# 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 DEFMArray counters . . . 12

5.3.1 Detailed Description . . . 13

5.3.2 Function Documentation . . . 14

5.3.2.1 counter_absdiff() . . . 14

5.3.2.2 counter_ctriads() [1/2] . . . 14

5.3.2.3 counter_ctriads() [2/2] . . . 14

5.3.2.4 counter_degree() . . . 14

5.3.2.5 counter_density() . . . 15

5.3.2.6 counter_diff() . . . 15

5.3.2.7 counter_edges() . . . 15

5.3.2.8 counter_fixed_effect() . . . 15

5.3.2.9 counter_idegree() [1/2] . . . 16

5.3.2.10 counter_idegree() [2/2] . . . 16

5.3.2.11 counter_idegree15() [1/2] . . . 16

5.3.2.12 counter_idegree15() [2/2] . . . 16

5.3.2.13 counter_isolates() [1/2] . . . 17

5.3.2.14 counter_isolates() [2/2] . . . 17

5.3.2.15 counter_istar2() [1/2] . . . 17

5.3.2.16 counter_istar2() [2/2] . . . 17

5.3.2.17 counter_mutual() . . . 17

5.3.2.18 counter_nodecov() . . . 18

5.3.2.19 counter_nodeicov() . . . 18

5.3.2.20 counter_nodematch() . . . 18

5.3.2.21 counter_nodeocov() . . . 18

5.3.2.22 counter_odegree() [1/2] . . . 18

5.3.2.23 counter_odegree() [2/2] . . . 19

5.3.2.24 counter_odegree15() [1/2] . . . 19

5.3.2.25 counter_odegree15() [2/2] . . . 19

5.3.2.26 counter_ones() . . . 19
5.3.2.27 counter_ostar2() [1/2] . . . 20
5.3.2.28 counter_ostar2() [2/2] . . . 20
5.3.2.29 counter_transition() . . . 20
5.3.2.30 counter_ttriads() [1/2] . . . 20
5.3.2.31 counter_ttriads() [2/2] . . . 21
5.3.2.32 NETWORK_COUNTER() . . . 21
5.3.2.33 rules_markov_fixed() . . . 21
5.4 Phylo counters . . . 21
5.4.1 Detailed Description . . . 22
5.4.2 Function Documentation . . . 23
5.4.2.1 counter_co_opt() . . . 23
5.4.2.2 counter_cogain() . . . 23
5.4.2.3 counter_gains() . . . 23
5.4.2.4 counter_gains_from_0() . . . 24
5.4.2.5 counter_gains_k_offspring() . . . 24
5.4.2.6 counter_genes_changing() . . . 24
5.4.2.7 counter_k_genes_changing() . . . 24
5.4.2.8 counter_less_than_p_prop_genes_changing() . . . 25
5.4.2.9 counter_longest() . . . 25
5.4.2.10 counter_loss() . . . 25
5.4.2.11 counter_maxfuns() . . . 25
5.4.2.12 counter_neofun() . . . 26
5.4.2.13 counter_neofun_a2b() . . . 26
5.4.2.14 counter_overall_changes() . . . 26
5.4.2.15 counter_overall_gains() . . . 26
5.4.2.16 counter_overall_gains_from_0() . . . 27
5.4.2.17 counter_overall_loss() . . . 27
5.4.2.18 counter_pairwise_first_gain() . . . 27
5.4.2.19 counter_pairwise_neofun_singlefun() . . . 27
5.4.2.20 counter_pairwise_overall_change() . . . 28
5.4.2.21 counter_pairwise_preserving() . . . 28
5.4.2.22 counter_preserve_pseudogene() . . . 28
5.4.2.23 counter_prop_genes_changing() . . . 28
5.4.2.24 counter_subfun() . . . 29
5.5 Phylo rules . . . 29
5.5.1 Detailed Description . . . 29
5.5.2 Function Documentation . . . 29
5.5.2.1 rule_dyn_limit_changes() . . . 29

