff
DEFMArray counters, 23
counter_edges
DEFMArray counters, 23
counter_fixed_effect
DEFMArray counters, 23
Counter_fun_type
typedefs.hpp, 364
COUNTER_FUNCTION
barry.hpp, 284
counter_gains
Phylo counters, 32
counter_gains_from_0
Phylo counters, 32
counter_gains_k_offspring
Phylo counters, 33
counter_genes_changing
Phylo counters, 33
counter_idegree
DEFMArray counters, 24
counter_idegree15
DEFMArray counters, 24
counter_isolates
DEFMArray counters, 24, 25
counter_istar2
DEFMArray counters, 25
counter_k_genes_changing
Phylo counters, 33
COUNTER_LAMBDA
barry.hpp, 284
counter_less_than_p_prop_genes_changing
Phylo counters, 33
counter_logit_intercept
DEFMArray counters, 25
counter_longest
Phylo counters, 34
counter_loss
Phylo counters, 34
counter_maxfuns
Phylo counters, 34
counter_mutual
DEFMArray counters, 25
counter_neofun
Phylo counters, 34
counter_neofun_a2b
Phylo counters, 35
counter_nodecov
DEFMArray counters, 26
counter_nodeicov
DEFMArray counters, 26
counter_nodematch
DEFMArray counters, 26
counter_nodeocov
DEFMArray counters, 26
counter_odegree
DEFMArray counters, 26, 27
counter_odegree15
DEFMArray counters, 27
counter_ones
DEFMArray counters, 27
counter_ostar2
DEFMArray counters, 28
counter_overall_changes
Phylo counters, 35
counter_overall_gains
Phylo counters, 35
counter_overall_gains_from_0
Phylo counters, 35
counter_overall_loss
Phylo counters, 36
counter_pairwise_first_gain
Phylo counters, 36
counter_pairwise_neofun_singlefun
Phylo counters, 36
counter_pairwise_overall_change
Phylo counters, 36
counter_pairwise_preserving
Phylo counters, 37
counter_preserve_pseudogene
Phylo counters, 37
counter_prop_genes_changing
Phylo counters, 37
counter_subfun

Phylo counters, 37
COUNTER_TEMPLATE
counters-meat.hpp, 288, 290, 291
COUNTER_TEMPLATE_ARGS
counters-meat.hpp, 288
counter_transition
DEFMArray counters, 28
counter_transition_formula
DEFMArray counters, 29
counter_ttriads
DEFMArray counters, 29
COUNTER_TYPE
counters-meat.hpp, 289
Counters
Counters< Array_Type, Data_Type >, 116, 117
counters
statscounter-meat.hpp, 349
support-meat.hpp, 358
Counters< Array_Type, Data_Type >, 116
~Counters, 117
add_counter, 117, 118
add_hash, 118
Counters, 116, 117
gen_hash, 118
get_descriptions, 118
get_names, 119
operator=, 119
operator[], 120
size, 120
counters-meat.hpp
add_dims, 295
count_fun, 290
count_fun_, 295
counter, 296
counter_, 296
COUNTER_TEMPLATE, 288, 290, 291
COUNTER_TEMPLATE_ARGS, 288
COUNTER_TYPE, 289
COUNTERS_TEMPLATE, 289, 291–293
COUNTERS_TEMPLATE_ARGS, 289
COUNTERS_TYPE, 289
data, 293
data_, 296
desc, 293
desc_, 296
for, 293
fun, 297
fun_, 297
hasher, 293, 294
hasher_fun, 294
hasher_fun_, 297
i, 297
if, 294
init_fun, 295
init_fun_, 298
j, 298
name, 295
name_, 298
noexcept, 298
res, 298
return, 299
TMP_HASHER_CALL, 289
counters_
statscounter-meat.hpp, 349
support-meat.hpp, 359
COUNTERS_TEMPLATE
counters-meat.hpp, 289, 291–293
COUNTERS_TEMPLATE_ARGS
counters-meat.hpp, 289
COUNTERS_TYPE
counters-meat.hpp, 289
Counting, 13
counts
DEFMRuleDynData, 136
PhyloRuleDynData, 191
Counts_type
typedefs.hpp, 364
covar_sort
DEFMData, 130
covar_used
DEFMData, 130
covariates
DEFMData, 131
CSS_APPEND
network-css.hpp, 306
CSS_CASE_ELSE
network-css.hpp, 306
CSS_CASE_PERCEIVED
network-css.hpp, 306
CSS_CASE_TRUTH
network-css.hpp, 306
CSS_CHECK_SIZE
network-css.hpp, 306
CSS_CHECK_SIZE_INIT
network-css.hpp, 307
CSS_NET_COUNTER_LAMBDA_INIT
network-css.hpp, 307
CSS_PERCEIVED_CELLS
network-css.hpp, 307
CSS_SIZE
network-css.hpp, 307
CSS_TRUE_CELLS
network-css.hpp, 308
current_col
ConstBArrayRowIter< Cell_Type, Data_Type >, 110
current_row
ConstBArrayRowIter< Cell_Type, Data_Type >, 110
current_stats
statscounter-meat.hpp, 349
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 219

