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 Class Index 7

3.1 Class List . . . 7

4 File Index 9

4.1 File List . . . 9

5 Module Documentation 11

5.1 Counting . . . 11

5.1.1 Detailed Description . . . 11

5.2 Statistical Models . . . 11

5.2.1 Detailed Description . . . 12

5.3 Network counters . . . 12

5.3.1 Detailed Description . . . 12

5.3.2 Function Documentation . . . 13

5.3.2.1 counter_absdiff() . . . 13

5.3.2.2 counter_ctriads() . . . 13

5.3.2.3 counter_degree() . . . 13

5.3.2.4 counter_density() . . . 13

5.3.2.5 counter_diff() . . . 14

5.3.2.6 counter_edges() . . . 14

5.3.2.7 counter_idegree() . . . 14

5.3.2.8 counter_idegree15() . . . 14

5.3.2.9 counter_isolates() . . . 14

5.3.2.10 counter_istar2() . . . 15

5.3.2.11 counter_mutual() . . . 15

5.3.2.12 counter_nodecov() . . . 15

5.3.2.13 counter_nodeicov() . . . 15

5.3.2.14 counter_nodematch() . . . 15

5.3.2.15 counter_nodeocov() . . . 16

5.3.2.16 counter_odegree() . . . 16

5.3.2.17 counter_odegree15() . . . 16

5.3.2.18 counter_ostar2() . . . 16

5.3.2.19 counter_ttriads() . . . 16

5.3.2.20 NETWORK_COUNTER() . . . 17

5.4 Phylo counters . . . 17

5.4.1 Detailed Description . . . 17

5.4.2 Function Documentation . . . 18

5.4.2.1 counter_co_opt() . . . 18

5.4.2.2 counter_cogain() . . . 18

5.4.2.3 counter_gains() . . . 19
5.4.2.4 counter_gains_k_offspring() . . . 19
5.4.2.5 counter_genes_changing() . . . 19
5.4.2.6 counter_k_genes_changing() . . . 19
5.4.2.7 counter_longest() . . . 20
5.4.2.8 counter_loss() . . . 20
5.4.2.9 counter_maxfuns() . . . 20
5.4.2.10 counter_neofun() . . . 20
5.4.2.11 counter_neofun_a2b() . . . 21
5.4.2.12 counter_overall_changes() . . . 21
5.4.2.13 counter_overall_gains() . . . 21
5.4.2.14 counter_overall_loss() . . . 21
5.4.2.15 counter_prop_genes_changing() . . . 22
5.4.2.16 counter_subfun() . . . 22
5.5 Phylo rules . . . 22
5.5.1 Detailed Description . . . 22
5.5.2 Function Documentation . . . 23
5.5.2.1 rule_dyn_limit_changes() . . . 23

6 Namespace Documentation **25**
6.1 barry Namespace Reference . . . 25
6.1.1 Detailed Description . . . 25
6.2 barry::counters Namespace Reference . . . 25
6.2.1 Detailed Description . . . 25
6.3 barry::counters::network Namespace Reference . . . 26
6.4 barry::counters::phylo Namespace Reference . . . 26
6.5 CHECK Namespace Reference . . . 26
6.5.1 Detailed Description . . . 26
6.5.2 Variable Documentation . . . 26
6.5.2.1 BOTH . . . 26
6.5.2.2 NONE . . . 26
6.5.2.3 ONE . . . 26
6.5.2.4 TWO . . . 27
6.6 EXISTS Namespace Reference . . . 27
6.6.1 Detailed Description . . . 27
6.6.2 Variable Documentation . . . 27
6.6.2.1 AS_ONE . . . 27
6.6.2.2 AS_ZERO . . . 27
6.6.2.3 BOTH . . . 28
6.6.2.4 NONE . . . 28
6.6.2.5 ONE . . . 28
6.6.2.6 TWO . . . 28

6.6.2.7 UKNOWN 28

7 Class Documentation **29**

7.1 BArray< Cell_Type, Data_Type > Class Template Reference 29

7.1.1 Detailed Description 31

7.1.2 Constructor & Destructor Documentation 32

7.1.2.1 BArray() [1/6] 32

7.1.2.2 BArray() [2/6] 32

7.1.2.3 BArray() [3/6] 32

7.1.2.4 BArray() [4/6] 33

7.1.2.5 BArray() [5/6] 33

7.1.2.6 BArray() [6/6] 33

7.1.2.7 ~BArray() 33

7.1.3 Member Function Documentation 33

7.1.3.1 clear() 33

7.1.3.2 col() 34

7.1.3.3 D() [1/2] 34

7.1.3.4 D() [2/2] 34

7.1.3.5 default_val() 34

7.1.3.6 flush_data() 34

7.1.3.7 get_cell() 34

7.1.3.8 get_col_vec() [1/2] 35

7.1.3.9 get_col_vec() [2/2] 35

7.1.3.10 get_entries() 35

7.1.3.11 get_row_vec() [1/2] 35

7.1.3.12 get_row_vec() [2/2] 35

7.1.3.13 insert_cell() [1/3] 36

7.1.3.14 insert_cell() [2/3] 36

7.1.3.15 insert_cell() [3/3] 36

7.1.3.16 is_empty() 36

7.1.3.17 ncol() 36

7.1.3.18 nnozero() 37

7.1.3.19 nrow() 37

7.1.3.20 operator()() [1/2] 37

7.1.3.21 operator()() [2/2] 37

7.1.3.22 operator*=() 37

7.1.3.23 operator+=() [1/3] 37

7.1.3.24 operator+=() [2/3] 38

7.1.3.25 operator+=() [3/3] 38

7.1.3.26 operator-=() [1/3] 38

7.1.3.27 operator-=() [2/3] 38

7.1.3.28 operator-=() [3/3] 38

7.1.3.29 operator/=() . . . 38
7.1.3.30 operator=() [1/2] . . . 39
7.1.3.31 operator=() [2/2] . . . 39
7.1.3.32 operator==() . . . 39
7.1.3.33 out_of_range() . . . 39
7.1.3.34 print() . . . 39
7.1.3.35 reserve() . . . 39
7.1.3.36 resize() . . . 40
7.1.3.37 rm_cell() . . . 40
7.1.3.38 row() . . . 40
7.1.3.39 set_data() . . . 40
7.1.3.40 swap_cells() . . . 40
7.1.3.41 swap_cols() . . . 41
7.1.3.42 swap_rows() . . . 41
7.1.3.43 toggle_cell() . . . 41
7.1.3.44 toggle_lock() . . . 41
7.1.3.45 transpose() . . . 42
7.1.3.46 zero_col() . . . 42
7.1.3.47 zero_row() . . . 42
7.1.4 Friends And Related Function Documentation . . . 42
7.1.4.1 BArrayCell< Cell_Type, Data_Type > . . . 42
7.1.4.2 BArrayCell_const< Cell_Type, Data_Type > . . . 42
7.1.5 Member Data Documentation . . . 42
7.1.5.1 visited . . . 43
7.2 BArrayCell< Cell_Type, Data_Type > Class Template Reference . . . 43
7.2.1 Detailed Description . . . 43
7.2.2 Constructor & Destructor Documentation . . . 43
7.2.2.1 BArrayCell() . . . 44
7.2.2.2 ~BArrayCell() . . . 44
7.2.3 Member Function Documentation . . . 44
7.2.3.1 operator Cell_Type() . . . 44
7.2.3.2 operator*=() . . . 44
7.2.3.3 operator+=() . . . 44
7.2.3.4 operator-=() . . . 45
7.2.3.5 operator/=() . . . 45
7.2.3.6 operator=() . . . 45
7.2.3.7 operator==() . . . 45
7.3 BArrayCell_const< Cell_Type, Data_Type > Class Template Reference . . . 45
7.3.1 Detailed Description . . . 46
7.3.2 Constructor & Destructor Documentation . . . 46
7.3.2.1 BArrayCell_const() . . . 46
7.3.2.2 ~BArrayCell_const() . . . 46

7.3.3 Member Function Documentation . . . 46
7.3.3.1 operator Cell_Type() . . . 47
7.3.3.2 operator"!=() . . . 47
7.3.3.3 operator<() . . . 47
7.3.3.4 operator<=() . . . 47
7.3.3.5 operator==() . . . 47
7.3.3.6 operator>() . . . 48
7.3.3.7 operator>=() . . . 48
7.4 BArrayDense< Cell_Type, Data_Type > Class Template Reference . . . 48
7.4.1 Detailed Description . . . 50
7.4.2 Constructor & Destructor Documentation . . . 51
7.4.2.1 BArrayDense() [1/6] . . . 51
7.4.2.2 BArrayDense() [2/6] . . . 51
7.4.2.3 BArrayDense() [3/6] . . . 51
7.4.2.4 BArrayDense() [4/6] . . . 52
7.4.2.5 BArrayDense() [5/6] . . . 52
7.4.2.6 BArrayDense() [6/6] . . . 52
7.4.2.7 ~BArrayDense() . . . 52
7.4.3 Member Function Documentation . . . 52
7.4.3.1 clear() . . . 52
7.4.3.2 col() . . . 53
7.4.3.3 D() [1/2] . . . 53
7.4.3.4 D() [2/2] . . . 53
7.4.3.5 default_val() . . . 53
7.4.3.6 get_cell() . . . 53
7.4.3.7 get_col_vec() [1/2] . . . 53
7.4.3.8 get_col_vec() [2/2] . . . 54
7.4.3.9 get_entries() . . . 54
7.4.3.10 get_row_vec() [1/2] . . . 54
7.4.3.11 get_row_vec() [2/2] . . . 54
7.4.3.12 insert_cell() [1/3] . . . 54
7.4.3.13 insert_cell() [2/3] . . . 55
7.4.3.14 insert_cell() [3/3] . . . 55
7.4.3.15 is_empty() . . . 55
7.4.3.16 ncol() . . . 55
7.4.3.17 nnozero() . . . 55
7.4.3.18 nrow() . . . 56
7.4.3.19 operator()() [1/2] . . . 56
7.4.3.20 operator()() [2/2] . . . 56
7.4.3.21 operator*=() . . . 56
7.4.3.22 operator+=() [1/3] . . . 56
7.4.3.23 operator+=() [2/3] . . . 56

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

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

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

7.11.2 Constructor & Destructor Documentation . . . 82
7.11.2.1 Counter() [1/4] . . . 83
7.11.2.2 Counter() [2/4] . . . 83
7.11.2.3 Counter() [3/4] . . . 83
7.11.2.4 Counter() [4/4] . . . 83
7.11.2.5 ∼Counter() . . . 83
7.11.3 Member Function Documentation . . . 84
7.11.3.1 count() . . . 84
7.11.3.2 get_description() . . . 84
7.11.3.3 get_name() . . . 84
7.11.3.4 init() . . . 84
7.11.3.5 operator=() [1/2] . . . 84
7.11.3.6 operator=() [2/2] . . . 85
7.11.4 Member Data Documentation . . . 85
7.11.4.1 count_fun . . . 85
7.11.4.2 data . . . 85
7.11.4.3 delete_data . . . 85
7.11.4.4 desc . . . 85
7.11.4.5 init_fun . . . 86
7.11.4.6 name . . . 86
7.12 Counters< Array_Type, Data_Type > Class Template Reference . . . 86
7.12.1 Detailed Description . . . 87
7.12.2 Constructor & Destructor Documentation . . . 87
7.12.2.1 Counters() [1/3] . . . 87
7.12.2.2 ∼Counters() . . . 87
7.12.2.3 Counters() [2/3] . . . 87
7.12.2.4 Counters() [3/3] . . . 88
7.12.3 Member Function Documentation . . . 88
7.12.3.1 add_counter() [1/3] . . . 88
7.12.3.2 add_counter() [2/3] . . . 88
7.12.3.3 add_counter() [3/3] . . . 88
7.12.3.4 clear() . . . 89
7.12.3.5 get_descriptions() . . . 89
7.12.3.6 get_names() . . . 89
7.12.3.7 operator=() [1/2] . . . 89
7.12.3.8 operator=() [2/2] . . . 89
7.12.3.9 operator[]() . . . 90
7.12.3.10 size() . . . 90
7.13 Entries< Cell_Type > Class Template Reference . . . 91
7.13.1 Detailed Description . . . 91
7.13.2 Constructor & Destructor Documentation . . . 91
7.13.2.1 Entries() [1/2] . . . 91

7.13.2.2 Entries() [2/2] . . . 92
7.13.2.3 ~Entries() . . . 92
7.13.3 Member Function Documentation . . . 92
7.13.3.1 resize() . . . 92
7.13.4 Member Data Documentation . . . 92
7.13.4.1 source . . . 92
7.13.4.2 target . . . 92
7.13.4.3 val . . . 93
7.14 Flock Class Reference . . . 93
7.14.1 Detailed Description . . . 94
7.14.2 Constructor & Destructor Documentation . . . 94
7.14.2.1 Flock() . . . 94
7.14.2.2 ~Flock() . . . 94
7.14.3 Member Function Documentation . . . 94
7.14.3.1 add_data() . . . 94
7.14.3.2 colnames() . . . 95
7.14.3.3 get_counters() . . . 95
7.14.3.4 get_model() . . . 95
7.14.3.5 get_support() . . . 95
7.14.3.6 init() . . . 96
7.14.3.7 likelihood_joint() . . . 96
7.14.3.8 nfuns() . . . 96
7.14.3.9 nleafs() . . . 96
7.14.3.10 nnodes() . . . 97
7.14.3.11 nterms() . . . 97
7.14.3.12 ntrees() . . . 97
7.14.3.13 operator()() . . . 97
7.14.3.14 parse_polytomies() . . . 97
7.14.3.15 print() . . . 98
7.14.3.16 set_seed() . . . 98
7.14.3.17 support_size() . . . 98
7.14.4 Member Data Documentation . . . 98
7.14.4.1 dat . . . 98
7.14.4.2 initialized . . . 98
7.14.4.3 model . . . 99
7.14.4.4 nfunctions . . . 99
7.14.4.5 rengine . . . 99
7.15 FreqTable< T > Class Template Reference . . . 99
7.15.1 Detailed Description . . . 100
7.15.2 Constructor & Destructor Documentation . . . 100
7.15.2.1 FreqTable() . . . 100
7.15.2.2 ~FreqTable() . . . 100

7.15.3 Member Function Documentation . . . 100
7.15.3.1 add() . . . 100
7.15.3.2 as_vector() . . . 101
7.15.3.3 clear() . . . 101
7.15.3.4 get_data() . . . 101
7.15.3.5 get_data_ptr() . . . 101
7.15.3.6 print() . . . 101
7.15.3.7 reserve() . . . 102
7.15.3.8 size() . . . 102
7.16 Geese Class Reference . . . 102
7.16.1 Detailed Description . . . 104
7.16.2 Constructor & Destructor Documentation . . . 104
7.16.2.1 Geese() [1/4] . . . 105
7.16.2.2 Geese() [2/4] . . . 105
7.16.2.3 Geese() [3/4] . . . 105
7.16.2.4 Geese() [4/4] . . . 105
7.16.2.5 ∼Geese() . . . 105
7.16.3 Member Function Documentation . . . 105
7.16.3.1 calc_reduced_sequence() . . . 106
7.16.3.2 calc_sequence() . . . 106
7.16.3.3 colnames() . . . 106
7.16.3.4 get_annotated_nodes() . . . 106
7.16.3.5 get_counters() . . . 106
7.16.3.6 get_model() . . . 106
7.16.3.7 get_probabilities() . . . 107
7.16.3.8 get_rengine() . . . 107
7.16.3.9 get_states() . . . 107
7.16.3.10 get_support() . . . 107
7.16.3.11 inherit_support() . . . 107
7.16.3.12 init() . . . 108
7.16.3.13 init_node() . . . 108
7.16.3.14 likelihood() . . . 108
7.16.3.15 likelihood_exhaust() . . . 108
7.16.3.16 nannotations() . . . 108
7.16.3.17 nfuns() . . . 109
7.16.3.18 nleafs() . . . 109
7.16.3.19 nnodes() . . . 109
7.16.3.20 nterms() . . . 109
7.16.3.21 observed_counts() . . . 109
7.16.3.22 operator=() [1/2] . . . 110
7.16.3.23 operator=() [2/2] . . . 110
7.16.3.24 parse_polytomies() . . . 110

7.16.3.25 predict() . . . 110
7.16.3.26 predict_backend() . . . 110
7.16.3.27 predict_exhaust() . . . 111
7.16.3.28 predict_exhaust_backend() . . . 111
7.16.3.29 predict_sim() . . . 111
7.16.3.30 print() . . . 111
7.16.3.31 print_observed_counts() . . . 111
7.16.3.32 set_seed() . . . 112
7.16.3.33 simulate() . . . 112
7.16.3.34 support_size() . . . 112
7.16.3.35 update_annotations() . . . 112
7.16.4 Member Data Documentation . . . 112
7.16.4.1 delete_rengine . . . 112
7.16.4.2 delete_support . . . 113
7.16.4.3 initialized . . . 113
7.16.4.4 map_to_nodes . . . 113
7.16.4.5 nfunctions . . . 113
7.16.4.6 nodes . . . 113
7.16.4.7 reduced_sequence . . . 113
7.16.4.8 sequence . . . 114
7.17 Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference . . . 114
7.17.1 Detailed Description . . . 116
7.17.2 Constructor & Destructor Documentation . . . 116
7.17.2.1 Model() [1/3] . . . 117
7.17.2.2 Model() [2/3] . . . 117
7.17.2.3 Model() [3/3] . . . 117
7.17.2.4 ~Model() . . . 117
7.17.3 Member Function Documentation . . . 117
7.17.3.1 add_array() . . . 117
7.17.3.2 add_counter() [1/3] . . . 118
7.17.3.3 add_counter() [2/3] . . . 118
7.17.3.4 add_counter() [3/3] . . . 118
7.17.3.5 add_rule() [1/3] . . . 118
7.17.3.6 add_rule() [2/3] . . . 119
7.17.3.7 add_rule() [3/3] . . . 119
7.17.3.8 add_rule_dyn() [1/3] . . . 119
7.17.3.9 add_rule_dyn() [2/3] . . . 119
7.17.3.10 add_rule_dyn() [3/3] . . . 119
7.17.3.11 colnames() . . . 120
7.17.3.12 conditional_prob() . . . 120
7.17.3.13 gen_key() . . . 120

7.17.3.14 get_counters() . . . 121
7.17.3.15 get_norm_const() . . . 121
7.17.3.16 get_pset() . . . 121
7.17.3.17 get_pset_stats() . . . 121
7.17.3.18 get_rengine() . . . 121
7.17.3.19 get_rules() . . . 122
7.17.3.20 get_rules_dyn() . . . 122
7.17.3.21 get_support() . . . 122
7.17.3.22 likelihood() [1/3] . . . 122
7.17.3.23 likelihood() [2/3] . . . 122
7.17.3.24 likelihood() [3/3] . . . 123
7.17.3.25 likelihood_total() . . . 123
7.17.3.26 nterms() . . . 123
7.17.3.27 operator=() . . . 123
7.17.3.28 print() . . . 123
7.17.3.29 print_stats() . . . 124
7.17.3.30 sample() [1/2] . . . 124
7.17.3.31 sample() [2/2] . . . 124
7.17.3.32 set_counters() . . . 124
7.17.3.33 set_keygen() . . . 124
7.17.3.34 set_rengine() . . . 125
7.17.3.35 set_rules() . . . 125
7.17.3.36 set_rules_dyn() . . . 125
7.17.3.37 set_seed() . . . 125
7.17.3.38 size() . . . 125
7.17.3.39 size_unique() . . . 126
7.17.3.40 store_psets() . . . 126
7.17.3.41 support_size() . . . 126
7.18 NetCounterData Class Reference . . . 126
7.18.1 Detailed Description . . . 127
7.18.2 Constructor & Destructor Documentation . . . 127
7.18.2.1 NetCounterData() [1/2] . . . 127
7.18.2.2 NetCounterData() [2/2] . . . 127
7.18.2.3 ~NetCounterData() . . . 127
7.18.3 Member Data Documentation . . . 127
7.18.3.1 indices . . . 127
7.18.3.2 numbers . . . 128
7.19 NetworkData Class Reference . . . 128
7.19.1 Detailed Description . . . 128
7.19.2 Constructor & Destructor Documentation . . . 128
7.19.2.1 NetworkData() [1/3] . . . 129
7.19.2.2 NetworkData() [2/3] . . . 129

7.19.2.3 NetworkData() [3/3] . . . 130
7.19.2.4 ~NetworkData() . . . 130
7.19.3 Member Data Documentation . . . 130
7.19.3.1 directed . . . 130
7.19.3.2 vertex_attr . . . 131
7.20 Node Class Reference . . . 131
7.20.1 Detailed Description . . . 132
7.20.2 Constructor & Destructor Documentation . . . 132
7.20.2.1 Node() [1/5] . . . 132
7.20.2.2 Node() [2/5] . . . 133
7.20.2.3 Node() [3/5] . . . 133
7.20.2.4 Node() [4/5] . . . 133
7.20.2.5 Node() [5/5] . . . 133
7.20.2.6 ~Node() . . . 133
7.20.3 Member Function Documentation . . . 133
7.20.3.1 get_parent() . . . 134
7.20.3.2 is_leaf() . . . 134
7.20.3.3 noffspring() . . . 134
7.20.4 Member Data Documentation . . . 134
7.20.4.1 annotations . . . 134
7.20.4.2 array . . . 134
7.20.4.3 arrays . . . 135
7.20.4.4 duplication . . . 135
7.20.4.5 id . . . 135
7.20.4.6 narray . . . 135
7.20.4.7 offspring . . . 135
7.20.4.8 ord . . . 136
7.20.4.9 parent . . . 136
7.20.4.10 probability . . . 136
7.20.4.11 subtree_prob . . . 136
7.20.4.12 visited . . . 136
7.21 NodeData Class Reference . . . 137
7.21.1 Detailed Description . . . 137
7.21.2 Constructor & Destructor Documentation . . . 137
7.21.2.1 NodeData() . . . 137
7.21.3 Member Data Documentation . . . 137
7.21.3.1 blengths . . . 138
7.21.3.2 duplication . . . 138
7.21.3.3 states . . . 138
7.22 PhyloCounterData Class Reference . . . 138
7.22.1 Detailed Description . . . 139
7.22.2 Constructor & Destructor Documentation . . . 139

7.22.2.1 PhyloCounterData() . . . 139
7.22.3 Member Function Documentation . . . 139
7.22.3.1 at() . . . 139
7.22.3.2 begin() . . . 139
7.22.3.3 empty() . . . 139
7.22.3.4 end() . . . 140
7.22.3.5 get_counters() . . . 140
7.22.3.6 operator()() . . . 140
7.22.3.7 push_back() . . . 140
7.22.3.8 reserve() . . . 140
7.22.3.9 shrink_to_fit() . . . 140
7.22.3.10 size() . . . 141
7.23 PhyloRuleDynData Class Reference . . . 141
7.23.1 Detailed Description . . . 141
7.23.2 Constructor & Destructor Documentation . . . 141
7.23.2.1 PhyloRuleDynData() . . . 141
7.23.2.2 ∼PhyloRuleDynData() . . . 142
7.23.3 Member Data Documentation . . . 142
7.23.3.1 counts . . . 142
7.23.3.2 duplication . . . 142
7.23.3.3 lb . . . 142
7.23.3.4 pos . . . 142
7.23.3.5 ub . . . 142
7.24 PowerSet< Array_Type, Data_Rule_Type > Class Template Reference . . . 143
7.24.1 Detailed Description . . . 144
7.24.2 Constructor & Destructor Documentation . . . 144
7.24.2.1 PowerSet() [1/3] . . . 144
7.24.2.2 PowerSet() [2/3] . . . 144
7.24.2.3 PowerSet() [3/3] . . . 145
7.24.2.4 ∼PowerSet() . . . 145
7.24.3 Member Function Documentation . . . 145
7.24.3.1 add_rule() [1/3] . . . 145
7.24.3.2 add_rule() [2/3] . . . 145
7.24.3.3 add_rule() [3/3] . . . 145
7.24.3.4 begin() . . . 146
7.24.3.5 calc() . . . 146
7.24.3.6 end() . . . 146
7.24.3.7 get_data() . . . 146
7.24.3.8 get_data_ptr() . . . 146
7.24.3.9 init_support() . . . 147
7.24.3.10 operator[]() . . . 147
7.24.3.11 reset() . . . 147

7.24.3.12 size() . . . 147
7.24.4 Member Data Documentation . . . 147
7.24.4.1 coordinates_free . . . 147
7.24.4.2 coordinates_locked . . . 148
7.24.4.3 data . . . 148
7.24.4.4 EmptyArray . . . 148
7.24.4.5 M . . . 148
7.24.4.6 N . . . 148
7.24.4.7 rules . . . 149
7.24.4.8 rules_deleted . . . 149
7.25 Progress Class Reference . . . 149
7.25.1 Detailed Description . . . 149
7.25.2 Constructor & Destructor Documentation . . . 149
7.25.2.1 Progress() . . . 150
7.25.2.2 ∼Progress() . . . 150
7.25.3 Member Function Documentation . . . 150
7.25.3.1 end() . . . 150
7.25.3.2 next() . . . 150
7.26 Rule< Array_Type, Data_Type > Class Template Reference . . . 150
7.26.1 Detailed Description . . . 151
7.26.2 Constructor & Destructor Documentation . . . 151
7.26.2.1 Rule() [1/2] . . . 151
7.26.2.2 Rule() [2/2] . . . 151
7.26.2.3 ∼Rule() . . . 152
7.26.3 Member Function Documentation . . . 152
7.26.3.1 D() . . . 152
7.26.3.2 operator()() . . . 152
7.27 Rules< Array_Type, Data_Type > Class Template Reference . . . 152
7.27.1 Detailed Description . . . 153
7.27.2 Constructor & Destructor Documentation . . . 153
7.27.2.1 Rules() [1/2] . . . 153
7.27.2.2 Rules() [2/2] . . . 153
7.27.2.3 ∼Rules() . . . 154
7.27.3 Member Function Documentation . . . 154
7.27.3.1 add_rule() [1/3] . . . 154
7.27.3.2 add_rule() [2/3] . . . 154
7.27.3.3 add_rule() [3/3] . . . 154
7.27.3.4 clear() . . . 154
7.27.3.5 get_seq() . . . 154
7.27.3.6 operator()() . . . 155
7.27.3.7 operator=() . . . 155
7.27.3.8 size() . . . 156

7.28 StatsCounter< Array_Type, Data_Type > Class Template Reference . . . 156
7.28.1 Detailed Description . . . 156
7.28.2 Constructor & Destructor Documentation . . . 157
7.28.2.1 StatsCounter() [1/2] . . . 157
7.28.2.2 StatsCounter() [2/2] . . . 157
7.28.2.3 ~StatsCounter() . . . 157
7.28.3 Member Function Documentation . . . 157
7.28.3.1 add_counter() [1/2] . . . 157
7.28.3.2 add_counter() [2/2] . . . 158
7.28.3.3 count_all() . . . 158
7.28.3.4 count_current() . . . 158
7.28.3.5 count_init() . . . 158
7.28.3.6 get_counters() . . . 158
7.28.3.7 get_descriptions() . . . 158
7.28.3.8 get_names() . . . 159
7.28.3.9 reset_array() . . . 159
7.28.3.10 set_counters() . . . 159
7.29 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference . . . 159
7.29.1 Detailed Description . . . 161
7.29.2 Constructor & Destructor Documentation . . . 161
7.29.2.1 Support() [1/3] . . . 161
7.29.2.2 Support() [2/3] . . . 162
7.29.2.3 Support() [3/3] . . . 162
7.29.2.4 ~Support() . . . 162
7.29.3 Member Function Documentation . . . 162
7.29.3.1 add_counter() [1/2] . . . 162
7.29.3.2 add_counter() [2/2] . . . 163
7.29.3.3 add_rule() [1/2] . . . 163
7.29.3.4 add_rule() [2/2] . . . 163
7.29.3.5 add_rule_dyn() [1/2] . . . 163
7.29.3.6 add_rule_dyn() [2/2] . . . 163
7.29.3.7 calc() . . . 164
7.29.3.8 eval_rules_dyn() . . . 165
7.29.3.9 get_counters() . . . 165
7.29.3.10 get_counts() . . . 165
7.29.3.11 get_counts_ptr() . . . 165
7.29.3.12 get_current_stats() . . . 166
7.29.3.13 get_data() . . . 166
7.29.3.14 get_rules() . . . 166
7.29.3.15 get_rules_dyn() . . . 166
7.29.3.16 init_support() . . . 166

7.29.3.17 print() . . . 167
7.29.3.18 reset_array() [1/2] . . . 167
7.29.3.19 reset_array() [2/2] . . . 167
7.29.3.20 set_counters() . . . 167
7.29.3.21 set_rules() . . . 167
7.29.3.22 set_rules_dyn() . . . 168
7.29.4 Member Data Documentation . . . 168
7.29.4.1 change_stats . . . 168
7.29.4.2 coordinates_free . . . 168
7.29.4.3 coordinates_locked . . . 168
7.29.4.4 current_stats . . . 169
7.29.4.5 delete_counters . . . 169
7.29.4.6 delete_rules . . . 169
7.29.4.7 delete_rules_dyn . . . 169
7.29.4.8 M . . . 169
7.29.4.9 max_num_elements . . . 170
7.29.4.10 N . . . 170
7.30 vecHasher< T > Struct Template Reference . . . 170
7.30.1 Detailed Description . . . 170
7.30.2 Member Function Documentation . . . 170
7.30.2.1 operator()() . . . 170

8 File Documentation **171**
8.1 include/barry/barray-bones.hpp File Reference . . . 171
8.1.1 Macro Definition Documentation . . . 172
8.1.1.1 BARRAY_BONES_HPP . . . 172
8.2 include/barry/barray-iterator.hpp File Reference . . . 172
8.3 include/barry/barray-meat-operators.hpp File Reference . . . 173
8.3.1 Macro Definition Documentation . . . 174
8.3.1.1 BARRY_BARRAY_MEAT_OPERATORS_HPP . . . 174
8.3.1.2 COL . . . 174
8.3.1.3 ROW . . . 174
8.3.2 Function Documentation . . . 174
8.3.2.1 checkdim_() . . . 174
8.4 include/barry/barray-meat.hpp File Reference . . . 175
8.4.1 Macro Definition Documentation . . . 177
8.4.1.1 BARRAY_TEMPLATE . . . 177
8.4.1.2 BARRAY_TEMPLATE_ARGS . . . 177
8.4.1.3 BARRAY_TYPE . . . 178
8.4.1.4 COL . . . 178
8.4.1.5 ROW . . . 178
8.4.2 Function Documentation . . . 178

8.4.2.1 ans() . . . 178
8.4.2.2 BARRAY_TEMPLATE() [1/23] . . . 178
8.4.2.3 BARRAY_TEMPLATE() [2/23] . . . 179
8.4.2.4 BARRAY_TEMPLATE() [3/23] . . . 179
8.4.2.5 BARRAY_TEMPLATE() [4/23] . . . 179
8.4.2.6 BARRAY_TEMPLATE() [5/23] . . . 179
8.4.2.7 BARRAY_TEMPLATE() [6/23] . . . 179
8.4.2.8 BARRAY_TEMPLATE() [7/23] . . . 179
8.4.2.9 BARRAY_TEMPLATE() [8/23] . . . 180
8.4.2.10 BARRAY_TEMPLATE() [9/23] . . . 180
8.4.2.11 BARRAY_TEMPLATE() [10/23] . . . 180
8.4.2.12 BARRAY_TEMPLATE() [11/23] . . . 180
8.4.2.13 BARRAY_TEMPLATE() [12/23] . . . 180
8.4.2.14 BARRAY_TEMPLATE() [13/23] . . . 181
8.4.2.15 BARRAY_TEMPLATE() [14/23] . . . 181
8.4.2.16 BARRAY_TEMPLATE() [15/23] . . . 181
8.4.2.17 BARRAY_TEMPLATE() [16/23] . . . 181
8.4.2.18 BARRAY_TEMPLATE() [17/23] . . . 181
8.4.2.19 BARRAY_TEMPLATE() [18/23] . . . 181
8.4.2.20 BARRAY_TEMPLATE() [19/23] . . . 182
8.4.2.21 BARRAY_TEMPLATE() [20/23] . . . 182
8.4.2.22 BARRAY_TEMPLATE() [21/23] . . . 182
8.4.2.23 BARRAY_TEMPLATE() [22/23] . . . 182
8.4.2.24 BARRAY_TEMPLATE() [23/23] . . . 182
8.4.2.25 COL() . . . 182
8.4.2.26 for() [1/3] . . . 183
8.4.2.27 for() [2/3] . . . 183
8.4.2.28 for() [3/3] . . . 183
8.4.2.29 if() [1/17] . . . 183
8.4.2.30 if() [2/17] . . . 183
8.4.2.31 if() [3/17] . . . 183
8.4.2.32 if() [4/17] . . . 184
8.4.2.33 if() [5/17] . . . 184
8.4.2.34 if() [6/17] . . . 184
8.4.2.35 if() [7/17] . . . 184
8.4.2.36 if() [8/17] . . . 184
8.4.2.37 if() [9/17] . . . 184
8.4.2.38 if() [10/17] . . . 185
8.4.2.39 if() [11/17] . . . 185
8.4.2.40 if() [12/17] . . . 185
8.4.2.41 if() [13/17] . . . 185
8.4.2.42 if() [14/17] . . . 185

8.4.2.43 if() [15/17] . . . 185
8.4.2.44 if() [16/17] . . . 185
8.4.2.45 if() [17/17] . . . 186
8.4.2.46 M() . . . 186
8.4.2.47 resize() [1/2] . . . 186
8.4.2.48 resize() [2/2] . . . 186
8.4.2.49 return() . . . 186
8.4.2.50 ROW() [1/2] . . . 186
8.4.2.51 ROW() [2/2] . . . 186
8.4.3 Variable Documentation . . . 187
8.4.3.1 add . . . 187
8.4.3.2 ans . . . 187
8.4.3.3 Array_ . . . 187
8.4.3.4 check_bounds . . . 187
8.4.3.5 check_exists . . . 188
8.4.3.6 col0 . . . 188
8.4.3.7 const . . . 188
8.4.3.8 copy_data . . . 188
8.4.3.9 data . . . 188
8.4.3.10 delete_data . . . 189
8.4.3.11 delete_data_ . . . 189
8.4.3.12 else . . . 189
8.4.3.13 false . . . 189
8.4.3.14 first . . . 189
8.4.3.15 i1 . . . 190
8.4.3.16 j . . . 190
8.4.3.17 j0 . . . 190
8.4.3.18 j1 . . . 190
8.4.3.19 M . . . 190
8.4.3.20 M_ . . . 191
8.4.3.21 N . . . 191
8.4.3.22 NCells . . . 191
8.4.3.23 report . . . 191
8.4.3.24 return . . . 191
8.4.3.25 row0 . . . 192
8.4.3.26 search . . . 192
8.4.3.27 source . . . 192
8.4.3.28 target . . . 192
8.4.3.29 v . . . 192
8.4.3.30 value . . . 192
8.5 include/barry/barraycell-bones.hpp File Reference . . . 193
8.6 include/barry/barraycell-meat.hpp File Reference . . . 193

8.7 include/barry/barraydense-bones.hpp File Reference . . . 194
8.8 include/barry/barraydense-meet.hpp File Reference . . . 195
8.8.1 Macro Definition Documentation . . . 198
8.8.1.1 BARRY_BARRAYDENSE_MEAT_HPP . . . 198
8.8.1.2 BDENSE_TEMPLATE . . . 198
8.8.1.3 BDENSE_TEMPLATE_ARGS . . . 198
8.8.1.4 BDENSE_TYPE . . . 198
8.8.1.5 COL . . . 199
8.8.1.6 POS . . . 199
8.8.1.7 ROW . . . 199
8.8.1.8 ZERO_CELL . . . 199
8.8.2 Function Documentation . . . 199
8.8.2.1 ans() . . . 199
8.8.2.2 BDENSE_TEMPLATE() [1/25] . . . 200
8.8.2.3 BDENSE_TEMPLATE() [2/25] . . . 200
8.8.2.4 BDENSE_TEMPLATE() [3/25] . . . 200
8.8.2.5 BDENSE_TEMPLATE() [4/25] . . . 200
8.8.2.6 BDENSE_TEMPLATE() [5/25] . . . 200
8.8.2.7 BDENSE_TEMPLATE() [6/25] . . . 200
8.8.2.8 BDENSE_TEMPLATE() [7/25] . . . 201
8.8.2.9 BDENSE_TEMPLATE() [8/25] . . . 201
8.8.2.10 BDENSE_TEMPLATE() [9/25] . . . 201
8.8.2.11 BDENSE_TEMPLATE() [10/25] . . . 201
8.8.2.12 BDENSE_TEMPLATE() [11/25] . . . 201
8.8.2.13 BDENSE_TEMPLATE() [12/25] . . . 202
8.8.2.14 BDENSE_TEMPLATE() [13/25] . . . 202
8.8.2.15 BDENSE_TEMPLATE() [14/25] . . . 202
8.8.2.16 BDENSE_TEMPLATE() [15/25] . . . 202
8.8.2.17 BDENSE_TEMPLATE() [16/25] . . . 202
8.8.2.18 BDENSE_TEMPLATE() [17/25] . . . 202
8.8.2.19 BDENSE_TEMPLATE() [18/25] . . . 203
8.8.2.20 BDENSE_TEMPLATE() [19/25] . . . 203
8.8.2.21 BDENSE_TEMPLATE() [20/25] . . . 203
8.8.2.22 BDENSE_TEMPLATE() [21/25] . . . 203
8.8.2.23 BDENSE_TEMPLATE() [22/25] . . . 203
8.8.2.24 BDENSE_TEMPLATE() [23/25] . . . 203
8.8.2.25 BDENSE_TEMPLATE() [24/25] . . . 204
8.8.2.26 BDENSE_TEMPLATE() [25/25] . . . 204
8.8.2.27 COL() . . . 204
8.8.2.28 for() [1/2] . . . 204
8.8.2.29 for() [2/2] . . . 204
8.8.2.30 if() [1/16] . . . 204

8.8.2.31 if() [2/16] . . . 205
8.8.2.32 if() [3/16] . . . 205
8.8.2.33 if() [4/16] . . . 205
8.8.2.34 if() [5/16] . . . 205
8.8.2.35 if() [6/16] . . . 205
8.8.2.36 if() [7/16] . . . 205
8.8.2.37 if() [8/16] . . . 206
8.8.2.38 if() [9/16] . . . 206
8.8.2.39 if() [10/16] . . . 206
8.8.2.40 if() [11/16] . . . 206
8.8.2.41 if() [12/16] . . . 206
8.8.2.42 if() [13/16] . . . 206
8.8.2.43 if() [14/16] . . . 206
8.8.2.44 if() [15/16] . . . 207
8.8.2.45 if() [16/16] . . . 207
8.8.2.46 M() . . . 207
8.8.2.47 resize() . . . 207
8.8.2.48 ROW() [1/2] . . . 207
8.8.2.49 ROW() [2/2] . . . 207
8.8.3 Variable Documentation . . . 207
8.8.3.1 add . . . 208
8.8.3.2 ans . . . 208
8.8.3.3 check_bounds . . . 208
8.8.3.4 check_exists . . . 208
8.8.3.5 col0 . . . 209
8.8.3.6 const . . . 209
8.8.3.7 copy_data . . . 209
8.8.3.8 data . . . 209
8.8.3.9 delete_data . . . 209
8.8.3.10 delete_data_ . . . 210
8.8.3.11 else . . . 210
8.8.3.12 false . . . 210
8.8.3.13 first . . . 210
8.8.3.14 i1 . . . 210
8.8.3.15 j . . . 211
8.8.3.16 j0 . . . 211
8.8.3.17 j1 . . . 211
8.8.3.18 M . . . 211
8.8.3.19 M_ . . . 211
8.8.3.20 N . . . 212
8.8.3.21 NCells . . . 212
8.8.3.22 report . . . 212