**6 Namespace Documentation** **31**
6.1 barry Namespace Reference . . . 31

6.1.1 Detailed Description . . . 31
6.2 barry::counters Namespace Reference . . . 31
6.2.1 Detailed Description . . . 31
6.3 barry::counters::defm Namespace Reference . . . 32
6.4 barry::counters::network Namespace Reference . . . 32
6.5 barry::counters::phylo Namespace Reference . . . 32
6.6 CHECK Namespace Reference . . . 32
6.6.1 Detailed Description . . . 32
6.6.2 Variable Documentation . . . 32
6.6.2.1 BOTH . . . 32
6.6.2.2 NONE . . . 32
6.6.2.3 ONE . . . 33
6.6.2.4 TWO . . . 33
6.7 EXISTS Namespace Reference . . . 33
6.7.1 Detailed Description . . . 33
6.7.2 Variable Documentation . . . 33
6.7.2.1 AS_ONE . . . 33
6.7.2.2 AS_ZERO . . . 34
6.7.2.3 BOTH . . . 34
6.7.2.4 NONE . . . 34
6.7.2.5 ONE . . . 34
6.7.2.6 TWO . . . 34
6.7.2.7 UKNOWN . . . 34

**7 Class Documentation** **35**
7.1 BArray< Cell_Type, Data_Type > Class Template Reference . . . 35
7.1.1 Detailed Description . . . 37
7.1.2 Constructor & Destructor Documentation . . . 38
7.1.2.1 BArray() [1/6] . . . 38
7.1.2.2 BArray() [2/6] . . . 38
7.1.2.3 BArray() [3/6] . . . 38
7.1.2.4 BArray() [4/6] . . . 39
7.1.2.5 BArray() [5/6] . . . 39
7.1.2.6 BArray() [6/6] . . . 39
7.1.2.7 ~BArray() . . . 39
7.1.3 Member Function Documentation . . . 39
7.1.3.1 clear() . . . 39
7.1.3.2 col() . . . 40
7.1.3.3 D() [1/2] . . . 40
7.1.3.4 D() [2/2] . . . 40
7.1.3.5 D_ptr() [1/2] . . . 40
7.1.3.6 D_ptr() [2/2] . . . 40

7.1.3.7 default_val() . . . 40
7.1.3.8 flush_data() . . . 40
7.1.3.9 get_cell() . . . 41
7.1.3.10 get_col_vec() [1/2] . . . 41
7.1.3.11 get_col_vec() [2/2] . . . 41
7.1.3.12 get_entries() . . . 41
7.1.3.13 get_row_vec() [1/2] . . . 41
7.1.3.14 get_row_vec() [2/2] . . . 42
7.1.3.15 insert_cell() [1/3] . . . 42
7.1.3.16 insert_cell() [2/3] . . . 42
7.1.3.17 insert_cell() [3/3] . . . 42
7.1.3.18 is_dense() . . . 42
7.1.3.19 is_empty() . . . 43
7.1.3.20 ncol() . . . 43
7.1.3.21 nnozero() . . . 43
7.1.3.22 nrow() . . . 43
7.1.3.23 operator()() [1/2] . . . 43
7.1.3.24 operator()() [2/2] . . . 43
7.1.3.25 operator*=() . . . 44
7.1.3.26 operator+=() [1/3] . . . 44
7.1.3.27 operator+=() [2/3] . . . 44
7.1.3.28 operator+=() [3/3] . . . 44
7.1.3.29 operator-=() [1/3] . . . 44
7.1.3.30 operator-=() [2/3] . . . 44
7.1.3.31 operator-=() [3/3] . . . 45
7.1.3.32 operator/=() . . . 45
7.1.3.33 operator=() [1/2] . . . 45
7.1.3.34 operator=() [2/2] . . . 45
7.1.3.35 operator==() . . . 45
7.1.3.36 out_of_range() . . . 45
7.1.3.37 print() . . . 46
7.1.3.38 reserve() . . . 46
7.1.3.39 resize() . . . 46
7.1.3.40 rm_cell() . . . 46
7.1.3.41 row() . . . 46
7.1.3.42 set_data() . . . 46
7.1.3.43 swap_cells() . . . 47
7.1.3.44 swap_cols() . . . 47
7.1.3.45 swap_rows() . . . 47
7.1.3.46 toggle_cell() . . . 47
7.1.3.47 toggle_lock() . . . 48
7.1.3.48 transpose() . . . 48