D
BArray< Cell_Type, Data_Type >, 48

BArrayDense< Cell_Type, Data_Type >, 67, 68
Rule< Array_Type, Data_Type >, 202
D_ptr
BArray< Cell_Type, Data_Type >, 48
BArrayDense< Cell_Type, Data_Type >, 68
dat
Flock, 144
data
barray-meat.hpp, 242
barraydense-meat.hpp, 266
Counter< Array_Type, Data_Type >, 114
counters-meat.hpp, 293
PowerSet< Array_Type, Data_Rule_Type >, 197
data_
counters-meat.hpp, 296
DEFAULT_DUPLICATION
phylo.hpp, 322
default_val
BArray< Cell_Type, Data_Type >, 48
BArrayDense< Cell_Type, Data_Type >, 68
DEFM, 120
∼DEFM, 121
DEFM, 121
get_ID, 122
get_m_order, 122
get_model, 122
get_n_covars, 122
get_n_obs, 122
get_n_rows, 122
get_n_y, 123
get_X, 123
get_X_names, 123
get_Y, 123
get_Y_names, 123
init, 123
likelihood, 124
logodds, 124
motif_census, 124
set_names, 124
simulate, 124
defm-formula.hpp
defm_motif_parser, 299
defm-meat.hpp
DEFM_LOOP_ARRAYS, 332
DEFM_RANGES, 332
keygen_defm, 332
defm.hpp
DEFMArray, 303
UNI_SUB, 303
DEFM_COUNTER
Phylo rules, 15
DEFM_COUNTER_LAMBDA
Phylo rules, 15
DEFM_LOOP_ARRAYS
defm-meat.hpp, 332
defm_motif_parser
defm-formula.hpp, 299
DEFM_RANGES
defm-meat.hpp, 332
DEFM_RULE
Phylo rules, 15
DEFM_RULE_LAMBDA
Phylo rules, 16
DEFM_RULEDYN_LAMBDA
Phylo rules, 16
DEFMArray
defm.hpp, 303
DEFMArray counters, 19
counter_absdiff, 21
counter_ctriads, 22
counter_degree, 22
counter_density, 22
counter_diff, 23
counter_edges, 23
counter_fixed_effect, 23
counter_idegree, 24
counter_idegree15, 24
counter_isolates, 24, 25
counter_istar2, 25
counter_logit_intercept, 25
counter_mutual, 25
counter_nodecov, 26
counter_nodeicov, 26
counter_nodematch, 26
counter_nodeocov, 26
counter_odegree, 26, 27
counter_odegree15, 27
counter_ones, 27
counter_ostar2, 28
counter_transition, 28
counter_transition_formula, 29
counter_ttriads, 29
NETWORK_COUNTER, 29
rules_dont_become_zero, 29
rules_markov_fixed, 30
DEFMCounter
Phylo rules, 17
DEFMCounterData, 125
∼DEFMCounterData, 126
DEFMCounterData, 125, 126
idx, 126
indices, 127
is_true, 126
logical, 127
num, 126
numbers, 127
DEFMCounters
Phylo rules, 17
DEFMData, 127
∼DEFMData, 130
array, 130
at, 130
covar_sort, 130
covar_used, 130
covariates, 131
DEFMData, 128

obs_start, 131
X_ncol, 131
X_nrow, 131
DEFMModel
Phylo rules, 17
DEFMRule
Phylo rules, 17
DEFMRuleData, 132
DEFMRuleData, 132, 133
idx, 133
indices, 134
init, 134
is_true, 133
logical, 134
num, 133
numbers, 134
DEFMRuleDyn
Phylo rules, 17
DEFMRuleDynData, 135
∼DEFMRuleDynData, 136
counts, 136
DEFMRuleDynData, 136
DEFMRules
Phylo rules, 17
DEFMRulesDyn
Phylo rules, 18
DEFMStatsCounter
Phylo rules, 18
DEFMSupport
Phylo rules, 18
delete_counters
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 219
support-meat.hpp, 359
delete_data
barray-meat.hpp, 242
barraydense-meat.hpp, 266
delete_data_
barray-meat.hpp, 242
barraydense-meat.hpp, 267
delete_rengine
Geese, 160
delete_rules
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 219
support-meat.hpp, 359
delete_rules_dyn
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 219
support-meat.hpp, 359
delete_support
Geese, 160
desc
Counter< Array_Type, Data_Type >, 115
counters-meat.hpp, 293
desc_
counters-meat.hpp, 296
directed
NetworkData, 180
DUPL_DUPL
phylo.hpp, 322
DUPL_EITH
phylo.hpp, 322
DUPL_SPEC
phylo.hpp, 322
duplication
Node, 184
NodeData, 187
PhyloRuleDynData, 191

el
barraydense-meat.hpp, 267
el_colsums
barraydense-meat.hpp, 267
el_rowsums
barraydense-meat.hpp, 267
else
barray-meat.hpp, 242
barraydense-meat.hpp, 267
support-meat.hpp, 359
empty
PhyloCounterData, 188
EmptyArray
PowerSet< Array_Type, Data_Rule_Type >, 197
statscounter-meat.hpp, 349
end
BArrayDenseCol< Cell_Type, Data_Type >, 83
BArrayDenseCol_const< Cell_Type, Data_Type >, 85
BArrayDenseRow< Cell_Type, Data_Type >, 87
BArrayDenseRow_const< Cell_Type, Data_Type >, 90
BArrayVector< Cell_Type, Data_Type >, 97
BArrayVector_const< Cell_Type, Data_Type >, 101
PhyloCounterData, 189
PowerSet< Array_Type, Data_Rule_Type >, 195
Progress, 200
Entries
Entries< Cell_Type >, 137
Entries< Cell_Type >, 136
∼Entries, 137
Entries, 137
resize, 138
source, 138
target, 138
val, 138
eval_rules_dyn
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 215
EXISTS, 41
AS_ONE, 41
AS_ZERO, 41

BOTH, 42
NONE, 42
ONE, 42
TWO, 42
UKNOWN, 42
f_
statscounter-meat.hpp, 350
support-meat.hpp, 360
false
barray-meat.hpp, 242
barraydense-meat.hpp, 268
first
barray-meat.hpp, 243
Flock, 139
∼Flock, 140
add_data, 140
colnames, 141
dat, 144
Flock, 140
get_counters, 141
get_model, 141
get_stats_support, 141
get_stats_target, 141
get_support_fun, 141
init, 142
initialized, 145
likelihood_joint, 142
model, 145
nfunctions, 145
nfuns, 142
nleafs, 142
nnodes, 143
nterms, 143
ntrees, 143
operator(), 143
parse_polytomies, 144
print, 144
rengine, 145
set_seed, 144
support_size, 144
flush_data
BArray< Cell_Type, Data_Type >, 48
for
barray-meat-operators.hpp, 227
barray-meat.hpp, 236
barraydense-meat.hpp, 261
counters-meat.hpp, 293
statscounter-meat.hpp, 346
support-meat.hpp, 353
FreqTable
FreqTable< T >, 146
FreqTable< T >, 145
∼FreqTable, 146
add, 147
as_vector, 147
clear, 147
FreqTable, 146
get_data, 147
get_index, 147
make_hash, 148
print, 148
reserve, 148
size, 148
fun
counters-meat.hpp, 297
fun_
counters-meat.hpp, 297