8.8.3.23 return . . . 212
8.8.3.24 row0 . . . 212
8.8.3.25 source . . . 212
8.8.3.26 target . . . 213
8.8.3.27 v . . . 213
8.8.3.28 value . . . 213
8.9 include/barry/barraydensecell-bones.hpp File Reference . . . 213
8.10 include/barry/barraydensecell-meat.hpp File Reference . . . 214
8.10.1 Macro Definition Documentation . . . 215
8.10.1.1 BARRY_BARRAYDENSECELL_MEAT_HPP . . . 215
8.10.1.2 POS . . . 215
8.11 include/barry/barrayvector-bones.hpp File Reference . . . 215
8.12 include/barry/barrayvector-meat.hpp File Reference . . . 216
8.12.1 Macro Definition Documentation . . . 217
8.12.1.1 BARRY_BARRAYVECTOR_MEAT_HPP . . . 217
8.13 include/barry/barry-configuration.hpp File Reference . . . 217
8.13.1 Macro Definition Documentation . . . 218
8.13.1.1 BARRY_CHECK_SUPPORT . . . 218
8.13.1.2 BARRY_ISFINITE . . . 218
8.13.1.3 BARRY_MAX_NUM_ELEMENTS . . . 218
8.13.1.4 BARRY_SAFE_EXP . . . 218
8.13.1.5 printf_barry . . . 218
8.13.2 Typedef Documentation . . . 218
8.13.2.1 Map . . . 219
8.14 include/barry/barry-debug.hpp File Reference . . . 219
8.14.1 Macro Definition Documentation . . . 219
8.14.1.1 BARRY_DEBUG_LEVEL . . . 219
8.15 include/barry/barry.hpp File Reference . . . 220
8.15.1 Macro Definition Documentation . . . 221
8.15.1.1 BARRY_HPP . . . 221
8.15.1.2 BARRY_VERSION . . . 221
8.15.1.3 COUNTER_FUNCTION . . . 221
8.15.1.4 COUNTER_LAMBDA . . . 221
8.15.1.5 RULE_FUNCTION . . . 222
8.15.1.6 RULE_LAMBDA . . . 222
8.16 include/barry/cell-bones.hpp File Reference . . . 222
8.17 include/barry/cell-meat.hpp File Reference . . . 223
8.18 include/barry/col-bones.hpp File Reference . . . 224
8.19 include/barry/counters-bones.hpp File Reference . . . 224
8.20 include/barry/counters-meat.hpp File Reference . . . 225
8.20.1 Macro Definition Documentation . . . 227
8.20.1.1 COUNTER_TEMPLATE . . . 227

8.20.1.2 COUNTER_TEMPLATE_ARGS . . . 227
8.20.1.3 COUNTER_TYPE . . . 228
8.20.1.4 COUNTERS_TEMPLATE . . . 228
8.20.1.5 COUNTERS_TEMPLATE_ARGS . . . 228
8.20.1.6 COUNTERS_TYPE . . . 228
8.20.2 Function Documentation . . . 228
8.20.2.1 count_fun() . . . 228
8.20.2.2 COUNTER_TEMPLATE() [1/7] . . . 229
8.20.2.3 COUNTER_TEMPLATE() [2/7] . . . 229
8.20.2.4 COUNTER_TEMPLATE() [3/7] . . . 229
8.20.2.5 COUNTER_TEMPLATE() [4/7] . . . 229
8.20.2.6 COUNTER_TEMPLATE() [5/7] . . . 229
8.20.2.7 COUNTER_TEMPLATE() [6/7] . . . 229
8.20.2.8 COUNTER_TEMPLATE() [7/7] . . . 230
8.20.2.9 COUNTERS_TEMPLATE() [1/8] . . . 230
8.20.2.10 COUNTERS_TEMPLATE() [2/8] . . . 230
8.20.2.11 COUNTERS_TEMPLATE() [3/8] . . . 230
8.20.2.12 COUNTERS_TEMPLATE() [4/8] . . . 230
8.20.2.13 COUNTERS_TEMPLATE() [5/8] . . . 231
8.20.2.14 COUNTERS_TEMPLATE() [6/8] . . . 231
8.20.2.15 COUNTERS_TEMPLATE() [7/8] . . . 231
8.20.2.16 COUNTERS_TEMPLATE() [8/8] . . . 231
8.20.2.17 data() . . . 231
8.20.2.18 delete_data() [1/3] . . . 231
8.20.2.19 delete_data() [2/3] . . . 232
8.20.2.20 delete_data() [3/3] . . . 232
8.20.2.21 delete_to_be_deleted() [1/2] . . . 232
8.20.2.22 delete_to_be_deleted() [2/2] . . . 232
8.20.2.23 desc() . . . 232
8.20.2.24 init_fun() [1/3] . . . 232
8.20.2.25 init_fun() [2/3] . . . 233
8.20.2.26 init_fun() [3/3] . . . 233
8.20.2.27 name() . . . 233
8.20.2.28 push_back() [1/2] . . . 233
8.20.2.29 push_back() [2/2] . . . 233
8.20.2.30 to_be_deleted() [1/2] . . . 233
8.20.2.31 to_be_deleted() [2/2] . . . 233
8.20.3 Variable Documentation . . . 234
8.20.3.1 count_fun_ . . . 234
8.20.3.2 counter . . . 234
8.20.3.3 counter_ . . . 234
8.20.3.4 data_ . . . 234

8.20.3.5 delete_data_ . . . 235
8.20.3.6 desc_ . . . 235
8.20.3.7 i . . . 235
8.20.3.8 init_fun_ . . . 235
8.20.3.9 j . . . 235
8.20.3.10 name_ . . . 236
8.20.3.11 noexcept . . . 236
8.20.3.12 return . . . 236
8.21 include/barry/counters/network-css.hpp File Reference . . . 237
8.21.1 Macro Definition Documentation . . . 238
8.21.1.1 CSS_APPEND . . . 238
8.21.1.2 CSS_CASE_ELSE . . . 238
8.21.1.3 CSS_CASE_PERCEIVED . . . 238
8.21.1.4 CSS_CASE_TRUTH . . . 238
8.21.1.5 CSS_CHECK_SIZE . . . 238
8.21.1.6 CSS_CHECK_SIZE_INIT . . . 239
8.21.1.7 CSS_PERCEIVED_CELLS . . . 239
8.21.1.8 CSS_SIZE . . . 239
8.21.1.9 CSS_TRUE_CELLS . . . 239
8.21.2 Function Documentation . . . 239
8.21.2.1 counter_css_completely_false_recip_comiss() . . . 240
8.21.2.2 counter_css_completely_false_recip_omiss() . . . 240
8.21.2.3 counter_css_mixed_recip() . . . 240
8.21.2.4 counter_css_partially_false_recip_commi() . . . 240
8.21.2.5 counter_css_partially_false_recip_omiss() . . . 241
8.22 include/barry/counters/network.hpp File Reference . . . 241
8.22.1 Macro Definition Documentation . . . 243
8.22.1.1 NET_C_DATA_IDX . . . 243
8.22.1.2 NET_C_DATA_NUM . . . 243
8.22.1.3 NETWORK_COUNTER . . . 244
8.22.1.4 NETWORK_COUNTER_LAMBDA . . . 244
8.22.1.5 NETWORK_RULE . . . 244
8.22.1.6 NETWORK_RULE_LAMBDA . . . 244
8.22.2 Typedef Documentation . . . 245
8.22.2.1 NetCounter . . . 245
8.22.2.2 NetCounters . . . 245
8.22.2.3 NetModel . . . 245
8.22.2.4 NetRule . . . 245
8.22.2.5 NetRules . . . 245
8.22.2.6 NetStatsCounter . . . 246
8.22.2.7 NetSupport . . . 246
8.22.2.8 Network . . . 246

8.22.3 Function Documentation . . . 246
8.22.3.1 rules_zerodiag() . . . 246
8.23 include/barry/counters/phylo.hpp File Reference . . . 246
8.23.1 Macro Definition Documentation . . . 248
8.23.1.1 DEFAULT_DUPLICATION . . . 248
8.23.1.2 DUPL_DUPL . . . 249
8.23.1.3 DUPL_EITH . . . 249
8.23.1.4 DUPL_SPEC . . . 249
8.23.1.5 IF_MATCHES . . . 249
8.23.1.6 IF_NOTMATCHES . . . 249
8.23.1.7 IS_DUPLICATION . . . 250
8.23.1.8 IS_EITHER . . . 250
8.23.1.9 IS_SPECIATION . . . 250
8.23.1.10 MAKE_DUPL_VARS . . . 250
8.23.1.11 PHYLO_CHECK_MISSING . . . 250
8.23.1.12 PHYLO_COUNTER_LAMBDA . . . 251
8.23.1.13 PHYLO_RULE_DYN_LAMBDA . . . 251
8.23.2 Typedef Documentation . . . 251
8.23.2.1 PhyloArray . . . 251
8.23.2.2 PhyloCounter . . . 251
8.23.2.3 PhyloCounters . . . 252
8.23.2.4 PhyloModel . . . 252
8.23.2.5 PhyloPowerSet . . . 252
8.23.2.6 PhyloRule . . . 252
8.23.2.7 PhyloRuleData . . . 252
8.23.2.8 PhyloRuleDyn . . . 252
8.23.2.9 PhyloRules . . . 253
8.23.2.10 PhyloRulesDyn . . . 253
8.23.2.11 PhyloStatsCounter . . . 253
8.23.2.12 PhyloSupport . . . 253
8.23.3 Function Documentation . . . 253
8.23.3.1 get_last_name() . . . 253
8.24 include/barry/model-bones.hpp File Reference . . . 254
8.24.1 Function Documentation . . . 255
8.24.1.1 keygen_default() . . . 255
8.25 include/barry/model-meat.hpp File Reference . . . 255
8.25.1 Macro Definition Documentation . . . 256
8.25.1.1 MODEL_TEMPLATE . . . 256
8.25.1.2 MODEL_TEMPLATE_ARGS . . . 256
8.25.1.3 MODEL_TYPE . . . 256
8.25.2 Function Documentation . . . 256
8.25.2.1 likelihood_() . . . 257

8.25.2.2 MODEL_TEMPLATE() [1/2] . . . 257
8.25.2.3 MODEL_TEMPLATE() [2/2] . . . 257
8.25.2.4 update_normalizing_constant() . . . 257
8.26 include/barry/models/geese.hpp File Reference . . . 257
8.27 include/barry/models/geese/flock-bones.hpp File Reference . . . 258
8.28 include/barry/models/geese/flock-meat.hpp File Reference . . . 258
8.29 include/barry/models/geese/geese-bones.hpp File Reference . . . 259
8.29.1 Macro Definition Documentation . . . 259
8.29.1.1 INITIALIZED . . . 260
8.29.2 Function Documentation . . . 260
8.29.2.1 keygen_full() . . . 260
8.29.2.2 RULE_FUNCTION() . . . 260
8.29.2.3 vec_diff() . . . 260
8.29.2.4 vector_caster() . . . 260
8.30 include/barry/models/geese/geese-meat-constructors.hpp File Reference . . . 261
8.31 include/barry/models/geese/geese-meat-likelihood.hpp File Reference . . . 261
8.32 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference . . . 262
8.33 include/barry/models/geese/geese-meat-predict.hpp File Reference . . . 263
8.34 include/barry/models/geese/geese-meat-predict_exhaust.hpp File Reference . . . 263
8.35 include/barry/models/geese/geese-meat-predict_sim.hpp File Reference . . . 264
8.36 include/barry/models/geese/geese-meat-simulate.hpp File Reference . . . 264
8.37 include/barry/models/geese/geese-meat.hpp File Reference . . . 265
8.38 include/barry/models/geese/geese-node-bones.hpp File Reference . . . 265
8.39 include/barry/powerset-bones.hpp File Reference . . . 266
8.40 include/barry/powerset-meat.hpp File Reference . . . 267
8.41 include/barry/progress.hpp File Reference . . . 268
8.41.1 Macro Definition Documentation . . . 268
8.41.1.1 BARRY_PROGRESS_BAR_WIDTH . . . 268
8.42 include/barry/rules-bones.hpp File Reference . . . 269
8.42.1 Function Documentation . . . 270
8.42.1.1 rule_fun_default() . . . 270
8.43 include/barry/rules-meat.hpp File Reference . . . 270
8.44 include/barry/statscounter-bones.hpp File Reference . . . 271
8.45 include/barry/statscounter-meat.hpp File Reference . . . 272
8.45.1 Macro Definition Documentation . . . 274
8.45.1.1 STATSCOUNTER_TEMPLATE . . . 274
8.45.1.2 STATSCOUNTER_TEMPLATE_ARGS . . . 274
8.45.1.3 STATSCOUNTER_TYPE . . . 274
8.45.2 Function Documentation . . . 275
8.45.2.1 for() . . . 275
8.45.2.2 resize() . . . 275
8.45.2.3 STATSCOUNTER_TEMPLATE() [1/9] . . . 275

8.45.2.4 STATSCOUNTER_TEMPLATE() [2/9] . . . 275
8.45.2.5 STATSCOUNTER_TEMPLATE() [3/9] . . . 275
8.45.2.6 STATSCOUNTER_TEMPLATE() [4/9] . . . 276
8.45.2.7 STATSCOUNTER_TEMPLATE() [5/9] . . . 276
8.45.2.8 STATSCOUNTER_TEMPLATE() [6/9] . . . 276
8.45.2.9 STATSCOUNTER_TEMPLATE() [7/9] . . . 276
8.45.2.10 STATSCOUNTER_TEMPLATE() [8/9] . . . 276
8.45.2.11 STATSCOUNTER_TEMPLATE() [9/9] . . . 276
8.45.3 Variable Documentation . . . 277
8.45.3.1 counter_deleted . . . 277
8.45.3.2 counters . . . 277
8.45.3.3 counters_ . . . 277
8.45.3.4 f_ . . . 277
8.45.3.5 j . . . 278
8.45.3.6 return . . . 278
8.46 include/barry/statsdb.hpp File Reference . . . 278
8.47 include/barry/support-bones.hpp File Reference . . . 279
8.48 include/barry/support-meat.hpp File Reference . . . 281
8.48.1 Macro Definition Documentation . . . 282
8.48.1.1 BARRY_SUPPORT_MEAT_HPP . . . 283
8.48.1.2 SUPPORT_TEMPLATE . . . 283
8.48.1.3 SUPPORT_TEMPLATE_ARGS . . . 283
8.48.1.4 SUPPORT_TYPE . . . 283
8.48.2 Function Documentation . . . 283
8.48.2.1 calc_backend() . . . 284
8.48.2.2 for() . . . 284
8.48.2.3 if() [1/3] . . . 284
8.48.2.4 if() [2/3] . . . 284
8.48.2.5 if() [3/3] . . . 284
8.48.2.6 insert_cell() . . . 284
8.48.2.7 rm_cell() . . . 285
8.48.2.8 SUPPORT_TEMPLATE() [1/17] . . . 285
8.48.2.9 SUPPORT_TEMPLATE() [2/17] . . . 285
8.48.2.10 SUPPORT_TEMPLATE() [3/17] . . . 285
8.48.2.11 SUPPORT_TEMPLATE() [4/17] . . . 285
8.48.2.12 SUPPORT_TEMPLATE() [5/17] . . . 286
8.48.2.13 SUPPORT_TEMPLATE() [6/17] . . . 286
8.48.2.14 SUPPORT_TEMPLATE() [7/17] . . . 286
8.48.2.15 SUPPORT_TEMPLATE() [8/17] . . . 286
8.48.2.16 SUPPORT_TEMPLATE() [9/17] . . . 286
8.48.2.17 SUPPORT_TEMPLATE() [10/17] . . . 286
8.48.2.18 SUPPORT_TEMPLATE() [11/17] . . . 287

8.48.2.19 SUPPORT_TEMPLATE() [12/17] . . . 287
8.48.2.20 SUPPORT_TEMPLATE() [13/17] . . . 287
8.48.2.21 SUPPORT_TEMPLATE() [14/17] . . . 287
8.48.2.22 SUPPORT_TEMPLATE() [15/17] . . . 287
8.48.2.23 SUPPORT_TEMPLATE() [16/17] . . . 288
8.48.2.24 SUPPORT_TEMPLATE() [17/17] . . . 288
8.48.3 Variable Documentation . . . 288
8.48.3.1 array_bank . . . 288
8.48.3.2 cfree . . . 288
8.48.3.3 counters . . . 288
8.48.3.4 counters_ . . . 289
8.48.3.5 delete_counters . . . 289
8.48.3.6 delete_rules . . . 289
8.48.3.7 delete_rules_dyn . . . 289
8.48.3.8 else . . . 289
8.48.3.9 f_ . . . 290
8.48.3.10 return . . . 290
8.48.3.11 rules . . . 290
8.48.3.12 rules_ . . . 290
8.48.3.13 rules_dyn . . . 290
8.48.3.14 stats_bank . . . 291
8.49 include/barry/typedefs.hpp File Reference . . . 291
8.49.1 Typedef Documentation . . . 293
8.49.1.1 Col_type . . . 293
8.49.1.2 Counter_fun_type . . . 293
8.49.1.3 Counts_type . . . 293
8.49.1.4 MapVec_type . . . 293
8.49.1.5 Row_type . . . 294
8.49.1.6 Rule_fun_type . . . 294
8.49.1.7 uint . . . 294
8.49.2 Function Documentation . . . 294
8.49.2.1 vec_equal() . . . 294
8.49.2.2 vec_equal_approx() . . . 295
8.49.2.3 vec_inner_prod() . . . 295
8.50 README.md File Reference . . . 295

Index **297**

Chapter 1

Main Page

Barry: your to-go motif accountant

`This repository` contains a C++ template library that essentially counts sufficient statistics on binary arrays. The goal of the library 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.

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 11
Statistical Models 11
Network counters 12
Phylo counters 17
Phylo rules 22

Chapter 3

Class Index

3.1 Class List

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

BArray< Cell_Type, Data_Type >
: Baseline class for binary arrays . . . 29

BArrayCell< Cell_Type, Data_Type > . . . 43

BArrayCell_const< Cell_Type, Data_Type > . . . 45

BArrayDense< Cell_Type, Data_Type >
: Baseline class for binary arrays . . . 48

BArrayDenseCell< Cell_Type, Data_Type > . . . 62

BArrayDenseCell_const< Cell_Type, Data_Type > . . . 64

BArrayVector< Cell_Type, Data_Type >
: Row or column of a `BArray` . . . 67

BArrayVector_const< Cell_Type, Data_Type > . . . 71

Cell< Cell_Type >
: Entries in BArray. For now, it only has two members: . . . 74

ConstBArrayRowIter< Cell_Type, Data_Type > . . . 79

Counter< Array_Type, Data_Type >
: A counter function based on change statistics . . . 81

Counters< Array_Type, Data_Type >
: Vector of counters . . . 86

Entries< Cell_Type >
: A wrapper class to store `source`, `target`, `val` from a `BArray` object . . . 91

Flock
: A Flock is a group of Geese . . . 93

FreqTable< T >
: Database of statistics . . . 99

Geese
: Annotated Phylo Model . . . 102

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: . . . 114

NetCounterData
: Data class used to store arbitrary uint or double vectors . . . 126

NetworkData
: Data class for Networks . . . 128

Node
: A single node for the model . . . 131

NodeData
Data definition for the PhyloArray class . . . 137
PhyloCounterData . . . 138
PhyloRuleDynData . . . 141
PowerSet< Array_Type, Data_Rule_Type >
Powerset of a binary array . . . 143
Progress
A simple progress bar . . . 149
Rule< Array_Type, Data_Type >
Rule for determining if a cell should be included in a sequence . . . 150
Rules< Array_Type, Data_Type >
Vector of objects of class Rule . . . 152
StatsCounter< Array_Type, Data_Type >
Count stats for a single Array . . . 156
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >
Compute the support of sufficient statistics . . . 159
vecHasher< T > . . . 170

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

include/barry/barray-bones.hpp . . . 171
include/barry/barray-iterator.hpp . . . 172
include/barry/barray-meat-operators.hpp . . . 173
include/barry/barray-meat.hpp . . . 175
include/barry/barraycell-bones.hpp . . . 193
include/barry/barraycell-meat.hpp . . . 193
include/barry/barraydense-bones.hpp . . . 194
include/barry/barraydense-meet.hpp . . . 195
include/barry/barraydensecell-bones.hpp . . . 213
include/barry/barraydensecell-meat.hpp . . . 214
include/barry/barrayvector-bones.hpp . . . 215
include/barry/barrayvector-meat.hpp . . . 216
include/barry/barry-configuration.hpp . . . 217
include/barry/barry-debug.hpp . . . 219
include/barry/barry.hpp . . . 220
include/barry/cell-bones.hpp . . . 222
include/barry/cell-meat.hpp . . . 223
include/barry/col-bones.hpp . . . 224
include/barry/counters-bones.hpp . . . 224
include/barry/counters-meat.hpp . . . 225
include/barry/model-bones.hpp . . . 254
include/barry/model-meat.hpp . . . 255
include/barry/powerset-bones.hpp . . . 266
include/barry/powerset-meat.hpp . . . 267
include/barry/progress.hpp . . . 268
include/barry/rules-bones.hpp . . . 269
include/barry/rules-meat.hpp . . . 270
include/barry/statscounter-bones.hpp . . . 271
include/barry/statscounter-meat.hpp . . . 272
include/barry/statsdb.hpp . . . 278
include/barry/support-bones.hpp . . . 279
include/barry/support-meat.hpp . . . 281
include/barry/typedefs.hpp . . . 291
include/barry/counters/network-css.hpp . . . 237
include/barry/counters/network.hpp . . . 241

include/barry/counters/phylo.hpp . . . 246
include/barry/models/geese.hpp . . . 257
include/barry/models/geese/flock-bones.hpp . . . 258
include/barry/models/geese/flock-meat.hpp . . . 258
include/barry/models/geese/geese-bones.hpp . . . 259
include/barry/models/geese/geese-meat-constructors.hpp . . . 261
include/barry/models/geese/geese-meat-likelihood.hpp . . . 261
include/barry/models/geese/geese-meat-likelihood_exhaust.hpp . . . 262
include/barry/models/geese/geese-meat-predict.hpp . . . 263
include/barry/models/geese/geese-meat-predict_exhaust.hpp . . . 263
include/barry/models/geese/geese-meat-predict_sim.hpp . . . 264
include/barry/models/geese/geese-meat-simulate.hpp . . . 264
include/barry/models/geese/geese-meat.hpp . . . 265
include/barry/models/geese/geese-node-bones.hpp . . . 265

Chapter 5

Module Documentation

5.1 Counting

Classes

- class NetworkData

  Data class for Networks.
- class NodeData

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

  A counter function based on change statistics.

5.1.1 Detailed Description

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

In particular, let the motif be defined as $s(y)$, with y as the binary array. The change statistic when adding cell y_{ij}, i.e. when the cell moves from being 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.

5.2 Statistical Models

Statistical models available in `barry`.

Classes

- class Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >

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

  A Flock is a group of Geese.
- class Geese

  Annotated Phylo Model.

5.2.1 Detailed Description

Statistical models available in `barry`.

5.3 Network counters

Counters for network models.

Functions

- void counter_edges (NetCounters *counters)

  Number of edges.
- void counter_isolates (NetCounters *counters)

  Number of isolated vertices.
- void counter_mutual (NetCounters *counters)

  Number of mutual ties.
- void counter_istar2 (NetCounters *counters)
- void counter_ostar2 (NetCounters *counters)
- void counter_ttriads (NetCounters *counters)
- void counter_ctriads (NetCounters *counters)
- void counter_density (NetCounters *counters)
- void counter_idegree15 (NetCounters *counters)
- void counter_odegree15 (NetCounters *counters)
- void counter_absdiff (NetCounters *counters, uint attr_id, double alpha=1.0)

  Sum of absolute attribute difference between ego and alter.
- void counter_diff (NetCounters *counters, uint attr_id, double alpha=1.0, double tail_head=true)

  Sum of attribute difference between ego and alter to pow(alpha)
- NETWORK_COUNTER (init_single_attr)
- void counter_nodeicov (NetCounters *counters, uint attr_id)
- void counter_nodeocov (NetCounters *counters, uint attr_id)
- void counter_nodecov (NetCounters *counters, uint attr_id)
- void counter_nodematch (NetCounters *counters, uint attr_id)
- void counter_idegree (NetCounters *counters, std::vector< uint > d)

  Counts number of vertices with a given in-degree.
- void counter_odegree (NetCounters *counters, std::vector< uint > d)

  Counts number of vertices with a given out-degree.
- void counter_degree (NetCounters *counters, std::vector< uint > d)

  Counts number of vertices with a given out-degree.

5.3.1 Detailed Description

Counters for network models.

Parameters

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

5.3.2 Function Documentation

5.3.2.1 counter_absdiff()

```
void counter_absdiff (
            NetCounters * counters,
            uint attr_id,
            double alpha = 1.0 )  [inline]
```

Sum of absolute attribute difference between ego and alter.

Definition at line 501 of file network.hpp.

5.3.2.2 counter_ctriads()

```
void counter_ctriads (
            NetCounters * counters )  [inline]
```

Definition at line 369 of file network.hpp.

5.3.2.3 counter_degree()

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

Counts number of vertices with a given out-degree.

Definition at line 798 of file network.hpp.

5.3.2.4 counter_density()

```
void counter_density (
            NetCounters * counters )  [inline]
```

Definition at line 422 of file network.hpp.

5.3.2.5 counter_diff()

```
void counter_diff (
            NetCounters * counters,
            uint attr_id,
            double alpha = 1.0,
            double tail_head = true )  [inline]
```

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

Definition at line 545 of file network.hpp.

5.3.2.6 counter_edges()

```
void counter_edges (
            NetCounters * counters )  [inline]
```

Number of edges.

Definition at line 123 of file network.hpp.

5.3.2.7 counter_idegree()

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

Counts number of vertices with a given in-degree.

Definition at line 701 of file network.hpp.

5.3.2.8 counter_idegree15()

```
void counter_idegree15 (
            NetCounters * counters )  [inline]
```

Definition at line 447 of file network.hpp.

5.3.2.9 counter_isolates()

```
void counter_isolates (
            NetCounters * counters )  [inline]
```

Number of isolated vertices.

Definition at line 144 of file network.hpp.

5.3.2.10 counter_istar2()

```
void counter_istar2 (
            NetCounters * counters )  [inline]
```

Definition at line 235 of file network.hpp.

5.3.2.11 counter_mutual()

```
void counter_mutual (
            NetCounters * counters )  [inline]
```

Number of mutual ties.

Definition at line 182 of file network.hpp.

5.3.2.12 counter_nodecov()

```
void counter_nodecov (
            NetCounters * counters,
            uint attr_id )  [inline]
```

Definition at line 651 of file network.hpp.

5.3.2.13 counter_nodeicov()

```
void counter_nodeicov (
            NetCounters * counters,
            uint attr_id )  [inline]
```

Definition at line 607 of file network.hpp.

5.3.2.14 counter_nodematch()

```
void counter_nodematch (
            NetCounters * counters,
            uint attr_id )  [inline]
```

Definition at line 673 of file network.hpp.

5.3.2.15 counter_nodeocov()

```
void counter_nodeocov (
            NetCounters * counters,
            uint attr_id )  [inline]
```

Definition at line 629 of file network.hpp.

5.3.2.16 counter_odegree()

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

Counts number of vertices with a given out-degree.

Definition at line 749 of file network.hpp.

5.3.2.17 counter_odegree15()

```
void counter_odegree15 (
            NetCounters * counters )  [inline]
```

Definition at line 473 of file network.hpp.

5.3.2.18 counter_ostar2()

```
void counter_ostar2 (
            NetCounters * counters )  [inline]
```

Definition at line 256 of file network.hpp.

5.3.2.19 counter_ttriads()

```
void counter_ttriads (
            NetCounters * counters )  [inline]
```

Definition at line 280 of file network.hpp.

5.3.2.20 NETWORK_COUNTER()

```
NETWORK_COUNTER (
              init_single_attr  )
```

Definition at line 589 of file network.hpp.

5.4 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_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_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.

5.4.1 Detailed Description

Counters for phylogenetic modeling.

Parameters

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

5.4.2 Function Documentation

5.4.2.1 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:

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

This algorithm implements the change statistic.

Definition at line 1194 of file phylo.hpp.

5.4.2.2 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 819 of file phylo.hpp.

5.4.2.3 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 192 of file phylo.hpp.

5.4.2.4 counter_gains_k_offspring()

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

k genes gain function nfun

Definition at line 238 of file phylo.hpp.

5.4.2.5 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 311 of file phylo.hpp.

5.4.2.6 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.

$<$ How many genes diverge the parent

Definition at line 1293 of file phylo.hpp.

5.4.2.7 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 877 of file phylo.hpp.

5.4.2.8 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 622 of file phylo.hpp.

5.4.2.9 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 540 of file phylo.hpp.

5.4.2.10 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 989 of file phylo.hpp.

5.4.2.11 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 1075 of file phylo.hpp.

5.4.2.12 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 672 of file phylo.hpp.

5.4.2.13 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 154 of file phylo.hpp.

5.4.2.14 counter_overall_loss()

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

Overall functional loss.

Definition at line 497 of file phylo.hpp.

5.4.2.15 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 383 of file phylo.hpp.

5.4.2.16 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 732 of file phylo.hpp.

5.5 Phylo rules

Rules for phylogenetic modeling.

Classes

- class PhyloRuleDynData

Functions

- void rule_dyn_limit_changes (PhyloSupport *support, uint pos, uint lb, uint ub, unsigned int duplication=DEFAULT_DUPLICATIO

  Overall functional gains.

5.5.1 Detailed Description

Rules for phylogenetic modeling.

Parameters

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

5.5.2 Function Documentation

5.5.2.1 rule_dyn_limit_changes()

```
void rule_dyn_limit_changes (
            PhyloSupport * support,
            uint pos,
            uint lb,
            uint ub,
            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 1442 of file phylo.hpp.

Chapter 6

Namespace Documentation

6.1 barry Namespace Reference

barry: Your go-to motif accountant

Namespaces

- counters

  Tree class and TreeIterator class.

6.1.1 Detailed Description

barry: Your go-to motif accountant

6.2 barry::counters Namespace Reference

Tree class and TreeIterator class.

Namespaces

- network
- phylo

6.2.1 Detailed Description

Tree class and TreeIterator class.

6.3 barry::counters::network Namespace Reference

6.4 barry::counters::phylo Namespace Reference

6.5 CHECK Namespace Reference

Integer constants used to specify which cell should be check.

Variables

- const int BOTH = -1
- const int NONE = 0
- const int ONE = 1
- const int TWO = 2

6.5.1 Detailed Description

Integer constants used to specify which cell should be check.

6.5.2 Variable Documentation

6.5.2.1 BOTH

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

Definition at line 26 of file typedefs.hpp.

6.5.2.2 NONE

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

Definition at line 27 of file typedefs.hpp.

6.5.2.3 ONE

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

Definition at line 28 of file typedefs.hpp.

6.5.2.4 TWO

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

Definition at line 29 of file typedefs.hpp.

6.6 EXISTS Namespace Reference

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

Variables

- const int BOTH = -1
- const int NONE = 0
- const int ONE = 1
- const int TWO = 1
- const int UKNOWN = -1
- const int AS_ZERO = 0
- const int AS_ONE = 1

6.6.1 Detailed Description

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

6.6.2 Variable Documentation

6.6.2.1 AS_ONE

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

Definition at line 44 of file typedefs.hpp.

6.6.2.2 AS_ZERO

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

Definition at line 43 of file typedefs.hpp.

6.6.2.3 BOTH

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

Definition at line 37 of file typedefs.hpp.

6.6.2.4 NONE

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

Definition at line 38 of file typedefs.hpp.

6.6.2.5 ONE

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

Definition at line 39 of file typedefs.hpp.

6.6.2.6 TWO

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

Definition at line 40 of file typedefs.hpp.

6.6.2.7 UKNOWN

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

Definition at line 42 of file typedefs.hpp.

Chapter 7

Class Documentation

7.1 BArray< Cell_Type, Data_Type > Class Template Reference

Baseline class for binary arrays.

`#include <barray-bones.hpp>`

Public Member Functions

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

  Get the edgelist.
- void transpose ()
- void clear (bool hard=true)
- void resize (uint N_, uint M_)
- void reserve ()
- void print (const char *fmt=nullptr,...) const

Constructors

Parameters

N_	*Number of rows*
M_	*Number of columns*
source	*An unsigned vector ranging from 0 to N_*
target	*An unsigned int vector ranging from 0 to M_*
target	*When `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 ()
- 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 BArrayCell_const< Cell_Type, Data_Type > operator() (uint i, uint j, bool check_bounds=true) const
- void rm_cell (uint i, uint j, bool check_bounds=true, bool check_exists=true)
- void insert_cell (uint i, uint j, const Cell< Cell_Type > &v, bool check_bounds, bool check_exists)
- void insert_cell (uint i, uint j, Cell< Cell_Type > &&v, bool check_bounds, bool check_exists)
- void insert_cell (uint i, uint j, Cell_Type v, bool check_bounds, bool check_exists)
- void swap_cells (uint i0, uint j0, uint i1, uint j1, bool check_bounds=true, int check_exists=CHECK::BOTH, int *report=nullptr)
- void toggle_cell (uint i, uint j, bool check_bounds=true, int check_exists=EXISTS::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 >

7.1.1 Detailed Description

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

Baseline class for binary arrays.

`BArray` class objects are arbitrary arrays in which non-empty cells hold data of type `Cell_Type`. The non-empty cells are stored by row and indexed using `unordered_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 22 of file barray-bones.hpp.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 BArray() [1/6]

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

Zero-size array.

Definition at line 60 of file barray-bones.hpp.

7.1.2.2 BArray() [2/6]

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

Empty array.

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

7.1.2.3 BArray() [3/6]

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

Edgelist with data.

7.1.2.4 BArray() [4/6]

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

Edgelist with no data (simpler)

7.1.2.5 BArray() [5/6]

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

Copy constructor.

7.1.2.6 BArray() [6/6]

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

Move operator.

7.1.2.7 ~BArray()

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

7.1.3 Member Function Documentation

7.1.3.1 clear()

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

7.1.3.2 col()

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

7.1.3.3 D() [1/2]

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

7.1.3.4 D() [2/2]

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

7.1.3.5 default_val()

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

7.1.3.6 flush_data()

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

7.1.3.7 get_cell()

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

7.1.3.8 get_col_vec() [1/2]

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

7.1.3.9 get_col_vec() [2/2]

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

7.1.3.10 get_entries()

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

Get the edgelist.

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

Returns

Entries<Cell_Type>

7.1.3.11 get_row_vec() [1/2]

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

7.1.3.12 get_row_vec() [2/2]

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

7.1.3.13 insert_cell() [1/3]

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

7.1.3.14 insert_cell() [2/3]

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

7.1.3.15 insert_cell() [3/3]

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

7.1.3.16 is_empty()

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

7.1.3.17 ncol()

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

7.1.3.18 nnozero()

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

7.1.3.19 nrow()

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

7.1.3.20 operator()() [1/2]

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

7.1.3.21 operator()() [2/2]

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

7.1.3.22 operator*=()

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

7.1.3.23 operator+=() [1/3]

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

7.1.3.24 operator+=() [2/3]

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

7.1.3.25 operator+=() [3/3]

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

7.1.3.26 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 )
```

7.1.3.27 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 )
```

7.1.3.28 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 )
```

7.1.3.29 operator/=()

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

7.1.3.30 operator=() [1/2]

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

Move assignment.

7.1.3.31 operator=() [2/2]

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

Assignment constructor.

7.1.3.32 operator==()

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

7.1.3.33 out_of_range()

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

7.1.3.34 print()

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

7.1.3.35 reserve()

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

7.1.3.36 resize()

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

7.1.3.37 rm_cell()

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

7.1.3.38 row()

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

7.1.3.39 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_*	

7.1.3.40 swap_cells()

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

```
            uint i0,
            uint j0,
            uint i1,
            uint j1,
            bool check_bounds = true,
            int check_exists = CHECK::BOTH,
            int * report = nullptr )
```

7.1.3.41 swap_cols()

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

7.1.3.42 swap_rows()

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

7.1.3.43 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 )
```

7.1.3.44 toggle_lock()

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

7.1.3.45 transpose()

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

7.1.3.46 zero_col()

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

7.1.3.47 zero_row()

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

7.1.4 Friends And Related Function Documentation

7.1.4.1 BArrayCell< Cell_Type, Data_Type >

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

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

7.1.4.2 BArrayCell_const< Cell_Type, Data_Type >

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

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

7.1.5 Member Data Documentation

7.1.5.1 visited

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

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

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

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

- include/barry/barray-bones.hpp

7.2 BArrayCell< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

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

7.2.1 Detailed Description

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

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

7.2.2 Constructor & Destructor Documentation

7.2.2.1 BArrayCell()

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

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

7.2.2.2 ~BArrayCell()

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

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

7.2.3 Member Function Documentation

7.2.3.1 operator Cell_Type()

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

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

7.2.3.2 operator*=()

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

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

7.2.3.3 operator+=()

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

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

7.2.3.4 operator-=()

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

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

7.2.3.5 operator/=()

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

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

7.2.3.6 operator=()

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

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

7.2.3.7 operator==()

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

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

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

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

7.3 BArrayCell_const< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

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

7.3.1 Detailed Description

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

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

7.3.2 Constructor & Destructor Documentation

7.3.2.1 BArrayCell_const()

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

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

7.3.2.2 ~BArrayCell_const()

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

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

7.3.3 Member Function Documentation

7.3.3.1 operator Cell_Type()

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

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

7.3.3.2 operator"!=()

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

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

7.3.3.3 operator<()

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

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

7.3.3.4 operator<=()

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

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

7.3.3.5 operator==()

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

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

7.3.3.6 operator>()

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

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

7.3.3.7 operator>=()

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

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

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

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

7.4 BArrayDense< Cell_Type, Data_Type > Class Template Reference

Baseline class for binary arrays.

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

Public Member Functions

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

  Get the edgelist.
- void transpose ()
- void clear (bool hard=true)
- void resize (uint N_, uint M_)
- void reserve ()
- void print () const

Constructors

Parameters

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

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

- void set_data (Data_Type *data_, bool delete_data_=false)
  Set the data object.
- Data_Type * D ()
- const Data_Type * D () const

Queries

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

Parameters

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

7.4.1 Detailed Description

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

Baseline class for binary arrays.

`BArrayDense` class objects are arbitrary arrays in which non-empty cells hold data of type `Cell_Type`. The non-empty cells are stored by row and indexed using `unordered_maps`, i.e. `std::vector< std::unordered_map<unsigned int,Cell_Type> >`.

Template Parameters

Cell_Type	Type of cell (any type).
Data_Type	Data type of the array (bool default).

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

7.4.2 Constructor & Destructor Documentation

7.4.2.1 BArrayDense() [1/6]

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

Zero-size array.

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

7.4.2.2 BArrayDense() [2/6]

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

Empty array.

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

7.4.2.3 BArrayDense() [3/6]

```
template<typename Cell_Type = 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.