7.1.3.49 zero_col() . . . 48
7.1.3.50 zero_row() . . . 48
7.1.4 Friends And Related Function Documentation . . . 48
7.1.4.1 BArrayCell< Cell_Type, Data_Type > . . . 48
7.1.4.2 BArrayCell_const< Cell_Type, Data_Type > . . . 49
7.1.5 Member Data Documentation . . . 49
7.1.5.1 visited . . . 49
7.2 BArrayCell< Cell_Type, Data_Type > Class Template Reference . . . 49
7.2.1 Detailed Description . . . 49
7.2.2 Constructor & Destructor Documentation . . . 50
7.2.2.1 BArrayCell() . . . 50
7.2.2.2 ~BArrayCell() . . . 50
7.2.3 Member Function Documentation . . . 50
7.2.3.1 operator Cell_Type() . . . 50
7.2.3.2 operator*=() . . . 50
7.2.3.3 operator+=() . . . 51
7.2.3.4 operator-=() . . . 51
7.2.3.5 operator/=() . . . 51
7.2.3.6 operator=() . . . 51
7.2.3.7 operator==() . . . 51
7.3 BArrayCell_const< Cell_Type, Data_Type > Class Template Reference . . . 52
7.3.1 Detailed Description . . . 52
7.3.2 Constructor & Destructor Documentation . . . 52
7.3.2.1 BArrayCell_const() . . . 52
7.3.2.2 ~BArrayCell_const() . . . 52
7.3.3 Member Function Documentation . . . 53
7.3.3.1 operator Cell_Type() . . . 53
7.3.3.2 operator"!=() . . . 53
7.3.3.3 operator<() . . . 53
7.3.3.4 operator<=() . . . 53
7.3.3.5 operator==() . . . 53
7.3.3.6 operator>() . . . 54
7.3.3.7 operator>=() . . . 54
7.4 BArrayDense< Cell_Type, Data_Type > Class Template Reference . . . 54
7.4.1 Detailed Description . . . 56
7.4.2 Constructor & Destructor Documentation . . . 57
7.4.2.1 BArrayDense() [1/6] . . . 57
7.4.2.2 BArrayDense() [2/6] . . . 57
7.4.2.3 BArrayDense() [3/6] . . . 57
7.4.2.4 BArrayDense() [4/6] . . . 58
7.4.2.5 BArrayDense() [5/6] . . . 58
7.4.2.6 BArrayDense() [6/6] . . . 58

7.4.2.7 ~BArrayDense() . . . 58
7.4.3 Member Function Documentation . . . 58
7.4.3.1 clear() . . . 58
7.4.3.2 col() [1/2] . . . 59
7.4.3.3 col() [2/2] . . . 59
7.4.3.4 colsum() . . . 59
7.4.3.5 D() [1/2] . . . 59
7.4.3.6 D() [2/2] . . . 59
7.4.3.7 D_ptr() [1/2] . . . 59
7.4.3.8 D_ptr() [2/2] . . . 60
7.4.3.9 default_val() . . . 60
7.4.3.10 get_cell() . . . 60
7.4.3.11 get_col_vec() [1/2] . . . 60
7.4.3.12 get_col_vec() [2/2] . . . 60
7.4.3.13 get_data() . . . 60
7.4.3.14 get_entries() . . . 61
7.4.3.15 get_row_vec() [1/2] . . . 61
7.4.3.16 get_row_vec() [2/2] . . . 61
7.4.3.17 insert_cell() [1/2] . . . 61
7.4.3.18 insert_cell() [2/2] . . . 62
7.4.3.19 is_dense() . . . 62
7.4.3.20 is_empty() . . . 62
7.4.3.21 ncol() . . . 62
7.4.3.22 nnozero() . . . 62
7.4.3.23 nrow() . . . 62
7.4.3.24 operator()() [1/2] . . . 63
7.4.3.25 operator()() [2/2] . . . 63
7.4.3.26 operator*=() . . . 63
7.4.3.27 operator+=() [1/3] . . . 63
7.4.3.28 operator+=() [2/3] . . . 63
7.4.3.29 operator+=() [3/3] . . . 63
7.4.3.30 operator-=() [1/3] . . . 64
7.4.3.31 operator-=() [2/3] . . . 64
7.4.3.32 operator-=() [3/3] . . . 64
7.4.3.33 operator/=() . . . 64
7.4.3.34 operator=() [1/2] . . . 64
7.4.3.35 operator=() [2/2] . . . 64
7.4.3.36 operator==() . . . 65
7.4.3.37 out_of_range() . . . 65
7.4.3.38 print() . . . 65
7.4.3.39 reserve() . . . 65
7.4.3.40 resize() . . . 65