Geese, 149
∼Geese, 153
calc_reduced_sequence, 153
calc_sequence, 153
colnames, 154
delete_rengine, 160
delete_support, 160
Geese, 152, 153
get_annotated_nodes, 154
get_counters, 154
get_model, 154
get_probabilities, 154
get_rengine, 154
get_states, 155
get_support_fun, 155
inherit_support, 155
init, 155
init_node, 155
initialized, 160
likelihood, 156
likelihood_exhaust, 156
map_to_nodes, 161
nannotations, 156
nfunctions, 161
nfuns, 156
nleafs, 156
nnodes, 157
nodes, 161
nterms, 157
observed_counts, 157
operator=, 157
parse_polytomies, 157
predict, 158
predict_backend, 158
predict_exhaust, 158
predict_exhaust_backend, 158
predict_sim, 159
print, 159
print_observed_counts, 159
pset_loc, 161
reduced_sequence, 161
sequence, 161
set_seed, 159
simulate, 159
support_size, 160
update_annotations, 160
geese-bones.hpp
INITIALIZED, 335
keygen_full, 335

RULE_FUNCTION, 335
vec_diff, 335
vector_caster, 336
gen_hash
Counters< Array_Type, Data_Type >, 118
gen_key
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 168
get_annotated_nodes
Geese, 154
get_arrays2support
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 168
get_cell
BArray< Cell_Type, Data_Type >, 48
BArrayDense< Cell_Type, Data_Type >, 68
get_col_vec
BArray< Cell_Type, Data_Type >, 49
BArrayDense< Cell_Type, Data_Type >, 68, 69
get_counters
Flock, 141
Geese, 154
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 169
PhyloCounterData, 189
StatsCounter< Array_Type, Data_Type >, 209
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 215
get_counts
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 215
get_current_stats
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 215
get_data
BArrayDense< Cell_Type, Data_Type >, 69
FreqTable< T >, 147
PowerSet< Array_Type, Data_Rule_Type >, 195
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 216
get_data_ptr
PowerSet< Array_Type, Data_Rule_Type >, 196
get_description
Counter< Array_Type, Data_Type >, 113
Rule< Array_Type, Data_Type >, 202
get_descriptions
Counters< Array_Type, Data_Type >, 118
Rules< Array_Type, Data_Type >, 205
StatsCounter< Array_Type, Data_Type >, 209
get_entries
BArray< Cell_Type, Data_Type >, 49
BArrayDense< Cell_Type, Data_Type >, 69
get_hasher
Counter< Array_Type, Data_Type >, 113
get_ID
DEFM, 122
get_index
FreqTable< T >, 147
get_last_name
phylo.hpp, 327
get_m_order
DEFM, 122
get_model
DEFM, 122
Flock, 141
Geese, 154
get_n_covars
DEFM, 122
get_n_obs
DEFM, 122
get_n_rows
DEFM, 122
get_n_y
DEFM, 123
get_name
Counter< Array_Type, Data_Type >, 113
Rule< Array_Type, Data_Type >, 202
get_names
Counters< Array_Type, Data_Type >, 119
Rules< Array_Type, Data_Type >, 205
StatsCounter< Array_Type, Data_Type >, 210
get_norm_const
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 169
get_parent
Node, 183
get_probabilities
Geese, 154
get_pset
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 169
get_pset_arrays
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 169
get_pset_probs
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 169
get_pset_stats
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 170
get_rengine
Geese, 154
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >,

170
get_row_vec
BArray< Cell_Type, Data_Type >, 49
BArrayDense< Cell_Type, Data_Type >, 69
get_rules
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 170
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 216
get_rules_dyn
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 170
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 216
get_seq
Rules< Array_Type, Data_Type >, 205
get_states
Geese, 155
get_stats_support
Flock, 141
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 171
get_stats_target
Flock, 141
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 171
get_support_fun
Flock, 141
Geese, 155
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 171
get_X
DEFM, 123
get_X_names
DEFM, 123
get_Y
DEFM, 123
get_Y_names
DEFM, 123

hasher
counters-meat.hpp, 293, 294
hasher_fun
Counter< Array_Type, Data_Type >, 115
counters-meat.hpp, 294
hasher_fun_
counters-meat.hpp, 297
Hasher_fun_type
typedefs.hpp, 364
hashes
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 220
support-meat.hpp, 360
hashes_initialized
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 220

i
counters-meat.hpp, 297
i1
barray-meat.hpp, 243
barraydense-meat.hpp, 268
id
Node, 184
idx
DEFMCounterData, 126
DEFMRuleData, 133
if
barray-meat.hpp, 236–239
barraydense-meat.hpp, 262
counters-meat.hpp, 294
support-meat.hpp, 353, 354
IF_MATCHES
phylo.hpp, 322
IF_NOTMATCHES
phylo.hpp, 323
include/barry/barray-bones.hpp, 223
include/barry/barray-iterator.hpp, 223
include/barry/barray-meat-operators.hpp, 224
include/barry/barray-meat.hpp, 228
include/barry/barraycell-bones.hpp, 246
include/barry/barraycell-meat.hpp, 246
include/barry/barraydense-bones.hpp, 247
include/barry/barraydense-meat-operators.hpp, 247
include/barry/barraydense-meat.hpp, 250
include/barry/barraydensecell-bones.hpp, 271
include/barry/barraydensecell-meat.hpp, 272
include/barry/barraydensecol-bones.hpp, 272
include/barry/barraydenserow-bones.hpp, 274
include/barry/barrayrow-bones.hpp, 275
include/barry/barrayrow-meat.hpp, 275
include/barry/barrayvector-bones.hpp, 277
include/barry/barrayvector-meat.hpp, 278
include/barry/barry-configuration.hpp, 278
include/barry/barry-debug.hpp, 280
include/barry/barry-macros.hpp, 281
include/barry/barry.hpp, 282
include/barry/cell-bones.hpp, 285
include/barry/cell-meat.hpp, 286
include/barry/col-bones.hpp, 286
include/barry/counters-bones.hpp, 286
include/barry/counters-meat.hpp, 287
include/barry/counters/defm-formula.hpp, 299
include/barry/counters/defm.hpp, 301
include/barry/counters/network-css.hpp, 304
include/barry/counters/network.hpp, 312
include/barry/counters/phylo.hpp, 319
include/barry/freqtable.hpp, 327
include/barry/model-bones.hpp, 328