7.4.2.4 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)

7.4.2.5 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.

7.4.2.6 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.

7.4.2.7 ~BArrayDense()

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

7.4.3 Member Function Documentation

7.4.3.1 clear()

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

7.4.3.2 col()

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

7.4.3.3 D() [1/2]

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

7.4.3.4 D() [2/2]

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

7.4.3.5 default_val()

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

7.4.3.6 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
```

7.4.3.7 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
```

7.4.3.8 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
```

7.4.3.9 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>

7.4.3.10 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
```

7.4.3.11 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
```

7.4.3.12 insert_cell() [1/3]

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

7.4.3.13 insert_cell() [2/3]

```
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 )
```

7.4.3.14 insert_cell() [3/3]

```
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 )
```

7.4.3.15 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
```

7.4.3.16 ncol()

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

7.4.3.17 nnozero()

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

7.4.3.18 nrow()

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

7.4.3.19 operator()() [1/2]

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

7.4.3.20 operator()() [2/2]

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

7.4.3.21 operator*=()

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

7.4.3.22 operator+=() [1/3]

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

7.4.3.23 operator+=() [2/3]

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

7.4.3.24 operator+=() [3/3]

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

7.4.3.25 operator-=() [1/3]

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

7.4.3.26 operator-=() [2/3]

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

7.4.3.27 operator-=() [3/3]

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

7.4.3.28 operator/=()

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

7.4.3.29 operator=() [1/2]

```
template<typename Cell_Type = 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.

7.4.3.30 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.

7.4.3.31 operator==()

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

7.4.3.32 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
```

7.4.3.33 print()

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

7.4.3.34 reserve()

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

7.4.3.35 resize()

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

7.4.3.36 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 )
```

7.4.3.37 row()

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

7.4.3.38 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_*	

7.4.3.39 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 )
```

7.4.3.40 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 )
```

7.4.3.41 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 )
```

7.4.3.42 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 )
```

7.4.3.43 toggle_lock()

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

7.4.3.44 transpose()

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

7.4.3.45 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 )
```

7.4.3.46 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 )
```

7.4.4 Friends And Related Function Documentation

7.4.4.1 BArrayCell< Cell_Type, Data_Type >

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

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

7.4.4.2 BArrayCell_const< Cell_Type, Data_Type >

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

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

7.4.5 Member Data Documentation

7.4.5.1 visited

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

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

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

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

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

7.5 BArrayDenseCell< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

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

7.5.1 Detailed Description

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

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

7.5.2 Constructor & Destructor Documentation

7.5.2.1 BArrayDenseCell()

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

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

7.5.2.2 ~BArrayDenseCell()

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

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

7.5.3 Member Function Documentation

7.5.3.1 operator Cell_Type()

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

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

7.5.3.2 operator*=()

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

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

7.5.3.3 operator+=()

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

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

7.5.3.4 operator-=()

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

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

7.5.3.5 operator/=()

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

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

7.5.3.6 operator=()

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

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

7.5.3.7 operator==()

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

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

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

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

7.6 BArrayDenseCell_const< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

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

7.6.1 Detailed Description

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

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

7.6.2 Constructor & Destructor Documentation

7.6.2.1 BArrayDenseCell_const()

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

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

7.6.2.2 ~BArrayDenseCell_const()

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

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

7.6.3 Member Function Documentation

7.6.3.1 operator Cell_Type()

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

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

7.6.3.2 operator"!=()

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

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

7.6.3.3 operator<()

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

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

7.6.3.4 operator<=()

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

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

7.6.3.5 operator==()

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

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

7.6.3.6 operator>()

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

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

7.6.3.7 operator>=()

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

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

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

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

7.7 BArrayVector< Cell_Type, Data_Type > Class Template Reference

Row or column of a BArray

`#include <barrayvector-bones.hpp>`

Public Member Functions

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

7.7.1 Detailed Description

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

Row or column of a BArray

Template Parameters

Cell_Type	
Data_Type	

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

7.7.2 Constructor & Destructor Documentation

7.7.2.1 BArrayVector()

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

Construct a new BArrayVector object.

Parameters

*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 34 of file barrayvector-bones.hpp.

7.7.2.2 ~BArrayVector()

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

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

7.7.3 Member Function Documentation

7.7.3.1 begin()

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

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

7.7.3.2 end()

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

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

7.7.3.3 is_col()

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

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

7.7.3.4 is_row()

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

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

7.7.3.5 operator std::vector< Cell_Type >()

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

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

7.7.3.6 operator*=()

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

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

7.7.3.7 operator+=()

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

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

7.7.3.8 operator-=()

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

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

7.7.3.9 operator/=()

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

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

7.7.3.10 operator=()

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

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

7.7.3.11 operator==()

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

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

7.7.3.12 size()

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

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

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

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

7.8 BArrayVector_const< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

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

7.8.1 Detailed Description

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

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

7.8.2 Constructor & Destructor Documentation

7.8.2.1 BArrayVector_const()

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

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

7.8.2.2 ~BArrayVector_const()

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

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

7.8.3 Member Function Documentation

7.8.3.1 begin()

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

7.8.3.2 end()

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

7.8.3.3 is_col()

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

7.8.3.4 is_row()

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

7.8.3.5 operator std::vector< Cell_Type >()

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

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

7.8.3.6 operator"!=()

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

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

7.8.3.7 operator<()

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

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

7.8.3.8 operator<=()

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

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

7.8.3.9 operator==()

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

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

7.8.3.10 operator>()

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

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

7.8.3.11 operator>=()

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

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

7.8.3.12 size()

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

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

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

7.9 Cell< Cell_Type > Class Template Reference

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

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

Public Member Functions

- Cell ()
- Cell (Cell_Type value_, bool visited_=false)
- ~Cell ()
- Cell (const Cell< Cell_Type > &arg)
- Cell< Cell_Type > & operator= (Cell< Cell_Type > &other)
- Cell (Cell< Cell_Type > &&arg) noexcept
- Cell< Cell_Type > & operator= (Cell< Cell_Type > &&other) noexcept
- void add (Cell_Type x)
- operator Cell_Type () const
- bool operator== (const Cell< Cell_Type > &rhs) const
- bool operator!= (const Cell< Cell_Type > &rhs) const
- void add (double x)
- void add (unsigned int x)
- void add (int x)
- Cell ()
- Cell ()
- Cell ()

Public Attributes

- Cell_Type value
- bool visited

7.9.1 Detailed Description

template<class Cell_Type>
class Cell< Cell_Type >

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

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

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

7.9.2 Constructor & Destructor Documentation

7.9.2.1 Cell() [1/7]

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

7.9.2.2 Cell() [2/7]

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

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

7.9.2.3 ~Cell()

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

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

7.9.2.4 Cell() [3/7]

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

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

7.9.2.5 Cell() [4/7]

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

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

7.9.2.6 Cell() [5/7]

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

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

7.9.2.7 Cell() [6/7]

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

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

7.9.2.8 Cell() [7/7]

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

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

7.9.3 Member Function Documentation

7.9.3.1 add() [1/4]

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

7.9.3.2 add() [2/4]

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

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

7.9.3.3 add() [3/4]

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

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

7.9.3.4 add() [4/4]

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

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

7.9.3.5 operator Cell_Type()

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

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

7.9.3.6 operator"!=()

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

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

7.9.3.7 operator=() [1/2]

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

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

7.9.3.8 operator=() [2/2]

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

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

7.9.3.9 operator==()

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

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

7.9.4 Member Data Documentation

7.9.4.1 value

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

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

7.9.4.2 visited

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

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

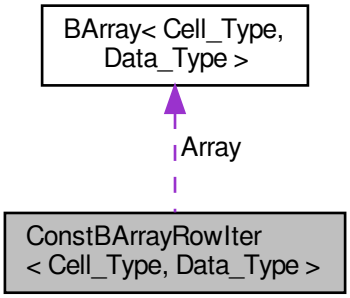
The documentation for this class was generated from the following files:

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

7.10 ConstBArrayRowIter< Cell_Type, Data_Type > Class Template Reference

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

Collaboration diagram for ConstBArrayRowIter< Cell_Type, Data_Type >:



Public Member Functions

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

Public Attributes

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

7.10.1 Detailed Description

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

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

7.10.2 Constructor & Destructor Documentation

7.10.2.1 ConstBArrayRowIter()

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

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

7.10.2.2 ~ConstBArrayRowIter()

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

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

7.10.3 Member Data Documentation

7.10.3.1 Array

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

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

7.10.3.2 current_col

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

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

7.10.3.3 current_row

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

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

7.10.3.4 iter

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

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

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

- include/barry/barray-iterator.hpp

7.11 Counter< 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_=nullptr, Data_Type *data_=nullptr, bool delete_data_=false, std::string name_="", std::string desc_="")
- Counter (const Counter< Array_Type, Data_Type > &counter_)
  Copy constructor.
- Counter (Counter< Array_Type, Data_Type > &&counter_) noexcept
  Move constructor.
- Counter< Array_Type, Data_Type > operator= (const Counter< Array_Type, Data_Type > &counter_)
  Copy assignment.
- Counter< Array_Type, Data_Type > & operator= (Counter< Array_Type, Data_Type > &&counter_) noexcept
  Move assignment.

Public Attributes

- Counter_fun_type< Array_Type, Data_Type > count_fun
- Counter_fun_type< Array_Type, Data_Type > init_fun
- Data_Type * data = nullptr
- bool delete_data = false
- std::string name = ""
- std::string desc = ""

7.11.1 Detailed Description

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

A counter function based on change statistics.

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

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

7.11.2 Constructor & Destructor Documentation

7.11.2.1 Counter() [1/4]

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

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

7.11.2.2 Counter() [2/4]

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

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

7.11.2.3 Counter() [3/4]

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

Copy constructor.

7.11.2.4 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.

7.11.2.5 ~Counter()

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

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

7.11.3 Member Function Documentation

7.11.3.1 count()

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

7.11.3.2 get_description()

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

7.11.3.3 get_name()

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

7.11.3.4 init()

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

7.11.3.5 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.

7.11.3.6 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.

7.11.4 Member Data Documentation

7.11.4.1 count_fun

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

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

7.11.4.2 data

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

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

7.11.4.3 delete_data

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

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

7.11.4.4 desc

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

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

7.11.4.5 init_fun

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

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

7.11.4.6 name

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

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

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

- include/barry/counters-bones.hpp

7.12 Counters< Array_Type, Data_Type > Class Template Reference

Vector of counters.

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

Public Member Functions

- Counters ()
- ~Counters ()
- Counters (const Counters< Array_Type, Data_Type > &counter_)

  Copy constructor.
- Counters (Counters< Array_Type, Data_Type > &&counters_) noexcept

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

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

  Move assignment constructor.
- Counter< Array_Type, Data_Type > & operator[ ] (uint idx)

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

  Number of counters in the set.
- void add_counter (Counter< Array_Type, Data_Type > &counter)
- void add_counter (Counter< Array_Type, Data_Type > *counter)
- void add_counter (Counter_fun_type< Array_Type, Data_Type > count_fun_, Counter_fun_type< Array_Type, Data_Type > init_fun_=nullptr, Data_Type *data_=nullptr, bool delete_data_=false, std::string name_="", std::string desc_="")
- void clear ()
- std::vector< std::string > get_names () const
- std::vector< std::string > get_descriptions () const

7.12.1 Detailed Description

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

Vector of counters.

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

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

7.12.2 Constructor & Destructor Documentation

7.12.2.1 Counters() [1/3]

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

7.12.2.2 ~Counters()

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

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

7.12.2.3 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_*	

7.12.2.4 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_*	

7.12.3 Member Function Documentation

7.12.3.1 add_counter() [1/3]

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

7.12.3.2 add_counter() [2/3]

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

7.12.3.3 add_counter() [3/3]

```
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_ = nullptr,
            Data_Type * data_ = nullptr,
            bool delete_data_ = false,
            std::string name_ = "",
            std::string desc_ = "" )
```

7.12.3.4 clear()

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

7.12.3.5 get_descriptions()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::vector< std::string > Counters< Array_Type, Data_Type >::get_descriptions ( ) const
```

7.12.3.6 get_names()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::vector< std::string > Counters< Array_Type, Data_Type >::get_names ( ) const
```

7.12.3.7 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>

7.12.3.8 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>&

7.12.3.9 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>*

7.12.3.10 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 161 of file counters-bones.hpp.

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

- include/barry/counters-bones.hpp

7.13 Entries< Cell_Type > Class Template Reference

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

`#include <typedefs.hpp>`

Public Member Functions

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

Public Attributes

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

7.13.1 Detailed Description

template<typename Cell_Type>
class Entries< Cell_Type >

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

Template Parameters

Cell_Type	Any type

Definition at line 65 of file typedefs.hpp.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 Entries() [1/2]

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

Definition at line 71 of file typedefs.hpp.

7.13.2.2 Entries() [2/2]

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

Definition at line 72 of file typedefs.hpp.

7.13.2.3 ~Entries()

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

Definition at line 79 of file typedefs.hpp.

7.13.3 Member Function Documentation

7.13.3.1 resize()

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

Definition at line 81 of file typedefs.hpp.

7.13.4 Member Data Documentation

7.13.4.1 source

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

Definition at line 67 of file typedefs.hpp.

7.13.4.2 target

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

Definition at line 68 of file typedefs.hpp.

7.13.4.3 val

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

Definition at line 69 of file typedefs.hpp.

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

- include/barry/typedefs.hpp

7.14 Flock Class Reference

A Flock is a group of Geese.

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

Public Member Functions

- Flock ()
- ~Flock ()
- unsigned int add_data (std::vector< std::vector< unsigned int > > &annotations, std::vector< unsigned int > &geneid, std::vector< int > &parent, std::vector< bool > &duplication)

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

  Set the seed of the model.
- void init (unsigned int bar_width=BARRY_PROGRESS_BAR_WIDTH)
- phylocounters::PhyloCounters * get_counters ()
- phylocounters::PhyloSupport * get_support ()
- 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) const noexcept
- void print () const

Public Attributes

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

7.14.1 Detailed Description

A Flock is a group of Geese.

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

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

7.14.2 Constructor & Destructor Documentation

7.14.2.1 Flock()

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

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

7.14.2.2 ~Flock()

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

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

7.14.3 Member Function Documentation

7.14.3.1 add_data()

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

Add a tree to the flock.

Parameters

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.

7.14.3.2 colnames()

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

Definition at line 176 of file flock-meat.hpp.

7.14.3.3 get_counters()

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

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

7.14.3.4 get_model()

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

Definition at line 99 of file flock-meat.hpp.

7.14.3.5 get_support()

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

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

7.14.3.6 init()

```
void Flock::init (
            unsigned int bar_width = BARRY_PROGRESS_BAR_WIDTH )  [inline]
```

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

7.14.3.7 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 103 of file flock-meat.hpp.

7.14.3.8 nfuns()

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

Definition at line 128 of file flock-meat.hpp.

7.14.3.9 nleafs()

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

Definition at line 151 of file flock-meat.hpp.

7.14.3.10 nnodes()

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

Definition at line 140 of file flock-meat.hpp.

7.14.3.11 nterms()

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

Definition at line 163 of file flock-meat.hpp.

7.14.3.12 ntrees()

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

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

7.14.3.13 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 234 of file flock-meat.hpp.

7.14.3.14 parse_polytomies()

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

Definition at line 182 of file flock-meat.hpp.

7.14.3.15 print()

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

Definition at line 201 of file flock-meat.hpp.

7.14.3.16 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 37 of file flock-meat.hpp.

7.14.3.17 support_size()

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

Definition at line 170 of file flock-meat.hpp.

7.14.4 Member Data Documentation

7.14.4.1 dat

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

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

7.14.4.2 initialized

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

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

7.14.4.3 model

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

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

7.14.4.4 nfunctions

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

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

7.14.4.5 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

7.15 FreqTable< T > Class Template Reference

Database of statistics.

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

Public Member Functions

- FreqTable ()
- ~FreqTable ()
- void add (const std::vector< T > &x)
- Counts_type as_vector () const
- MapVec_type< T, uint > get_data () const
- const MapVec_type< T, uint > * get_data_ptr () const
- void clear ()
- void reserve (unsigned int n)
- void print () const
- size_t size () const noexcept

7.15.1 Detailed Description

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

Database of statistics.

This is mostly used in `Support`.

Definition at line 16 of file statsdb.hpp.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 FreqTable()

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

Definition at line 28 of file statsdb.hpp.

7.15.2.2 ~FreqTable()

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

Definition at line 29 of file statsdb.hpp.

7.15.3 Member Function Documentation

7.15.3.1 add()

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

Definition at line 47 of file statsdb.hpp.

7.15.3.2 as_vector()

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

Definition at line 61 of file statsdb.hpp.

7.15.3.3 clear()

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

Definition at line 83 of file statsdb.hpp.

7.15.3.4 get_data()

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

Definition at line 73 of file statsdb.hpp.

7.15.3.5 get_data_ptr()

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

Definition at line 78 of file statsdb.hpp.

7.15.3.6 print()

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

Definition at line 102 of file statsdb.hpp.

7.15.3.7 reserve()

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

Definition at line 89 of file statsdb.hpp.

7.15.3.8 size()

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

Definition at line 126 of file statsdb.hpp.

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

- include/barry/statsdb.hpp

7.16 Geese Class Reference

Annotated 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) 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()` returns the computed support of the model.

- std::mt19937 * get_rengine ()
- phylocounters::PhyloCounters * get_counters ()
- phylocounters::PhyloModel * get_model ()
- phylocounters::PhyloSupport * get_support ()

Public Attributes

- unsigned int nfunctions
- std::map< unsigned int, Node > nodes
- barry::MapVec_type< unsigned int > map_to_nodes
- std::vector< unsigned int > sequence
- std::vector< unsigned int > reduced_sequence
- bool initialized = false
- bool delete_rengine = false
- bool delete_support = false

7.16.1 Detailed Description

Annotated Phylo Model.

A list of available terms for this model can be found in the Phylo counters section.

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

7.16.2 Constructor & Destructor Documentation

7.16.2.1 Geese() [1/4]

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

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

7.16.2.2 Geese() [2/4]

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

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

7.16.2.3 Geese() [3/4]

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

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

7.16.2.4 Geese() [4/4]

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

Definition at line 292 of file geese-meat-constructors.hpp.

7.16.2.5 ~Geese()

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

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

7.16.3 Member Function Documentation

7.16.3.1 calc_reduced_sequence()

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

Definition at line 305 of file geese-meat.hpp.

7.16.3.2 calc_sequence()

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

Definition at line 266 of file geese-meat.hpp.

7.16.3.3 colnames()

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

Names of the terms in the model.

Definition at line 406 of file geese-meat.hpp.

7.16.3.4 get_annotated_nodes()

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

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

Definition at line 587 of file geese-meat.hpp.

7.16.3.5 get_counters()

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

Definition at line 570 of file geese-meat.hpp.

7.16.3.6 get_model()

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

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

7.16.3.7 get_probabilities()

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

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

7.16.3.8 get_rengine()

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

Definition at line 565 of file geese-meat.hpp.

7.16.3.9 get_states()

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

Powerset of a gene's possible states.

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

Returns

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

Definition at line 583 of file geese-meat.hpp.

7.16.3.10 get_support()

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

Definition at line 579 of file geese-meat.hpp.

7.16.3.11 inherit_support()

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

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

7.16.3.12 init()

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

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

7.16.3.13 init_node()

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

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

7.16.3.14 likelihood()

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

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

7.16.3.15 likelihood_exhaust()

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

Definition at line 7 of file geese-meat-likelihood_exhaust.hpp.

7.16.3.16 nannotations()

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

Number of annotations.

Definition at line 400 of file geese-meat.hpp.

7.16.3.17 nfuns()

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

Number of functions analyzed.

Definition at line 364 of file geese-meat.hpp.

7.16.3.18 nleafs()

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

Number of leaf.

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

7.16.3.19 nnodes()

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

Number of nodes (interior + leaf)

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

7.16.3.20 nterms()

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

Number of terms included.

Definition at line 382 of file geese-meat.hpp.

7.16.3.21 observed_counts()

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

Definition at line 437 of file geese-meat.hpp.

7.16.3.22 operator=() [1/2]

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

7.16.3.23 operator=() [2/2]

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

7.16.3.24 parse_polytomies()

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

Check polytomies and return the largest.

Definition at line 413 of file geese-meat.hpp.

7.16.3.25 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 240 of file geese-meat-predict.hpp.

7.16.3.26 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.

7.16.3.27 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.

7.16.3.28 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.

7.16.3.29 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.

7.16.3.30 print()

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

Prints information about the GEESE.

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

7.16.3.31 print_observed_counts()

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

Definition at line 485 of file geese-meat.hpp.

7.16.3.32 set_seed()

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

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

7.16.3.33 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.

7.16.3.34 support_size()

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

Number of unique sets of sufficient stats.

Definition at line 390 of file geese-meat.hpp.

7.16.3.35 update_annotations()

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

Definition at line 237 of file geese-meat.hpp.

7.16.4 Member Data Documentation

7.16.4.1 delete_rengine

```
bool Geese::delete_rengine = false
```

Definition at line 117 of file geese-bones.hpp.

7.16.4.2 delete_support

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

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

7.16.4.3 initialized

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

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

7.16.4.4 map_to_nodes

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

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

7.16.4.5 nfunctions

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

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

7.16.4.6 nodes

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

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

7.16.4.7 reduced_sequence

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

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

7.16.4.8 sequence

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

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

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

- include/barry/models/geese/geese-bones.hpp
- include/barry/models/geese/geese-meat-constructors.hpp
- include/barry/models/geese/geese-meat-likelihood.hpp
- include/barry/models/geese/geese-meat-likelihood_exhaust.hpp
- include/barry/models/geese/geese-meat-predict.hpp
- include/barry/models/geese/geese-meat-predict_exhaust.hpp
- include/barry/models/geese/geese-meat-predict_sim.hpp
- include/barry/models/geese/geese-meat-simulate.hpp
- include/barry/models/geese/geese-meat.hpp

7.17 Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference

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

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

Public Member Functions

- void set_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
- void set_keygen (std::function< std::vector< double >(const Array_Type &)> keygen_)
- 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 ()

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

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

- void add_counter (Counter< Array_Type, Data_Counter_Type > &counter)
- void add_counter (Counter< Array_Type, Data_Counter_Type > *counter)
- void add_counter (Counter_fun_type< Array_Type, Data_Counter_Type > count_fun_, Counter_fun_type< Array_Type, Data_Counter_Type > init_fun_=nullptr, Data_Counter_Type *data_=nullptr, bool delete_data_=false)
- void set_counters (Counters< Array_Type, Data_Counter_Type > *counters_)

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

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

- void add_rule (Rule< Array_Type, Data_Rule_Type > &rule)
- void add_rule (Rule< Array_Type, Data_Rule_Type > *rule)
- void add_rule (Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_, Data_Rule_Type *data_=nullptr, bool delete_data_=false)
- void set_rules (Rules< Array_Type, Data_Rule_Type > *rules_)
- void add_rule_dyn (Rule< Array_Type, Data_Rule_Dyn_Type > &rule)
- void add_rule_dyn (Rule< Array_Type, Data_Rule_Dyn_Type > *rule)
- void add_rule_dyn (Rule_fun_type< Array_Type, Data_Rule_Dyn_Type > count_fun_, Data_Rule_Dyn↩ _Type *data_=nullptr, bool delete_data_=false)
- void set_rules_dyn (Rules< Array_Type, Data_Rule_Dyn_Type > *rules_)

Likelihood functions.

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

Parameters

params	*Vector of parameters*
as_log	*When `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_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< 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`.

Returns

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

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

7.17.1 Detailed Description

template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
class Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >

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

$$\frac{\exp\left(\theta^{\mathsf{t}} c(A)\right)}{\sum_{A' \in \mathcal{A}} \exp\left(\theta^{\mathsf{t}} c(A')\right)}$$

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 47 of file model-bones.hpp.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 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 ( )
```

7.17.2.2 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_ )
```

7.17.2.3 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_ )
```

7.17.2.4 ~Model()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::~Model ( )  [inline]
```

Definition at line 149 of file model-bones.hpp.

7.17.3 Member Function Documentation

7.17.3.1 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.

7.17.3.2 add_counter() [1/3]

```
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 )
```

7.17.3.3 add_counter() [2/3]

```
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 )
```

7.17.3.4 add_counter() [3/3]

```
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,
            bool delete_data_ = false )
```

7.17.3.5 add_rule() [1/3]

```
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 )
```

7.17.3.6 add_rule() [2/3]

```
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 )
```

7.17.3.7 add_rule() [3/3]

```
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_ = nullptr,
            bool delete_data_ = false )
```

7.17.3.8 add_rule_dyn() [1/3]

```
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 )
```

7.17.3.9 add_rule_dyn() [2/3]

```
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 )
```

7.17.3.10 add_rule_dyn() [3/3]

```
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_ = nullptr,
            bool delete_data_ = false )
```

7.17.3.11 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
```

7.17.3.12 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

7.17.3.13 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_ )
```

7.17.3.14 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 ( )
```

7.17.3.15 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 )
```

7.17.3.16 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 )
```

7.17.3.17 get_pset_stats()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
const std::vector< std::vector< double > >* Model< Array_Type, Data_Counter_Type, Data_Rule↩
_Type, Data_Rule_Dyn_Type >::get_pset_stats (
            const uint & i )
```

7.17.3.18 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
```

7.17.3.19 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 ( )
```

7.17.3.20 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 ( )
```

7.17.3.21 get_support()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support<Array_Type,Data_Counter_Type,Data_Rule_Type,Data_Rule_Dyn_Type>* Model< Array_Type,
Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_support ( )
```

7.17.3.22 likelihood() [1/3]

```
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 )
```

7.17.3.23 likelihood() [2/3]

```
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 )
```

7.17.3.24 likelihood() [3/3]

```
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 )
```

7.17.3.25 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 )
```

7.17.3.26 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]
```

7.17.3.27 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_ )
```

7.17.3.28 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.

7.17.3.29 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
```

7.17.3.30 sample() [1/2]

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

7.17.3.31 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 )
```

7.17.3.32 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_ )
```

7.17.3.33 set_keygen()

```
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_keygen (
            std::function< std::vector< double >(const Array_Type &)> keygen_ )
```

7.17.3.34 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 119 of file model-bones.hpp.

7.17.3.35 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_ )
```

7.17.3.36 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_ )
```

7.17.3.37 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 129 of file model-bones.hpp.

7.17.3.38 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]
```

7.17.3.39 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]
```

7.17.3.40 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]
```

7.17.3.41 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]
```

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

- include/barry/model-bones.hpp

7.18 NetCounterData Class Reference

Data class used to store arbitrary uint or double vectors.

`#include <network.hpp>`

Public Member Functions

- NetCounterData ()
- NetCounterData (const std::vector< uint > indices_, const std::vector< double > numbers_)
- ~NetCounterData ()

Public Attributes

- std::vector< uint > indices
- std::vector< double > numbers

7.18.1 Detailed Description

Data class used to store arbitrary uint or double vectors.

Definition at line 56 of file network.hpp.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 NetCounterData() [1/2]

```
NetCounterData::NetCounterData ( )  [inline]
```

Definition at line 62 of file network.hpp.

7.18.2.2 NetCounterData() [2/2]

```
NetCounterData::NetCounterData (
            const std::vector< uint > indices_,
            const std::vector< double > numbers_ )  [inline]
```

Definition at line 63 of file network.hpp.

7.18.2.3 ~NetCounterData()

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

Definition at line 68 of file network.hpp.

7.18.3 Member Data Documentation

7.18.3.1 indices

```
std::vector< uint > NetCounterData::indices
```

Definition at line 59 of file network.hpp.

7.18.3.2 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

7.19 NetworkData Class Reference

Data class for Networks.

```
#include <network.hpp>
```

Public Member Functions

- NetworkData ()
- NetworkData (std::vector< double > vertex_attr_, bool directed_=true)

  Constructor using a single attribute.
- NetworkData (std::vector< std::vector< double > > vertex_attr_, bool directed_=true)

  Constructor using multiple attributes.
- ~NetworkData ()

Public Attributes

- bool directed = true
- std::vector< std::vector< double > > vertex_attr

7.19.1 Detailed Description

Data class for Networks.

Details on the available counters for `NetworkData` can be found in the Network counters section.

This holds information about whether the graph is directed or not, and, if defined, vectors of node (vertex) attributes (`vertex_attr`).

Definition at line 19 of file network.hpp.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 NetworkData() [1/3]

```
NetworkData::NetworkData ( )  [inline]
```

Definition at line 25 of file network.hpp.

7.19.2.2 NetworkData() [2/3]

```
NetworkData::NetworkData (
            std::vector< double > vertex_attr_,
            bool directed_ = true )  [inline]
```

Constructor using a single attribute.

Parameters

*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.

7.19.2.3 NetworkData() [3/3]

```
NetworkData::NetworkData (
            std::vector< std::vector< double > > vertex_attr_,
            bool directed_ = true )  [inline]
```

Constructor using multiple attributes.

Parameters

*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.

7.19.2.4 ∼NetworkData()

```
NetworkData::∼NetworkData ( )  [inline]
```

Definition at line 51 of file network.hpp.

7.19.3 Member Data Documentation

7.19.3.1 directed

```
bool NetworkData::directed = true
```

Definition at line 22 of file network.hpp.

7.19.3.2 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:

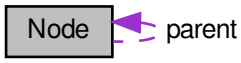
- include/barry/counters/network.hpp

7.20 Node Class Reference

A single node for the model.

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

Collaboration diagram for Node:



Public Member Functions

- ~Node ()
- int get_parent () const
- unsigned int noffspring () const noexcept
- bool is_leaf () const noexcept

Construct a new Node object

- Node ()
- Node (unsigned int id_, unsigned int ord_, bool duplication_)
- Node (unsigned int id_, unsigned int ord_, std::vector< unsigned int > annotations_, bool duplication_)
- Node (Node &&x) noexcept
- Node (const Node &x)

Public Attributes

- unsigned int id

  Id of the node (as specified in the input)
- unsigned int ord

  Order in which the node was created.
- phylocounters::PhyloArray array
- std::vector< unsigned int > annotations

  Observed annotations (only defined for Geese)
- bool duplication
- std::vector< phylocounters::PhyloArray > arrays = {}

  Arrays given all possible states.
- Node * parent = nullptr

  Parent node.
- std::vector< Node * > offspring = {}

  Offspring nodes.
- std::vector< unsigned int > narray = {}

  ID of the array in the model.
- bool visited = false
- std::vector< double > subtree_prob

  Induced subtree probabilities.
- std::vector< double > probability

  The probability of observing each state.

7.20.1 Detailed Description

A single node for the model.

Each node contains all the information to compute the conditional probability of the pruning algorithm at that node.

Definition at line 11 of file geese-node-bones.hpp.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 Node() [1/5]

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

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

7.20.2.2 Node() [2/5]

```
Node::Node (
            unsigned int id_,
            unsigned int ord_,
            bool duplication_ )  [inline]
```

Definition at line 56 of file geese-node-bones.hpp.

7.20.2.3 Node() [3/5]

```
Node::Node (
            unsigned int id_,
            unsigned int ord_,
            std::vector< unsigned int > annotations_,
            bool duplication_ )  [inline]
```

Definition at line 62 of file geese-node-bones.hpp.

7.20.2.4 Node() [4/5]

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

Definition at line 69 of file geese-node-bones.hpp.

7.20.2.5 Node() [5/5]

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

Definition at line 83 of file geese-node-bones.hpp.

7.20.2.6 ~Node()

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

Definition at line 47 of file geese-node-bones.hpp.

7.20.3 Member Function Documentation

7.20.3.1 get_parent()

```
int Node::get_parent ( ) const  [inline]
```

Definition at line 97 of file geese-node-bones.hpp.

7.20.3.2 is_leaf()

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

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

7.20.3.3 noffspring()

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

Definition at line 103 of file geese-node-bones.hpp.

7.20.4 Member Data Documentation

7.20.4.1 annotations

```
std::vector< unsigned int > Node::annotations
```

Observed annotations (only defined for Geese)

Definition at line 18 of file geese-node-bones.hpp.

7.20.4.2 array

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

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

7.20.4.3 arrays

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

Arrays given all possible states.

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

7.20.4.4 duplication

```
bool Node::duplication
```

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

7.20.4.5 id

```
unsigned int Node::id
```

Id of the node (as specified in the input)

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

7.20.4.6 narray

```
std::vector< unsigned int > Node::narray = {}
```

ID of the array in the model.

Definition at line 24 of file geese-node-bones.hpp.

7.20.4.7 offspring

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

Offspring nodes.

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

7.20.4.8 ord

```
unsigned int Node::ord
```

Order in which the node was created.

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

7.20.4.9 parent

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

Parent node.

Definition at line 22 of file geese-node-bones.hpp.

7.20.4.10 probability

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

The probability of observing each state.

Definition at line 28 of file geese-node-bones.hpp.

7.20.4.11 subtree_prob

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

Induced subtree probabilities.

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

7.20.4.12 visited

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

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

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

- include/barry/models/geese/geese-node-bones.hpp

7.21 NodeData Class Reference

Data definition for the `PhyloArray` class.

`#include <phylo.hpp>`

Public Member Functions

- NodeData (const std::vector< double > &blengths_, const std::vector< bool > &states_, bool duplication_=true)

Public Attributes

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

7.21.1 Detailed Description

Data definition for the `PhyloArray` class.

Details about the available counters for `PhyloArray` objects can be found in the Phylo counters section.

This holds basic information about a given node.

Definition at line 38 of file phylo.hpp.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 NodeData()

```
NodeData::NodeData (
            const std::vector< double > & blengths_,
            const std::vector< bool > & states_,
            bool duplication_ = true )  [inline]
```

Definition at line 58 of file phylo.hpp.

7.21.3 Member Data Documentation

7.21.3.1 blengths

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

Branch length.

Definition at line 44 of file phylo.hpp.

7.21.3.2 duplication

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

Definition at line 54 of file phylo.hpp.

7.21.3.3 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

7.22 PhyloCounterData Class Reference

```
#include <phylo.hpp>
```

Public Member Functions

- PhyloCounterData (std::vector< uint > data_, std::vector< double > *counters_=nullptr)
- uint at (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 ()

7.22.1 Detailed Description

Definition at line 69 of file phylo.hpp.

7.22.2 Constructor & Destructor Documentation

7.22.2.1 PhyloCounterData()

```
PhyloCounterData::PhyloCounterData (
            std::vector< uint > data_,
            std::vector< double > * counters_ = nullptr )  [inline]
```

Definition at line 75 of file phylo.hpp.

7.22.3 Member Function Documentation

7.22.3.1 at()

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

Definition at line 80 of file phylo.hpp.

7.22.3.2 begin()

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

Definition at line 87 of file phylo.hpp.

7.22.3.3 empty()

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

Definition at line 90 of file phylo.hpp.

7.22.3.4 end()

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

Definition at line 88 of file phylo.hpp.

7.22.3.5 get_counters()

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

Definition at line 91 of file phylo.hpp.

7.22.3.6 operator()()

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

Definition at line 81 of file phylo.hpp.

7.22.3.7 push_back()

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

Definition at line 83 of file phylo.hpp.

7.22.3.8 reserve()

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

Definition at line 82 of file phylo.hpp.

7.22.3.9 shrink_to_fit()

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

Definition at line 84 of file phylo.hpp.

7.22.3.10 size()

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

Definition at line 85 of file phylo.hpp.

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

- include/barry/counters/phylo.hpp

7.23 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

7.23.1 Detailed Description

Definition at line 1414 of file phylo.hpp.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 PhyloRuleDynData()

```
PhyloRuleDynData::PhyloRuleDynData (
            const std::vector< double > * counts_,
            uint pos_,
            uint lb_,
            uint ub_,
            uint duplication_ )  [inline]
```

Definition at line 1421 of file phylo.hpp.

7.23.2.2 ~PhyloRuleDynData()

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

Definition at line 1430 of file phylo.hpp.

7.23.3 Member Data Documentation

7.23.3.1 counts

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

Definition at line 1416 of file phylo.hpp.

7.23.3.2 duplication

```
uint PhyloRuleDynData::duplication
```

Definition at line 1420 of file phylo.hpp.

7.23.3.3 lb

```
uint PhyloRuleDynData::lb
```

Definition at line 1418 of file phylo.hpp.

7.23.3.4 pos

```
uint PhyloRuleDynData::pos
```

Definition at line 1417 of file phylo.hpp.

7.23.3.5 ub

```
uint PhyloRuleDynData::ub
```

Definition at line 1419 of file phylo.hpp.

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

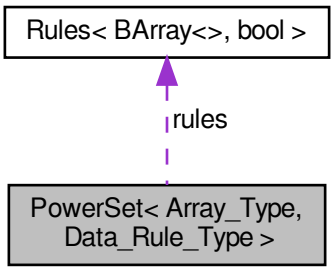
- include/barry/counters/phylo.hpp

7.24 PowerSet< Array_Type, Data_Rule_Type > Class Template Reference

Powerset of a binary array.

`#include <powerset-bones.hpp>`

Collaboration diagram for PowerSet< Array_Type, Data_Rule_Type >:



Public Member Functions

- void init_support ()
- void calc ()
- void reset (uint N_, uint M_)

Construct and destroy a PowerSet object

- PowerSet ()
- PowerSet (uint N_, uint M_)
- PowerSet (const Array_Type &array)
- ~PowerSet ()

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

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

- void add_rule (Rule< Array_Type, Data_Rule_Type > &rule)
- void add_rule (Rule< Array_Type, Data_Rule_Type > *rule)
- void add_rule (Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_, Data_Rule_Type *data_=nullptr, bool delete_data_=false)

Getter functions

- const std::vector< Array_Type > * get_data_ptr () const
- std::vector< Array_Type > get_data () const
- std::vector< Array_Type >::iterator begin ()
- std::vector< Array_Type >::iterator end ()
- std::size_t size () const noexcept
- const Array_Type & operator[] (const unsigned int &i) const

Public Attributes

- Array_Type EmptyArray
- std::vector< Array_Type > data
- Rules< Array_Type, Data_Rule_Type > * rules
- uint N
- uint M
- bool rules_deleted = false
- std::vector< std::pair< uint, uint > > coordinates_free
- std::vector< std::pair< uint, uint > > coordinates_locked

7.24.1 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 17 of file powerset-bones.hpp.

7.24.2 Constructor & Destructor Documentation

7.24.2.1 PowerSet() [1/3]

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

Definition at line 39 of file powerset-bones.hpp.

7.24.2.2 PowerSet() [2/3]

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
PowerSet< Array_Type, Data_Rule_Type >::PowerSet (
            uint N_,
            uint M_ )  [inline]
```

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

7.24.2.3 PowerSet() [3/3]

```
template<typename Array_Type , typename Data_Rule_Type >
PowerSet< Array_Type, Data_Rule_Type >::PowerSet (
            const Array_Type & array )  [inline]
```

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

7.24.2.4 ~PowerSet()

```
template<typename Array_Type , typename Data_Rule_Type >
PowerSet< Array_Type, Data_Rule_Type >::~PowerSet  [inline]
```

Definition at line 15 of file powerset-meat.hpp.