7.4.3.41 rm_cell() . . . 65
7.4.3.42 row() [1/2] . . . 66
7.4.3.43 row() [2/2] . . . 66
7.4.3.44 rowsum() . . . 66
7.4.3.45 set_data() . . . 66
7.4.3.46 swap_cells() . . . 66
7.4.3.47 swap_cols() . . . 67
7.4.3.48 swap_rows() . . . 67
7.4.3.49 toggle_cell() . . . 67
7.4.3.50 toggle_lock() . . . 67
7.4.3.51 transpose() . . . 68
7.4.3.52 zero_col() . . . 68
7.4.3.53 zero_row() . . . 68
7.4.4 Friends And Related Function Documentation . . . 68
7.4.4.1 BArrayDenseCell< Cell_Type, Data_Type > . . . 68
7.4.4.2 BArrayDenseCol< Cell_Type, Data_Type > . . . 68
7.4.4.3 BArrayDenseCol_const< Cell_Type, Data_Type > . . . 69
7.4.4.4 BArrayDenseRow< Cell_Type, Data_Type > . . . 69
7.4.4.5 BArrayDenseRow_const< Cell_Type, Data_Type > . . . 69
7.4.5 Member Data Documentation . . . 69
7.4.5.1 visited . . . 69
7.5 BArrayDenseCell< Cell_Type, Data_Type > Class Template Reference . . . 69
7.5.1 Detailed Description . . . 70
7.5.2 Constructor & Destructor Documentation . . . 70
7.5.2.1 BArrayDenseCell() . . . 70
7.5.2.2 ~BArrayDenseCell() . . . 70
7.5.3 Member Function Documentation . . . 71
7.5.3.1 operator Cell_Type() . . . 71
7.5.3.2 operator*=() . . . 71
7.5.3.3 operator+=() . . . 71
7.5.3.4 operator-=() . . . 71
7.5.3.5 operator/=() . . . 71
7.5.3.6 operator=() [1/2] . . . 72
7.5.3.7 operator=() [2/2] . . . 72
7.5.3.8 operator==() . . . 72
7.5.4 Friends And Related Function Documentation . . . 72
7.5.4.1 BArrayDense< Cell_Type, Data_Type > . . . 72
7.5.4.2 BArrayDenseCol< Cell_Type, Data_Type > . . . 72
7.5.4.3 BArrayDenseCol_const< Cell_Type, Data_Type > . . . 73
7.6 BArrayDenseCell_const< Cell_Type, Data_Type > Class Template Reference . . . 73
7.6.1 Detailed Description . . . 73
7.7 BArrayDenseCol< Cell_Type, Data_Type > Class Template Reference . . . 73