include/barry/model-meat.hpp, 328
include/barry/models/defm.hpp, 304
include/barry/models/defm/defm-bones.hpp, 331
include/barry/models/defm/defm-meat.hpp, 331
include/barry/models/geese.hpp, 333
include/barry/models/geese/flock-bones.hpp, 333
include/barry/models/geese/flock-meat.hpp, 334
include/barry/models/geese/geese-bones.hpp, 334
include/barry/models/geese/geese-meat-constructors.hpp, 336
include/barry/models/geese/geese-meat-likelihood.hpp, 337
include/barry/models/geese/geese-meat-likelihood_exhaust.hpp, 338
include/barry/models/geese/geese-meat-predict.hpp, 338
include/barry/models/geese/geese-meat-predict_exhaust.hpp, 339
include/barry/models/geese/geese-meat-predict_sim.hpp, 339
include/barry/models/geese/geese-meat-simulate.hpp, 340
include/barry/models/geese/geese-meat.hpp, 340
include/barry/models/geese/geese-node-bones.hpp, 341
include/barry/powerset-bones.hpp, 341
include/barry/powerset-meat.hpp, 342
include/barry/progress.hpp, 342
include/barry/rules-bones.hpp, 343
include/barry/rules-meat.hpp, 344
include/barry/statscounter-bones.hpp, 344
include/barry/statscounter-meat.hpp, 345
include/barry/support-bones.hpp, 350
include/barry/support-meat.hpp, 351
include/barry/typedefs.hpp, 362
indices
  DEFMCounterData, 127
  DEFMRuleData, 134
  NetCounterData, 178
inherit_support
  Geese, 155
init
  Counter< Array_Type, Data_Type >, 113
  DEFM, 123
  DEFMRuleData, 134
  Flock, 142
  Geese, 155
init_fun
  Counter< Array_Type, Data_Type >, 115
  counters-meat.hpp, 295
init_fun_
  counters-meat.hpp, 298
init_node
  Geese, 155
init_support
  PowerSet< Array_Type, Data_Rule_Type >, 196
  Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 216
INITIALIZED
  geese-bones.hpp, 335
initialized
  Flock, 145
  Geese, 160
insert_cell
  BArray< Cell_Type, Data_Type >, 50
  BArrayDense< Cell_Type, Data_Type >, 70
  barraydense-meat.hpp, 262
  support-meat.hpp, 354
is_col
  BArrayVector< Cell_Type, Data_Type >, 97
  BArrayVector_const< Cell_Type, Data_Type >, 101
is_dense
  BArray< Cell_Type, Data_Type >, 50
  BArrayDense< Cell_Type, Data_Type >, 70
IS_DUPLICATION
  phylo.hpp, 323
IS_EITHER
  phylo.hpp, 323
is_empty
  BArray< Cell_Type, Data_Type >, 50
  BArrayDense< Cell_Type, Data_Type >, 70
is_leaf
  Node, 183
is_row
  BArrayVector< Cell_Type, Data_Type >, 97
  BArrayVector_const< Cell_Type, Data_Type >, 101
IS_SPECIATION
  phylo.hpp, 323
is_true
  DEFMCounterData, 126
  DEFMRuleData, 133
iter
  ConstBArrayRowIter< Cell_Type, Data_Type >, 110

j
  barray-meat.hpp, 243
  barraydense-meat.hpp, 268
  counters-meat.hpp, 298
  statscounter-meat.hpp, 350
j0
  barray-meat.hpp, 243
  barraydense-meat.hpp, 268
j1
  barray-meat.hpp, 243
  barraydense-meat.hpp, 268

keygen_defm
  defm-meat.hpp, 332
keygen_full
  geese-bones.hpp, 335

lb
  PhyloRuleDynData, 191

likelihood
DEFM, 124
Geese, 156
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 171, 172
likelihood_
model-meat.hpp, 329
likelihood_exhaust
Geese, 156
likelihood_joint
Flock, 142
likelihood_total
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 172
logical
DEFMCounterData, 127
DEFMRuleData, 134
logodds
DEFM, 124

M
barray-meat.hpp, 239, 243
barraydense-meat.hpp, 263, 268
PowerSet< Array_Type, Data_Rule_Type >, 197
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 220
M_
barray-meat.hpp, 244
barraydense-meat.hpp, 269
MAKE_DEFM_HASHER
Phylo rules, 16
MAKE_DUPL_VARS
phylo.hpp, 323
make_hash
FreqTable< T >, 148
Map
barry-configuration.hpp, 279
map_to_nodes
Geese, 161
MapVec_type
typedefs.hpp, 365
max_num_elements
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 220
Model
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 165
model
Flock, 145
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 162
~Model, 165
add_array, 166
add_counter, 166
add_hasher, 166
add_rule, 167
add_rule_dyn, 167
colnames, 167
conditional_prob, 168
gen_key, 168
get_arrays2support, 168
get_counters, 169
get_norm_const, 169
get_pset, 169
get_pset_arrays, 169
get_pset_probs, 169
get_pset_stats, 170
get_rengine, 170
get_rules, 170
get_rules_dyn, 170
get_stats_support, 171
get_stats_target, 171
get_support_fun, 171
likelihood, 171, 172
likelihood_total, 172
Model, 165
nrules, 172
nrules_dyn, 173
nterms, 173
operator=, 173
print, 173
print_stats, 173
sample, 174
set_counters, 174
set_rengine, 174
set_rules, 174
set_rules_dyn, 175
set_seed, 175
set_transform_model, 175
size, 176
size_unique, 176
store_psets, 176
support_size, 176
transform_model, 176
model-meat.hpp
likelihood_, 329
MODEL_TEMPLATE, 329, 330
MODEL_TEMPLATE_ARGS, 329
MODEL_TYPE, 329
update_normalizing_constant, 330
MODEL_TEMPLATE
model-meat.hpp, 329, 330
MODEL_TEMPLATE_ARGS
model-meat.hpp, 329
MODEL_TYPE
model-meat.hpp, 329
motif_census
DEFM, 124

N
barray-meat.hpp, 244
barraydense-meat.hpp, 269
PowerSet< Array_Type, Data_Rule_Type >, 198

Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 220
n_counters
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 221
n_free
PowerSet< Array_Type, Data_Rule_Type >, 198
n_locked
PowerSet< Array_Type, Data_Rule_Type >, 198
name
Counter< Array_Type, Data_Type >, 115
counters-meat.hpp, 295
name_
counters-meat.hpp, 298
nannotations
Geese, 156
narray
Node, 184
NCells
barray-meat.hpp, 244
ncol
BArray< Cell_Type, Data_Type >, 51
BArrayDense< Cell_Type, Data_Type >, 70
Phylo rules, 18
NET_C_DATA_IDX
network.hpp, 316
NET_C_DATA_NUM
network.hpp, 316
NetCounter
network.hpp, 317
NetCounterData, 177
∼NetCounterData, 177
indices, 178
NetCounterData, 177
numbers, 178
NetCounters
network.hpp, 317
NetModel
network.hpp, 318
NetRule
network.hpp, 318
NetRules
network.hpp, 318
NetStatsCounter
network.hpp, 318
NetSupport
network.hpp, 318
Network
network.hpp, 318
network-css.hpp
counter_css_census01, 308
counter_css_census02, 308
counter_css_census03, 308
counter_css_census04, 309
counter_css_census05, 309
counter_css_census06, 309
counter_css_census07, 309
counter_css_census08, 310
counter_css_census09, 310
counter_css_census10, 310
counter_css_completely_false_recip_comiss, 310
counter_css_completely_false_recip_omiss, 311
counter_css_mixed_recip, 311
counter_css_partially_false_recip_commi, 311
counter_css_partially_false_recip_omiss, 312
CSS_APPEND, 306
CSS_CASE_ELSE, 306
CSS_CASE_PERCEIVED, 306
CSS_CASE_TRUTH, 306
CSS_CHECK_SIZE, 306
CSS_CHECK_SIZE_INIT, 307
CSS_NET_COUNTER_LAMBDA_INIT, 307
CSS_PERCEIVED_CELLS, 307
CSS_SIZE, 307
CSS_TRUE_CELLS, 308
network.hpp
BARRY_ZERO_NETWORK, 315
BARRY_ZERO_NETWORK_DENSE, 315
NET_C_DATA_IDX, 316
NET_C_DATA_NUM, 316
NetCounter, 317
NetCounters, 317
NetModel, 318
NetRule, 318
NetRules, 318
NetStatsCounter, 318
NetSupport, 318
Network, 318
NETWORK_COUNTER, 316
NETWORK_COUNTER_LAMBDA, 316
NETWORK_RULE, 316
NETWORK_RULE_LAMBDA, 317
NetworkDense, 319
NETWORKDENSE_COUNTER_LAMBDA, 317
rules_zerodiag, 319
NETWORK_COUNTER
DEFMArray counters, 29
network.hpp, 316
NETWORK_COUNTER_LAMBDA
network.hpp, 316
NETWORK_RULE
network.hpp, 316
NETWORK_RULE_LAMBDA
network.hpp, 317
NetworkData, 178
∼NetworkData, 180
directed, 180
NetworkData, 179
vertex_attr, 180
NetworkDense
network.hpp, 319
NETWORKDENSE_COUNTER_LAMBDA
network.hpp, 317
next

Progress, 200
nfunctions
Flock, 145
Geese, 161
nfuns
Flock, 142
Geese, 156
nleafs
Flock, 142
Geese, 156
nnodes
Flock, 143
Geese, 157
nnozero
BArray< Cell_Type, Data_Type >, 51
BArrayDense< Cell_Type, Data_Type >, 71
Node, 180
~Node, 182
annotations, 183
array, 183
arrays, 184
duplication, 184
get_parent, 183
id, 184
is_leaf, 183
narray, 184
Node, 181, 182
noffspring, 183
offspring, 184
ord, 185
parent, 185
probability, 185
subtree_prob, 185
visited, 185
NodeData, 186
blengths, 187
duplication, 187
NodeData, 186
states, 187
nodes
Geese, 161
noexcept
counters-meat.hpp, 298
noffspring
Node, 183
NONE
CHECK, 40
EXISTS, 42
nrow
BArray< Cell_Type, Data_Type >, 51
BArrayDense< Cell_Type, Data_Type >, 71
Phylo rules, 18
nrules
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 172
nrules_dyn
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 173
nterms
Flock, 143
Geese, 157
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 173
ntrees
Flock, 143
num
DEFMCounterData, 126
DEFMRuleData, 133
numbers
DEFMCounterData, 127
DEFMRuleData, 134
NetCounterData, 178

obs_start
DEFMData, 131
observed_counts
Geese, 157
offspring
Node, 184
ONE
CHECK, 40
EXISTS, 42
operator BArrayRow< Cell_Type, Data_Type >
BArrayRow< Cell_Type, Data_Type >, 92
operator BArrayRow_const< Cell_Type, Data_Type >
BArrayRow_const< Cell_Type, Data_Type >, 94
operator Cell_Type
BArrayCell< Cell_Type, Data_Type >, 58
BArrayCell_const< Cell_Type, Data_Type >, 61
BArrayDenseCell< Cell_Type, Data_Type >, 79
Cell< Cell_Type >, 106
operator std::vector< Cell_Type >
BArrayVector< Cell_Type, Data_Type >, 98
BArrayVector_const< Cell_Type, Data_Type >, 101
operator!=
BArrayCell_const< Cell_Type, Data_Type >, 61
BArrayRow_const< Cell_Type, Data_Type >, 94
BArrayVector_const< Cell_Type, Data_Type >, 101
Cell< Cell_Type >, 106
operator<
BArrayCell_const< Cell_Type, Data_Type >, 61
BArrayRow_const< Cell_Type, Data_Type >, 94
BArrayVector_const< Cell_Type, Data_Type >, 102
operator<=
BArrayCell_const< Cell_Type, Data_Type >, 61
BArrayRow_const< Cell_Type, Data_Type >, 95
BArrayVector_const< Cell_Type, Data_Type >, 102
operator>
BArrayCell_const< Cell_Type, Data_Type >, 61
BArrayRow_const< Cell_Type, Data_Type >, 95