7.24.3 Member Function Documentation

7.24.3.1 add_rule() [1/3]

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > & rule )  [inline]
```

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

7.24.3.2 add_rule() [2/3]

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > * rule )  [inline]
```

Definition at line 122 of file powerset-meat.hpp.

7.24.3.3 add_rule() [3/3]

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::add_rule (
            Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_,
            Data_Rule_Type * data_ = nullptr,
            bool delete_data_ = false )  [inline]
```

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

7.24.3.4 begin()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type >::iterator PowerSet< Array_Type, Data_Rule_Type >::begin ( )  [inline]
```

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

7.24.3.5 calc()

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::calc  [inline]
```

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

7.24.3.6 end()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type >::iterator PowerSet< Array_Type, Data_Rule_Type >::end ( )  [inline]
```

Definition at line 74 of file powerset-bones.hpp.

7.24.3.7 get_data()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type > PowerSet< Array_Type, Data_Rule_Type >::get_data ( ) const  [inline]
```

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

7.24.3.8 get_data_ptr()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
const std::vector< Array_Type >* PowerSet< Array_Type, Data_Rule_Type >::get_data_ptr ( )
const  [inline]
```

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

7.24.3.9 init_support()

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::init_support  [inline]
```

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

7.24.3.10 operator[]()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
const Array_Type& PowerSet< Array_Type, Data_Rule_Type >::operator[] (
            const unsigned int & i ) const  [inline]
```

Definition at line 76 of file powerset-bones.hpp.

7.24.3.11 reset()

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::reset (
            uint N_,
            uint M_ )  [inline]
```

Definition at line 101 of file powerset-meat.hpp.

7.24.3.12 size()

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

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

7.24.4 Member Data Documentation

7.24.4.1 coordinates_free

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< std::pair<uint,uint> > PowerSet< Array_Type, Data_Rule_Type >::coordinates_free
```

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

7.24.4.2 coordinates_locked

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< std::pair<uint,uint> > PowerSet< Array_Type, Data_Rule_Type >::coordinates_↩
locked
```

Definition at line 32 of file powerset-bones.hpp.

7.24.4.3 data

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type > PowerSet< Array_Type, Data_Rule_Type >::data
```

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

7.24.4.4 EmptyArray

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
Array_Type PowerSet< Array_Type, Data_Rule_Type >::EmptyArray
```

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

7.24.4.5 M

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
uint PowerSet< Array_Type, Data_Rule_Type >::M
```

Definition at line 27 of file powerset-bones.hpp.

7.24.4.6 N

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
uint PowerSet< Array_Type, Data_Rule_Type >::N
```

Definition at line 27 of file powerset-bones.hpp.

7.24.4.7 rules

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
Rules<Array_Type,Data_Rule_Type>* PowerSet< Array_Type, Data_Rule_Type >::rules
```

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

7.24.4.8 rules_deleted

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
bool PowerSet< Array_Type, Data_Rule_Type >::rules_deleted = false
```

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

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

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

7.25 Progress Class Reference

A simple progress bar.

```
#include <progress.hpp>
```

Public Member Functions

- Progress (int n_, int width_)
- ~Progress ()
- void next ()
- void end ()

7.25.1 Detailed Description

A simple progress bar.

Definition at line 11 of file progress.hpp.

7.25.2 Constructor & Destructor Documentation

7.25.2.1 Progress()

```
Progress::Progress (
            int n_,
            int width_ )  [inline]
```

Definition at line 30 of file progress.hpp.

7.25.2.2 ~Progress()

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

Definition at line 23 of file progress.hpp.

7.25.3 Member Function Documentation

7.25.3.1 end()

```
void Progress::end ( )  [inline]
```

Definition at line 52 of file progress.hpp.

7.25.3.2 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

7.26 Rule< Array_Type, Data_Type > Class Template Reference

Rule for determining if a cell should be included in a sequence.

`#include <rules-bones.hpp>`

Public Member Functions

- ~Rule ()
- Data_Type * D ()

  Read/Write access to the data.
- bool operator() (const Array_Type &a, uint i, uint j)

Construct a new Rule object

Construct a new Rule object

Parameters

fun_	*A function of type* `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_=nullptr, bool delete_dat_=false)

7.26.1 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 22 of file rules-bones.hpp.

7.26.2 Constructor & Destructor Documentation

7.26.2.1 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.

7.26.2.2 Rule() [2/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Rule< Array_Type, Data_Type >::Rule (
            Rule_fun_type< Array_Type, Data_Type > fun_,
            Data_Type * dat_ = nullptr,
            bool delete_dat_ = false )  [inline]
```

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

7.26.2.3 ~Rule()

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

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

7.26.3 Member Function Documentation

7.26.3.1 D()

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

Read/Write access to the data.

7.26.3.2 operator()()

```
template<typename Array_Type , typename Data_Type >
bool Rule< Array_Type, Data_Type >::operator() (
            const Array_Type & a,
            uint i,
            uint j )  [inline]
```

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

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

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

7.27 Rules< Array_Type, Data_Type > Class Template Reference

Vector of objects of class Rule.

`#include <rules-bones.hpp>`

Public Member Functions

- Rules ()
- Rules (const Rules< Array_Type, Data_Type > &rules_)
- Rules< Array_Type, Data_Type > operator= (const Rules< Array_Type, Data_Type > &rules_)
- ~Rules ()
- uint size () const noexcept
- bool operator() (const Array_Type &a, uint i, uint j)

  Check whether a given cell is free or locked.
- void clear ()
- void get_seq (const Array_Type &a, std::vector< std::pair< uint, uint > > *free, std::vector< std::pair< uint, uint > > *locked=nullptr)

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

Rule adding

Parameters

rule	

- void add_rule (Rule< Array_Type, Data_Type > &rule)
- void add_rule (Rule< Array_Type, Data_Type > *rule)
- void add_rule (Rule_fun_type< Array_Type, Data_Type > rule_, Data_Type *data_=nullptr, bool delete_data_=false)

7.27.1 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 68 of file rules-bones.hpp.

7.27.2 Constructor & Destructor Documentation

7.27.2.1 Rules() [1/2]

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::Rules ( )  [inline]
```

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

7.27.2.2 Rules() [2/2]

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

Definition at line 10 of file rules-meat.hpp.

7.27.2.3 ~Rules()

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::~Rules ( )  [inline]
```

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

7.27.3 Member Function Documentation

7.27.3.1 add_rule() [1/3]

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::add_rule (
            Rule< Array_Type, Data_Type > & rule )  [inline]
```

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

7.27.3.2 add_rule() [2/3]

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::add_rule (
            Rule< Array_Type, Data_Type > * rule )  [inline]
```

Definition at line 79 of file rules-meat.hpp.

7.27.3.3 add_rule() [3/3]

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::add_rule (
            Rule_fun_type< Array_Type, Data_Type > rule_,
            Data_Type * data_ = nullptr,
            bool delete_data_ = false )  [inline]
```

Definition at line 89 of file rules-meat.hpp.

7.27.3.4 clear()

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

Definition at line 127 of file rules-meat.hpp.

7.27.3.5 get_seq()

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::get_seq (
            const Array_Type & a,
            std::vector< std::pair< uint, uint > > * free,
            std::vector< std::pair< uint, uint > > * locked = nullptr )  [inline]
```

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

Parameters

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 139 of file rules-meat.hpp.

7.27.3.6 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 111 of file rules-meat.hpp.

7.27.3.7 operator=()

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type > Rules< Array_Type, Data_Type >::operator= (
            const Rules< Array_Type, Data_Type > & rules_ )
```

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

7.27.3.8 size()

```
template<typename Array_Type , typename Data_Type >
uint Rules< Array_Type, Data_Type >::size ( ) const  [inline], [noexcept]
```

Definition at line 85 of file rules-bones.hpp.

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

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

7.28 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 ()
  Can be created without setting the array.
- ~StatsCounter ()
- void reset_array (const Array_Type *Array_)
  Changes the reference array for the counting.
- void add_counter (Counter< Array_Type, Data_Type > *f_)
- void add_counter (Counter< Array_Type, Data_Type > f_)
- void set_counters (Counters< Array_Type, Data_Type > *counters_)
- void count_init (uint i, uint j)
  Counter functions This function recurses through the entries of `Array` and at each step of adding a new cell it uses the functions to list the statistics.
- void count_current (uint i, uint j)
- std::vector< double > count_all ()
- Counters< Array_Type, Data_Type > * get_counters ()
- std::vector< std::string > get_names () const
- std::vector< std::string > get_descriptions () const

7.28.1 Detailed Description

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

Count stats for a single Array.

Users can a list of functions that can be used with this. The baseline set of arguments is a pointer to a binary array and a dataset to add the counts to.

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

7.28.2 Constructor & Destructor Documentation

7.28.2.1 StatsCounter() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
StatsCounter< Array_Type, Data_Type >::StatsCounter (
            const Array_Type * Array_ )  [inline]
```

Creator of a StatsCounter

Parameters

*Array_*	A const pointer to a BArray.

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

7.28.2.2 StatsCounter() [2/2]

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

Can be created without setting the array.

Definition at line 51 of file statscounter-bones.hpp.

7.28.2.3 ~StatsCounter()

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

7.28.3 Member Function Documentation

7.28.3.1 add_counter() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
void StatsCounter< Array_Type, Data_Type >::add_counter (
            Counter< Array_Type, Data_Type > * f_ )
```

7.28.3.2 add_counter() [2/2]

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

7.28.3.3 count_all()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::vector< double > StatsCounter< Array_Type, Data_Type >::count_all ( )
```

7.28.3.4 count_current()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
void StatsCounter< Array_Type, Data_Type >::count_current (
            uint i,
            uint j )
```

7.28.3.5 count_init()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
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.

7.28.3.6 get_counters()

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

7.28.3.7 get_descriptions()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::vector< std::string > StatsCounter< Array_Type, Data_Type >::get_descriptions ( ) const
```

7.28.3.8 get_names()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::vector< std::string > StatsCounter< Array_Type, Data_Type >::get_names ( ) const
```

7.28.3.9 reset_array()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
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`.

7.28.3.10 set_counters()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
void StatsCounter< Array_Type, Data_Type >::set_counters (
            Counters< Array_Type, Data_Type > * counters_ )
```

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

- include/barry/statscounter-bones.hpp

7.29 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference

Compute the support of sufficient statistics.

`#include <support-bones.hpp>`

Public Member Functions

- Support (const Array_Type &Array_)
  Constructor passing a reference Array.
- Support (uint N_, uint M_)
  Constructor specifying the dimensions of the array (empty).
- Support ()
- ~Support ()

- void init_support (std::vector< Array_Type > *array_bank=nullptr, std::vector< std::vector< double > > *stats_bank=nullptr)
- void calc (std::vector< Array_Type > *array_bank=nullptr, std::vector< std::vector< double > > *stats_bank=nullptr, unsigned int max_num_elements_=0u)

  Computes the entire support.
- Counts_type get_counts () const
- const MapVec_type * get_counts_ptr () const
- std::vector< double > * get_current_stats ()

  List current statistics.
- void print () const
- const FreqTable & get_data () const
- Counters< Array_Type, Data_Counter_Type > * get_counters ()

  Vector of 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 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< std::pair< uint, uint > > coordinates_free
- std::vector< std::pair< uint, uint > > coordinates_locked
- std::vector< std::vector< double > > change_stats

7.29.1 Detailed Description

template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
class 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 35 of file support-bones.hpp.

7.29.2 Constructor & Destructor Documentation

7.29.2.1 Support() [1/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Support (
            const Array_Type & Array_ )  [inline]
```

Constructor passing a reference Array.

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

7.29.2.2 Support() [2/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Support (
            uint N_,
            uint M_ )  [inline]
```

Constructor specifying the dimensions of the array (empty).

Definition at line 78 of file support-bones.hpp.

7.29.2.3 Support() [3/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Support ( )
[inline]
```

Definition at line 85 of file support-bones.hpp.

7.29.2.4 ~Support()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::~Support ( )
[inline]
```

Definition at line 92 of file support-bones.hpp.

7.29.3 Member Function Documentation

7.29.3.1 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 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_counter
(
            Counter< Array_Type, Data_Counter_Type > * f_ )
```

7.29.3.2 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 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_counter
(
            Counter< Array_Type, Data_Counter_Type > f_ )
```

7.29.3.3 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 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > * f_ )
```

7.29.3.4 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 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > f_ )
```

7.29.3.5 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 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_↩
dyn (
            Rule< Array_Type, Data_Rule_Dyn_Type > * f_ )
```

7.29.3.6 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 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_↩
dyn (
            Rule< Array_Type, Data_Rule_Dyn_Type > f_ )
```

7.29.3.7 calc()

```
template<typename Array_Type = BArray<>, 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< 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.

7.29.3.8 eval_rules_dyn()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::eval_↩
rules_dyn (
            const std::vector< double > & counts,
            const uint & i,
            const uint & j )
```

7.29.3.9 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>* Support< Array_Type, Data_Counter_Type, Data_Rule_↩
Type, Data_Rule_Dyn_Type >::get_counters ( )
```

Vector of couter functions.

7.29.3.10 get_counts()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Counts_type Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >↩
::get_counts ( ) const
```

7.29.3.11 get_counts_ptr()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
const MapVec_type* Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type
>::get_counts_ptr ( ) const
```

7.29.3.12 get_current_stats()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< double >* Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_↩
Dyn_Type >::get_current_stats ( )
```

List current statistics.

7.29.3.13 get_data()

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

7.29.3.14 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>* Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_rules ( )
```

Vector of static rules (cells to iterate).

7.29.3.15 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>* 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).

7.29.3.16 init_support()

```
template<typename Array_Type = BArray<>, 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< std::vector< double > > * stats_bank = nullptr )
```

7.29.3.17 print()

```
template<typename Array_Type = BArray<>, 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
```

7.29.3.18 reset_array() [1/2]

```
template<typename Array_Type = BArray<>, 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
( )
```

7.29.3.19 reset_array() [2/2]

```
template<typename Array_Type = BArray<>, 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_ )
```

7.29.3.20 set_counters()

```
template<typename Array_Type = BArray<>, 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_ )
```

7.29.3.21 set_rules()

```
template<typename Array_Type = BArray<>, 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_ )
```

7.29.3.22 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 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_rules↩
_dyn (
            Rules< Array_Type, Data_Rule_Dyn_Type > * rules_ )
```

7.29.4 Member Data Documentation

7.29.4.1 change_stats

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector< double > > Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::change_stats
```

Definition at line 65 of file support-bones.hpp.

7.29.4.2 coordinates_free

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::pair<uint,uint> > Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::coordinates_free
```

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

7.29.4.3 coordinates_locked

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::pair<uint,uint> > Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::coordinates_locked
```

Definition at line 64 of file support-bones.hpp.

7.29.4.4 current_stats

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< double > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn↩
_Type >::current_stats
```

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

7.29.4.5 delete_counters

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↩
counters = true
```

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

7.29.4.6 delete_rules

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↩
rules = true
```

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

7.29.4.7 delete_rules_dyn

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↩
rules_dyn = true
```

Definition at line 58 of file support-bones.hpp.

7.29.4.8 M

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::M
```

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

7.29.4.9 max_num_elements

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::max_num_↩
elements = BARRY_MAX_NUM_ELEMENTS
```

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

7.29.4.10 N

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::N
```

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

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

- include/barry/support-bones.hpp

7.30 vecHasher< T > Struct Template Reference

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

Public Member Functions

- std::size_t operator() (std::vector< T > const &dat) const noexcept

7.30.1 Detailed Description

template<typename T>
struct vecHasher< T >

Definition at line 92 of file typedefs.hpp.

7.30.2 Member Function Documentation

7.30.2.1 operator()()

```
template<typename T >
std::size_t vecHasher< T >::operator() (
            std::vector< T > const & dat ) const  [inline], [noexcept]
```

Definition at line 93 of file typedefs.hpp.

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

- include/barry/typedefs.hpp

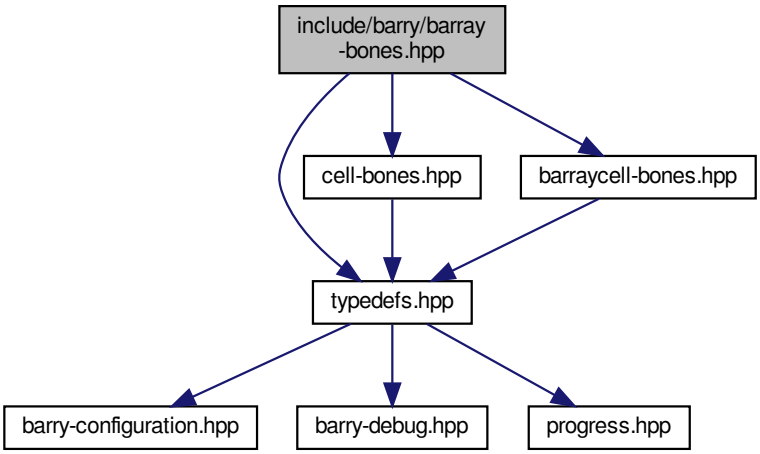
Chapter 8

File Documentation

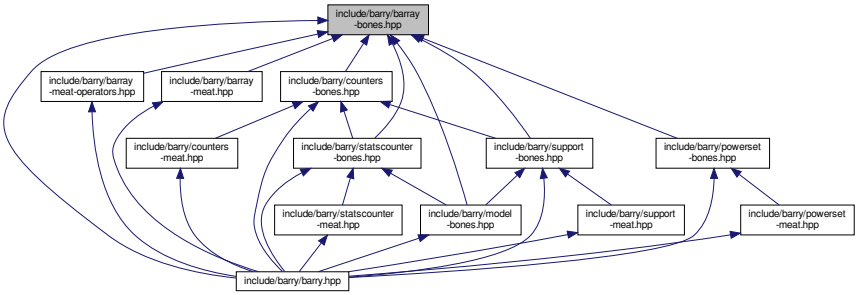
8.1 include/barry/barray-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "cell-bones.hpp"
#include "barraycell-bones.hpp"
```

Include dependency graph for barray-bones.hpp:



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



Classes

- class BArray< Cell_Type, Data_Type >

  Baseline class for binary arrays.

Macros

- #define BARRAY_BONES_HPP 1

8.1.1 Macro Definition Documentation

8.1.1.1 BARRAY_BONES_HPP

```
#define BARRAY_BONES_HPP 1
```

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

8.2 include/barry/barray-iterator.hpp File Reference

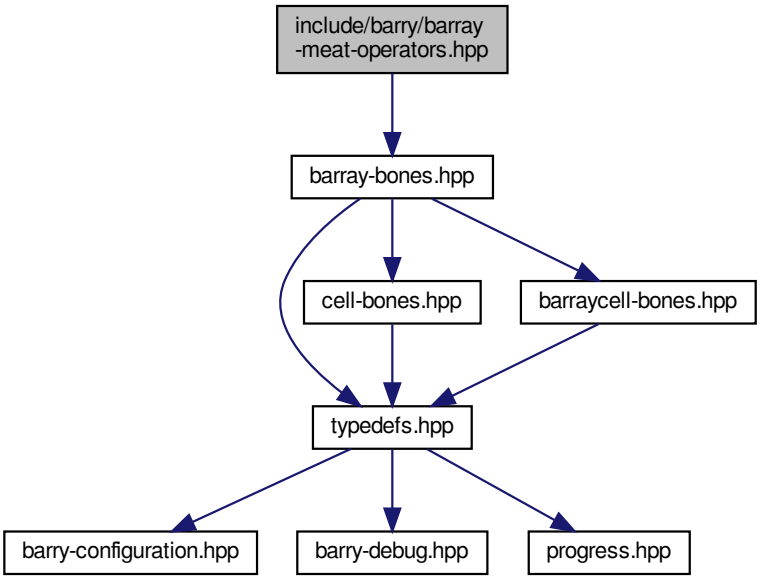
Classes

- class ConstBArrayRowIter< Cell_Type, Data_Type >

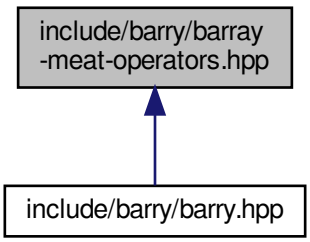
8.3 include/barry/barray-meat-operators.hpp File Reference

```
#include "barray-bones.hpp"
```

Include dependency graph for barray-meat-operators.hpp:



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



Macros

- #define BARRY_BARRAY_MEAT_OPERATORS_HPP 1
- #define ROW(a) this->el_ij[a]
- #define COL(a) this->el_ji[a]

Functions

- template<typename Cell_Type , typename Data_Type >
  void checkdim_ (const BArray< Cell_Type, Data_Type > &lhs, const BArray< Cell_Type, Data_Type > &rhs)

8.3.1 Macro Definition Documentation

8.3.1.1 BARRY_BARRAY_MEAT_OPERATORS_HPP

```
#define BARRY_BARRAY_MEAT_OPERATORS_HPP 1
```

Definition at line 5 of file barray-meat-operators.hpp.

8.3.1.2 COL

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

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

8.3.1.3 ROW

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

Definition at line 7 of file barray-meat-operators.hpp.

8.3.2 Function Documentation

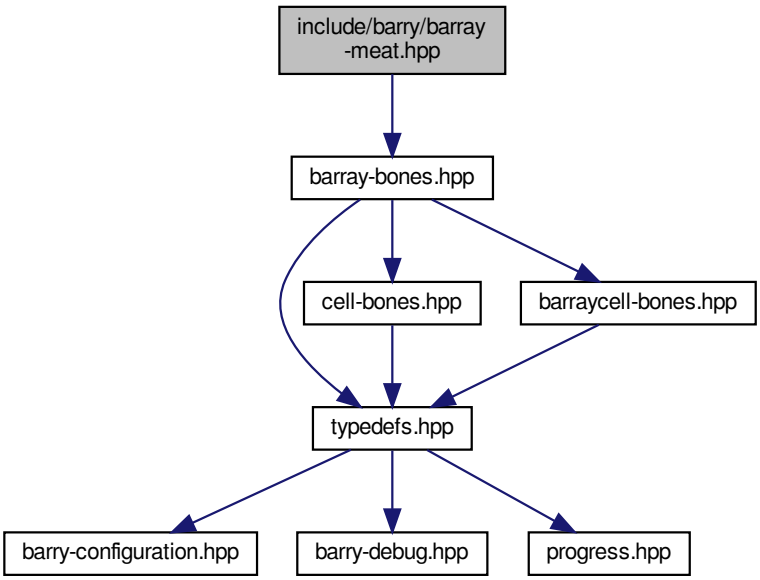
8.3.2.1 checkdim_()

```
template<typename Cell_Type , typename Data_Type >
void checkdim_ (
             const BArray< Cell_Type, Data_Type > & lhs,
             const BArray< Cell_Type, Data_Type > & rhs )  [inline]
```

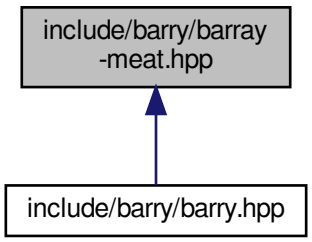
Definition at line 11 of file barray-meat-operators.hpp.

8.4 include/barry/barray-meat.hpp File Reference

`#include "barray-bones.hpp"`
Include dependency graph for barray-meat.hpp:



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



Macros

- #define 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) = el[POS(i, j)]
- 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)()
- 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)

8.4.1 Macro Definition Documentation

8.4.1.1 BARRAY_TEMPLATE

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

Definition at line 11 of file barray-meat.hpp.

8.4.1.2 BARRAY_TEMPLATE_ARGS

```
template BARRAY_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

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

8.4.1.3 BARRAY_TYPE

```
template Data_Type BARRAY_TYPE( ) BArray<Cell_Type, Data_Type>
```

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

8.4.1.4 COL

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

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

8.4.1.5 ROW

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

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

8.4.2 Function Documentation

8.4.2.1 ans()

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

8.4.2.2 BARRAY_TEMPLATE() [1/23]

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

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

8.4.2.3 BARRAY_TEMPLATE() [2/23]

```
BARRAY_TEMPLATE (
            BArray  )
```

8.4.2.4 BARRAY_TEMPLATE() [3/23]

```
BARRAY_TEMPLATE (
            ~ BArray )
```

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

8.4.2.5 BARRAY_TEMPLATE() [4/23]

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

Definition at line 580 of file barray-meat.hpp.

8.4.2.6 BARRAY_TEMPLATE() [5/23]

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

Definition at line 266 of file barray-meat.hpp.

8.4.2.7 BARRAY_TEMPLATE() [6/23]

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

8.4.2.8 BARRAY_TEMPLATE() [7/23]

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

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

8.4.2.9 BARRAY_TEMPLATE() [8/23]

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

8.4.2.10 BARRAY_TEMPLATE() [9/23]

```
BARRAY_TEMPLATE (
            Data_Type * ,
            D  )
```

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

8.4.2.11 BARRAY_TEMPLATE() [10/23]

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

8.4.2.12 BARRAY_TEMPLATE() [11/23]

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

Definition at line 1112 of file barray-meat.hpp.

8.4.2.13 BARRAY_TEMPLATE() [12/23]

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

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

8.4.2.14 BARRAY_TEMPLATE() [13/23]

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

8.4.2.15 BARRAY_TEMPLATE() [14/23]

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

8.4.2.16 BARRAY_TEMPLATE() [15/23]

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

8.4.2.17 BARRAY_TEMPLATE() [16/23]

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

8.4.2.18 BARRAY_TEMPLATE() [17/23]

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

8.4.2.19 BARRAY_TEMPLATE() [18/23]

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

8.4.2.20 BARRAY_TEMPLATE() [19/23]

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

8.4.2.21 BARRAY_TEMPLATE() [20/23]

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

8.4.2.22 BARRAY_TEMPLATE() [21/23]

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

Definition at line 1051 of file barray-meat.hpp.

8.4.2.23 BARRAY_TEMPLATE() [22/23]

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

8.4.2.24 BARRAY_TEMPLATE() [23/23]

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

8.4.2.25 COL()

```
COL (
            j  )
```

8.4.2.26 for() [1/3]

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

8.4.2.27 for() [2/3]

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

8.4.2.28 for() [3/3]

```
for ( ) = el[POS(i, j)]
```

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

8.4.2.29 if() [1/17]

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

Definition at line 990 of file barray-meat.hpp.

8.4.2.30 if() [2/17]

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

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

8.4.2.31 if() [3/17]

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

8.4.2.32 if() [4/17]

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

Definition at line 846 of file barray-meat.hpp.

8.4.2.33 if() [5/17]

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

Definition at line 981 of file barray-meat.hpp.

8.4.2.34 if() [6/17]

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

Definition at line 954 of file barray-meat.hpp.

8.4.2.35 if() [7/17]

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

Definition at line 820 of file barray-meat.hpp.

8.4.2.36 if() [8/17]

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

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

8.4.2.37 if() [9/17]

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

8.4.2.38 if() [10/17]

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

8.4.2.39 if() [11/17]

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

8.4.2.40 if() [12/17]

```
if ( ) = N_
```

Definition at line 80 of file barray-meat.hpp.

8.4.2.41 if() [13/17]

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

8.4.2.42 if() [14/17]

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

8.4.2.43 if() [15/17]

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

8.4.2.44 if() [16/17]

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

8.4.2.45 if() [17/17]

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

8.4.2.46 M()

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

Definition at line 130 of file barray-meat.hpp.

8.4.2.47 resize() [1/2]

```
el_ji resize (
             M  )
```

8.4.2.48 resize() [2/2]

```
el_ij resize (
             N  )
```

8.4.2.49 return()

```
return (
             Cell_Type  )
```

8.4.2.50 ROW() [1/2]

```
& ROW (
             i  )
```

8.4.2.51 ROW() [2/2]

```
ROW (
             i0  )
```

8.4.3 Variable Documentation

8.4.3.1 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 28 of file barray-meat.hpp.

8.4.3.2 ans

```
return ans
```

Definition at line 432 of file barray-meat.hpp.

8.4.3.3 Array_

```
Data_Type & Array_
```

Definition at line 128 of file barray-meat.hpp.

8.4.3.4 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 654 of file barray-meat.hpp.

8.4.3.5 check_exists

```
uint bool int check_exists
```

Initial value:

```
{

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

Definition at line 655 of file barray-meat.hpp.

8.4.3.6 col0

```
auto col0 = COL(j)
```

Definition at line 1043 of file barray-meat.hpp.

8.4.3.7 const

```
bool check_bounds const
```

Initial value:

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

Definition at line 385 of file barray-meat.hpp.

8.4.3.8 copy_data

```
Data_Type bool copy_data
```

Definition at line 129 of file barray-meat.hpp.

8.4.3.9 data

```
data = data_
```

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

8.4.3.10 delete_data

```
delete_data = delete_data_
```

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

8.4.3.11 delete_data_

```
bool delete_data_
```

Initial value:

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

Definition at line 342 of file barray-meat.hpp.

8.4.3.12 else

```
else
```

Initial value:

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

Definition at line 685 of file barray-meat.hpp.

8.4.3.13 false

```
row false
```

Definition at line 1024 of file barray-meat.hpp.

8.4.3.14 first

```
row first
```

Definition at line 1024 of file barray-meat.hpp.

8.4.3.15 i1

```
uint i1
```

Definition at line 758 of file barray-meat.hpp.

8.4.3.16 j

```
uint j
```

Initial value:

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

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

8.4.3.17 j0

```
uint j0
```

Definition at line 757 of file barray-meat.hpp.

8.4.3.18 j1

```
uint j1
```

Definition at line 758 of file barray-meat.hpp.

8.4.3.19 M

```
M = M_
```

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

8.4.3.20 M_

```
uint M_
```

Initial value:

```
{

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

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

8.4.3.21 N

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

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

8.4.3.22 NCells

```
NCells
```

Definition at line 689 of file barray-meat.hpp.

8.4.3.23 report

```
uint uint uint bool int int* report
```

Definition at line 761 of file barray-meat.hpp.

8.4.3.24 return

```
return
```

Definition at line 60 of file barray-meat.hpp.

8.4.3.25 row0

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

Definition at line 1022 of file barray-meat.hpp.

8.4.3.26 search

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

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

8.4.3.27 source

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

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

8.4.3.28 target

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

Definition at line 26 of file barray-meat.hpp.

8.4.3.29 v

```
uint Cell_Type v
```

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

8.4.3.30 value

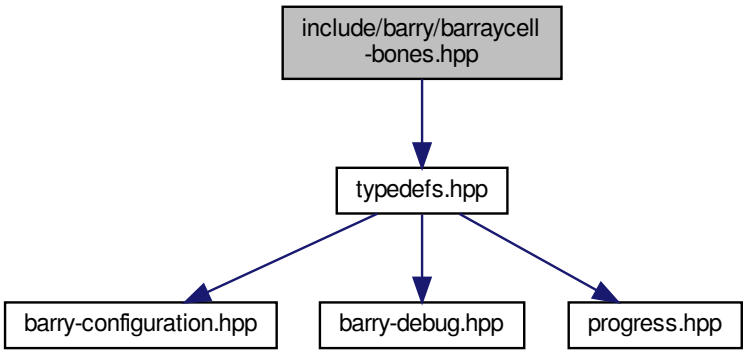
```
uint const std::vector< uint > const std::vector< uint > const std::vector< Cell_Type >&
value
```

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

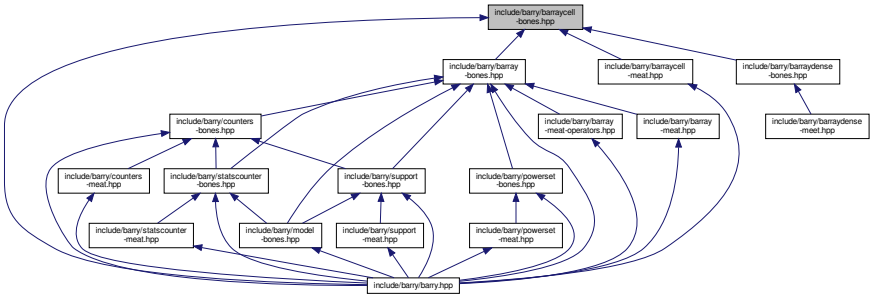
8.5 include/barry/barraycell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barraycell-bones.hpp:



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



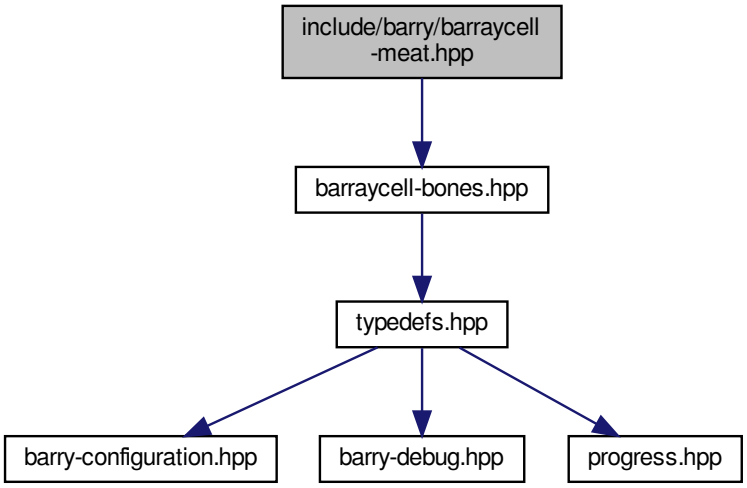
Classes

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

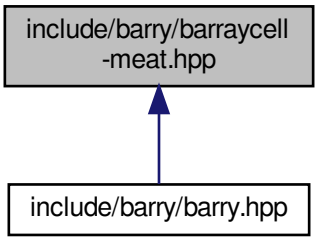
8.6 include/barry/barraycell-meat.hpp File Reference

```
#include "barraycell-bones.hpp"
```

Include dependency graph for barraycell-meat.hpp:



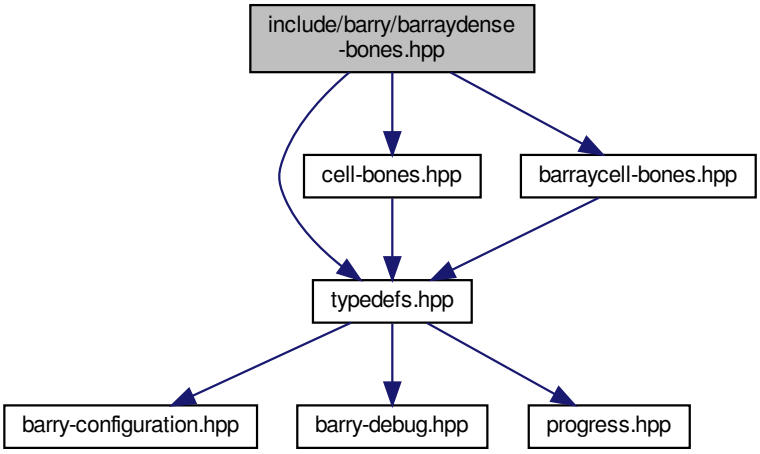
This graph shows which files directly or indirectly include this file:



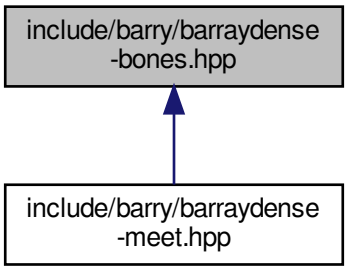
8.7 include/barry/barraydense-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "cell-bones.hpp"
#include "barraycell-bones.hpp"
```

Include dependency graph for barraydense-bones.hpp:



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



Classes

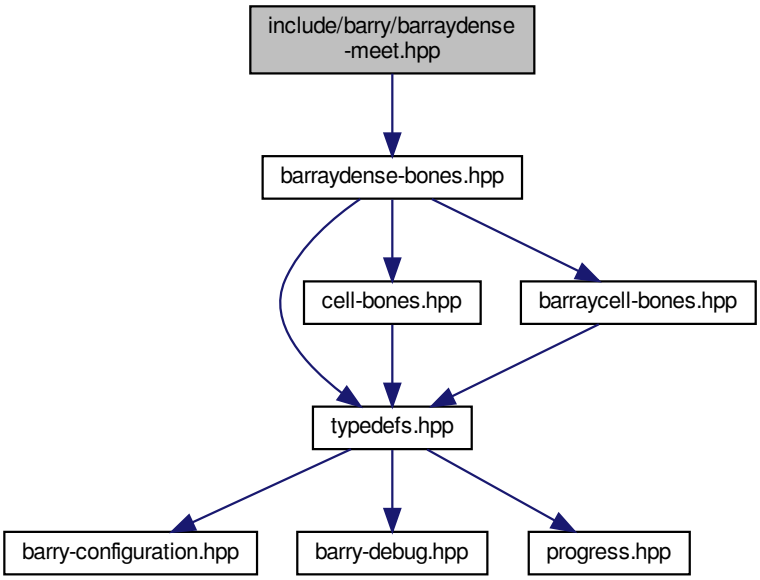
- class BArrayDense< Cell_Type, Data_Type >

  Baseline class for binary arrays.

8.8 include/barry/barraydense-meet.hpp File Reference

```
#include "barraydense-bones.hpp"
```

Include dependency graph for barraydense-meet.hpp:



Macros

- #define BARRY_BARRAYDENSE_MEAT_HPP
- #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 ZERO_CELL Cell< Cell_Type >(static_cast< Cell_Type >(0.0))

Functions

- BDENSE_TEMPLATE (, BArrayDense)(uint N_
- el resize (N *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)()
- 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(),(Cell_Type) false)
- BDENSE_TEMPLATE (void, get_row_vec)(std
- BDENSE_TEMPLATE (BDENSE_TYPE() &, operator-=)(const std
- BDENSE_TEMPLATE (void, insert_cell)(uint i
- if (check_exists)
- COL (j).emplace(i
- & ROW (i)[j])
- BDENSE_TEMPLATE (void, swap_cells)(uint i0
- if (report !=nullptr)(*report)
- if (check0 &check1)
- else if (!check0 &check1)
- else if (check0 &!check1)
- BDENSE_TEMPLATE (void, toggle_cell)(uint i
- BDENSE_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))
- BDENSE_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)
- BDENSE_TEMPLATE (void, zero_row)(uint i
- if (ROW(i).size()==0u) return
- for (auto row=row0.begin();row !=row0.end();++row) rm_cell(i
- BDENSE_TEMPLATE (void, zero_col)(uint j
- if (COL(j).size()==0u) return
- BDENSE_TEMPLATE (void, transpose)()
- BDENSE_TEMPLATE (void, clear)(bool hard)
- BDENSE_TEMPLATE (void, resize)(uint N_
- if (M_< M) for(uint j

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 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)

8.8.1 Macro Definition Documentation

8.8.1.1 BARRY_BARRAYDENSE_MEAT_HPP

```
#define BARRY_BARRAYDENSE_MEAT_HPP
```

Definition at line 5 of file barraydense-meet.hpp.

8.8.1.2 BDENSE_TEMPLATE

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

Definition at line 11 of file barraydense-meet.hpp.

8.8.1.3 BDENSE_TEMPLATE_ARGS

```
template BDENSE_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

Definition at line 9 of file barraydense-meet.hpp.

8.8.1.4 BDENSE_TYPE

```
template Data_Type BDENSE_TYPE( ) BArrayDense<Cell_Type, Data_Type>
```

Definition at line 7 of file barraydense-meet.hpp.

8.8.1.5 COL

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

Definition at line 15 of file barraydense-meet.hpp.

8.8.1.6 POS

```
#define POS(
              a,
              b ) (b)*N + (a)
```

Definition at line 16 of file barraydense-meet.hpp.