7.7.1 Detailed Description . . . 74
7.7.2 Constructor & Destructor Documentation . . . 74
7.7.2.1 BArrayDenseCol() . . . 74
7.7.3 Member Function Documentation . . . 74
7.7.3.1 begin() . . . 74
7.7.3.2 end() . . . 74
7.7.3.3 operator()() . . . 75
7.7.3.4 size() . . . 75
7.7.4 Friends And Related Function Documentation . . . 75
7.7.4.1 BArrayDense< Cell_Type, Data_Type > . . . 75
7.7.4.2 BArrayDenseCell< Cell_Type, Data_Type > . . . 75
7.7.4.3 BArrayDenseCell_const< Cell_Type, Data_Type > . . . 75
7.8 BArrayDenseCol_const< Cell_Type, Data_Type > Class Template Reference . . . 76
7.8.1 Detailed Description . . . 76
7.8.2 Constructor & Destructor Documentation . . . 76
7.8.2.1 BArrayDenseCol_const() . . . 76
7.8.3 Member Function Documentation . . . 76
7.8.3.1 begin() . . . 77
7.8.3.2 end() . . . 77
7.8.3.3 operator()() . . . 77
7.8.3.4 size() . . . 77
7.8.4 Friends And Related Function Documentation . . . 77
7.8.4.1 BArrayDenseCell< Cell_Type, Data_Type > . . . 77
7.8.4.2 BArrayDenseCell_const< Cell_Type, Data_Type > . . . 78
7.9 BArrayDenseRow< Cell_Type, Data_Type > Class Template Reference . . . 78
7.9.1 Detailed Description . . . 78
7.9.2 Constructor & Destructor Documentation . . . 78
7.9.2.1 BArrayDenseRow() . . . 79
7.9.3 Member Function Documentation . . . 79
7.9.3.1 begin() . . . 79
7.9.3.2 end() . . . 79
7.9.3.3 operator()() . . . 79
7.9.3.4 size() . . . 79
7.9.4 Friends And Related Function Documentation . . . 80
7.9.4.1 BArrayDense< Cell_Type, Data_Type > . . . 80
7.9.4.2 BArrayDenseCell< Cell_Type, Data_Type > . . . 80
7.9.4.3 BArrayDenseCell_const< Cell_Type, Data_Type > . . . 80
7.10 BArrayDenseRow_const< Cell_Type, Data_Type > Class Template Reference . . . 80
7.10.1 Detailed Description . . . 81
7.10.2 Constructor & Destructor Documentation . . . 81
7.10.2.1 BArrayDenseRow_const() . . . 81
7.10.3 Member Function Documentation . . . 81

7.10.3.1 begin() . . . 81
7.10.3.2 end() . . . 81
7.10.3.3 operator()() . . . 82
7.10.3.4 size() . . . 82
7.10.4 Friends And Related Function Documentation . . . 82
7.10.4.1 BArrayDenseCell< Cell_Type, Data_Type > . . . 82
7.10.4.2 BArrayDenseCell_const< Cell_Type, Data_Type > . . . 82
7.11 BArrayRow< Cell_Type, Data_Type > Class Template Reference . . . 82
7.11.1 Detailed Description . . . 83
7.11.2 Constructor & Destructor Documentation . . . 83
7.11.2.1 BArrayRow() . . . 83
7.11.2.2 ~BArrayRow() . . . 83
7.11.3 Member Function Documentation . . . 83
7.11.3.1 operator BArrayRow< Cell_Type, Data_Type >() . . . 84
7.11.3.2 operator*=() . . . 84
7.11.3.3 operator+=() . . . 84
7.11.3.4 operator-=() . . . 84
7.11.3.5 operator/=() . . . 84
7.11.3.6 operator=() . . . 84
7.11.3.7 operator==() . . . 85
7.12 BArrayRow_const< Cell_Type, Data_Type > Class Template Reference . . . 85
7.12.1 Detailed Description . . . 85
7.12.2 Constructor & Destructor Documentation . . . 85
7.12.2.1 BArrayRow_const() . . . 85
7.12.2.2 ~BArrayRow_const() . . . 86
7.12.3 Member Function Documentation . . . 86
7.12.3.1 operator BArrayRow_const< Cell_Type, Data_Type >() . . . 86
7.12.3.2 operator"!=() . . . 86
7.12.3.3 operator<() . . . 86
7.12.3.4 operator<=() . . . 86
7.12.3.5 operator==() . . . 86
7.12.3.6 operator>() . . . 87
7.12.3.7 operator>=() . . . 87
7.13 BArrayVector< Cell_Type, Data_Type > Class Template Reference . . . 87
7.13.1 Detailed Description . . . 87
7.13.2 Constructor & Destructor Documentation . . . 88
7.13.2.1 BArrayVector() . . . 88
7.13.2.2 ~BArrayVector() . . . 88
7.13.3 Member Function Documentation . . . 88
7.13.3.1 begin() . . . 89
7.13.3.2 end() . . . 89
7.13.3.3 is_col() . . . 89