BArrayVector_const< Cell_Type, Data_Type >, 102
operator>=
BArrayCell_const< Cell_Type, Data_Type >, 62
BArrayRow_const< Cell_Type, Data_Type >, 95
BArrayVector_const< Cell_Type, Data_Type >, 102
operator*=
BArray< Cell_Type, Data_Type >, 51
BArrayCell< Cell_Type, Data_Type >, 58
BArrayDense< Cell_Type, Data_Type >, 71
BArrayDenseCell< Cell_Type, Data_Type >, 79
BArrayRow< Cell_Type, Data_Type >, 92
BArrayVector< Cell_Type, Data_Type >, 98
operator()
BArray< Cell_Type, Data_Type >, 51
barray-meat-operators.hpp, 227
BArrayDense< Cell_Type, Data_Type >, 71
BArrayDenseCol< Cell_Type, Data_Type >, 83
BArrayDenseCol_const< Cell_Type, Data_Type >, 85
BArrayDenseRow< Cell_Type, Data_Type >, 88
BArrayDenseRow_const< Cell_Type, Data_Type >, 90
Flock, 143
Phylo rules, 18
PhyloCounterData, 189
Rule< Array_Type, Data_Type >, 203
Rules< Array_Type, Data_Type >, 206
vecHasher< T >, 221
operator+=
BArray< Cell_Type, Data_Type >, 52
BArrayCell< Cell_Type, Data_Type >, 58
BArrayDense< Cell_Type, Data_Type >, 71, 72
BArrayDenseCell< Cell_Type, Data_Type >, 79
BArrayRow< Cell_Type, Data_Type >, 92
BArrayVector< Cell_Type, Data_Type >, 98
operator-=
BArray< Cell_Type, Data_Type >, 52
BArrayCell< Cell_Type, Data_Type >, 59
BArrayDense< Cell_Type, Data_Type >, 72
BArrayDenseCell< Cell_Type, Data_Type >, 80
BArrayRow< Cell_Type, Data_Type >, 92
BArrayVector< Cell_Type, Data_Type >, 98
operator/=
BArray< Cell_Type, Data_Type >, 53
BArrayCell< Cell_Type, Data_Type >, 59
BArrayDense< Cell_Type, Data_Type >, 72
BArrayDenseCell< Cell_Type, Data_Type >, 80
BArrayRow< Cell_Type, Data_Type >, 93
BArrayVector< Cell_Type, Data_Type >, 98
operator=
BArray< Cell_Type, Data_Type >, 53
BArrayCell< Cell_Type, Data_Type >, 59
BArrayDense< Cell_Type, Data_Type >, 73
BArrayDenseCell< Cell_Type, Data_Type >, 80
BArrayRow< Cell_Type, Data_Type >, 93
BArrayVector< Cell_Type, Data_Type >, 99
Cell< Cell_Type >, 106, 107
Counter< Array_Type, Data_Type >, 113, 114
Counters< Array_Type, Data_Type >, 119
Geese, 157
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 173
Rules< Array_Type, Data_Type >, 206
operator==
BArray< Cell_Type, Data_Type >, 53
BArrayCell< Cell_Type, Data_Type >, 59
BArrayCell_const< Cell_Type, Data_Type >, 61
BArrayDense< Cell_Type, Data_Type >, 73
BArrayDenseCell< Cell_Type, Data_Type >, 80
BArrayRow< Cell_Type, Data_Type >, 93
BArrayRow_const< Cell_Type, Data_Type >, 95
BArrayVector< Cell_Type, Data_Type >, 99
BArrayVector_const< Cell_Type, Data_Type >, 102
Cell< Cell_Type >, 107
operator[]
Counters< Array_Type, Data_Type >, 120
PhyloCounterData, 189
PowerSet< Array_Type, Data_Rule_Type >, 196
ord
Node, 185
out_of_range
BArray< Cell_Type, Data_Type >, 53
BArrayDense< Cell_Type, Data_Type >, 73

parent
Node, 185
parse_polytomies
Flock, 144
Geese, 157
Phylo counters, 30
counter_co_opt, 32
counter_cogain, 32
counter_gains, 32
counter_gains_from_0, 32
counter_gains_k_offspring, 33
counter_genes_changing, 33
counter_k_genes_changing, 33
counter_less_than_p_prop_genes_changing, 33
counter_longest, 34
counter_loss, 34
counter_maxfuns, 34
counter_neofun, 34
counter_neofun_a2b, 35
counter_overall_changes, 35
counter_overall_gains, 35
counter_overall_gains_from_0, 35
counter_overall_loss, 36
counter_pairwise_first_gain, 36
counter_pairwise_neofun_singlefun, 36
counter_pairwise_overall_change, 36
counter_pairwise_preserving, 37
counter_preserve_pseudogene, 37
counter_prop_genes_changing, 37

counter_subfun, 37
Phylo rules, 14
DEFM_COUNTER, 15
DEFM_COUNTER_LAMBDA, 15
DEFM_RULE, 15
DEFM_RULE_LAMBDA, 16
DEFM_RULEDYN_LAMBDA, 16
DEFMCounter, 17
DEFMCounters, 17
DEFMModel, 17
DEFMRule, 17
DEFMRuleDyn, 17
DEFMRules, 17
DEFMRulesDyn, 18
DEFMStatsCounter, 18
DEFMSupport, 18
MAKE_DEFM_HASHER, 16
ncol, 18
nrow, 18
operator(), 18
print, 19
rule_dyn_limit_changes, 19
phylo.hpp
DEFAULT_DUPLICATION, 322
DUPL_DUPL, 322
DUPL_EITH, 322
DUPL_SPEC, 322
get_last_name, 327
IF_MATCHES, 322
IF_NOTMATCHES, 323
IS_DUPLICATION, 323
IS_EITHER, 323
IS_SPECIATION, 323
MAKE_DUPL_VARS, 323
PHYLO_CHECK_MISSING, 324
PHYLO_COUNTER_LAMBDA, 324
PHYLO_RULE_DYN_LAMBDA, 324
PhyloArray, 325
PhyloCounter, 325
PhyloCounters, 325
PhyloModel, 325
PhyloPowerSet, 325
PhyloRule, 325
PhyloRuleData, 326
PhyloRuleDyn, 326
PhyloRules, 326
PhyloRulesDyn, 326
PhyloStatsCounter, 326
PhyloSupport, 326
PHYLO_CHECK_MISSING
phylo.hpp, 324
PHYLO_COUNTER_LAMBDA
phylo.hpp, 324
PHYLO_RULE_DYN_LAMBDA
phylo.hpp, 324
PhyloArray
phylo.hpp, 325
PhyloCounter
phylo.hpp, 325
PhyloCounterData, 187
at, 188
begin, 188
empty, 188
end, 189
get_counters, 189
operator(), 189
operator[], 189
PhyloCounterData, 188
push_back, 189
reserve, 189
shrink_to_fit, 190
size, 190
PhyloCounters
phylo.hpp, 325
PhyloModel
phylo.hpp, 325
PhyloPowerSet
phylo.hpp, 325
PhyloRule
phylo.hpp, 325
PhyloRuleData
phylo.hpp, 326
PhyloRuleDyn
phylo.hpp, 326
PhyloRuleDynData, 190
∼PhyloRuleDynData, 191
counts, 191
duplication, 191
lb, 191
PhyloRuleDynData, 191
pos, 191
ub, 192
PhyloRules
phylo.hpp, 326
PhyloRulesDyn
phylo.hpp, 326
PhyloStatsCounter
phylo.hpp, 326
PhyloSupport
phylo.hpp, 326
POS
barraydense-meat-operators.hpp, 248
barraydense-meat.hpp, 253
barraydensecell-bones.hpp, 271
barraydensecell-meat.hpp, 272
barraydensecol-bones.hpp, 273
barraydenserow-bones.hpp, 274
pos
PhyloRuleDynData, 191
POS_N
barraydense-meat-operators.hpp, 249
barraydense-meat.hpp, 253
barraydensecol-bones.hpp, 273
barraydenserow-bones.hpp, 274
PowerSet
PowerSet< Array_Type, Data_Rule_Type >, 194