8.8.1.7 ROW

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

Definition at line 14 of file barraydense-meet.hpp.

8.8.1.8 ZERO_CELL

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

Definition at line 21 of file barraydense-meet.hpp.

8.8.2 Function Documentation

8.8.2.1 ans()

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

8.8.2.2 BDENSE_TEMPLATE() [1/25]

```
BDENSE_TEMPLATE (
            BArrayDense  ) &&  [noexcept]
```

Definition at line 198 of file barraydense-meet.hpp.

8.8.2.3 BDENSE_TEMPLATE() [2/25]

```
BDENSE_TEMPLATE (
            BArrayDense  ) const &
```

8.8.2.4 BDENSE_TEMPLATE() [3/25]

```
BDENSE_TEMPLATE (
            BArrayDense  )
```

8.8.2.5 BDENSE_TEMPLATE() [4/25]

```
BDENSE_TEMPLATE (
            ~ BArrayDense )
```

Definition at line 272 of file barraydense-meet.hpp.

8.8.2.6 BDENSE_TEMPLATE() [5/25]

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

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

8.8.2.7 BDENSE_TEMPLATE() [6/25]

```
BDENSE_TEMPLATE (
            BDENSE_TYPE() & ,
            operator  ) &&  [noexcept]
```

Definition at line 213 of file barraydense-meet.hpp.

8.8.2.8 BDENSE_TEMPLATE() [7/25]

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

Definition at line 159 of file barraydense-meet.hpp.

8.8.2.9 BDENSE_TEMPLATE() [8/25]

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

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

8.8.2.10 BDENSE_TEMPLATE() [9/25]

```
BDENSE_TEMPLATE (
            Cell_Type ,
            get_cell  )
```

8.8.2.11 BDENSE_TEMPLATE() [10/25]

```
BDENSE_TEMPLATE (
            const Data_Type * ,
            D  ) const
```

Definition at line 298 of file barraydense-meet.hpp.

8.8.2.12 BDENSE_TEMPLATE() [11/25]

```
BDENSE_TEMPLATE (
            Data_Type * ,
            D  )
```

Definition at line 294 of file barraydense-meet.hpp.

8.8.2.13 BDENSE_TEMPLATE() [12/25]

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

8.8.2.14 BDENSE_TEMPLATE() [13/25]

```
BDENSE_TEMPLATE (
            void ,
            clear  )
```

Definition at line 979 of file barraydense-meet.hpp.

8.8.2.15 BDENSE_TEMPLATE() [14/25]

```
BDENSE_TEMPLATE (
            void ,
            get_row_vec  )
```

Definition at line 337 of file barraydense-meet.hpp.

8.8.2.16 BDENSE_TEMPLATE() [15/25]

```
BDENSE_TEMPLATE (
            void ,
            insert_cell  )
```

8.8.2.17 BDENSE_TEMPLATE() [16/25]

```
BDENSE_TEMPLATE (
            void ,
            out_of_range  )
```

8.8.2.18 BDENSE_TEMPLATE() [17/25]

```
BDENSE_TEMPLATE (
            void ,
            resize  )
```

8.8.2.19 BDENSE_TEMPLATE() [18/25]

```
BDENSE_TEMPLATE (
            void ,
            set_data  )
```

8.8.2.20 BDENSE_TEMPLATE() [19/25]

```
BDENSE_TEMPLATE (
            void ,
            swap_cells  )
```

8.8.2.21 BDENSE_TEMPLATE() [20/25]

```
BDENSE_TEMPLATE (
            void ,
            swap_cols  )
```

8.8.2.22 BDENSE_TEMPLATE() [21/25]

```
BDENSE_TEMPLATE (
            void ,
            swap_rows  )
```

8.8.2.23 BDENSE_TEMPLATE() [22/25]

```
BDENSE_TEMPLATE (
            void ,
            toggle_cell  )
```

8.8.2.24 BDENSE_TEMPLATE() [23/25]

```
BDENSE_TEMPLATE (
            void ,
            transpose  )
```

Definition at line 923 of file barraydense-meet.hpp.

8.8.2.25 BDENSE_TEMPLATE() [24/25]

```
BDENSE_TEMPLATE (
            void ,
            zero_col  )
```

8.8.2.26 BDENSE_TEMPLATE() [25/25]

```
BDENSE_TEMPLATE (
            void ,
            zero_row  )
```

8.8.2.27 COL()

```
COL (
            j  )
```

8.8.2.28 for() [1/2]

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

8.8.2.29 for() [2/2]

```
for ( )
```

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

8.8.2.30 if() [1/16]

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

Definition at line 862 of file barraydense-meet.hpp.

8.8.2.31 if() [2/16]

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

Definition at line 713 of file barraydense-meet.hpp.

8.8.2.32 if() [3/16]

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

8.8.2.33 if() [4/16]

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

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

8.8.2.34 if() [5/16]

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

Definition at line 853 of file barraydense-meet.hpp.

8.8.2.35 if() [6/16]

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

Definition at line 826 of file barraydense-meet.hpp.

8.8.2.36 if() [7/16]

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

Definition at line 696 of file barraydense-meet.hpp.

8.8.2.37 if() [8/16]

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

Definition at line 544 of file barraydense-meet.hpp.

8.8.2.38 if() [9/16]

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

8.8.2.39 if() [10/16]

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

8.8.2.40 if() [11/16]

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

8.8.2.41 if() [12/16]

```
if ( )
```

8.8.2.42 if() [13/16]

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

8.8.2.43 if() [14/16]

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

8.8.2.44 if() [15/16]

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

8.8.2.45 if() [16/16]

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

8.8.2.46 M()

```
bool M (
              Array_.  M )
```

Definition at line 127 of file barraydense-meet.hpp.

8.8.2.47 resize()

```
el resize (
              N * M,
              ZERO_CELL  )
```

8.8.2.48 ROW() [1/2]

```
& ROW (
              i  )
```

8.8.2.49 ROW() [2/2]

```
ROW (
              i0  )
```

8.8.3 Variable Documentation

8.8.3.1 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 29 of file barraydense-meet.hpp.

8.8.3.2 ans

```
return ans
```

Definition at line 334 of file barraydense-meet.hpp.

8.8.3.3 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 537 of file barraydense-meet.hpp.

8.8.3.4 check_exists

```
uint bool int check_exists
```

Initial value:

```
{

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

Definition at line 538 of file barraydense-meet.hpp.

8.8.3.5 col0

```
auto col0 = COL(j)
```

Definition at line 915 of file barraydense-meet.hpp.

8.8.3.6 const

```
bool check_bounds const
```

Initial value:

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

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

8.8.3.7 copy_data

```
bool copy_data
```

Definition at line 126 of file barraydense-meet.hpp.

8.8.3.8 data

```
data = data_
```

Definition at line 287 of file barraydense-meet.hpp.

8.8.3.9 delete_data

```
delete_data = delete_data_
```

Definition at line 288 of file barraydense-meet.hpp.

8.8.3.10 delete_data_

```
bool delete_data_
```

Initial value:

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

Definition at line 281 of file barraydense-meet.hpp.

8.8.3.11 else

```
else
```

Initial value:

```
{

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

Definition at line 568 of file barraydense-meet.hpp.

8.8.3.12 false

```
row false
```

Definition at line 896 of file barraydense-meet.hpp.

8.8.3.13 first

```
row first
```

Definition at line 896 of file barraydense-meet.hpp.

8.8.3.14 i1

```
uint i1
```

Definition at line 636 of file barraydense-meet.hpp.

8.8.3.15 j

```
uint j
```

Definition at line 311 of file barraydense-meet.hpp.

8.8.3.16 j0

```
uint j0
```

Definition at line 635 of file barraydense-meet.hpp.

8.8.3.17 j1

```
uint j1
```

Definition at line 636 of file barraydense-meet.hpp.

8.8.3.18 M

```
M = M_
```

Definition at line 39 of file barraydense-meet.hpp.

8.8.3.19 M_

```
uint M_
```

Initial value:

```
{

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

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

8.8.3.20 N

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

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

8.8.3.21 NCells

```
NCells
```

Definition at line 572 of file barraydense-meet.hpp.

8.8.3.22 report

```
uint uint uint bool int int* report
```

Definition at line 639 of file barraydense-meet.hpp.

8.8.3.23 return

```
return
```

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

8.8.3.24 row0

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

Definition at line 894 of file barraydense-meet.hpp.

8.8.3.25 source

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

Definition at line 26 of file barraydense-meet.hpp.

8.8.3.26 target

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

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

8.8.3.27 v

```
uint Cell_Type v
```

Definition at line 536 of file barraydense-meet.hpp.

8.8.3.28 value

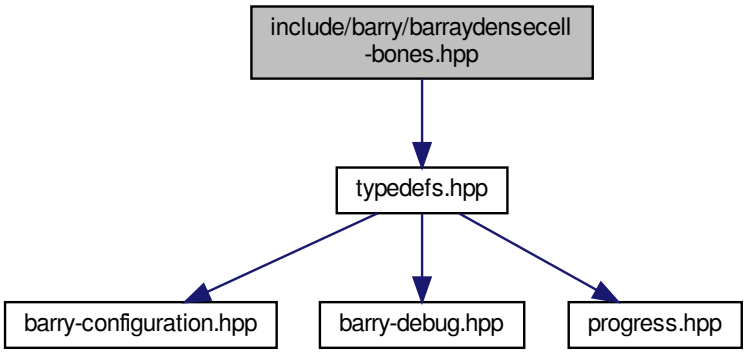
```
return el [POS(i, j)] value
```

Definition at line 28 of file barraydense-meet.hpp.

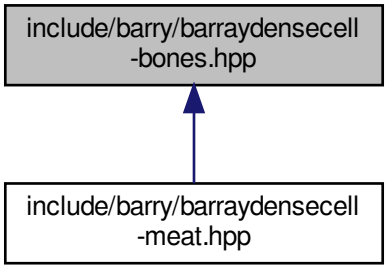
8.9 include/barry/barraydensecell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barraydensecell-bones.hpp:



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



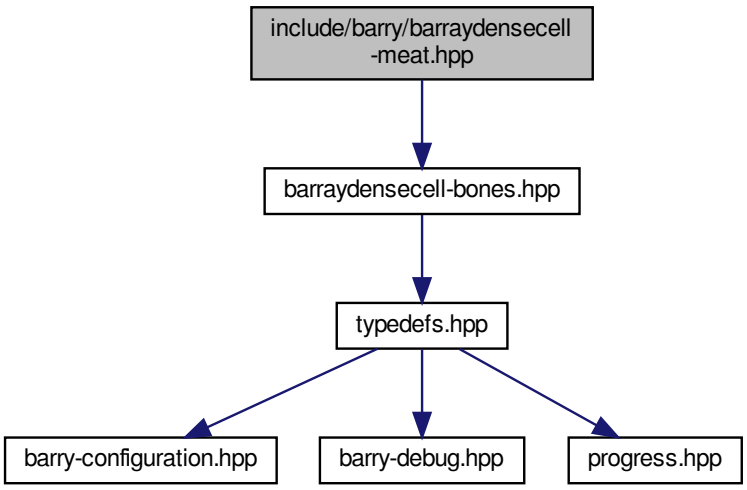
Classes

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

8.10 include/barry/barraydensecell-meat.hpp File Reference

```
#include "barraydensecell-bones.hpp"
```

Include dependency graph for barraydensecell-meat.hpp:



Macros

- #define BARRY_BARRAYDENSECELL_MEAT_HPP 1
- #define POS(a, b) (a) + (b) * Array->N

8.10.1 Macro Definition Documentation

8.10.1.1 BARRY_BARRAYDENSECELL_MEAT_HPP

```
#define BARRY_BARRAYDENSECELL_MEAT_HPP 1
```

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

8.10.1.2 POS

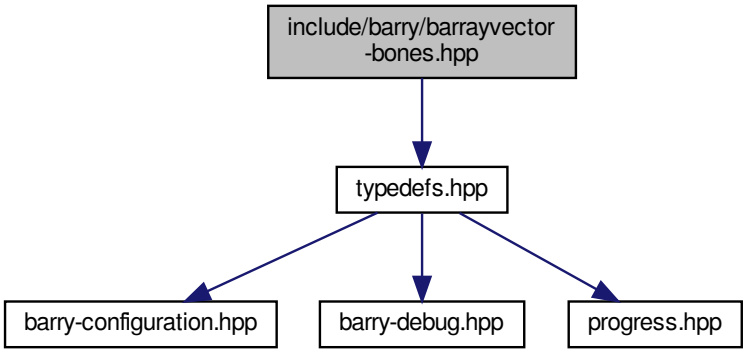
```
#define POS(
              a,
              b ) (a) + (b) * Array->N
```

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

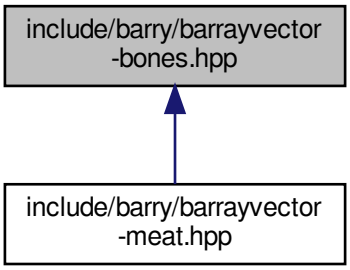
8.11 include/barry/barrayvector-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barrayvector-bones.hpp:



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



Classes

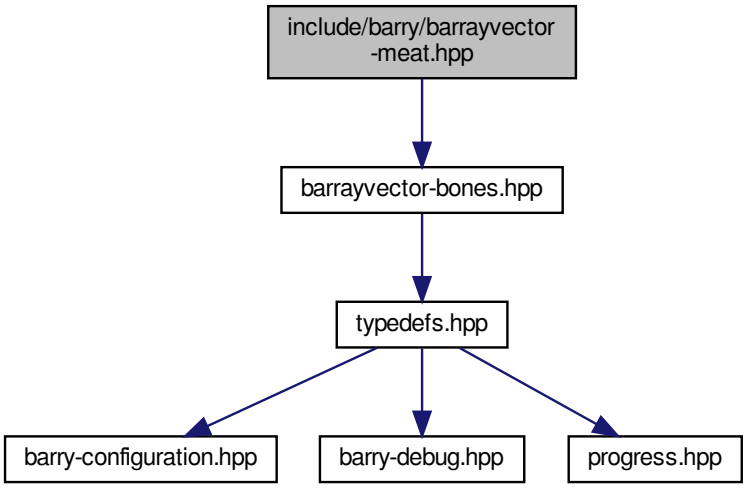
- class BArrayVector< Cell_Type, Data_Type >

  Row or column of a `BArray`
- class BArrayVector_const< Cell_Type, Data_Type >

8.12 include/barry/barrayvector-meat.hpp File Reference

```
#include "barrayvector-bones.hpp"
```

Include dependency graph for barrayvector-meat.hpp:



Macros

- #define BARRY_BARRAYVECTOR_MEAT_HPP 1

8.12.1 Macro Definition Documentation

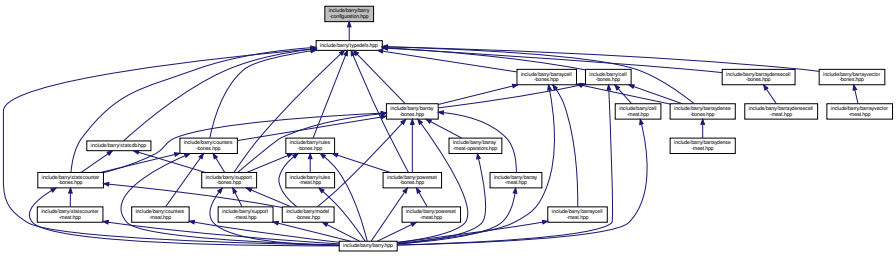
8.12.1.1 BARRY_BARRAYVECTOR_MEAT_HPP

```
#define BARRY_BARRAYVECTOR_MEAT_HPP 1
```

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

8.13 include/barry/barry-configuration.hpp File Reference

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



Configuration MACROS

These are mostly related to performance. The definitions follow:

- `BARRY_USE_UNORDERED_MAP` If specified, then barry is compiled using `std::unordered_map`. Otherwise it will use `std::map` for the arrays.
- `BARRY_USE_SAFE_EXP` When specified, it will multiply all likelihoods in `Model` by (1/-100)/(1/-100) so that numerical overflows are avoided.
- `BARRY_USE_ISFINITE` When specified, it will introduce a macro that checks whether the likelihood is finite or not.
- `printf_barry` If not specified, will be defined as `printf`.
- `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< unsigned int >(UINT_MAX/2u)
- template<typename Ta , typename Tb >
  using Map = std::map< Ta, Tb >

8.13.1 Macro Definition Documentation

8.13.1.1 BARRY_CHECK_SUPPORT

```
#define BARRY_CHECK_SUPPORT(
              x,
              maxs )
```

Definition at line 47 of file barry-configuration.hpp.

8.13.1.2 BARRY_ISFINITE

```
#define BARRY_ISFINITE(
              a )
```

Definition at line 40 of file barry-configuration.hpp.

8.13.1.3 BARRY_MAX_NUM_ELEMENTS

```
#define BARRY_MAX_NUM_ELEMENTS static_cast< unsigned int >(UINT_MAX/2u)
```

Definition at line 55 of file barry-configuration.hpp.

8.13.1.4 BARRY_SAFE_EXP

```
#define BARRY_SAFE_EXP -100.0
```

Definition at line 33 of file barry-configuration.hpp.

8.13.1.5 printf_barry

```
#define printf_barry printf
```

Definition at line 51 of file barry-configuration.hpp.

8.13.2 Typedef Documentation

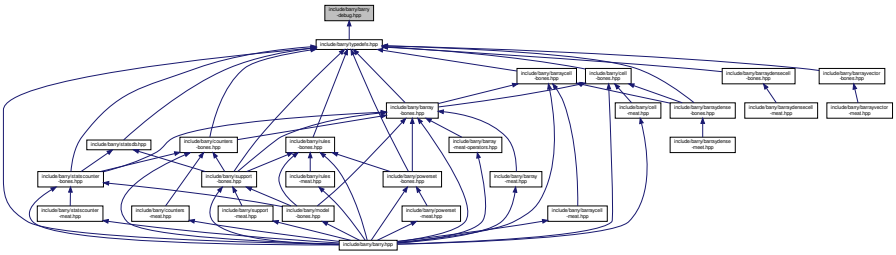
8.13.2.1 Map

```
template<typename Ta , typename Tb >
using Map = std::map<Ta,Tb>
```

Definition at line 27 of file barry-configuration.hpp.

8.14 include/barry/barry-debug.hpp File Reference

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



Macros

- #define BARRY_DEBUG_LEVEL 0

8.14.1 Macro Definition Documentation

8.14.1.1 BARRY_DEBUG_LEVEL

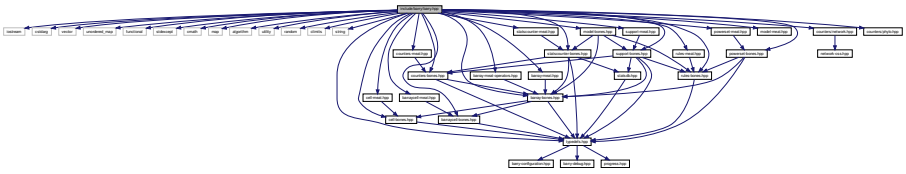
```
#define BARRY_DEBUG_LEVEL 0
```

Definition at line 5 of file barry-debug.hpp.

8.15 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 <string>
#include "typedefs.hpp"
#include "cell-bones.hpp"
#include "cell-meat.hpp"
#include "barray-bones.hpp"
#include "barraycell-bones.hpp"
#include "barray-meat.hpp"
#include "barraycell-meat.hpp"
#include "barray-meat-operators.hpp"
#include "counters-bones.hpp"
#include "counters-meat.hpp"
#include "statscounter-bones.hpp"
#include "statscounter-meat.hpp"
#include "support-bones.hpp"
#include "support-meat.hpp"
#include "powerset-bones.hpp"
#include "powerset-meat.hpp"
#include "model-bones.hpp"
#include "model-meat.hpp"
#include "rules-bones.hpp"
#include "rules-meat.hpp"
#include "counters/network.hpp"
#include "counters/phylo.hpp"
```

Include dependency graph for barry.hpp:



Namespaces

- barry

  barry: Your go-to motif accountant
- barry::counters

  Tree class and TreeIterator class.
- barry::counters::network
- barry::counters::phylo

Macros

- #define BARRY_HPP
- #define BARRY_VERSION 0.1
- #define COUNTER_FUNCTION(a)
- #define COUNTER_LAMBDA(a)
- #define RULE_FUNCTION(a)
- #define RULE_LAMBDA(a)

8.15.1 Macro Definition Documentation

8.15.1.1 BARRY_HPP

```
#define BARRY_HPP
```

Definition at line 20 of file barry.hpp.

8.15.1.2 BARRY_VERSION

```
#define BARRY_VERSION 0.1
```

Definition at line 22 of file barry.hpp.

8.15.1.3 COUNTER_FUNCTION

```
#define COUNTER_FUNCTION(
              a )
```

Value:

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
inline double (a) (const Array_Type & Array, uint i, uint j, Data_Type * data)\
```

Definition at line 73 of file barry.hpp.

8.15.1.4 COUNTER_LAMBDA

```
#define COUNTER_LAMBDA(
              a )
```

Value:

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
Counter_fun_type<Array_Type, Data_Type> a = \
[](const Array_Type & Array, uint i, uint j, Data_Type * data)
```

Definition at line 76 of file barry.hpp.

8.15.1.5 RULE_FUNCTION

```
#define RULE_FUNCTION(
                a )
```

Value:

```
    template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
    inline bool (a) (const Array_Type & Array, uint i, uint j, Data_Type * data)\
```

Definition at line 80 of file barry.hpp.

8.15.1.6 RULE_LAMBDA

```
#define RULE_LAMBDA(
                a )
```

Value:

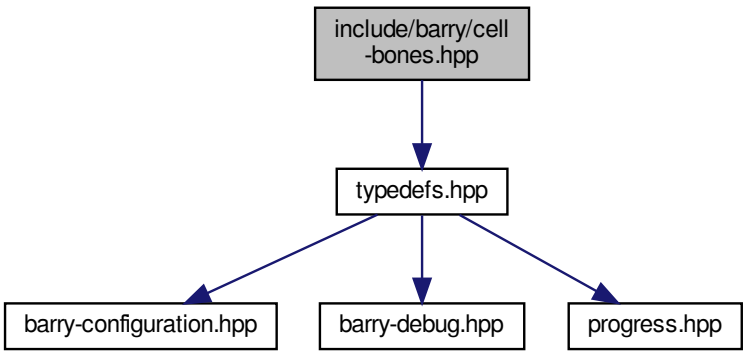
```
    template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
    Rule_fun_type<Array_Type, Data_Type> a = \
    [](const Array_Type & Array, uint i, uint j, Data_Type * data)
```

Definition at line 83 of file barry.hpp.

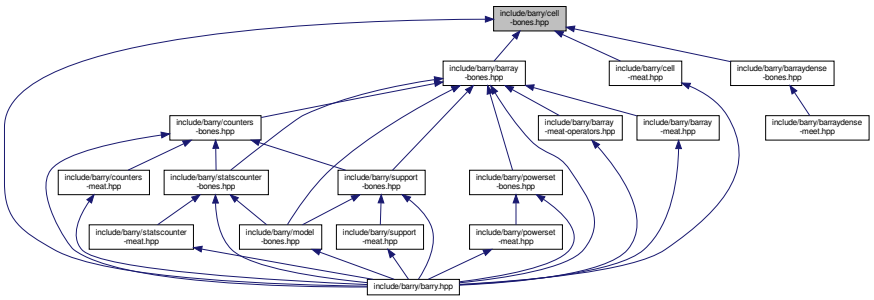
8.16 include/barry/cell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for cell-bones.hpp:



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



Classes

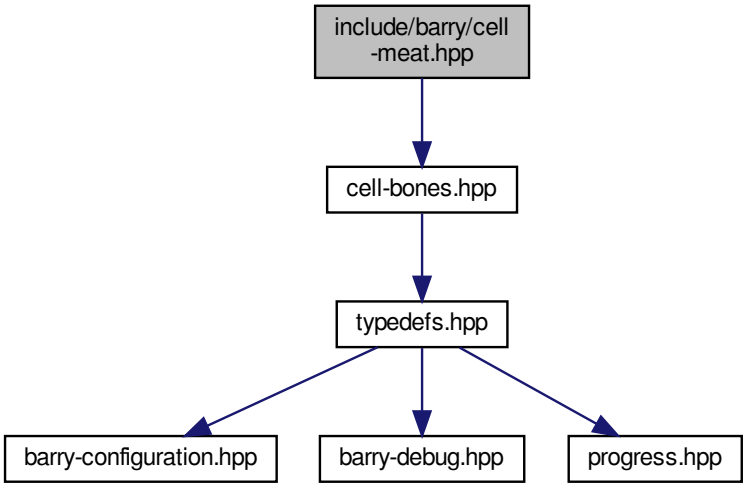
- class Cell< Cell_Type >

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

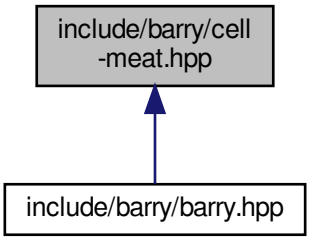
8.17 include/barry/cell-meat.hpp File Reference

```
#include "cell-bones.hpp"
```

Include dependency graph for cell-meat.hpp:



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

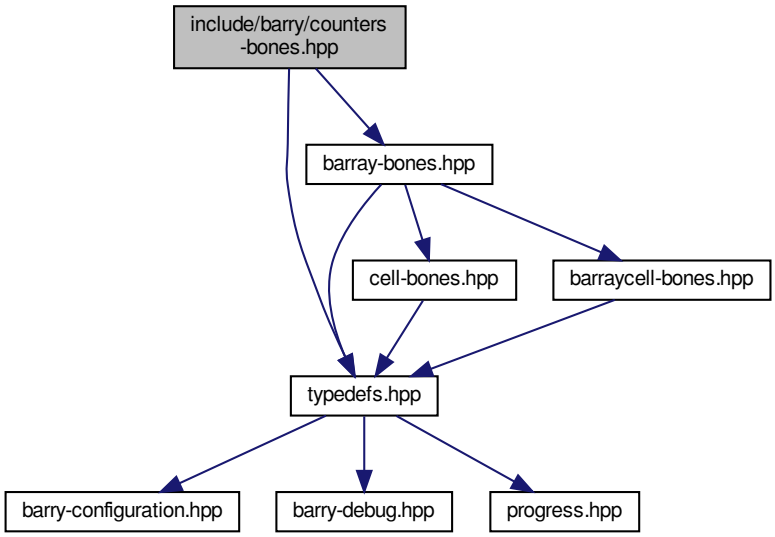


8.18 include/barry/col-bones.hpp File Reference

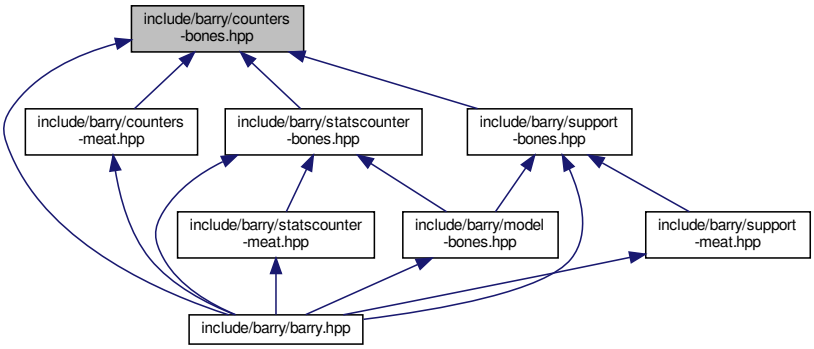
8.19 include/barry/counters-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "barray-bones.hpp"
```

Include dependency graph for counters-bones.hpp:



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



Classes

- class Counter< Array_Type, Data_Type >

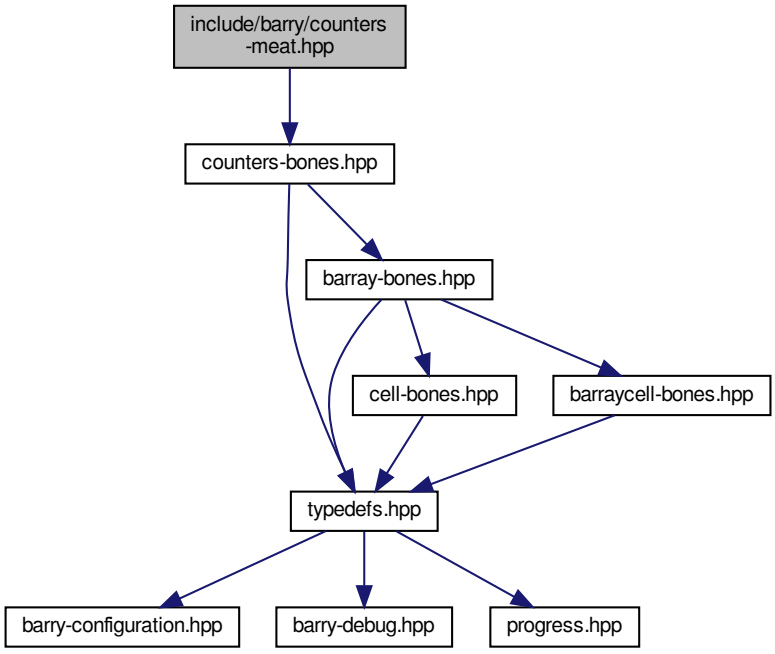
  A counter function based on change statistics.
- class Counters< Array_Type, Data_Type >

  Vector of counters.

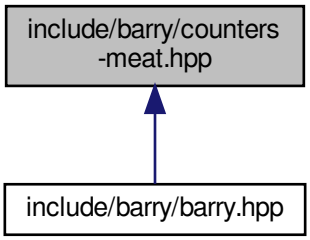
8.20 include/barry/counters-meat.hpp File Reference

```
#include "counters-bones.hpp"
```

Include dependency graph for counters-meat.hpp:



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



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 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 &&counter_ init_fun (std::move(counter_.init_fun))
- Data_Type &&counter_ data (std::move(counter_.data))
- Data_Type &&counter_ delete_data (std::move(counter_.delete_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
- COUNTERS_TEMPLATE (, Counters)()
- COUNTERS_TEMPLATE (COUNTER_TYPE() &, operator[])(uint idx)
- Data_Type Data_Type to_be_deleted (new std::vector< uint >(0u))
- Data_Type Data_Type delete_data (true)
- Data_Type Data_Type delete_to_be_deleted (true)

- Data_Type &&counters_ to_be_deleted (std::move(counters_.to_be_deleted))
- Data_Type &&counters_ delete_data (std::move(counters_.delete_data))
- Data_Type &&counters_ delete_to_be_deleted (std::move(counters_.delete_to_be_deleted))
- 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
- data push_back (new Counter< Array_Type, Data_Type >(counter))
- data push_back (new Counter< Array_Type, Data_Type >(count_fun_, init_fun_, data_, delete_data_, name_, desc_))
- COUNTERS_TEMPLATE (void, clear)()
- COUNTERS_TEMPLATE (std::vector< std::string >, get_names)() const
- COUNTERS_TEMPLATE (std::vector< std::string >, get_descriptions)() const

Variables

- Data_Type & counter_
- Data_Type &&counter_ noexcept
- uint i
- uint uint j
- 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 > Data_Type * data_
- Data_Type Counter_fun_type< Array_Type, Data_Type > Data_Type bool delete_data_
- Data_Type Counter_fun_type< Array_Type, Data_Type > Data_Type bool std::string name_
- Data_Type Counter_fun_type< Array_Type, Data_Type > Data_Type bool std::string std::string desc_

8.20.1 Macro Definition Documentation

8.20.1.1 COUNTER_TEMPLATE

```
#define COUNTER_TEMPLATE(
              a,
              b )  template COUNTER_TEMPLATE_ARGS() inline a COUNTER_TYPE()::b
```

Definition at line 10 of file counters-meat.hpp.

8.20.1.2 COUNTER_TEMPLATE_ARGS

```
#define COUNTER_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

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

8.20.1.3 COUNTER_TYPE

```
#define COUNTER_TYPE( ) Counter<Array_Type,Data_Type>
```

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

8.20.1.4 COUNTERS_TEMPLATE

```
#define COUNTERS_TEMPLATE(
              a,
              b )  template COUNTERS_TEMPLATE_ARGS() inline a COUNTERS_TYPE()::b
```

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

8.20.1.5 COUNTERS_TEMPLATE_ARGS

```
#define COUNTERS_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

Definition at line 151 of file counters-meat.hpp.

8.20.1.6 COUNTERS_TYPE

```
#define COUNTERS_TYPE( ) Counters<Array_Type,Data_Type>
```

Definition at line 149 of file counters-meat.hpp.

8.20.2 Function Documentation

8.20.2.1 count_fun()

```
return count_fun (
             Array ,
             i ,
             j ,
             data  )
```

8.20.2.2 COUNTER_TEMPLATE() [1/7]

```
COUNTER_TEMPLATE (
            Counter  ) const
```

8.20.2.3 COUNTER_TEMPLATE() [2/7]

```
COUNTER_TEMPLATE (
            COUNTER_TYPE() & ,
            operator  )
```

8.20.2.4 COUNTER_TEMPLATE() [3/7]

```
COUNTER_TEMPLATE (
            COUNTER_TYPE() ,
            operator  ) const
```

8.20.2.5 COUNTER_TEMPLATE() [4/7]

```
COUNTER_TEMPLATE (
            double ,
            count  ) &
```

< Move assignment

8.20.2.6 COUNTER_TEMPLATE() [5/7]

```
COUNTER_TEMPLATE (
            double ,
            init  ) &
```

8.20.2.7 COUNTER_TEMPLATE() [6/7]

```
COUNTER_TEMPLATE (
            std::string ,
            get_description  ) const
```

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

8.20.2.8 COUNTER_TEMPLATE() [7/7]

```
COUNTER_TEMPLATE (
            std::string ,
            get_name  ) const
```

Definition at line 137 of file counters-meat.hpp.

8.20.2.9 COUNTERS_TEMPLATE() [1/8]

```
COUNTERS_TEMPLATE (
            Counters  )
```

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

8.20.2.10 COUNTERS_TEMPLATE() [2/8]

```
COUNTERS_TEMPLATE (
            COUNTER_TYPE() & ,
            operator [] )
```

Definition at line 163 of file counters-meat.hpp.

8.20.2.11 COUNTERS_TEMPLATE() [3/8]

```
COUNTERS_TEMPLATE (
            COUNTERS_TYPE() & ,
            operator  )
```

8.20.2.12 COUNTERS_TEMPLATE() [4/8]

```
COUNTERS_TEMPLATE (
            COUNTERS_TYPE() ,
            operator  ) const
```

8.20.2.13 COUNTERS_TEMPLATE() [5/8]

```
COUNTERS_TEMPLATE (
            std::vector< std::string > ,
            get_descriptions  ) const
```

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

8.20.2.14 COUNTERS_TEMPLATE() [6/8]

```
COUNTERS_TEMPLATE (
            std::vector< std::string > ,
            get_names  ) const
```

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

8.20.2.15 COUNTERS_TEMPLATE() [7/8]

```
COUNTERS_TEMPLATE (
            void ,
            add_counter  )
```

8.20.2.16 COUNTERS_TEMPLATE() [8/8]

```
COUNTERS_TEMPLATE (
            void ,
            clear  )
```

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

8.20.2.17 data()

```
Data_Type&& counter_ data (
            std::move(counter_.data)  )
```

8.20.2.18 delete_data() [1/3]

```
Data_Type&& counter_ delete_data (
            std::move(counter_.delete_data)  )
```

8.20.2.19 delete_data() [2/3]

```
Data_Type&& counters_ delete_data (
            std::move(counters_.delete_data)  )
```

8.20.2.20 delete_data() [3/3]

```
Data_Type Data_Type delete_data (
            true  )
```

8.20.2.21 delete_to_be_deleted() [1/2]

```
Data_Type&& counters_ delete_to_be_deleted (
            std::move(counters_.delete_to_be_deleted)  )
```

Definition at line 201 of file counters-meat.hpp.

8.20.2.22 delete_to_be_deleted() [2/2]

```
Data_Type Data_Type delete_to_be_deleted (
            true  )
```

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

8.20.2.23 desc()

```
Data_Type&& counter_ desc (
            std::move(counter_.desc)  )
```

Move constructor.

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

8.20.2.24 init_fun() [1/3]

```
return init_fun (
            Array ,
            i ,
            j ,
            data  )
```

8.20.2.25 init_fun() [2/3]

```
Data_Type init_fun (
            counter_.  init_fun )
```

Definition at line 15 of file counters-meat.hpp.

8.20.2.26 init_fun() [3/3]

```
Data_Type&& counter_ init_fun (
            std::move(counter_.init_fun)  )
```

8.20.2.27 name()

```
Data_Type&& counter_ name (
            std::move(counter_.name)  )
```

8.20.2.28 push_back() [1/2]

```
data push_back (
            new Counter< Array_Type, Data_Type > count_fun_, init_fun_, data_, delete_data_,
name_, desc_ )
```

8.20.2.29 push_back() [2/2]

```
data push_back (
            new Counter< Array_Type, Data_Type > counter )
```

8.20.2.30 to_be_deleted() [1/2]

```
Data_Type Data_Type to_be_deleted (
            new std::vector< uint > 0u )
```

8.20.2.31 to_be_deleted() [2/2]

```
Data_Type&& counters_ to_be_deleted (
            std::move(counters_.to_be_deleted)  )
```

8.20.3 Variable Documentation

8.20.3.1 count_fun_

```
Data_Type count_fun_
```

Definition at line 291 of file counters-meat.hpp.

8.20.3.2 counter

```
Data_Type * counter
```

Initial value:

```
{
    to_be_deleted->push_back(data->size())
```

Definition at line 273 of file counters-meat.hpp.

8.20.3.3 counter_

```
Data_Type & counter_
```

Initial value:

```
{
    if (this != &counter_) {
        this->count_fun = counter_.count_fun;
        this->init_fun = counter_.init_fun;
        if (counter_.delete_data)
        {
            this->data = new Data_Type(*counter_.data);
            this->delete_data = true;

        } else {
            this->data = counter_.data;
            this->delete_data = false;
        }
        this->name = counter_.name;
        this->desc = counter_.desc;
    }
    return *this
```

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

8.20.3.4 data_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> Data_Type* data_
```

Definition at line 293 of file counters-meat.hpp.

8.20.3.5 delete_data_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> Data_Type bool delete_data_
```

Definition at line 294 of file counters-meat.hpp.

8.20.3.6 desc_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> Data_Type bool std::string std::string desc↩
_
```

Initial value:

```
{

    to_be_deleted->push_back(data->size())
```

Definition at line 296 of file counters-meat.hpp.

8.20.3.7 i

```
uint i
```

Definition at line 117 of file counters-meat.hpp.

8.20.3.8 init_fun_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> init_fun_
```

Definition at line 292 of file counters-meat.hpp.

8.20.3.9 j

```
uint uint j
```

Initial value:

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

Definition at line 117 of file counters-meat.hpp.

8.20.3.10 name_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> Data_Type bool std::string name_
```

Definition at line 295 of file counters-meat.hpp.