7.13.3.4 is_row() . . . 89
7.13.3.5 operator std::vector< Cell_Type >() . . . 89
7.13.3.6 operator*=() . . . 90
7.13.3.7 operator+=() . . . 90
7.13.3.8 operator-=() . . . 90
7.13.3.9 operator/=() . . . 90
7.13.3.10 operator=() . . . 90
7.13.3.11 operator==() . . . 91
7.13.3.12 size() . . . 91
7.14 BArrayVector_const< Cell_Type, Data_Type > Class Template Reference . . . 91
7.14.1 Detailed Description . . . 91
7.14.2 Constructor & Destructor Documentation . . . 92
7.14.2.1 BArrayVector_const() . . . 92
7.14.2.2 ~BArrayVector_const() . . . 92
7.14.3 Member Function Documentation . . . 92
7.14.3.1 begin() . . . 92
7.14.3.2 end() . . . 92
7.14.3.3 is_col() . . . 92
7.14.3.4 is_row() . . . 93
7.14.3.5 operator std::vector< Cell_Type >() . . . 93
7.14.3.6 operator"!=() . . . 93
7.14.3.7 operator<() . . . 93
7.14.3.8 operator<=() . . . 93
7.14.3.9 operator==() . . . 94
7.14.3.10 operator>() . . . 94
7.14.3.11 operator>=() . . . 94
7.14.3.12 size() . . . 94
7.15 Cell< Cell_Type > Class Template Reference . . . 94
7.15.1 Detailed Description . . . 95
7.15.2 Constructor & Destructor Documentation . . . 95
7.15.2.1 Cell() [1/7] . . . 95
7.15.2.2 Cell() [2/7] . . . 96
7.15.2.3 ~Cell() . . . 96
7.15.2.4 Cell() [3/7] . . . 96
7.15.2.5 Cell() [4/7] . . . 96
7.15.2.6 Cell() [5/7] . . . 96
7.15.2.7 Cell() [6/7] . . . 97
7.15.2.8 Cell() [7/7] . . . 97
7.15.3 Member Function Documentation . . . 97
7.15.3.1 add() [1/4] . . . 97
7.15.3.2 add() [2/4] . . . 97
7.15.3.3 add() [3/4] . . . 97

7.15.3.4 add() [4/4] . . . 98
7.15.3.5 operator Cell_Type() . . . 98
7.15.3.6 operator"!=() . . . 98
7.15.3.7 operator=() [1/2] . . . 98
7.15.3.8 operator=() [2/2] . . . 98
7.15.3.9 operator==() . . . 99
7.15.4 Member Data Documentation . . . 99
7.15.4.1 active . . . 99
7.15.4.2 value . . . 99
7.15.4.3 visited . . . 99
7.16 Cell_const< Cell_Type > Class Template Reference . . . 100
7.16.1 Detailed Description . . . 100
7.17 ConstBArrayRowIter< Cell_Type, Data_Type > Class Template Reference . . . 100
7.17.1 Detailed Description . . . 101
7.17.2 Constructor & Destructor Documentation . . . 101
7.17.2.1 ConstBArrayRowIter() . . . 101
7.17.2.2 ~ConstBArrayRowIter() . . . 101
7.17.3 Member Data Documentation . . . 101
7.17.3.1 Array . . . 101
7.17.3.2 current_col . . . 101
7.17.3.3 current_row . . . 102
7.17.3.4 iter . . . 102
7.18 Counter< Array_Type, Data_Type > Class Template Reference . . . 102
7.18.1 Detailed Description . . . 103
7.18.2 Constructor & Destructor Documentation . . . 103
7.18.2.1 Counter() [1/4] . . . 103
7.18.2.2 Counter() [2/4] . . . 104
7.18.2.3 Counter() [3/4] . . . 104
7.18.2.4 Counter() [4/4] . . . 104
7.18.2.5 ~Counter() . . . 104
7.18.3 Member Function Documentation . . . 104
7.18.3.1 count() . . . 105
7.18.3.2 get_description() . . . 105
7.18.3.3 get_name() . . . 105
7.18.3.4 init() . . . 105
7.18.3.5 operator=() [1/2] . . . 105
7.18.3.6 operator=() [2/2] . . . 105
7.18.4 Member Data Documentation . . . 106
7.18.4.1 count_fun . . . 106
7.18.4.2 data . . . 106
7.18.4.3 desc . . . 106
7.18.4.4 init_fun . . . 106