PowerSet< Array_Type, Data_Rule_Type >, 192
~PowerSet, 194
add_rule, 194, 195
begin, 195
calc, 195
coordinates_free, 197
coordinates_locked, 197
data, 197
EmptyArray, 197
end, 195
get_data, 195
get_data_ptr, 196
init_support, 196
M, 197
N, 198
n_free, 198
n_locked, 198
operator[], 196
PowerSet, 194
reset, 196
rules, 198
rules_deleted, 198
size, 196
predict
Geese, 158
predict_backend
Geese, 158
predict_exhaust
Geese, 158
predict_exhaust_backend
Geese, 158
predict_sim
Geese, 159
print
BArray< Cell_Type, Data_Type >, 53
BArrayDense< Cell_Type, Data_Type >, 73
Flock, 144
FreqTable< T >, 148
Geese, 159
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 173
Phylo rules, 19
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 216
print_observed_counts
Geese, 159
print_stats
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 173
printf_barry
barry-configuration.hpp, 279
probability
Node, 185
Progress, 199
~Progress, 199
end, 200
next, 200
Progress, 199
progress.hpp
BARRY_PROGRESS_BAR_WIDTH, 343
pset_loc
Geese, 161
push_back
PhyloCounterData, 189

README.md, 367
reduced_sequence
Geese, 161
rengine
Flock, 145
report
barray-meat.hpp, 244
barraydense-meat.hpp, 269
res
counters-meat.hpp, 298
reserve
BArray< Cell_Type, Data_Type >, 54
BArrayDense< Cell_Type, Data_Type >, 73
FreqTable< T >, 148
PhyloCounterData, 189
reset
PowerSet< Array_Type, Data_Rule_Type >, 196
reset_array
StatsCounter< Array_Type, Data_Type >, 210
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 217
resize
BArray< Cell_Type, Data_Type >, 54
barray-meat.hpp, 239
BArrayDense< Cell_Type, Data_Type >, 74
barraydense-meat.hpp, 263, 264
Entries< Cell_Type >, 138
statscounter-meat.hpp, 347
return
barray-meat.hpp, 239, 244
barraydense-meat.hpp, 269
counters-meat.hpp, 299
statscounter-meat.hpp, 350
support-meat.hpp, 360
rhs
barray-meat-operators.hpp, 227
rm_cell
BArray< Cell_Type, Data_Type >, 54
BArrayDense< Cell_Type, Data_Type >, 74
barraydense-meat.hpp, 264
support-meat.hpp, 354
ROW
barray-meat-operators.hpp, 225
barray-meat.hpp, 231, 239, 240
barraydense-meat-operators.hpp, 249
barraydense-meat.hpp, 254
row
BArray< Cell_Type, Data_Type >, 54

BArrayDense< Cell_Type, Data_Type >, 74
row0
barray-meat.hpp, 245
Row_type
typedefs.hpp, 365
rowsum
BArrayDense< Cell_Type, Data_Type >, 74
Rule
Rule< Array_Type, Data_Type >, 201
Rule< Array_Type, Data_Type >, 200
~Rule, 201
D, 202
get_description, 202
get_name, 202
operator(), 203
Rule, 201
rule_dyn_limit_changes
Phylo rules, 19
rule_fun_default
rules-bones.hpp, 343
Rule_fun_type
typedefs.hpp, 365
RULE_FUNCTION
barry.hpp, 285
geese-bones.hpp, 335
RULE_LAMBDA
barry.hpp, 285
Rules
Rules< Array_Type, Data_Type >, 204
rules
PowerSet< Array_Type, Data_Rule_Type >, 198
support-meat.hpp, 360
Rules< Array_Type, Data_Type >, 203
~Rules, 204
add_rule, 205
get_descriptions, 205
get_names, 205
get_seq, 205
operator(), 206
operator=, 206
Rules, 204
size, 206
rules-bones.hpp
rule_fun_default, 343
rules_
support-meat.hpp, 360
rules_deleted
PowerSet< Array_Type, Data_Rule_Type >, 198
rules_dont_become_zero
DEFMArray counters, 29
rules_dyn
support-meat.hpp, 361
rules_markov_fixed
DEFMArray counters, 30
rules_zerodiag
network.hpp, 319

sample
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 174
search
barray-meat.hpp, 245
sequence
Geese, 161
set_counters
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 174
StatsCounter< Array_Type, Data_Type >, 210
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 217
set_data
BArray< Cell_Type, Data_Type >, 54
BArrayDense< Cell_Type, Data_Type >, 74
set_hasher
Counter< Array_Type, Data_Type >, 114
set_names
DEFM, 124
set_rengine
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 174
set_rules
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 174
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 217
set_rules_dyn
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 175
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 217
set_seed
Flock, 144
Geese, 159
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 175
set_transform_model
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 175
shrink_to_fit
PhyloCounterData, 190
simulate
DEFM, 124
Geese, 159
size
BArrayDenseCol< Cell_Type, Data_Type >, 83
BArrayDenseCol_const< Cell_Type, Data_Type >,