8.20.3.11 noexcept

```
Data_Type &&counters_ noexcept
```

Initial value:

```
{
    if (this != &counter_)
    {

        if (delete_data)
            delete data;

        this->data        = std::move(counter_.data);
        this->delete_data = std::move(counter_.delete_data);
        counter_.data        = nullptr;
        counter_.delete_data = false;

        this->count_fun = std::move(counter_.count_fun);
        this->init_fun = std::move(counter_.init_fun);

        this->name = std::move(counter_.name);
        this->desc = std::move(counter_.desc);
    }
    return *this
```

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

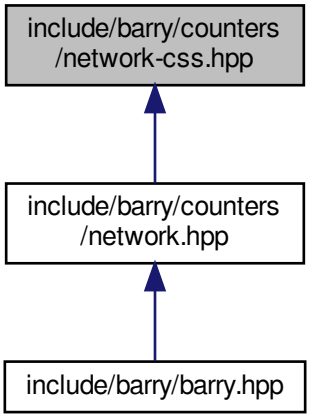
8.20.3.12 return

```
return
```

Definition at line 279 of file counters-meat.hpp.

8.21 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)

Functions

- void counter_css_partially_false_recip_commi (NetCounters *counters, uint netsize, const std::vector< uint > &end_)

  Counts errors of commission.
- void counter_css_partially_false_recip_omiss (NetCounters *counters, uint netsize, const std::vector< uint > &end_)

  Counts errors of omission.
- void counter_css_completely_false_recip_comiss (NetCounters *counters, uint netsize, const std::vector< uint > &end_)

  Counts completely false reciprocity (comission)
- void counter_css_completely_false_recip_omiss (NetCounters *counters, uint netsize, const std::vector< uint > &end_)

  Counts completely false reciprocity (omission)
- void counter_css_mixed_recip (NetCounters *counters, uint netsize, const std::vector< uint > &end_)

  Counts mixed reciprocity errors.

8.21.1 Macro Definition Documentation

8.21.1.1 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,\
            new NetCounterData({netsize, i == 0u ? netsize : end_[i-1], end_[i]}, {}),\
            true, tmpname);}
```

Definition at line 42 of file network-css.hpp.

8.21.1.2 CSS_CASE_ELSE

```
#define CSS_CASE_ELSE( )
```

Definition at line 27 of file network-css.hpp.

8.21.1.3 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.

8.21.1.4 CSS_CASE_TRUTH

```
#define CSS_CASE_TRUTH( ) if ((i < n) && (j < n))
```

Definition at line 13 of file network-css.hpp.

8.21.1.5 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.

8.21.1.6 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.

8.21.1.7 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.

8.21.1.8 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.

8.21.1.9 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.

8.21.2 Function Documentation

8.21.2.1 counter_css_completely_false_recip_comiss()

```
void counter_css_completely_false_recip_comiss (
            NetCounters * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Counts completely false reciprocity (comission)

Definition at line 160 of file network-css.hpp.

8.21.2.2 counter_css_completely_false_recip_omiss()

```
void counter_css_completely_false_recip_omiss (
            NetCounters * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Counts completely false reciprocity (omission)

Definition at line 206 of file network-css.hpp.

8.21.2.3 counter_css_mixed_recip()

```
void counter_css_mixed_recip (
            NetCounters * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Counts mixed reciprocity errors.

Definition at line 252 of file network-css.hpp.

8.21.2.4 counter_css_partially_false_recip_commi()

```
void counter_css_partially_false_recip_commi (
            NetCounters * 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 57 of file network-css.hpp.

8.21.2.5 counter_css_partially_false_recip_omiss()

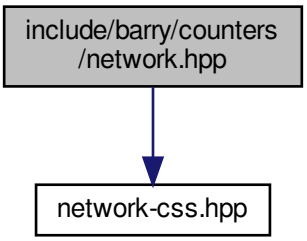
```
void counter_css_partially_false_recip_omiss (
            NetCounters * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Counts errors of omission.

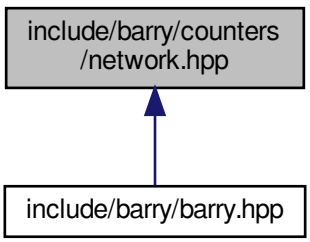
Definition at line 110 of file network-css.hpp.

8.22 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)

Macros for defining rules

- #define NETWORK_RULE(a)
- #define NETWORK_RULE_LAMBDA(a)

Typedefs

Convenient typedefs for network objects.

- typedef BArray< double, NetworkData > Network
- typedef Counter< Network, NetCounterData > NetCounter
- typedef Counters< Network, NetCounterData > NetCounters
- typedef Support< Network, NetCounterData > NetSupport
- typedef StatsCounter< Network, NetCounterData > NetStatsCounter
- typedef Model< Network, NetCounterData > NetModel
- typedef Rule< Network, bool > NetRule
- typedef Rules< Network, bool > NetRules

Functions

- void counter_edges (NetCounters *counters)

  Number of edges.
- void counter_isolates (NetCounters *counters)

  Number of isolated vertices.
- void counter_mutual (NetCounters *counters)

  Number of mutual ties.
- void counter_istar2 (NetCounters *counters)
- void counter_ostar2 (NetCounters *counters)
- void counter_ttriads (NetCounters *counters)
- void counter_ctriads (NetCounters *counters)
- void counter_density (NetCounters *counters)
- void counter_idegree15 (NetCounters *counters)

- void counter_odegree15 (NetCounters *counters)
- void counter_absdiff (NetCounters *counters, uint attr_id, double alpha=1.0)

  Sum of absolute attribute difference between ego and alter.
- void counter_diff (NetCounters *counters, uint attr_id, double alpha=1.0, double tail_head=true)

  Sum of attribute difference between ego and alter to pow(alpha)
- NETWORK_COUNTER (init_single_attr)
- void counter_nodeicov (NetCounters *counters, uint attr_id)
- void counter_nodeocov (NetCounters *counters, uint attr_id)
- void counter_nodecov (NetCounters *counters, uint attr_id)
- void counter_nodematch (NetCounters *counters, uint attr_id)
- void counter_idegree (NetCounters *counters, std::vector< uint > d)

  Counts number of vertices with a given in-degree.
- void counter_odegree (NetCounters *counters, std::vector< uint > d)

  Counts number of vertices with a given out-degree.
- void counter_degree (NetCounters *counters, std::vector< uint > d)

  Counts number of vertices with a given out-degree.

Rules for network models

Parameters

rules	*A pointer to a* `NetRules` *object (*`Rules<Network, bool>`*).*

- void rules_zerodiag (NetRules *rules)

  Number of edges.

8.22.1 Macro Definition Documentation

8.22.1.1 NET_C_DATA_IDX

```
#define NET_C_DATA_IDX(
              i ) (data->indices[i])
```

Definition at line 74 of file network.hpp.

8.22.1.2 NET_C_DATA_NUM

```
#define NET_C_DATA_NUM(
              i ) (data->numbers[i])
```

Definition at line 75 of file network.hpp.

8.22.1.3 NETWORK_COUNTER

```
#define NETWORK_COUNTER(
                a )
```

Value:
```
inline double (a) \
(const Network & Array, uint i, uint j, NetCounterData * data)
```

Function for definition of a network counter function

Definition at line 96 of file network.hpp.

8.22.1.4 NETWORK_COUNTER_LAMBDA

```
#define NETWORK_COUNTER_LAMBDA(
                a )
```

Value:
```
    Counter_fun_type<Network, NetCounterData> a = \
    [](const Network & Array, uint i, uint j, NetCounterData * data)
```

Lambda function for definition of a network counter function

Definition at line 99 of file network.hpp.

8.22.1.5 NETWORK_RULE

```
#define NETWORK_RULE(
                a )
```

Value:
```
inline bool (a) \
(const Network & Array, uint i, uint j, bool * data)
```

Function for definition of a network counter function

Definition at line 108 of file network.hpp.

8.22.1.6 NETWORK_RULE_LAMBDA

```
#define NETWORK_RULE_LAMBDA(
                a )
```

Value:
```
Rule_fun_type<Network, bool> a = \
[](const Network & Array, uint i, uint j, bool * data)
```

Lambda function for definition of a network counter function

Definition at line 111 of file network.hpp.

8.22.2 Typedef Documentation

8.22.2.1 NetCounter

```
typedef Counter<Network, NetCounterData > NetCounter
```

Definition at line 83 of file network.hpp.

8.22.2.2 NetCounters

```
typedef Counters< Network, NetCounterData> NetCounters
```

Definition at line 84 of file network.hpp.

8.22.2.3 NetModel

```
typedef Model<Network, NetCounterData> NetModel
```

Definition at line 87 of file network.hpp.

8.22.2.4 NetRule

```
typedef Rule<Network,bool> NetRule
```

Definition at line 88 of file network.hpp.

8.22.2.5 NetRules

```
typedef Rules<Network,bool> NetRules
```

Definition at line 89 of file network.hpp.

8.22.2.6 NetStatsCounter

```
typedef StatsCounter<Network, NetCounterData> NetStatsCounter
```

Definition at line 86 of file network.hpp.

8.22.2.7 NetSupport

```
typedef Support<Network, NetCounterData > NetSupport
```

Definition at line 85 of file network.hpp.

8.22.2.8 Network

```
typedef BArray<double, NetworkData> Network
```

Definition at line 82 of file network.hpp.

8.22.3 Function Documentation

8.22.3.1 rules_zerodiag()

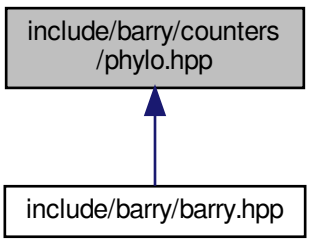
```
void rules_zerodiag (
            NetRules * rules )  [inline]
```

Number of edges.

Definition at line 853 of file network.hpp.

8.23 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 BArray< uint, NodeData > PhyloArray
- typedef Counter< PhyloArray, PhyloCounterData > PhyloCounter
- typedef Counters< PhyloArray, PhyloCounterData > PhyloCounters
- typedef Rule< PhyloArray, PhyloRuleData > PhyloRule
- typedef Rules< PhyloArray, PhyloRuleData > PhyloRules
- typedef Rule< PhyloArray, PhyloRuleDynData > PhyloRuleDyn
- typedef Rules< PhyloArray, PhyloRuleDynData > PhyloRulesDyn
- typedef Support< PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport
- typedef StatsCounter< PhyloArray, PhyloCounterData > PhyloStatsCounter
- typedef Model< PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel
- typedef PowerSet< PhyloArray, PhyloRuleData > PhyloPowerSet

Functions

- std::string get_last_name (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_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_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 rule_dyn_limit_changes (PhyloSupport *support, uint pos, uint lb, uint ub, unsigned int duplication=DEFAULT_DUPLICATIO

  Overall functional gains.

8.23.1 Macro Definition Documentation

8.23.1.1 DEFAULT_DUPLICATION

```
#define DEFAULT_DUPLICATION 1u
```

Definition at line 5 of file phylo.hpp.

8.23.1.2 DUPL_DUPL

```
#define DUPL_DUPL 1u
```

Definition at line 7 of file phylo.hpp.

8.23.1.3 DUPL_EITH

```
#define DUPL_EITH 2u
```

Definition at line 8 of file phylo.hpp.

8.23.1.4 DUPL_SPEC

```
#define DUPL_SPEC 0u
```

Definition at line 6 of file phylo.hpp.

8.23.1.5 IF_MATCHES

```
#define IF_MATCHES( )
```

Value:

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

Definition at line 19 of file phylo.hpp.

8.23.1.6 IF_NOTMATCHES

```
#define IF_NOTMATCHES( )
```

Value:

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

Definition at line 21 of file phylo.hpp.

8.23.1.7 IS_DUPLICATION

```
#define IS_DUPLICATION( ) ((DATA_AT == DUPL_DUPL) & (DPL))
```

Definition at line 16 of file phylo.hpp.

8.23.1.8 IS_EITHER

```
#define IS_EITHER( ) (DATA_AT == DUPL_EITH)
```

Definition at line 15 of file phylo.hpp.

8.23.1.9 IS_SPECIATION

```
#define IS_SPECIATION( ) ((DATA_AT == DUPL_SPEC) & (!DPL))
```

Definition at line 17 of file phylo.hpp.

8.23.1.10 MAKE_DUPL_VARS

```
#define MAKE_DUPL_VARS( )
```

Value:

```
bool DPL = Array.D()->duplication; \
unsigned int DATA_AT = data->at(0u);
```

Definition at line 11 of file phylo.hpp.

8.23.1.11 PHYLO_CHECK_MISSING

```
#define PHYLO_CHECK_MISSING( )
```

Value:

```
if (Array.D() == nullptr) \
throw std::logic_error("The array data is nullptr."); \
if (data == nullptr) \
throw std::logic_error("The counter/rule data is nullptr.")
```

Definition at line 136 of file phylo.hpp.

8.23.1.12 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 130 of file phylo.hpp.

8.23.1.13 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 133 of file phylo.hpp.

8.23.2 Typedef Documentation

8.23.2.1 PhyloArray

```
typedef BArray<uint, NodeData> PhyloArray
```

Definition at line 103 of file phylo.hpp.

8.23.2.2 PhyloCounter

```
typedef Counter<PhyloArray, PhyloCounterData > PhyloCounter
```

Definition at line 104 of file phylo.hpp.

8.23.2.3 PhyloCounters

```
typedef Counters< PhyloArray, PhyloCounterData> PhyloCounters
```

Definition at line 105 of file phylo.hpp.

8.23.2.4 PhyloModel

```
typedef Model<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel
```

Definition at line 115 of file phylo.hpp.

8.23.2.5 PhyloPowerSet

```
typedef PowerSet<PhyloArray, PhyloRuleData> PhyloPowerSet
```

Definition at line 116 of file phylo.hpp.

8.23.2.6 PhyloRule

```
typedef Rule<PhyloArray,PhyloRuleData> PhyloRule
```

Definition at line 107 of file phylo.hpp.

8.23.2.7 PhyloRuleData

```
typedef std::vector< std::pair< uint, uint > > PhyloRuleData
```

Definition at line 96 of file phylo.hpp.

8.23.2.8 PhyloRuleDyn

```
typedef Rule<PhyloArray,PhyloRuleDynData> PhyloRuleDyn
```

Definition at line 110 of file phylo.hpp.

8.23.2.9 PhyloRules

```
typedef Rules<PhyloArray,PhyloRuleData> PhyloRules
```

Definition at line 108 of file phylo.hpp.

8.23.2.10 PhyloRulesDyn

```
typedef Rules<PhyloArray,PhyloRuleDynData> PhyloRulesDyn
```

Definition at line 111 of file phylo.hpp.

8.23.2.11 PhyloStatsCounter

```
typedef StatsCounter<PhyloArray, PhyloCounterData> PhyloStatsCounter
```

Definition at line 114 of file phylo.hpp.

8.23.2.12 PhyloSupport

```
typedef Support<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport
```

Definition at line 113 of file phylo.hpp.

8.23.3 Function Documentation

8.23.3.1 get_last_name()

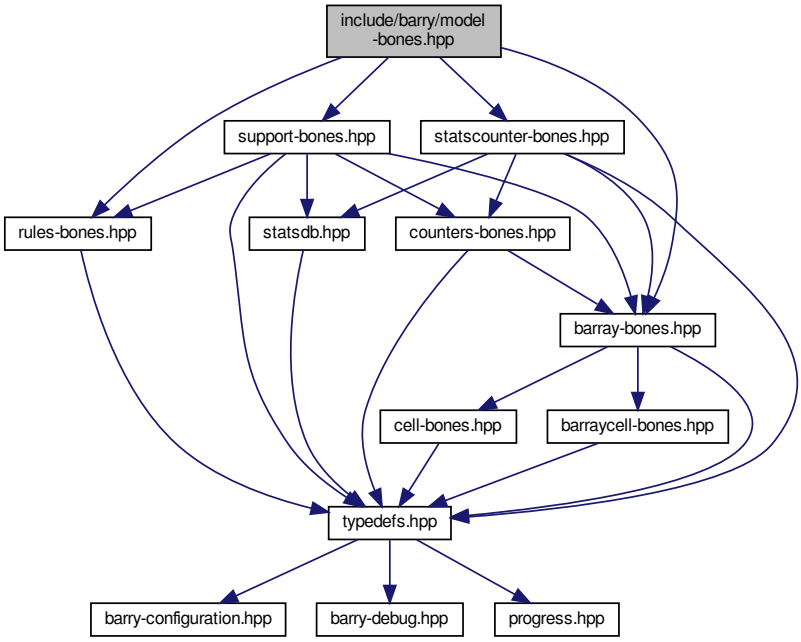
```
std::string get_last_name (
            unsigned int d )  [inline]
```

Definition at line 141 of file phylo.hpp.

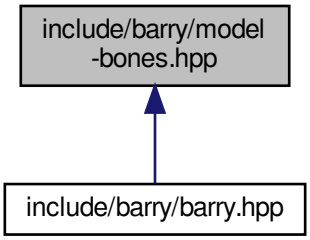
8.24 include/barry/model-bones.hpp File Reference

```
#include "barray-bones.hpp"
#include "support-bones.hpp"
#include "statscounter-bones.hpp"
#include "rules-bones.hpp"
```

Include dependency graph for model-bones.hpp:



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



Classes

- class Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >

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

Functions

- template<typename Array_Type >
  std::vector< double > keygen_default (const Array_Type &Array_)

  Array Hasher class (used for computing support)

8.24.1 Function Documentation

8.24.1.1 keygen_default()

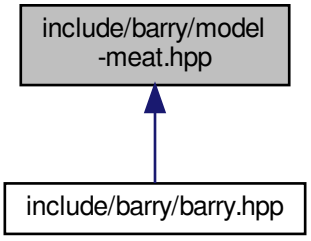
```
template<typename Array_Type >
std::vector< double > keygen_default (
            const Array_Type & Array_ )  [inline]
```

Array Hasher class (used for computing support)

Definition at line 17 of file model-bones.hpp.

8.25 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 std::vector< double > &params, const Counts_type &support)
- double likelihood_ (const std::vector< double > &target_stats, const std::vector< double > &params, const double normalizing_constant, bool log_=false)
- MODEL_TEMPLATE (, Model)()
- MODEL_TEMPLATE (, Model)(const MODEL_TYPE() &Model_)

8.25.1 Macro Definition Documentation

8.25.1.1 MODEL_TEMPLATE

```
#define MODEL_TEMPLATE(
              a,
              b )  template MODEL_TEMPLATE_ARGS() inline a MODEL_TYPE()::b
```

Definition at line 75 of file model-meat.hpp.

8.25.1.2 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 72 of file model-meat.hpp.

8.25.1.3 MODEL_TYPE

```
#define MODEL_TYPE( )
```

Value:

```
Model<Array_Type,Data_Counter_Type,Data_Rule_Type,\
Data_Rule_Dyn_Type>
```

Definition at line 69 of file model-meat.hpp.

8.25.2 Function Documentation

8.25.2.1 likelihood_()

```
double likelihood_ (
            const std::vector< double > & target_stats,
            const std::vector< double > & params,
            const double normalizing_constant,
            bool log_ = false )  [inline]
```

Definition at line 37 of file model-meat.hpp.

8.25.2.2 MODEL_TEMPLATE() [1/2]

```
MODEL_TEMPLATE (
            Model  )
```

Definition at line 79 of file model-meat.hpp.

8.25.2.3 MODEL_TEMPLATE() [2/2]

```
MODEL_TEMPLATE (
            Model  ) const &
```

Definition at line 134 of file model-meat.hpp.

8.25.2.4 update_normalizing_constant()

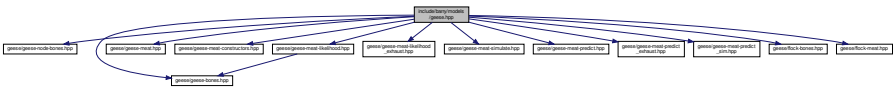
```
double update_normalizing_constant (
            const std::vector< double > & params,
            const Counts_type & support )  [inline]
```

Definition at line 11 of file model-meat.hpp.

8.26 include/barry/models/geese.hpp File Reference

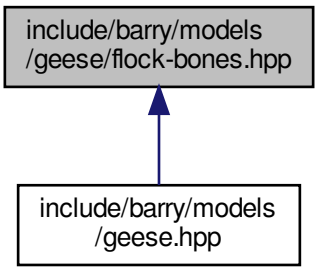
```
#include "geese/geese-node-bones.hpp"
#include "geese/geese-bones.hpp"
#include "geese/geese-meat.hpp"
#include "geese/geese-meat-constructors.hpp"
#include "geese/geese-meat-likelihood.hpp"
#include "geese/geese-meat-likelihood_exhaust.hpp"
#include "geese/geese-meat-simulate.hpp"
#include "geese/geese-meat-predict.hpp"
#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:



8.27 include/barry/models/geese/flock-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



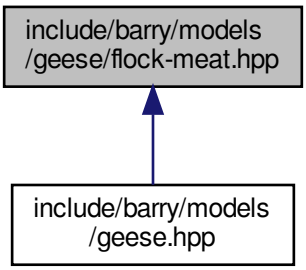
Classes

- class Flock

  A Flock is a group of Geese.

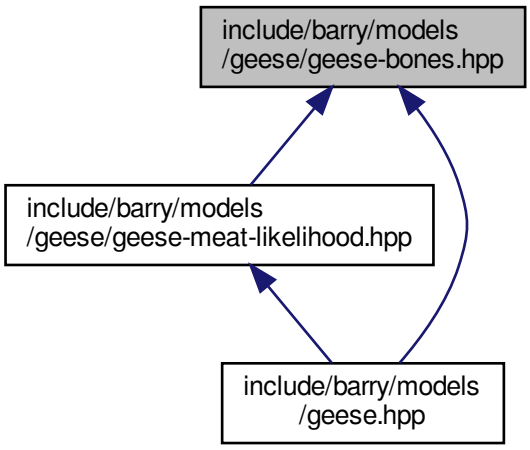
8.28 include/barry/models/geese/flock-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



8.29 include/barry/models/geese/geese-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class Geese

  Annotated Phylo Model.

Macros

- #define INITIALIZED()

Functions

- template<typename Ta , typename Tb >
  std::vector< Ta > vector_caster (const std::vector< Tb > &x)
- RULE_FUNCTION (rule_empty_free)
- std::vector< double > keygen_full (const phylocounters::PhyloArray &array)
- bool vec_diff (const std::vector< unsigned int > &s, const std::vector< unsigned int > &a)

8.29.1 Macro Definition Documentation

8.29.1.1 INITIALIZED

```
#define INITIALIZED( )
```

Value:

```
    if (!this->initialized) \
    throw std::logic_error("The model has not been initialized yet.");
```

Definition at line 22 of file geese-bones.hpp.

8.29.2 Function Documentation

8.29.2.1 keygen_full()

```
std::vector< double > keygen_full (
             const phylocounters::PhyloArray & array )  [inline]
```

Definition at line 35 of file geese-bones.hpp.

8.29.2.2 RULE_FUNCTION()

```
RULE_FUNCTION (
             rule_empty_free  )
```

Definition at line 26 of file geese-bones.hpp.

8.29.2.3 vec_diff()

```
bool vec_diff (
             const std::vector< unsigned int > & s,
             const std::vector< unsigned int > & a )  [inline]
```

Definition at line 59 of file geese-bones.hpp.

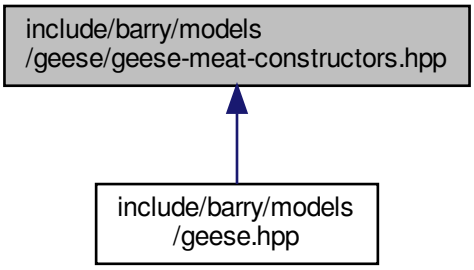
8.29.2.4 vector_caster()

```
template<typename Ta , typename Tb >
std::vector< Ta > vector_caster (
             const std::vector< Tb > & x )  [inline]
```

Definition at line 10 of file geese-bones.hpp.

8.30 include/barry/models/geese/geese-meat-constructors.hpp File Reference

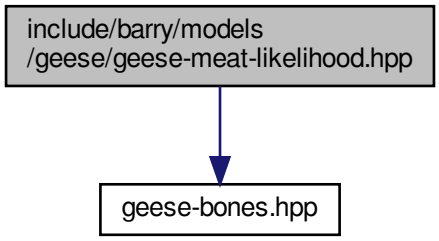
This graph shows which files directly or indirectly include this file:



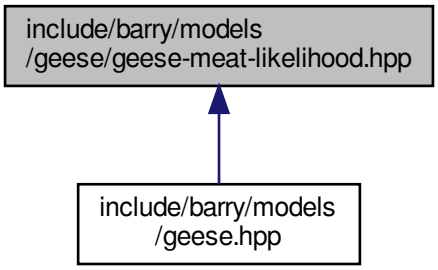
8.31 include/barry/models/geese/geese-meat-likelihood.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat-likelihood.hpp:

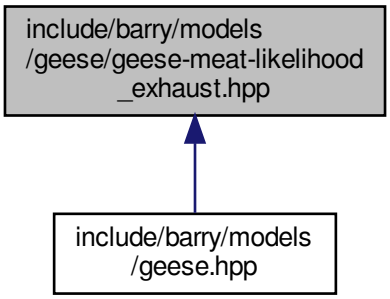


This graph shows which files directly or indirectly include this file:



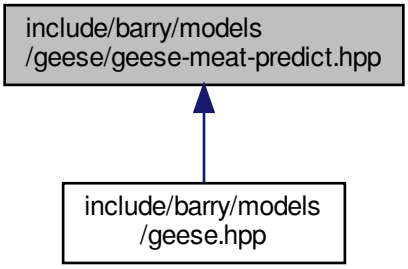
8.32 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference

This graph shows which files directly or indirectly include this file:



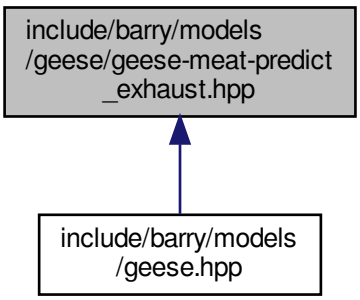
8.33 include/barry/models/geese/geese-meat-predict.hpp File Reference

This graph shows which files directly or indirectly include this file:



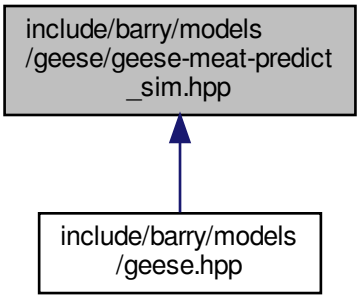
8.34 include/barry/models/geese/geese-meat-predict_exhaust.hpp File Reference

This graph shows which files directly or indirectly include this file:



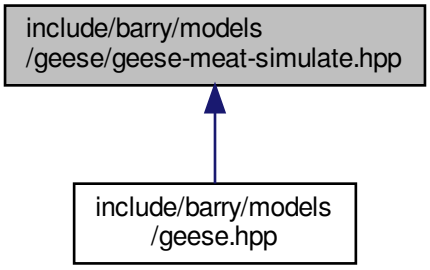
8.35 include/barry/models/geese/geese-meat-predict_sim.hpp File Reference

This graph shows which files directly or indirectly include this file:



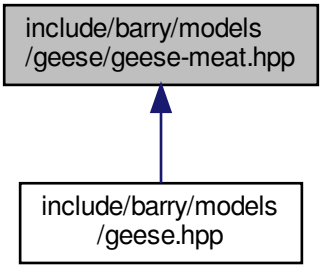
8.36 include/barry/models/geese/geese-meat-simulate.hpp File Reference

This graph shows which files directly or indirectly include this file:



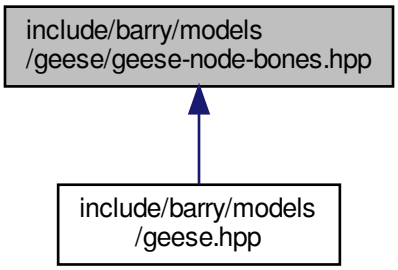
8.37 include/barry/models/geese/geese-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



8.38 include/barry/models/geese/geese-node-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

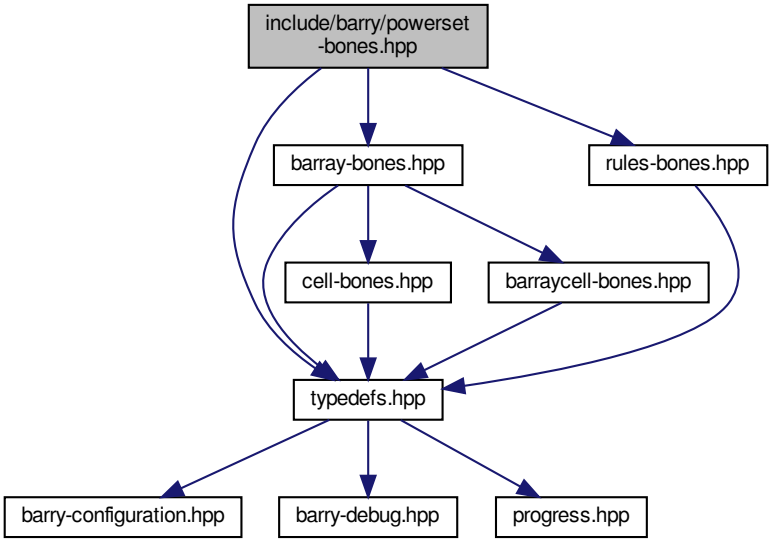
- class Node

  A single node for the model.

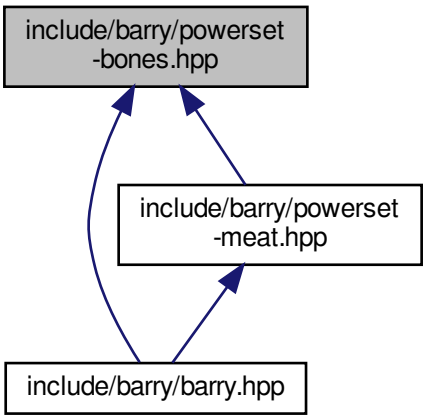
8.39 include/barry/powerset-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "barray-bones.hpp"
#include "rules-bones.hpp"
```

Include dependency graph for powerset-bones.hpp:



This graph shows which files directly or indirectly include this file:



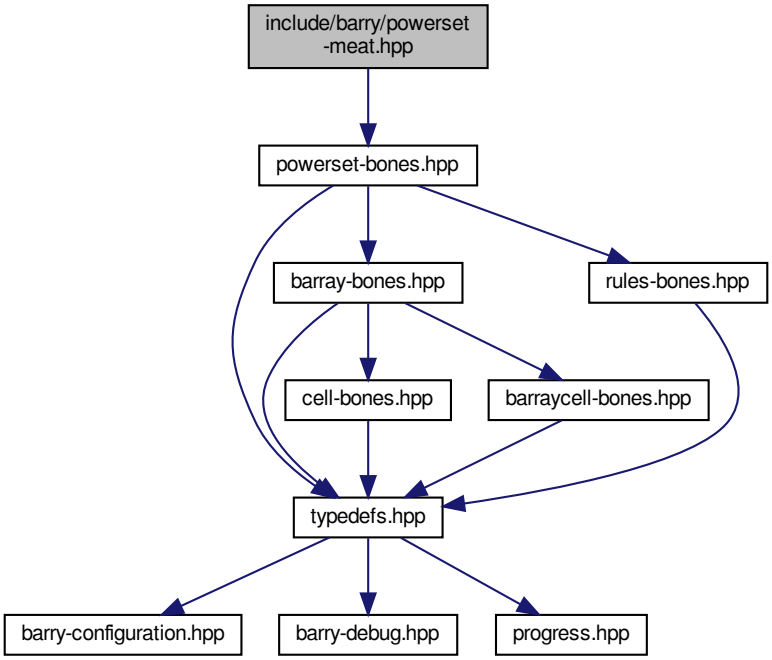
Classes

- class PowerSet< Array_Type, Data_Rule_Type >

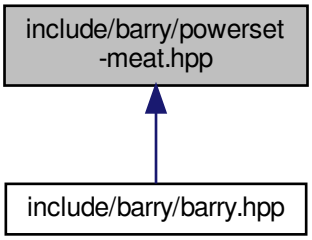
Powerset of a binary array.

8.40 include/barry/powerset-meat.hpp File Reference

`#include "powerset-bones.hpp"`
Include dependency graph for powerset-meat.hpp:

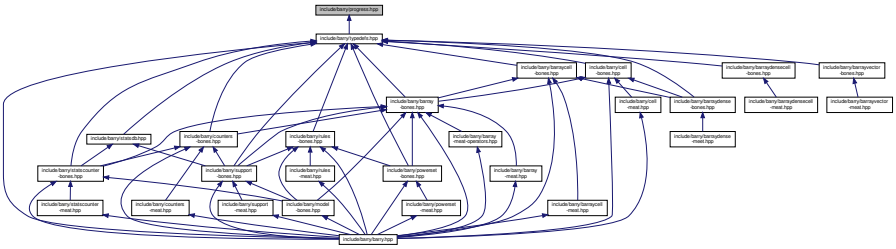


This graph shows which files directly or indirectly include this file:



8.41 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

8.41.1 Macro Definition Documentation

8.41.1.1 BARRY_PROGRESS_BAR_WIDTH

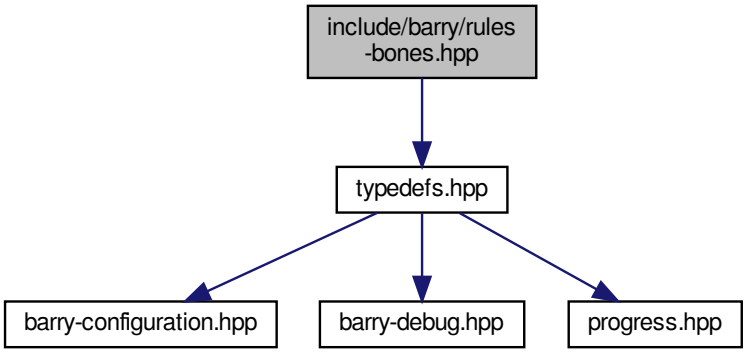
```
#define BARRY_PROGRESS_BAR_WIDTH 80
```

Definition at line 5 of file progress.hpp.

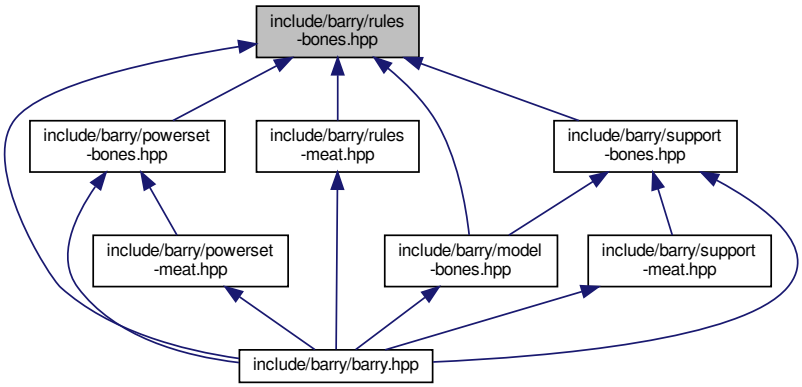
8.42 include/barry/rules-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for rules-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Rule< Array_Type, Data_Type >

  Rule for determining if a cell should be included in a sequence.
- class Rules< Array_Type, Data_Type >

  Vector of objects of class Rule.

Functions

- template<typename Array_Type , typename Data_Type >
  bool rule_fun_default (const Array_Type *array, uint i, uint j, Data_Type *dat)

8.42.1 Function Documentation

8.42.1.1 rule_fun_default()

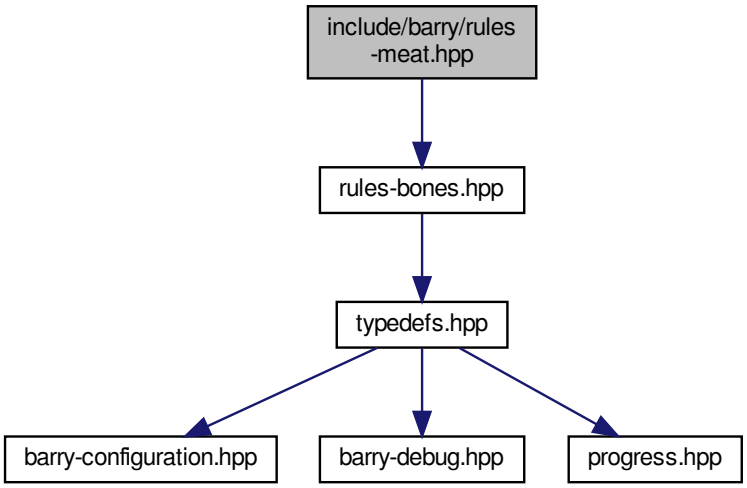
```
template<typename Array_Type , typename Data_Type >
bool rule_fun_default (
             const Array_Type * array,
             uint i,
             uint j,
             Data_Type * dat )
```

Definition at line 7 of file rules-bones.hpp.