86
BArrayDenseRow< Cell_Type, Data_Type >, 88
BArrayDenseRow_const< Cell_Type, Data_Type >, 90
BArrayVector< Cell_Type, Data_Type >, 99
BArrayVector_const< Cell_Type, Data_Type >, 103
Counters< Array_Type, Data_Type >, 120
FreqTable< T >, 148
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 176
PhyloCounterData, 190
PowerSet< Array_Type, Data_Rule_Type >, 196
Rules< Array_Type, Data_Type >, 206
StatsCounter< Array_Type, Data_Type >, 210
size_unique
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 176
sort_array
typedefs.hpp, 365
source
barray-meat.hpp, 245
barraydense-meat.hpp, 269
Entries< Cell_Type >, 138
states
NodeData, 187
Statistical Models, 13
stats_bank
support-meat.hpp, 361
StatsCounter
StatsCounter< Array_Type, Data_Type >, 208
StatsCounter< Array_Type, Data_Type >, 207
~StatsCounter, 208
add_counter, 209
count_all, 209
count_current, 209
count_init, 209
get_counters, 209
get_descriptions, 209
get_names, 210
reset_array, 210
set_counters, 210
size, 210
StatsCounter, 208
statscounter-meat.hpp
clear, 346
counter, 348
counter_deleted, 349
counters, 349
counters_, 349
current_stats, 349
EmptyArray, 349
f_, 350
for, 346
j, 350
resize, 347
return, 350
STATSCOUNTER_TEMPLATE, 346–348
STATSCOUNTER_TEMPLATE_ARGS, 346
STATSCOUNTER_TYPE, 346
STATSCOUNTER_TEMPLATE
statscounter-meat.hpp, 346–348
STATSCOUNTER_TEMPLATE_ARGS
statscounter-meat.hpp, 346
STATSCOUNTER_TYPE
statscounter-meat.hpp, 346
store_psets
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 176
subtree_prob
Node, 185
Support
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 213
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 211
~Support, 213
add_counter, 214
add_rule, 214
add_rule_dyn, 214
calc, 214
change_stats, 218
coordiantes_n_free, 218
coordiantes_n_locked, 218
coordinates_free, 218
coordinates_locked, 219
current_stats, 219
delete_counters, 219
delete_rules, 219
delete_rules_dyn, 219
eval_rules_dyn, 215
get_counters, 215
get_counts, 215
get_current_stats, 215
get_data, 216
get_rules, 216
get_rules_dyn, 216
hashes, 220
hashes_initialized, 220
init_support, 216
M, 220
max_num_elements, 220
N, 220
n_counters, 221
print, 216
reset_array, 217
set_counters, 217
set_rules, 217
set_rules_dyn, 217
Support, 213
support-meat.hpp
array_bank, 358

BARRY_SUPPORT_MEAT_HPP, 352
calc_backend_dense, 353
calc_backend_sparse, 353
change_stats_different, 358
coord_i, 358
coord_j, 358
counters, 358
counters_, 359
delete_counters, 359
delete_rules, 359
delete_rules_dyn, 359
else, 359
f_, 360
for, 353
hashes, 360
if, 353, 354
insert_cell, 354
return, 360
rm_cell, 354
rules, 360
rules_, 360
rules_dyn, 361
stats_bank, 361
SUPPORT_TEMPLATE, 352, 355–358
SUPPORT_TEMPLATE_ARGS, 352
SUPPORT_TYPE, 353
tmp_chng, 361
support_size
Flock, 144
Geese, 160
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 176
SUPPORT_TEMPLATE
support-meat.hpp, 352, 355–358
SUPPORT_TEMPLATE_ARGS
support-meat.hpp, 352
SUPPORT_TYPE
support-meat.hpp, 353
swap_cells
BArray< Cell_Type, Data_Type >, 55
BArrayDense< Cell_Type, Data_Type >, 75
swap_cols
BArray< Cell_Type, Data_Type >, 55
BArrayDense< Cell_Type, Data_Type >, 75
swap_rows
BArray< Cell_Type, Data_Type >, 55
BArrayDense< Cell_Type, Data_Type >, 75

target
barray-meat.hpp, 245
barraydense-meat.hpp, 270
Entries< Cell_Type >, 138
this
barray-meat-operators.hpp, 228
tmp_chng
support-meat.hpp, 361
TMP_HASHER_CALL
counters-meat.hpp, 289
toggle_cell
BArray< Cell_Type, Data_Type >, 55
BArrayDense< Cell_Type, Data_Type >, 75
toggle_lock
BArray< Cell_Type, Data_Type >, 55
BArrayDense< Cell_Type, Data_Type >, 76
transform_model
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 176
transpose
BArray< Cell_Type, Data_Type >, 56
BArrayDense< Cell_Type, Data_Type >, 76
TWO
CHECK, 41
EXISTS, 42
typedefs.hpp
Col_type, 364
Counter_fun_type, 364
Counts_type, 364
Hasher_fun_type, 364
MapVec_type, 365
Row_type, 365
Rule_fun_type, 365
sort_array, 365
uint, 365
vec_equal, 366
vec_equal_approx, 366
vec_inner_prod, 366, 367

ub
PhyloRuleDynData, 192
uint
typedefs.hpp, 365
UKNOWN
EXISTS, 42
UNI_SUB
defm.hpp, 303
update_annotations
Geese, 160
update_normalizing_constant
model-meat.hpp, 330

v
barray-meat.hpp, 245
barraydense-meat.hpp, 270
va_end
barraydense-meat.hpp, 264
va_start
barraydense-meat.hpp, 264
val
Entries< Cell_Type >, 138
val0
barraydense-meat.hpp, 270
val1
barraydense-meat.hpp, 270
value
barray-meat.hpp, 245
barraydense-meat.hpp, 270

Cell< Cell_Type >, 107
vec_diff
geese-bones.hpp, 335
vec_equal
typedefs.hpp, 366
vec_equal_approx
typedefs.hpp, 366
vec_inner_prod
typedefs.hpp, 366, 367
vecHasher< T >, 221
operator(), 221
vector_caster
geese-bones.hpp, 336
vertex_attr
NetworkData, 180
visited
BArray< Cell_Type, Data_Type >, 57
BArrayDense< Cell_Type, Data_Type >, 77
Cell< Cell_Type >, 107
Node, 185
vprintf
barraydense-meat.hpp, 265

X_ncol
DEFMData, 131
X_nrow
DEFMData, 131

ZERO_CELL
barraydense-meat.hpp, 254
barraydensecol-bones.hpp, 273
barraydenserow-bones.hpp, 275
zero_col
BArray< Cell_Type, Data_Type >, 56
BArrayDense< Cell_Type, Data_Type >, 76
zero_row
BArray< Cell_Type, Data_Type >, 56
BArrayDense< Cell_Type, Data_Type >, 76