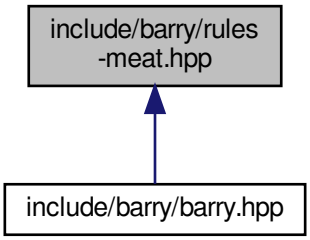
8.43 include/barry/rules-meat.hpp File Reference

```
#include "rules-bones.hpp"
```

Include dependency graph for rules-meat.hpp:



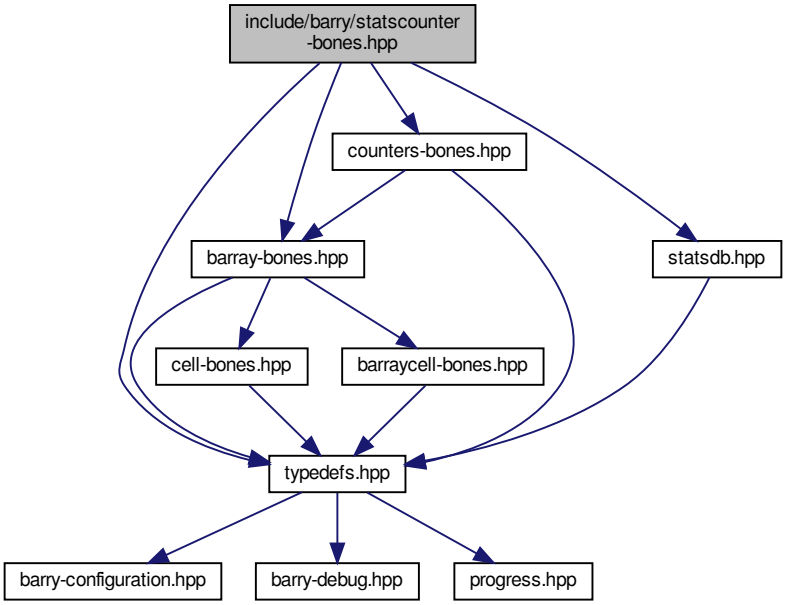
This graph shows which files directly or indirectly include this file:



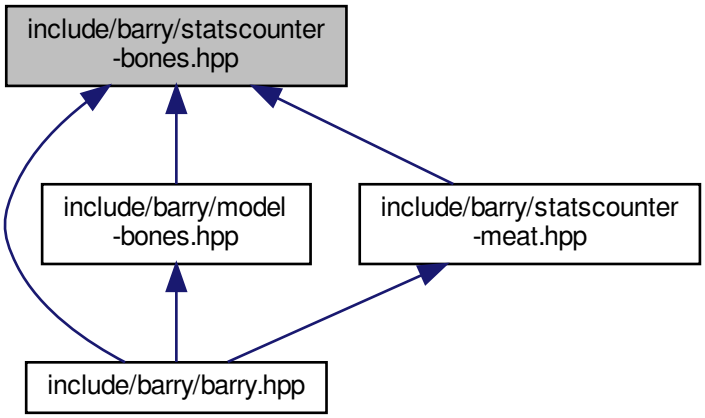
8.44 include/barry/statscounter-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "barray-bones.hpp"
#include "statsdb.hpp"
#include "counters-bones.hpp"
```

Include dependency graph for statscounter-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

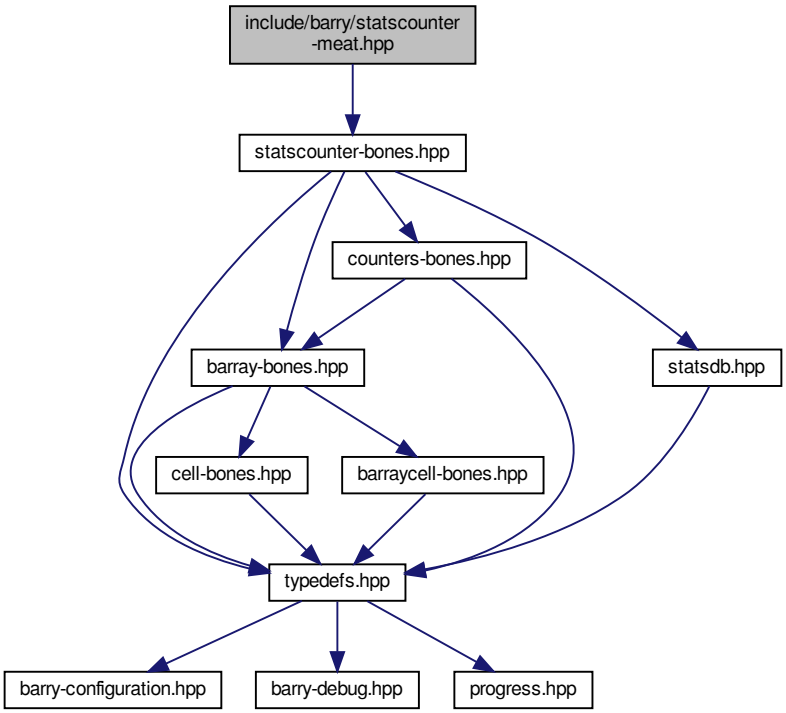
- class StatsCounter< Array_Type, Data_Type >

  Count stats for a single Array.

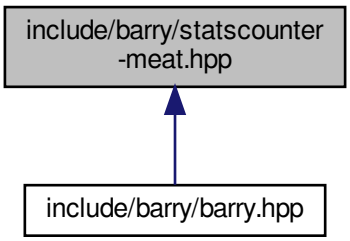
8.45 include/barry/statscounter-meat.hpp File Reference

```
#include "statscounter-bones.hpp"
```

Include dependency graph for statscounter-meat.hpp:



This graph shows which files directly or indirectly include this file:



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)()
- 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] = change_stats[pos][n]
- STATSCOUNTER_TEMPLATE (void, count_current)(uint i
- STATSCOUNTER_TEMPLATE (std::vector< double >, count_all)()
- STATSCOUNTER_TEMPLATE (std::vector< std::string >, get_names)() const
- STATSCOUNTER_TEMPLATE (std::vector< std::string >, get_descriptions)() const

Variables

- Data_Type * f_
- return
- Data_Type * counters_
- counter_deleted = true
- counters = counters_
- uint j

8.45.1 Macro Definition Documentation

8.45.1.1 STATSCOUNTER_TEMPLATE

```
#define STATSCOUNTER_TEMPLATE(
              a,
              b )  template STATSCOUNTER_TEMPLATE_ARGS() inline a STATSCOUNTER_TYPE()::b
```

Definition at line 10 of file statscounter-meat.hpp.

8.45.1.2 STATSCOUNTER_TEMPLATE_ARGS

```
template STATSCOUNTER_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

Definition at line 8 of file statscounter-meat.hpp.

8.45.1.3 STATSCOUNTER_TYPE

```
template Data_Type * STATSCOUNTER_TYPE( ) StatsCounter<Array_Type,Data_Type>
```

Definition at line 6 of file statscounter-meat.hpp.

8.45.2 Function Documentation

8.45.2.1 for()

```
for (
            uint n = 0u;n< counters->size();++n ) = change_stats[pos][n]
```

Definition at line 134 of file support-meat.hpp.

8.45.2.2 resize()

```
current_stats resize (
            counters-> size(),
            0.  0 )
```

8.45.2.3 STATSCOUNTER_TEMPLATE() [1/9]

```
STATSCOUNTER_TEMPLATE (
            ~ StatsCounter )
```

Definition at line 13 of file statscounter-meat.hpp.

8.45.2.4 STATSCOUNTER_TEMPLATE() [2/9]

```
STATSCOUNTER_TEMPLATE (
            std::vector< double > ,
            count_all  )
```

Definition at line 91 of file statscounter-meat.hpp.

8.45.2.5 STATSCOUNTER_TEMPLATE() [3/9]

```
STATSCOUNTER_TEMPLATE (
            std::vector< std::string > ,
            get_descriptions  ) const
```

Definition at line 171 of file statscounter-meat.hpp.

8.45.2.6 STATSCOUNTER_TEMPLATE() [4/9]

```
STATSCOUNTER_TEMPLATE (
            std::vector< std::string > ,
            get_names  ) const
```

Definition at line 166 of file statscounter-meat.hpp.

8.45.2.7 STATSCOUNTER_TEMPLATE() [5/9]

```
STATSCOUNTER_TEMPLATE (
            void ,
            add_counter  )
```

8.45.2.8 STATSCOUNTER_TEMPLATE() [6/9]

```
STATSCOUNTER_TEMPLATE (
            void ,
            count_current  )
```

8.45.2.9 STATSCOUNTER_TEMPLATE() [7/9]

```
STATSCOUNTER_TEMPLATE (
            void ,
            count_init  )
```

8.45.2.10 STATSCOUNTER_TEMPLATE() [8/9]

```
STATSCOUNTER_TEMPLATE (
            void ,
            reset_array  ) const
```

Definition at line 20 of file statscounter-meat.hpp.

8.45.2.11 STATSCOUNTER_TEMPLATE() [9/9]

```
STATSCOUNTER_TEMPLATE (
            void ,
            set_counters  )
```

8.45.3 Variable Documentation

8.45.3.1 counter_deleted

```
counter_deleted = true
```

Definition at line 52 of file statscounter-meat.hpp.

8.45.3.2 counters

```
counters = counters_
```

Definition at line 53 of file statscounter-meat.hpp.

8.45.3.3 counters_

```
Data_Type* counters_
```

Initial value:

```
{

    if (!counter_deleted)
        delete counters
```

Definition at line 46 of file statscounter-meat.hpp.

8.45.3.4 f_

```
Data_Rule_Dyn_Type f_
```

Initial value:

```
{

    counters->add_counter(f_)
```

Definition at line 29 of file statscounter-meat.hpp.

8.45.3.5 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 59 of file statscounter-meat.hpp.

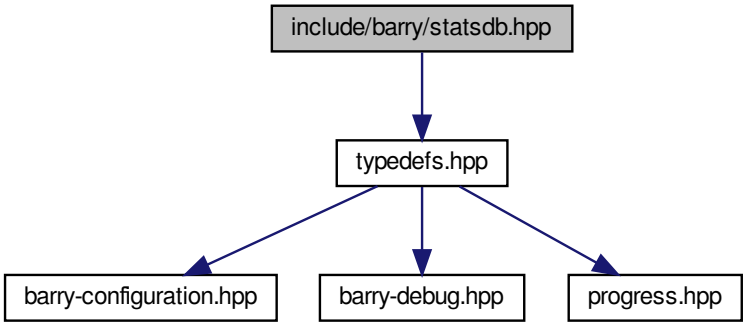
8.45.3.6 return

```
return
```

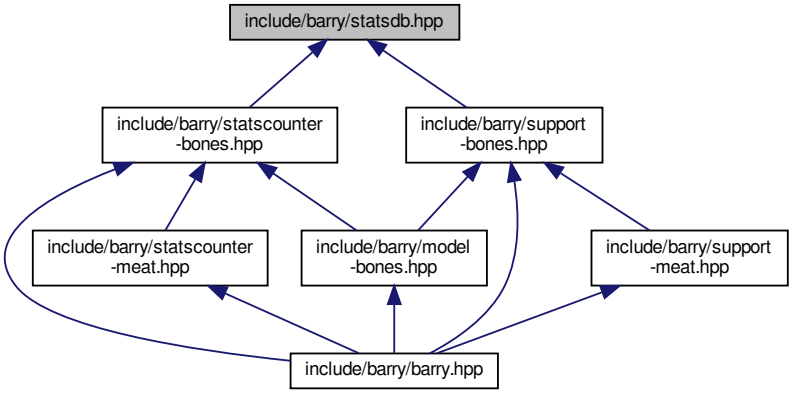
Definition at line 33 of file statscounter-meat.hpp.

8.46 include/barry/statsdb.hpp File Reference

```
#include "typedefs.hpp"
```
Include dependency graph for statsdb.hpp:



This graph shows which files directly or indirectly include this file:



Classes

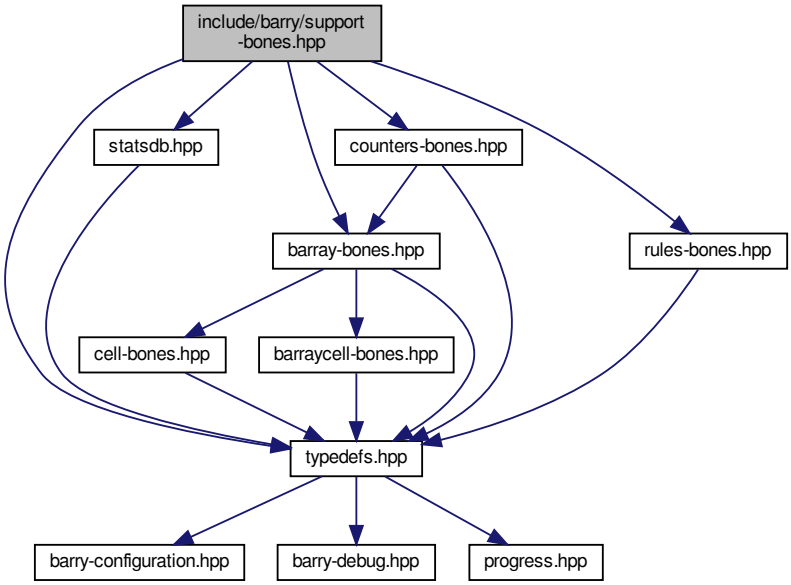
- class FreqTable< T >

  Database of statistics.

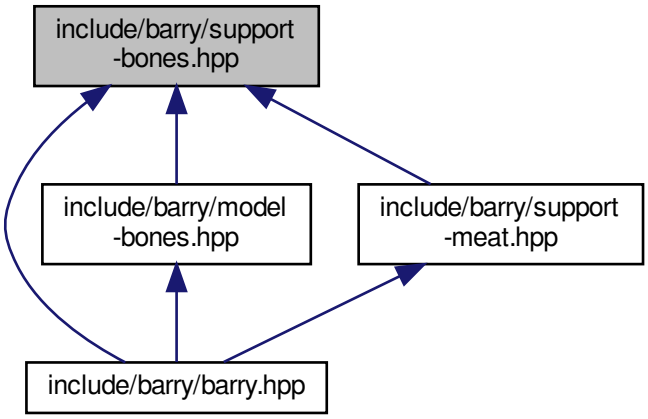
8.47 include/barry/support-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "barray-bones.hpp"
#include "statsdb.hpp"
#include "counters-bones.hpp"
#include "rules-bones.hpp"
```

Include dependency graph for support-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

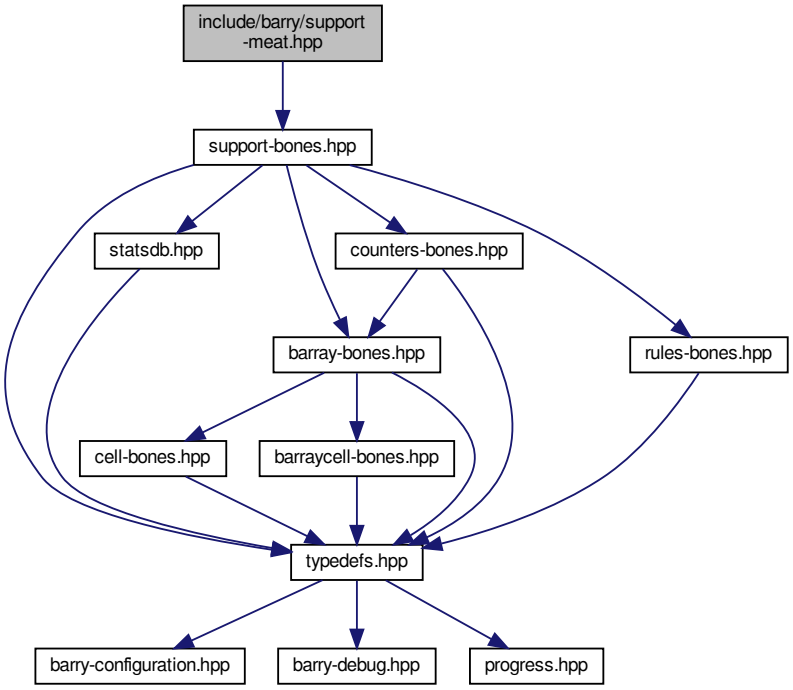
- class Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >

  Compute the support of sufficient statistics.

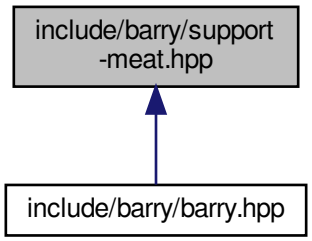
8.48 include/barry/support-meat.hpp File Reference

```
#include "support-bones.hpp"
```

Include dependency graph for support-meat.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define BARRY_SUPPORT_MEAT_HPP 1
- #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)(uint pos
- calc_backend (pos+1u, array_bank, stats_bank)
- EmptyArray insert_cell (cfree.first, cfree.second, EmptyArray.default_val().value, false, false)
- for (uint n=0u;n< counters->size();++n)
- if (rules_dyn->size() > 0u)
- if (array_bank !=nullptr) array_bank -> push_back(EmptyArray)
- if (stats_bank !=nullptr) stats_bank -> push_back(current_stats)
- EmptyArray rm_cell (cfree.first, cfree.second, 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 (Counts_type, get_counts)() const
- SUPPORT_TEMPLATE (const MapVec_type<> *, get_counts_ptr)() const
- SUPPORT_TEMPLATE (std::vector< double > *, get_current_stats)()
- SUPPORT_TEMPLATE (void, print)() const
- SUPPORT_TEMPLATE (const FreqTable<> &, get_data)() const

Variables

- std::vector< Array_Type > * array_bank
- std::vector< Array_Type > std::vector< std::vector< double > > * stats_bank
- const std::pair< uint, uint > & cfree = coordinates_free[pos]
- else
- 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_

8.48.1 Macro Definition Documentation

8.48.1.1 BARRY_SUPPORT_MEAT_HPP

```
#define BARRY_SUPPORT_MEAT_HPP 1
```

Definition at line 4 of file support-meat.hpp.

8.48.1.2 SUPPORT_TEMPLATE

```
#define SUPPORT_TEMPLATE(
              a,
              b )
```

Value:

```
template SUPPORT_TEMPLATE_ARGS() \
inline a SUPPORT_TYPE()::b
```

Definition at line 12 of file support-meat.hpp.

8.48.1.3 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 6 of file support-meat.hpp.

8.48.1.4 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 9 of file support-meat.hpp.

8.48.2 Function Documentation

8.48.2.1 calc_backend()

```
calc_backend (
            pos+ 1u,
            array_bank ,
            stats_bank  )
```

8.48.2.2 for()

```
for (
            uint n = 0u;n< counters->size();++n )
```

Definition at line 134 of file support-meat.hpp.

8.48.2.3 if() [1/3]

```
if (
            array_bank !    = nullptr ) -> push_back(EmptyArray)
```

8.48.2.4 if() [2/3]

```
if (
            rules_dyn-> size(),
            0u  )
```

Definition at line 158 of file support-meat.hpp.

8.48.2.5 if() [3/3]

```
if (
            stats_bank !    = nullptr ) -> push_back(current_stats)
```

8.48.2.6 insert_cell()

```
EmptyArray insert_cell (
            cfree.  first,
            cfree.  second,
            EmptyArray.default_val().  value,
            false ,
            false  )
```

8.48.2.7 rm_cell()

```
EmptyArray rm_cell (
            cfree.  first,
            cfree.  second,
            false ,
            false  )
```

8.48.2.8 SUPPORT_TEMPLATE() [1/17]

```
SUPPORT_TEMPLATE (
            bool ,
            eval_rules_dyn  ) const
```

Definition at line 330 of file support-meat.hpp.

8.48.2.9 SUPPORT_TEMPLATE() [2/17]

```
SUPPORT_TEMPLATE (
            const FreqTable<> & ,
            get_data  ) const
```

Definition at line 379 of file support-meat.hpp.

8.48.2.10 SUPPORT_TEMPLATE() [3/17]

```
SUPPORT_TEMPLATE (
            const MapVec_type<> * ,
            get_counts_ptr  ) const
```

Definition at line 358 of file support-meat.hpp.

8.48.2.11 SUPPORT_TEMPLATE() [4/17]

```
SUPPORT_TEMPLATE (
            Counts_type ,
            get_counts  ) const
```

Definition at line 352 of file support-meat.hpp.

8.48.2.12 SUPPORT_TEMPLATE() [5/17]

```
SUPPORT_TEMPLATE (
            std::vector< double > * ,
            get_current_stats  )
```

Definition at line 364 of file support-meat.hpp.

8.48.2.13 SUPPORT_TEMPLATE() [6/17]

```
SUPPORT_TEMPLATE (
            void ,
            add_counter  )
```

8.48.2.14 SUPPORT_TEMPLATE() [7/17]

```
SUPPORT_TEMPLATE (
            void ,
            add_rule  )
```

8.48.2.15 SUPPORT_TEMPLATE() [8/17]

```
SUPPORT_TEMPLATE (
            void ,
            add_rule_dyn  )
```

8.48.2.16 SUPPORT_TEMPLATE() [9/17]

```
SUPPORT_TEMPLATE (
            void ,
            calc  )
```

Definition at line 207 of file support-meat.hpp.

8.48.2.17 SUPPORT_TEMPLATE() [10/17]

```
SUPPORT_TEMPLATE (
            void ,
            calc_backend  )
```

8.48.2.18 SUPPORT_TEMPLATE() [11/17]

```
SUPPORT_TEMPLATE (
            void ,
            init_support  )
```

Definition at line 15 of file support-meat.hpp.

8.48.2.19 SUPPORT_TEMPLATE() [12/17]

```
SUPPORT_TEMPLATE (
            void ,
            print  ) const
```

Definition at line 368 of file support-meat.hpp.

8.48.2.20 SUPPORT_TEMPLATE() [13/17]

```
SUPPORT_TEMPLATE (
            void ,
            reset_array  )
```

Definition at line 91 of file support-meat.hpp.

8.48.2.21 SUPPORT_TEMPLATE() [14/17]

```
SUPPORT_TEMPLATE (
            void ,
            reset_array  ) const &
```

Definition at line 97 of file support-meat.hpp.

8.48.2.22 SUPPORT_TEMPLATE() [15/17]

```
SUPPORT_TEMPLATE (
            void ,
            set_counters  )
```

8.48.2.23 SUPPORT_TEMPLATE() [16/17]

```
SUPPORT_TEMPLATE (
            void ,
            set_rules  )
```

8.48.2.24 SUPPORT_TEMPLATE() [17/17]

```
SUPPORT_TEMPLATE (
            void ,
            set_rules_dyn  )
```

8.48.3 Variable Documentation

8.48.3.1 array_bank

```
std::vector< Array_Type >* array_bank
```

Definition at line 109 of file support-meat.hpp.

8.48.3.2 cfree

```
const std::pair<uint,uint>& cfree = coordinates_free[pos]
```

Definition at line 123 of file support-meat.hpp.

8.48.3.3 counters

```
counters = counters_
```

Definition at line 258 of file support-meat.hpp.

8.48.3.4 counters_

```
Data_Counter_Type* counters_
```

Initial value:

```
{

    if (delete_counters)
        delete counters
```

Definition at line 251 of file support-meat.hpp.

8.48.3.5 delete_counters

```
delete_counters = false
```

Definition at line 257 of file support-meat.hpp.

8.48.3.6 delete_rules

```
delete_rules = false
```

Definition at line 291 of file support-meat.hpp.

8.48.3.7 delete_rules_dyn

```
delete_rules_dyn = false
```

Definition at line 323 of file support-meat.hpp.

8.48.3.8 else

```
else
```

Initial value:

```
{
        data.add(current_stats)
```

Definition at line 176 of file support-meat.hpp.

8.48.3.9 f_

```
Data_Rule_Dyn_Type f_
```

Initial value:

```
{
    counters->add_counter(f_)
```

Definition at line 233 of file support-meat.hpp.

8.48.3.10 return

```
return
```

Definition at line 203 of file support-meat.hpp.

8.48.3.11 rules

```
rules = rules_
```

Definition at line 292 of file support-meat.hpp.

8.48.3.12 rules_

```
Data_Rule_Dyn_Type * rules_
```

Initial value:

```
{
    if (delete_rules)
        delete rules
```

Definition at line 285 of file support-meat.hpp.

8.48.3.13 rules_dyn

```
rules_dyn = rules_
```

Definition at line 324 of file support-meat.hpp.

8.48.3.14 stats_bank

```
std::vector< Array_Type > std::vector< std::vector< double > >* stats_bank
```

Initial value:

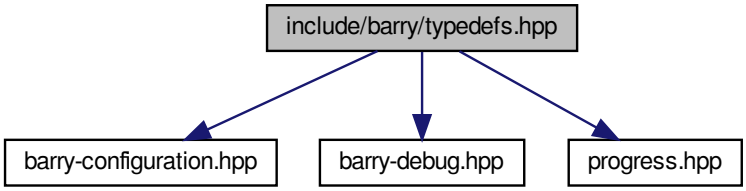
```
{

    if (pos >= coordinates_free.size())
        return
```

Definition at line 110 of file support-meat.hpp.

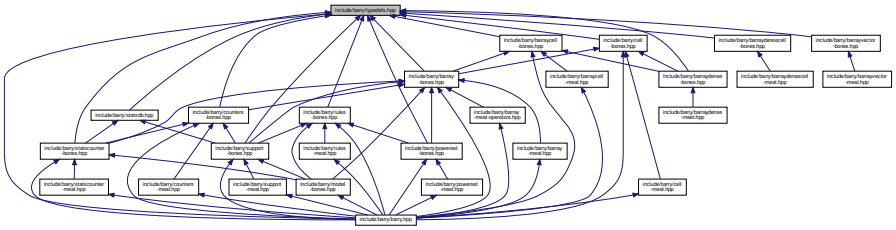
8.49 include/barry/typedefs.hpp File Reference

```
#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 Counter_fun_type = std::function< double(const Array_Type &, uint, uint, Data_Type *)>

  Counter and rule functions.
- template<typename Array_Type , typename Data_Type >
  using Rule_fun_type = std::function< bool(const Array_Type &, uint, uint, Data_Type *)>

Functions

- template<typename T >
  T vec_inner_prod (const std::vector< T > &a, const std::vector< T > &b)
- template<typename T >
  bool vec_equal (const std::vector< T > &a, const std::vector< T > &b)

  Compares if -a- and -b- are equal.
- template<typename T >
  bool vec_equal_approx (const std::vector< T > &a, const std::vector< T > &b, double eps=1e-10)

Variables

- const int CHECK::BOTH = -1
- const int CHECK::NONE = 0
- const int CHECK::ONE = 1
- const int CHECK::TWO = 2
- const int EXISTS::BOTH = -1
- const int EXISTS::NONE = 0
- const int EXISTS::ONE = 1
- const int EXISTS::TWO = 1
- const int EXISTS::UKNOWN = -1
- const int EXISTS::AS_ZERO = 0
- const int EXISTS::AS_ONE = 1

8.49.1 Typedef Documentation

8.49.1.1 Col_type

```
template<typename Cell_Type >
using Col_type = Map< uint, Cell<Cell_Type>* >
```

Definition at line 57 of file typedefs.hpp.

8.49.1.2 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 129 of file typedefs.hpp.

8.49.1.3 Counts_type

```
typedef std::vector< std::pair< std::vector<double>, uint > > Counts_type
```

Definition at line 50 of file typedefs.hpp.

8.49.1.4 MapVec_type

```
template<typename Ta = double, typename Tb = uint>
using MapVec_type = std::unordered_map< std::vector< Ta >, Tb, vecHasher<Ta> >
```

Definition at line 111 of file typedefs.hpp.

8.49.1.5 Row_type

```
template<typename Cell_Type >
using Row_type = Map< uint, Cell<Cell_Type> >
```

Definition at line 54 of file typedefs.hpp.

8.49.1.6 Rule_fun_type

```
template<typename Array_Type , typename Data_Type >
using Rule_fun_type = std::function<bool(const Array_Type &, uint, uint, Data_Type *)>
```

Definition at line 132 of file typedefs.hpp.

8.49.1.7 uint

```
typedef unsigned int uint
```

Definition at line 16 of file typedefs.hpp.

8.49.2 Function Documentation

8.49.2.1 vec_equal()

```
template<typename T >
bool vec_equal (
            const std::vector< T > & a,
            const std::vector< T > & b )  [inline]
```

Compares if -a- and -b- are equal.

Parameters

a,b	Two vectors of the same length

Returns

`true` if all elements are equal.

Definition at line 143 of file typedefs.hpp.

8.49.2.2 vec_equal_approx()

```
template<typename T >
bool vec_equal_approx (
             const std::vector< T > & a,
             const std::vector< T > & b,
             double eps = 1e-10 )  [inline]
```

Definition at line 161 of file typedefs.hpp.

8.49.2.3 vec_inner_prod()

```
template<typename T >
T vec_inner_prod (
             const std::vector< T > & a,
             const std::vector< T > & b )  [inline]
```

Definition at line 181 of file typedefs.hpp.

8.50 README.md File Reference

Index

~BArray
BArray< Cell_Type, Data_Type >, 33
~BArrayCell
BArrayCell< Cell_Type, Data_Type >, 44
~BArrayCell_const
BArrayCell_const< Cell_Type, Data_Type >, 46
~BArrayDense
BArrayDense< Cell_Type, Data_Type >, 52
~BArrayDenseCell
BArrayDenseCell< Cell_Type, Data_Type >, 63
~BArrayDenseCell_const
BArrayDenseCell_const< Cell_Type, Data_Type >, 65
~BArrayVector
BArrayVector< Cell_Type, Data_Type >, 68
~BArrayVector_const
BArrayVector_const< Cell_Type, Data_Type >, 72
~Cell
Cell< Cell_Type >, 76
~ConstBArrayRowIter
ConstBArrayRowIter< Cell_Type, Data_Type >, 80
~Counter
Counter< Array_Type, Data_Type >, 83
~Counters
Counters< Array_Type, Data_Type >, 87
~Entries
Entries< Cell_Type >, 92
~Flock
Flock, 94
~FreqTable
FreqTable< T >, 100
~Geese
Geese, 105
~Model
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 117
~NetCounterData
NetCounterData, 127
~NetworkData
NetworkData, 130
~Node
Node, 133
~PhyloRuleDynData
PhyloRuleDynData, 141
~PowerSet
PowerSet< Array_Type, Data_Rule_Type >, 145
~Progress
Progress, 150
~Rule
Rule< Array_Type, Data_Type >, 151
~Rules
Rules< Array_Type, Data_Type >, 153
~StatsCounter
StatsCounter< Array_Type, Data_Type >, 157
~Support
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 162

add
barray-meat.hpp, 187
barraydense-meet.hpp, 207
Cell< Cell_Type >, 77
FreqTable< T >, 100
add_array
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 117
add_counter
Counters< Array_Type, Data_Type >, 88
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 118
StatsCounter< Array_Type, Data_Type >, 157
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 162
add_data
Flock, 94
add_rule
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 118, 119
PowerSet< Array_Type, Data_Rule_Type >, 145
Rules< Array_Type, Data_Type >, 154
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 163
add_rule_dyn
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 119
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 163
annotations
Node, 134
ans

barray-meat.hpp, 178, 187
barraydense-meet.hpp, 199, 208
Array
ConstBArrayRowIter< Cell_Type, Data_Type >, 80
array
Node, 134
Array_
barray-meat.hpp, 187
array_bank
support-meat.hpp, 288
arrays
Node, 134
AS_ONE
EXISTS, 27
as_vector
FreqTable< T >, 100
AS_ZERO
EXISTS, 27
at
PhyloCounterData, 139

BArray
BArray< Cell_Type, Data_Type >, 32, 33
BArray< Cell_Type, Data_Type >, 29
∼BArray, 33
BArray, 32, 33
BArrayCell< Cell_Type, Data_Type >, 42
BArrayCell_const< Cell_Type, Data_Type >, 42
clear, 33
col, 33
D, 34
default_val, 34
flush_data, 34
get_cell, 34
get_col_vec, 34, 35
get_entries, 35
get_row_vec, 35
insert_cell, 35, 36
is_empty, 36
ncol, 36
nnozero, 36
nrow, 37
operator∗=, 37
operator(), 37
operator+=, 37, 38
operator-=, 38
operator/=, 38
operator=, 38, 39
operator==, 39
out_of_range, 39
print, 39
reserve, 39
resize, 39
rm_cell, 40
row, 40
set_data, 40
swap_cells, 40
swap_cols, 41
swap_rows, 41
toggle_cell, 41
toggle_lock, 41
transpose, 41
visited, 42
zero_col, 42
zero_row, 42
barray-bones.hpp
BARRAY_BONES_HPP, 172
barray-meat-operators.hpp
BARRY_BARRAY_MEAT_OPERATORS_HPP, 174
checkdim_, 174
COL, 174
ROW, 174
barray-meat.hpp
add, 187
ans, 178, 187
Array_, 187
BARRAY_TEMPLATE, 177–182
BARRAY_TEMPLATE_ARGS, 177
BARRAY_TYPE, 177
check_bounds, 187
check_exists, 187
COL, 178, 182
col0, 188
const, 188
copy_data, 188
data, 188
delete_data, 188
delete_data_, 189
else, 189
false, 189
first, 189
for, 182, 183
i1, 189
if, 183–185
j, 190
j0, 190
j1, 190
M, 186, 190
M_, 190
N, 191
NCells, 191
report, 191
resize, 186
return, 186, 191
ROW, 178, 186
row0, 191
search, 192
source, 192
target, 192
v, 192
value, 192
BARRAY_BONES_HPP
barray-bones.hpp, 172
BARRAY_TEMPLATE
barray-meat.hpp, 177–182
BARRAY_TEMPLATE_ARGS

barray-meat.hpp, 177
BARRAY_TYPE
barray-meat.hpp, 177
BArrayCell
BArrayCell< Cell_Type, Data_Type >, 43
BArrayCell< Cell_Type, Data_Type >, 43
~BArrayCell, 44
BArray< Cell_Type, Data_Type >, 42
BArrayCell, 43
BArrayDense< Cell_Type, Data_Type >, 61
operator Cell_Type, 44
operator*=, 44
operator+=, 44
operator-=, 44
operator/=, 45
operator=, 45
operator==, 45
BArrayCell_const
BArrayCell_const< Cell_Type, Data_Type >, 46
BArrayCell_const< Cell_Type, Data_Type >, 45
~BArrayCell_const, 46
BArray< Cell_Type, Data_Type >, 42
BArrayCell_const, 46
BArrayDense< Cell_Type, Data_Type >, 61
operator Cell_Type, 46
operator!=, 47
operator<, 47
operator<=, 47
operator>, 47
operator>=, 48
operator==, 47
BArrayDense
BArrayDense< Cell_Type, Data_Type >, 51, 52
BArrayDense< Cell_Type, Data_Type >, 48
~BArrayDense, 52
BArrayCell< Cell_Type, Data_Type >, 61
BArrayCell_const< Cell_Type, Data_Type >, 61
BArrayDense, 51, 52
clear, 52
col, 52
D, 53
default_val, 53
get_cell, 53
get_col_vec, 53
get_entries, 54
get_row_vec, 54
insert_cell, 54, 55
is_empty, 55
ncol, 55
nnozero, 55
nrow, 55
operator*=, 56
operator(), 56
operator+=, 56
operator-=, 57
operator/=, 57
operator=, 57
operator==, 58
out_of_range, 58
print, 58
reserve, 58
resize, 58
rm_cell, 58
row, 59
set_data, 59
swap_cells, 59
swap_cols, 59
swap_rows, 60
toggle_cell, 60
toggle_lock, 60
transpose, 60
visited, 61
zero_col, 60
zero_row, 61
barraydense-meet.hpp
add, 207
ans, 199, 208
BARRY_BARRAYDENSE_MEAT_HPP, 198
BDENSE_TEMPLATE, 198–204
BDENSE_TEMPLATE_ARGS, 198
BDENSE_TYPE, 198
check_bounds, 208
check_exists, 208
COL, 198, 204
col0, 208
const, 209
copy_data, 209
data, 209
delete_data, 209
delete_data_, 209
else, 210
false, 210
first, 210
for, 204
i1, 210
if, 204–207
j, 210
j0, 211
j1, 211
M, 207, 211
M_, 211
N, 211
NCells, 212
POS, 199
report, 212
resize, 207
return, 212
ROW, 199, 207
row0, 212
source, 212
target, 212
v, 213
value, 213
ZERO_CELL, 199
BArrayDenseCell
BArrayDenseCell< Cell_Type, Data_Type >, 62

BArrayDenseCell< Cell_Type, Data_Type >, 62
~BArrayDenseCell, 63
BArrayDenseCell, 62
operator Cell_Type, 63
operator*=, 63
operator+=, 63
operator-=, 63
operator/=, 64
operator=, 64
operator==, 64
barraydensecell-meat.hpp
BARRY_BARRAYDENSECELL_MEAT_HPP, 215
POS, 215
BArrayDenseCell_const
BArrayDenseCell_const< Cell_Type, Data_Type >, 65
BArrayDenseCell_const< Cell_Type, Data_Type >, 64
~BArrayDenseCell_const, 65
BArrayDenseCell_const, 65
operator Cell_Type, 65
operator!=, 66
operator<, 66
operator<=, 66
operator>, 66
operator>=, 67
operator==, 66
BArrayVector
BArrayVector< Cell_Type, Data_Type >, 68
BArrayVector< Cell_Type, Data_Type >, 67
~BArrayVector, 68
BArrayVector, 68
begin, 68
end, 69
is_col, 69
is_row, 69
operator std::vector< Cell_Type >, 69
operator*=, 69
operator+=, 70
operator-=, 70
operator/=, 70
operator=, 70
operator==, 70
size, 71
barrayvector-meat.hpp
BARRY_BARRAYVECTOR_MEAT_HPP, 217
BArrayVector_const
BArrayVector_const< Cell_Type, Data_Type >, 72
BArrayVector_const< Cell_Type, Data_Type >, 71
~BArrayVector_const, 72
BArrayVector_const, 72
begin, 72
end, 72
is_col, 72
is_row, 72
operator std::vector< Cell_Type >, 73
operator!=, 73
operator<, 73
operator<=, 73
operator>, 74
operator>=, 74
operator==, 73
size, 74
barry, 25
barry-configuration.hpp
BARRY_CHECK_SUPPORT, 218
BARRY_ISFINITE, 218
BARRY_MAX_NUM_ELEMENTS, 218
BARRY_SAFE_EXP, 218
Map, 218
printf_barry, 218
barry-debug.hpp
BARRY_DEBUG_LEVEL, 219
barry.hpp
BARRY_HPP, 221
BARRY_VERSION, 221
COUNTER_FUNCTION, 221
COUNTER_LAMBDA, 221
RULE_FUNCTION, 221
RULE_LAMBDA, 222
barry::counters, 25
barry::counters::network, 26
barry::counters::phylo, 26
BARRY_BARRAY_MEAT_OPERATORS_HPP
barray-meat-operators.hpp, 174
BARRY_BARRAYDENSE_MEAT_HPP
barraydense-meet.hpp, 198
BARRY_BARRAYDENSECELL_MEAT_HPP
barraydensecell-meat.hpp, 215
BARRY_BARRAYVECTOR_MEAT_HPP
barrayvector-meat.hpp, 217
BARRY_CHECK_SUPPORT
barry-configuration.hpp, 218
BARRY_DEBUG_LEVEL
barry-debug.hpp, 219
BARRY_HPP
barry.hpp, 221
BARRY_ISFINITE
barry-configuration.hpp, 218
BARRY_MAX_NUM_ELEMENTS
barry-configuration.hpp, 218
BARRY_PROGRESS_BAR_WIDTH
progress.hpp, 268
BARRY_SAFE_EXP
barry-configuration.hpp, 218
BARRY_SUPPORT_MEAT_HPP
support-meat.hpp, 282
BARRY_VERSION
barry.hpp, 221
BDENSE_TEMPLATE
barraydense-meet.hpp, 198–204
BDENSE_TEMPLATE_ARGS
barraydense-meet.hpp, 198
BDENSE_TYPE
barraydense-meet.hpp, 198
begin
BArrayVector< Cell_Type, Data_Type >, 68

BArrayVector_const< Cell_Type, Data_Type >, 72
PhyloCounterData, 139
PowerSet< Array_Type, Data_Rule_Type >, 145
blengths
NodeData, 137
BOTH
CHECK, 26
EXISTS, 27

calc
PowerSet< Array_Type, Data_Rule_Type >, 146
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 163
calc_backend
support-meat.hpp, 283
calc_reduced_sequence
Geese, 105
calc_sequence
Geese, 106
Cell
Cell< Cell_Type >, 75–77
Cell< Cell_Type >, 74
~Cell, 76
add, 77
Cell, 75–77
operator Cell_Type, 78
operator!=, 78
operator=, 78
operator==, 78
value, 79
visited, 79
cfree
support-meat.hpp, 288
change_stats
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 168
CHECK, 26
BOTH, 26
NONE, 26
ONE, 26
TWO, 26
check_bounds
barray-meat.hpp, 187
barraydense-meet.hpp, 208
check_exists
barray-meat.hpp, 187
barraydense-meet.hpp, 208
checkdim_
barray-meat-operators.hpp, 174
clear
BArray< Cell_Type, Data_Type >, 33
BArrayDense< Cell_Type, Data_Type >, 52
Counters< Array_Type, Data_Type >, 88
FreqTable< T >, 101
Rules< Array_Type, Data_Type >, 154
COL
barray-meat-operators.hpp, 174
barray-meat.hpp, 178, 182
barraydense-meet.hpp, 198, 204
col
BArray< Cell_Type, Data_Type >, 33
BArrayDense< Cell_Type, Data_Type >, 52
col0
barray-meat.hpp, 188
barraydense-meet.hpp, 208
Col_type
typedefs.hpp, 293
colnames
Flock, 95
Geese, 106
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 119
conditional_prob
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 120
const
barray-meat.hpp, 188
barraydense-meet.hpp, 209
ConstBArrayRowIter
ConstBArrayRowIter< Cell_Type, Data_Type >, 80
ConstBArrayRowIter< Cell_Type, Data_Type >, 79
~ConstBArrayRowIter, 80
Array, 80
ConstBArrayRowIter, 80
current_col, 81
current_row, 81
iter, 81
coordinates_free
PowerSet< Array_Type, Data_Rule_Type >, 147
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 168
coordinates_locked
PowerSet< Array_Type, Data_Rule_Type >, 147
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 168
copy_data
barray-meat.hpp, 188
barraydense-meet.hpp, 209
count
Counter< Array_Type, Data_Type >, 84
count_all
StatsCounter< Array_Type, Data_Type >, 158
count_current
StatsCounter< Array_Type, Data_Type >, 158
count_fun
Counter< Array_Type, Data_Type >, 85
counters-meat.hpp, 228
count_fun_
counters-meat.hpp, 234
count_init
StatsCounter< Array_Type, Data_Type >, 158

Counter
Counter< Array_Type, Data_Type >, 82, 83
counter
counters-meat.hpp, 234
Counter< Array_Type, Data_Type >, 81
~Counter, 83
count, 84
count_fun, 85
Counter, 82, 83
data, 85
delete_data, 85
desc, 85
get_description, 84
get_name, 84
init, 84
init_fun, 85
name, 86
operator=, 84
counter_
counters-meat.hpp, 234
counter_absdiff
Network counters, 13
counter_co_opt
Phylo counters, 18
counter_cogain
Phylo counters, 18
counter_css_completely_false_recip_comiss
network-css.hpp, 239
counter_css_completely_false_recip_omiss
network-css.hpp, 240
counter_css_mixed_recip
network-css.hpp, 240
counter_css_partially_false_recip_commi
network-css.hpp, 240
counter_css_partially_false_recip_omiss
network-css.hpp, 241
counter_ctriads
Network counters, 13
counter_degree
Network counters, 13
counter_deleted
statscounter-meat.hpp, 277
counter_density
Network counters, 13
counter_diff
Network counters, 13
counter_edges
Network counters, 14
Counter_fun_type
typedefs.hpp, 293
COUNTER_FUNCTION
barry.hpp, 221
counter_gains
Phylo counters, 18
counter_gains_k_offspring
Phylo counters, 19
counter_genes_changing
Phylo counters, 19
counter_idegree
Network counters, 14
counter_idegree15
Network counters, 14
counter_isolates
Network counters, 14
counter_istar2
Network counters, 14
counter_k_genes_changing
Phylo counters, 19
COUNTER_LAMBDA
barry.hpp, 221
counter_longest
Phylo counters, 19
counter_loss
Phylo counters, 20
counter_maxfuns
Phylo counters, 20
counter_mutual
Network counters, 15
counter_neofun
Phylo counters, 20
counter_neofun_a2b
Phylo counters, 20
counter_nodecov
Network counters, 15
counter_nodeicov
Network counters, 15
counter_nodematch
Network counters, 15
counter_nodeocov
Network counters, 15
counter_odegree
Network counters, 16
counter_odegree15
Network counters, 16
counter_ostar2
Network counters, 16
counter_overall_changes
Phylo counters, 21
counter_overall_gains
Phylo counters, 21
counter_overall_loss
Phylo counters, 21
counter_prop_genes_changing
Phylo counters, 21
counter_subfun
Phylo counters, 22
COUNTER_TEMPLATE
counters-meat.hpp, 227–229
COUNTER_TEMPLATE_ARGS
counters-meat.hpp, 227
counter_ttriads
Network counters, 16
COUNTER_TYPE
counters-meat.hpp, 227
Counters
Counters< Array_Type, Data_Type >, 87

counters
statscounter-meat.hpp, 277
support-meat.hpp, 288
Counters< Array_Type, Data_Type >, 86
~Counters, 87
add_counter, 88
clear, 88
Counters, 87
get_descriptions, 89
get_names, 89
operator=, 89
operator[], 90
size, 90
counters-meat.hpp
count_fun, 228
count_fun_, 234
counter, 234
counter_, 234
COUNTER_TEMPLATE, 227–229
COUNTER_TEMPLATE_ARGS, 227
COUNTER_TYPE, 227
COUNTERS_TEMPLATE, 228, 230, 231
COUNTERS_TEMPLATE_ARGS, 228
COUNTERS_TYPE, 228
data, 231
data_, 234
delete_data, 231, 232
delete_data_, 234
delete_to_be_deleted, 232
desc, 232
desc_, 235
i, 235
init_fun, 232, 233
init_fun_, 235
j, 235
name, 233
name_, 235
noexcept, 236
push_back, 233
return, 236
to_be_deleted, 233
counters_
statscounter-meat.hpp, 277
support-meat.hpp, 288
COUNTERS_TEMPLATE
counters-meat.hpp, 228, 230, 231
COUNTERS_TEMPLATE_ARGS
counters-meat.hpp, 228
COUNTERS_TYPE
counters-meat.hpp, 228
Counting, 11
counts
PhyloRuleDynData, 142
Counts_type
typedefs.hpp, 293
CSS_APPEND
network-css.hpp, 238
CSS_CASE_ELSE
network-css.hpp, 238
CSS_CASE_PERCEIVED
network-css.hpp, 238
CSS_CASE_TRUTH
network-css.hpp, 238
CSS_CHECK_SIZE
network-css.hpp, 238
CSS_CHECK_SIZE_INIT
network-css.hpp, 238
CSS_PERCEIVED_CELLS
network-css.hpp, 239
CSS_SIZE
network-css.hpp, 239
CSS_TRUE_CELLS
network-css.hpp, 239
current_col
ConstBArrayRowIter< Cell_Type, Data_Type >, 81
current_row
ConstBArrayRowIter< Cell_Type, Data_Type >, 81
current_stats
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 168

D
BArray< Cell_Type, Data_Type >, 34
BArrayDense< Cell_Type, Data_Type >, 53
Rule< Array_Type, Data_Type >, 152
dat
Flock, 98
data
barray-meat.hpp, 188
barraydense-meet.hpp, 209
Counter< Array_Type, Data_Type >, 85
counters-meat.hpp, 231
PowerSet< Array_Type, Data_Rule_Type >, 148
data_
counters-meat.hpp, 234
DEFAULT_DUPLICATION
phylo.hpp, 248
default_val
BArray< Cell_Type, Data_Type >, 34
BArrayDense< Cell_Type, Data_Type >, 53
delete_counters
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 169
support-meat.hpp, 289
delete_data
barray-meat.hpp, 188
barraydense-meet.hpp, 209
Counter< Array_Type, Data_Type >, 85
counters-meat.hpp, 231, 232
delete_data_
barray-meat.hpp, 189
barraydense-meet.hpp, 209
counters-meat.hpp, 234
delete_rengine
Geese, 112

delete_rules
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 169
support-meat.hpp, 289
delete_rules_dyn
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 169
support-meat.hpp, 289
delete_support
Geese, 112
delete_to_be_deleted
counters-meat.hpp, 232
desc
Counter< Array_Type, Data_Type >, 85
counters-meat.hpp, 232
desc_
counters-meat.hpp, 235
directed
NetworkData, 130
DUPL_DUPL
phylo.hpp, 248
DUPL_EITH
phylo.hpp, 249
DUPL_SPEC
phylo.hpp, 249
duplication
Node, 135
NodeData, 138
PhyloRuleDynData, 142

else
barray-meat.hpp, 189
barraydense-meet.hpp, 210
support-meat.hpp, 289
empty
PhyloCounterData, 139
EmptyArray
PowerSet< Array_Type, Data_Rule_Type >, 148
end
BArrayVector< Cell_Type, Data_Type >, 69
BArrayVector_const< Cell_Type, Data_Type >, 72
PhyloCounterData, 139
PowerSet< Array_Type, Data_Rule_Type >, 146
Progress, 150
Entries
Entries< Cell_Type >, 91
Entries< Cell_Type >, 91
~Entries, 92
Entries, 91
resize, 92
source, 92
target, 92
val, 92
eval_rules_dyn
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 165
EXISTS, 27
AS_ONE, 27
AS_ZERO, 27
BOTH, 27
NONE, 28
ONE, 28
TWO, 28
UKNOWN, 28

f_
statscounter-meat.hpp, 277
support-meat.hpp, 289
false
barray-meat.hpp, 189
barraydense-meet.hpp, 210
first
barray-meat.hpp, 189
barraydense-meet.hpp, 210
Flock, 93
~Flock, 94
add_data, 94
colnames, 95
dat, 98
Flock, 94
get_counters, 95
get_model, 95
get_support, 95
init, 95
initialized, 98
likelihood_joint, 96
model, 98
nfunctions, 99
nfuns, 96
nleafs, 96
nnodes, 96
nterms, 97
ntrees, 97
operator(), 97
parse_polytomies, 97
print, 97
rengine, 99
set_seed, 98
support_size, 98
flush_data
BArray< Cell_Type, Data_Type >, 34
for
barray-meat.hpp, 182, 183
barraydense-meet.hpp, 204
statscounter-meat.hpp, 275
support-meat.hpp, 284
FreqTable
FreqTable< T >, 100
FreqTable< T >, 99
~FreqTable, 100
add, 100
as_vector, 100
clear, 101
FreqTable, 100
get_data, 101

get_data_ptr, 101
print, 101
reserve, 101
size, 102

Geese, 102
~Geese, 105
calc_reduced_sequence, 105
calc_sequence, 106
colnames, 106
delete_rengine, 112
delete_support, 112
Geese, 104, 105
get_annotated_nodes, 106
get_counters, 106
get_model, 106
get_probabilities, 106
get_rengine, 107
get_states, 107
get_support, 107
inherit_support, 107
init, 107
init_node, 108
initialized, 113
likelihood, 108
likelihood_exhaust, 108
map_to_nodes, 113
nannotations, 108
nfunctions, 113
nfuns, 108
nleafs, 109
nnodes, 109
nodes, 113
nterms, 109
observed_counts, 109
operator=, 109, 110
parse_polytomies, 110
predict, 110
predict_backend, 110
predict_exhaust, 110
predict_exhaust_backend, 111
predict_sim, 111
print, 111
print_observed_counts, 111
reduced_sequence, 113
sequence, 113
set_seed, 111
simulate, 112
support_size, 112
update_annotations, 112
geese-bones.hpp
INITIALIZED, 259
keygen_full, 260
RULE_FUNCTION, 260
vec_diff, 260
vector_caster, 260
gen_key
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 120
get_annotated_nodes
Geese, 106
get_cell
BArray< Cell_Type, Data_Type >, 34
BArrayDense< Cell_Type, Data_Type >, 53
get_col_vec
BArray< Cell_Type, Data_Type >, 34, 35
BArrayDense< Cell_Type, Data_Type >, 53
get_counters
Flock, 95
Geese, 106
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 120
PhyloCounterData, 140
StatsCounter< Array_Type, Data_Type >, 158
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 165
get_counts
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 165
get_counts_ptr
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 165
get_current_stats
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 165
get_data
FreqTable< T >, 101
PowerSet< Array_Type, Data_Rule_Type >, 146
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 166
get_data_ptr
FreqTable< T >, 101
PowerSet< Array_Type, Data_Rule_Type >, 146
get_description
Counter< Array_Type, Data_Type >, 84
get_descriptions
Counters< Array_Type, Data_Type >, 89
StatsCounter< Array_Type, Data_Type >, 158
get_entries
BArray< Cell_Type, Data_Type >, 35
BArrayDense< Cell_Type, Data_Type >, 54
get_last_name
phylo.hpp, 253
get_model
Flock, 95
Geese, 106
get_name
Counter< Array_Type, Data_Type >, 84

get_names
Counters< Array_Type, Data_Type >, 89
StatsCounter< Array_Type, Data_Type >, 158
get_norm_const
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 121
get_parent
Node, 133
get_probabilities
Geese, 106
get_pset
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 121
get_pset_stats
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 121
get_rengine
Geese, 107
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 121
get_row_vec
BArray< Cell_Type, Data_Type >, 35
BArrayDense< Cell_Type, Data_Type >, 54
get_rules
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 121
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 166
get_rules_dyn
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 122
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 166
get_seq
Rules< Array_Type, Data_Type >, 154
get_states
Geese, 107
get_support
Flock, 95
Geese, 107
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 122

i
counters-meat.hpp, 235
i1
barray-meat.hpp, 189
barraydense-meet.hpp, 210
id
Node, 135
if
barray-meat.hpp, 183–185
barraydense-meet.hpp, 204–207
support-meat.hpp, 284
IF_MATCHES
phylo.hpp, 249
IF_NOTMATCHES
phylo.hpp, 249
include/barry/barray-bones.hpp, 171
include/barry/barray-iterator.hpp, 172
include/barry/barray-meat-operators.hpp, 173
include/barry/barray-meat.hpp, 175
include/barry/barraycell-bones.hpp, 193
include/barry/barraycell-meat.hpp, 193
include/barry/barraydense-bones.hpp, 194
include/barry/barraydense-meet.hpp, 195
include/barry/barraydensecell-bones.hpp, 213
include/barry/barraydensecell-meat.hpp, 214
include/barry/barrayvector-bones.hpp, 215
include/barry/barrayvector-meat.hpp, 216
include/barry/barry-configuration.hpp, 217
include/barry/barry-debug.hpp, 219
include/barry/barry.hpp, 220
include/barry/cell-bones.hpp, 222
include/barry/cell-meat.hpp, 223
include/barry/col-bones.hpp, 224
include/barry/counters-bones.hpp, 224
include/barry/counters-meat.hpp, 225
include/barry/counters/network-css.hpp, 237
include/barry/counters/network.hpp, 241
include/barry/counters/phylo.hpp, 246
include/barry/model-bones.hpp, 254
include/barry/model-meat.hpp, 255
include/barry/models/geese.hpp, 257
include/barry/models/geese/flock-bones.hpp, 258
include/barry/models/geese/flock-meat.hpp, 258
include/barry/models/geese/geese-bones.hpp, 259
include/barry/models/geese/geese-meat-constructors.hpp, 261
include/barry/models/geese/geese-meat-likelihood.hpp, 261
include/barry/models/geese/geese-meat-likelihood_exhaust.hpp, 262
include/barry/models/geese/geese-meat-predict.hpp, 263
include/barry/models/geese/geese-meat-predict_exhaust.hpp, 263
include/barry/models/geese/geese-meat-predict_sim.hpp, 264
include/barry/models/geese/geese-meat-simulate.hpp, 264
include/barry/models/geese/geese-meat.hpp, 265
include/barry/models/geese/geese-node-bones.hpp, 265
include/barry/powerset-bones.hpp, 266
include/barry/powerset-meat.hpp, 267
include/barry/progress.hpp, 268
include/barry/rules-bones.hpp, 269

include/barry/rules-meat.hpp, 270
include/barry/statscounter-bones.hpp, 271
include/barry/statscounter-meat.hpp, 272
include/barry/statsdb.hpp, 278
include/barry/support-bones.hpp, 279
include/barry/support-meat.hpp, 281
include/barry/typedefs.hpp, 291
indices
NetCounterData, 127
inherit_support
Geese, 107
init
Counter< Array_Type, Data_Type >, 84
Flock, 95
Geese, 107
init_fun
Counter< Array_Type, Data_Type >, 85
counters-meat.hpp, 232, 233
init_fun_
counters-meat.hpp, 235
init_node
Geese, 108
init_support
PowerSet< Array_Type, Data_Rule_Type >, 146
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 166
INITIALIZED
geese-bones.hpp, 259
initialized
Flock, 98
Geese, 113
insert_cell
BArray< Cell_Type, Data_Type >, 35, 36
BArrayDense< Cell_Type, Data_Type >, 54, 55
support-meat.hpp, 284
is_col
BArrayVector< Cell_Type, Data_Type >, 69
BArrayVector_const< Cell_Type, Data_Type >, 72
IS_DUPLICATION
phylo.hpp, 249
IS_EITHER
phylo.hpp, 250
is_empty
BArray< Cell_Type, Data_Type >, 36
BArrayDense< Cell_Type, Data_Type >, 55
is_leaf
Node, 134
is_row
BArrayVector< Cell_Type, Data_Type >, 69
BArrayVector_const< Cell_Type, Data_Type >, 72
IS_SPECIATION
phylo.hpp, 250
iter
ConstBArrayRowIter< Cell_Type, Data_Type >, 81

j
barray-meat.hpp, 190
barraydense-meet.hpp, 210
counters-meat.hpp, 235
statscounter-meat.hpp, 277
j0
barray-meat.hpp, 190
barraydense-meet.hpp, 211
j1
barray-meat.hpp, 190
barraydense-meet.hpp, 211

keygen_default
model-bones.hpp, 255
keygen_full
geese-bones.hpp, 260

lb
PhyloRuleDynData, 142
likelihood
Geese, 108
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 122
likelihood_
model-meat.hpp, 256
likelihood_exhaust
Geese, 108
likelihood_joint
Flock, 96
likelihood_total
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 123

M
barray-meat.hpp, 186, 190
barraydense-meet.hpp, 207, 211
PowerSet< Array_Type, Data_Rule_Type >, 148
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 169
M_
barray-meat.hpp, 190
barraydense-meet.hpp, 211
MAKE_DUPL_VARS
phylo.hpp, 250
Map
barry-configuration.hpp, 218
map_to_nodes
Geese, 113
MapVec_type
typedefs.hpp, 293
max_num_elements
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 169
Model
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 116, 117
model

Flock, 98
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 114
~Model, 117
add_array, 117
add_counter, 118
add_rule, 118, 119
add_rule_dyn, 119
colnames, 119
conditional_prob, 120
gen_key, 120
get_counters, 120
get_norm_const, 121
get_pset, 121
get_pset_stats, 121
get_rengine, 121
get_rules, 121
get_rules_dyn, 122
get_support, 122
likelihood, 122
likelihood_total, 123
Model, 116, 117
nterms, 123
operator=, 123
print, 123
print_stats, 123
sample, 124
set_counters, 124
set_keygen, 124
set_rengine, 124
set_rules, 125
set_rules_dyn, 125
set_seed, 125
size, 125
size_unique, 125
store_psets, 126
support_size, 126
model-bones.hpp
keygen_default, 255
model-meat.hpp
likelihood_, 256
MODEL_TEMPLATE, 256, 257
MODEL_TEMPLATE_ARGS, 256
MODEL_TYPE, 256
update_normalizing_constant, 257
MODEL_TEMPLATE
model-meat.hpp, 256, 257
MODEL_TEMPLATE_ARGS
model-meat.hpp, 256
MODEL_TYPE
model-meat.hpp, 256

N
barray-meat.hpp, 191
barraydense-meet.hpp, 211
PowerSet< Array_Type, Data_Rule_Type >, 148
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 170
name
Counter< Array_Type, Data_Type >, 86
counters-meat.hpp, 233
name_
counters-meat.hpp, 235
nannotations
Geese, 108
narray
Node, 135
NCells
barray-meat.hpp, 191
barraydense-meet.hpp, 212
ncol
BArray< Cell_Type, Data_Type >, 36
BArrayDense< Cell_Type, Data_Type >, 55
NET_C_DATA_IDX
network.hpp, 243
NET_C_DATA_NUM
network.hpp, 243
NetCounter
network.hpp, 245
NetCounterData, 126
~NetCounterData, 127
indices, 127
NetCounterData, 127
numbers, 127
NetCounters
network.hpp, 245
NetModel
network.hpp, 245
NetRule
network.hpp, 245
NetRules
network.hpp, 245
NetStatsCounter
network.hpp, 245
NetSupport
network.hpp, 246
Network
network.hpp, 246
Network counters, 12
counter_absdiff, 13
counter_ctriads, 13
counter_degree, 13
counter_density, 13
counter_diff, 13
counter_edges, 14
counter_idegree, 14
counter_idegree15, 14
counter_isolates, 14
counter_istar2, 14
counter_mutual, 15
counter_nodecov, 15
counter_nodeicov, 15
counter_nodematch, 15
counter_nodeocov, 15
counter_odegree, 16
counter_odegree15, 16

counter_ostar2, 16
counter_ttriads, 16
NETWORK_COUNTER, 16
network-css.hpp
counter_css_completely_false_recip_comiss, 239
counter_css_completely_false_recip_omiss, 240
counter_css_mixed_recip, 240
counter_css_partially_false_recip_commi, 240
counter_css_partially_false_recip_omiss, 241
CSS_APPEND, 238
CSS_CASE_ELSE, 238
CSS_CASE_PERCEIVED, 238
CSS_CASE_TRUTH, 238
CSS_CHECK_SIZE, 238
CSS_CHECK_SIZE_INIT, 238
CSS_PERCEIVED_CELLS, 239
CSS_SIZE, 239
CSS_TRUE_CELLS, 239
network.hpp
NET_C_DATA_IDX, 243
NET_C_DATA_NUM, 243
NetCounter, 245
NetCounters, 245
NetModel, 245
NetRule, 245
NetRules, 245
NetStatsCounter, 245
NetSupport, 246
Network, 246
NETWORK_COUNTER, 243
NETWORK_COUNTER_LAMBDA, 244
NETWORK_RULE, 244
NETWORK_RULE_LAMBDA, 244
rules_zerodiag, 246
NETWORK_COUNTER
Network counters, 16
network.hpp, 243
NETWORK_COUNTER_LAMBDA
network.hpp, 244
NETWORK_RULE
network.hpp, 244
NETWORK_RULE_LAMBDA
network.hpp, 244
NetworkData, 128
~NetworkData, 130
directed, 130
NetworkData, 128–130
vertex_attr, 130
next
Progress, 150
nfunctions
Flock, 99
Geese, 113
nfuns
Flock, 96
Geese, 108
nleafs
Flock, 96
Geese, 109
nnodes
Flock, 96
Geese, 109
nnozero
BArray< Cell_Type, Data_Type >, 36
BArrayDense< Cell_Type, Data_Type >, 55
Node, 131
~Node, 133
annotations, 134
array, 134
arrays, 134
duplication, 135
get_parent, 133
id, 135
is_leaf, 134
narray, 135
Node, 132, 133
noffspring, 134
offspring, 135
ord, 135
parent, 136
probability, 136
subtree_prob, 136
visited, 136
NodeData, 137
blengths, 137
duplication, 138
NodeData, 137
states, 138
nodes
Geese, 113
noexcept
counters-meat.hpp, 236
noffspring
Node, 134
NONE
CHECK, 26
EXISTS, 28
nrow
BArray< Cell_Type, Data_Type >, 37
BArrayDense< Cell_Type, Data_Type >, 55
nterms
Flock, 97
Geese, 109
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 123
ntrees
Flock, 97
numbers
NetCounterData, 127

observed_counts
Geese, 109
offspring
Node, 135
ONE
CHECK, 26

EXISTS, 28
operator Cell_Type
BArrayCell< Cell_Type, Data_Type >, 44
BArrayCell_const< Cell_Type, Data_Type >, 46
BArrayDenseCell< Cell_Type, Data_Type >, 63
BArrayDenseCell_const< Cell_Type, Data_Type >, 65
Cell< Cell_Type >, 78
operator std::vector< Cell_Type >
BArrayVector< Cell_Type, Data_Type >, 69
BArrayVector_const< Cell_Type, Data_Type >, 73
operator!=
BArrayCell_const< Cell_Type, Data_Type >, 47
BArrayDenseCell_const< Cell_Type, Data_Type >, 66
BArrayVector_const< Cell_Type, Data_Type >, 73
Cell< Cell_Type >, 78
operator<
BArrayCell_const< Cell_Type, Data_Type >, 47
BArrayDenseCell_const< Cell_Type, Data_Type >, 66
BArrayVector_const< Cell_Type, Data_Type >, 73
operator<=
BArrayCell_const< Cell_Type, Data_Type >, 47
BArrayDenseCell_const< Cell_Type, Data_Type >, 66
BArrayVector_const< Cell_Type, Data_Type >, 73
operator>
BArrayCell_const< Cell_Type, Data_Type >, 47
BArrayDenseCell_const< Cell_Type, Data_Type >, 66
BArrayVector_const< Cell_Type, Data_Type >, 74
operator>=
BArrayCell_const< Cell_Type, Data_Type >, 48
BArrayDenseCell_const< Cell_Type, Data_Type >, 67
BArrayVector_const< Cell_Type, Data_Type >, 74
operator*=
BArray< Cell_Type, Data_Type >, 37
BArrayCell< Cell_Type, Data_Type >, 44
BArrayDense< Cell_Type, Data_Type >, 56
BArrayDenseCell< Cell_Type, Data_Type >, 63
BArrayVector< Cell_Type, Data_Type >, 69
operator()
BArray< Cell_Type, Data_Type >, 37
BArrayDense< Cell_Type, Data_Type >, 56
Flock, 97
PhyloCounterData, 140
Rule< Array_Type, Data_Type >, 152
Rules< Array_Type, Data_Type >, 155
vecHasher< T >, 170
operator+=
BArray< Cell_Type, Data_Type >, 37, 38
BArrayCell< Cell_Type, Data_Type >, 44
BArrayDense< Cell_Type, Data_Type >, 56
BArrayDenseCell< Cell_Type, Data_Type >, 63
BArrayVector< Cell_Type, Data_Type >, 70
operator-=
BArray< Cell_Type, Data_Type >, 38
BArrayCell< Cell_Type, Data_Type >, 44
BArrayDense< Cell_Type, Data_Type >, 57
BArrayDenseCell< Cell_Type, Data_Type >, 63
BArrayVector< Cell_Type, Data_Type >, 70
operator/=
BArray< Cell_Type, Data_Type >, 38
BArrayCell< Cell_Type, Data_Type >, 45
BArrayDense< Cell_Type, Data_Type >, 57
BArrayDenseCell< Cell_Type, Data_Type >, 64
BArrayVector< Cell_Type, Data_Type >, 70
operator=
BArray< Cell_Type, Data_Type >, 38, 39
BArrayCell< Cell_Type, Data_Type >, 45
BArrayDense< Cell_Type, Data_Type >, 57
BArrayDenseCell< Cell_Type, Data_Type >, 64
BArrayVector< Cell_Type, Data_Type >, 70
Cell< Cell_Type >, 78
Counter< Array_Type, Data_Type >, 84
Counters< Array_Type, Data_Type >, 89
Geese, 109, 110
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 123
Rules< Array_Type, Data_Type >, 155
operator==
BArray< Cell_Type, Data_Type >, 39
BArrayCell< Cell_Type, Data_Type >, 45
BArrayCell_const< Cell_Type, Data_Type >, 47
BArrayDense< Cell_Type, Data_Type >, 58
BArrayDenseCell< Cell_Type, Data_Type >, 64
BArrayDenseCell_const< Cell_Type, Data_Type >, 66
BArrayVector< Cell_Type, Data_Type >, 70
BArrayVector_const< Cell_Type, Data_Type >, 73
Cell< Cell_Type >, 78
operator[]
Counters< Array_Type, Data_Type >, 90
PowerSet< Array_Type, Data_Rule_Type >, 147
ord
Node, 135
out_of_range
BArray< Cell_Type, Data_Type >, 39
BArrayDense< Cell_Type, Data_Type >, 58

parent
Node, 136
parse_polytomies
Flock, 97
Geese, 110
Phylo counters, 17
counter_co_opt, 18
counter_cogain, 18
counter_gains, 18
counter_gains_k_offspring, 19
counter_genes_changing, 19
counter_k_genes_changing, 19
counter_longest, 19
counter_loss, 20

counter_maxfuns, 20
counter_neofun, 20
counter_neofun_a2b, 20
counter_overall_changes, 21
counter_overall_gains, 21
counter_overall_loss, 21
counter_prop_genes_changing, 21
counter_subfun, 22
Phylo rules, 22
rule_dyn_limit_changes, 23
phylo.hpp
DEFAULT_DUPLICATION, 248
DUPL_DUPL, 248
DUPL_EITH, 249
DUPL_SPEC, 249
get_last_name, 253
IF_MATCHES, 249
IF_NOTMATCHES, 249
IS_DUPLICATION, 249
IS_EITHER, 250
IS_SPECIATION, 250
MAKE_DUPL_VARS, 250
PHYLO_CHECK_MISSING, 250
PHYLO_COUNTER_LAMBDA, 250
PHYLO_RULE_DYN_LAMBDA, 251
PhyloArray, 251
PhyloCounter, 251
PhyloCounters, 251
PhyloModel, 252
PhyloPowerSet, 252
PhyloRule, 252
PhyloRuleData, 252
PhyloRuleDyn, 252
PhyloRules, 252
PhyloRulesDyn, 253
PhyloStatsCounter, 253
PhyloSupport, 253
PHYLO_CHECK_MISSING
phylo.hpp, 250
PHYLO_COUNTER_LAMBDA
phylo.hpp, 250
PHYLO_RULE_DYN_LAMBDA
phylo.hpp, 251
PhyloArray
phylo.hpp, 251
PhyloCounter
phylo.hpp, 251
PhyloCounterData, 138
at, 139
begin, 139
empty, 139
end, 139
get_counters, 140
operator(), 140
PhyloCounterData, 139
push_back, 140
reserve, 140
shrink_to_fit, 140
size, 140
PhyloCounters
phylo.hpp, 251
PhyloModel
phylo.hpp, 252
PhyloPowerSet
phylo.hpp, 252
PhyloRule
phylo.hpp, 252
PhyloRuleData
phylo.hpp, 252
PhyloRuleDyn
phylo.hpp, 252
PhyloRuleDynData, 141
~PhyloRuleDynData, 141
counts, 142
duplication, 142
lb, 142
PhyloRuleDynData, 141
pos, 142
ub, 142
PhyloRules
phylo.hpp, 252
PhyloRulesDyn
phylo.hpp, 253
PhyloStatsCounter
phylo.hpp, 253
PhyloSupport
phylo.hpp, 253
POS
barraydense-meet.hpp, 199
barraydensecell-meat.hpp, 215
pos
PhyloRuleDynData, 142
PowerSet
PowerSet< Array_Type, Data_Rule_Type >, 144
PowerSet< Array_Type, Data_Rule_Type >, 143
~PowerSet, 145
add_rule, 145
begin, 145
calc, 146
coordinates_free, 147
coordinates_locked, 147
data, 148
EmptyArray, 148
end, 146
get_data, 146
get_data_ptr, 146
init_support, 146
M, 148
N, 148
operator[], 147
PowerSet, 144
reset, 147
rules, 148
rules_deleted, 149
size, 147
predict

Geese, 110
predict_backend
Geese, 110
predict_exhaust
Geese, 110
predict_exhaust_backend
Geese, 111
predict_sim
Geese, 111
print
BArray< Cell_Type, Data_Type >, 39
BArrayDense< Cell_Type, Data_Type >, 58
Flock, 97
FreqTable< T >, 101
Geese, 111
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 123
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 166
print_observed_counts
Geese, 111
print_stats
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 123
printf_barry
barry-configuration.hpp, 218
probability
Node, 136
Progress, 149
~Progress, 150
end, 150
next, 150
Progress, 149
progress.hpp
BARRY_PROGRESS_BAR_WIDTH, 268
push_back
counters-meat.hpp, 233
PhyloCounterData, 140

README.md, 295
reduced_sequence
Geese, 113
rengine
Flock, 99
report
barray-meat.hpp, 191
barraydense-meet.hpp, 212
reserve
BArray< Cell_Type, Data_Type >, 39
BArrayDense< Cell_Type, Data_Type >, 58
FreqTable< T >, 101
PhyloCounterData, 140
reset
PowerSet< Array_Type, Data_Rule_Type >, 147
reset_array
StatsCounter< Array_Type, Data_Type >, 159
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 167
resize
BArray< Cell_Type, Data_Type >, 39
barray-meat.hpp, 186
BArrayDense< Cell_Type, Data_Type >, 58
barraydense-meet.hpp, 207
Entries< Cell_Type >, 92
statscounter-meat.hpp, 275
return
barray-meat.hpp, 186, 191
barraydense-meet.hpp, 212
counters-meat.hpp, 236
statscounter-meat.hpp, 278
support-meat.hpp, 290
rm_cell
BArray< Cell_Type, Data_Type >, 40
BArrayDense< Cell_Type, Data_Type >, 58
support-meat.hpp, 284
ROW
barray-meat-operators.hpp, 174
barray-meat.hpp, 178, 186
barraydense-meet.hpp, 199, 207
row
BArray< Cell_Type, Data_Type >, 40
BArrayDense< Cell_Type, Data_Type >, 59
row0
barray-meat.hpp, 191
barraydense-meet.hpp, 212
Row_type
typedefs.hpp, 293
Rule
Rule< Array_Type, Data_Type >, 151
Rule< Array_Type, Data_Type >, 150
~Rule, 151
D, 152
operator(), 152
Rule, 151
rule_dyn_limit_changes
Phylo rules, 23
rule_fun_default
rules-bones.hpp, 270
Rule_fun_type
typedefs.hpp, 294
RULE_FUNCTION
barry.hpp, 221
geese-bones.hpp, 260
RULE_LAMBDA
barry.hpp, 222
Rules
Rules< Array_Type, Data_Type >, 153
rules
PowerSet< Array_Type, Data_Rule_Type >, 148
support-meat.hpp, 290
Rules< Array_Type, Data_Type >, 152
~Rules, 153
add_rule, 154

clear, 154
get_seq, 154
operator(), 155
operator=, 155
Rules, 153
size, 155
rules-bones.hpp
rule_fun_default, 270
rules_
support-meat.hpp, 290
rules_deleted
PowerSet< Array_Type, Data_Rule_Type >, 149
rules_dyn
support-meat.hpp, 290
rules_zerodiag
network.hpp, 246

sample
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 124
search
barray-meat.hpp, 192
sequence
Geese, 113
set_counters
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 124
StatsCounter< Array_Type, Data_Type >, 159
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 167
set_data
BArray< Cell_Type, Data_Type >, 40
BArrayDense< Cell_Type, Data_Type >, 59
set_keygen
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 124
set_rengine
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 124
set_rules
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 125
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 167
set_rules_dyn
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 125
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 167
set_seed
Flock, 98
Geese, 111
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 125
shrink_to_fit
PhyloCounterData, 140
simulate
Geese, 112
size
BArrayVector< Cell_Type, Data_Type >, 71
BArrayVector_const< Cell_Type, Data_Type >, 74
Counters< Array_Type, Data_Type >, 90
FreqTable< T >, 102
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 125
PhyloCounterData, 140
PowerSet< Array_Type, Data_Rule_Type >, 147
Rules< Array_Type, Data_Type >, 155
size_unique
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 125
source
barray-meat.hpp, 192
barraydense-meet.hpp, 212
Entries< Cell_Type >, 92
states
NodeData, 138
Statistical Models, 11
stats_bank
support-meat.hpp, 290
StatsCounter
StatsCounter< Array_Type, Data_Type >, 157
StatsCounter< Array_Type, Data_Type >, 156
~StatsCounter, 157
add_counter, 157
count_all, 158
count_current, 158
count_init, 158
get_counters, 158
get_descriptions, 158
get_names, 158
reset_array, 159
set_counters, 159
StatsCounter, 157
statscounter-meat.hpp
counter_deleted, 277
counters, 277
counters_, 277
f_, 277
for, 275
j, 277
resize, 275
return, 278
STATSCOUNTER_TEMPLATE, 274–276
STATSCOUNTER_TEMPLATE_ARGS, 274

STATSCOUNTER_TYPE, 274
STATSCOUNTER_TEMPLATE
statscounter-meat.hpp, 274–276
STATSCOUNTER_TEMPLATE_ARGS
statscounter-meat.hpp, 274
STATSCOUNTER_TYPE
statscounter-meat.hpp, 274
store_psets
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 126
subtree_prob
Node, 136
Support
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 161, 162
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 159
~Support, 162
add_counter, 162
add_rule, 163
add_rule_dyn, 163
calc, 163
change_stats, 168
coordinates_free, 168
coordinates_locked, 168
current_stats, 168
delete_counters, 169
delete_rules, 169
delete_rules_dyn, 169
eval_rules_dyn, 165
get_counters, 165
get_counts, 165
get_counts_ptr, 165
get_current_stats, 165
get_data, 166
get_rules, 166
get_rules_dyn, 166
init_support, 166
M, 169
max_num_elements, 169
N, 170
print, 166
reset_array, 167
set_counters, 167
set_rules, 167
set_rules_dyn, 167
Support, 161, 162
support-meat.hpp
array_bank, 288
BARRY_SUPPORT_MEAT_HPP, 282
calc_backend, 283
cfree, 288
counters, 288
counters_, 288
delete_counters, 289
delete_rules, 289
delete_rules_dyn, 289
else, 289
f_, 289
for, 284
if, 284
insert_cell, 284
return, 290
rm_cell, 284
rules, 290
rules_, 290
rules_dyn, 290
stats_bank, 290
SUPPORT_TEMPLATE, 283, 285–288
SUPPORT_TEMPLATE_ARGS, 283
SUPPORT_TYPE, 283
support_size
Flock, 98
Geese, 112
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 126
SUPPORT_TEMPLATE
support-meat.hpp, 283, 285–288
SUPPORT_TEMPLATE_ARGS
support-meat.hpp, 283
SUPPORT_TYPE
support-meat.hpp, 283
swap_cells
BArray< Cell_Type, Data_Type >, 40
BArrayDense< Cell_Type, Data_Type >, 59
swap_cols
BArray< Cell_Type, Data_Type >, 41
BArrayDense< Cell_Type, Data_Type >, 59
swap_rows
BArray< Cell_Type, Data_Type >, 41
BArrayDense< Cell_Type, Data_Type >, 60

target
barray-meat.hpp, 192
barraydense-meet.hpp, 212
Entries< Cell_Type >, 92
to_be_deleted
counters-meat.hpp, 233
toggle_cell
BArray< Cell_Type, Data_Type >, 41
BArrayDense< Cell_Type, Data_Type >, 60
toggle_lock
BArray< Cell_Type, Data_Type >, 41
BArrayDense< Cell_Type, Data_Type >, 60
transpose
BArray< Cell_Type, Data_Type >, 41
BArrayDense< Cell_Type, Data_Type >, 60
TWO
CHECK, 26
EXISTS, 28
typedefs.hpp
Col_type, 293
Counter_fun_type, 293
Counts_type, 293

MapVec_type, 293
Row_type, 293
Rule_fun_type, 294
uint, 294
vec_equal, 294
vec_equal_approx, 294
vec_inner_prod, 295

ub
PhyloRuleDynData, 142
uint
typedefs.hpp, 294
UKNOWN
EXISTS, 28
update_annotations
Geese, 112
update_normalizing_constant
model-meat.hpp, 257

v
barray-meat.hpp, 192
barraydense-meet.hpp, 213
val
Entries< Cell_Type >, 92
value
barray-meat.hpp, 192
barraydense-meet.hpp, 213
Cell< Cell_Type >, 79
vec_diff
geese-bones.hpp, 260
vec_equal
typedefs.hpp, 294
vec_equal_approx
typedefs.hpp, 294
vec_inner_prod
typedefs.hpp, 295
vecHasher< T >, 170
operator(), 170
vector_caster
geese-bones.hpp, 260
vertex_attr
NetworkData, 130
visited
BArray< Cell_Type, Data_Type >, 42
BArrayDense< Cell_Type, Data_Type >, 61
Cell< Cell_Type >, 79
Node, 136

ZERO_CELL
barraydense-meet.hpp, 199
zero_col
BArray< Cell_Type, Data_Type >, 42
BArrayDense< Cell_Type, Data_Type >, 60
zero_row
BArray< Cell_Type, Data_Type >, 42
BArrayDense< Cell_Type, Data_Type >, 61