7.18.4.5 name . . . 106
7.19 Counters< Array_Type, Data_Type > Class Template Reference . . . 107
7.19.1 Detailed Description . . . 107
7.19.2 Constructor & Destructor Documentation . . . 107
7.19.2.1 Counters() [1/3] . . . 108
7.19.2.2 ~Counters() . . . 108
7.19.2.3 Counters() [2/3] . . . 108
7.19.2.4 Counters() [3/3] . . . 108
7.19.3 Member Function Documentation . . . 108
7.19.3.1 add_counter() [1/2] . . . 109
7.19.3.2 add_counter() [2/2] . . . 109
7.19.3.3 get_descriptions() . . . 109
7.19.3.4 get_names() . . . 109
7.19.3.5 operator=() [1/2] . . . 109
7.19.3.6 operator=() [2/2] . . . 110
7.19.3.7 operator[]() . . . 110
7.19.3.8 size() . . . 110
7.20 DEFM Class Reference . . . 111
7.20.1 Detailed Description . . . 111
7.20.2 Constructor & Destructor Documentation . . . 111
7.20.2.1 DEFM() . . . 111
7.20.2.2 ~DEFM() . . . 112
7.20.3 Member Function Documentation . . . 112
7.20.3.1 get_m_order() . . . 112
7.20.3.2 get_model() . . . 112
7.20.3.3 get_n_covars() . . . 112
7.20.3.4 get_n_obs() . . . 112
7.20.3.5 get_n_rows() . . . 112
7.20.3.6 get_n_y() . . . 113
7.20.3.7 init() . . . 113
7.20.3.8 likelihood() . . . 113
7.20.3.9 simulate() . . . 113
7.21 DEFMCounterData Class Reference . . . 113
7.21.1 Detailed Description . . . 114
7.21.2 Constructor & Destructor Documentation . . . 114
7.21.2.1 DEFMCounterData() [1/2] . . . 114
7.21.2.2 DEFMCounterData() [2/2] . . . 114
7.21.2.3 ~DEFMCounterData() . . . 114
7.21.3 Member Function Documentation . . . 114
7.21.3.1 idx() . . . 115
7.21.3.2 num() . . . 115
7.21.4 Member Data Documentation . . . 115

7.21.4.1 indices . . . 115
7.21.4.2 markov_order . . . 115
7.21.4.3 numbers . . . 115
7.22 DEFMData Class Reference . . . 116
7.22.1 Detailed Description . . . 116
7.22.2 Constructor & Destructor Documentation . . . 117
7.22.2.1 DEFMData() [1/2] . . . 117
7.22.2.2 DEFMData() [2/2] . . . 117
7.22.2.3 ∼DEFMData() . . . 117
7.22.3 Member Function Documentation . . . 117
7.22.3.1 at() . . . 118
7.22.3.2 ncol() . . . 118
7.22.3.3 operator()() . . . 118
7.22.3.4 print() . . . 118
7.22.4 Member Data Documentation . . . 118
7.22.4.1 array . . . 119
7.22.4.2 covariates . . . 119
7.22.4.3 obs_start . . . 119
7.22.4.4 X_ncol . . . 119
7.22.4.5 X_nrow . . . 119
7.23 DEFMRuleData Class Reference . . . 120
7.23.1 Detailed Description . . . 120
7.23.2 Constructor & Destructor Documentation . . . 120
7.23.2.1 DEFMRuleData() [1/2] . . . 120
7.23.2.2 DEFMRuleData() [2/2] . . . 120
7.23.3 Member Function Documentation . . . 120
7.23.3.1 idx() . . . 120
7.23.3.2 num() . . . 121
7.24 Entries< Cell_Type > Class Template Reference . . . 121
7.24.1 Detailed Description . . . 121
7.24.2 Constructor & Destructor Documentation . . . 121
7.24.2.1 Entries() [1/2] . . . 122
7.24.2.2 Entries() [2/2] . . . 122
7.24.2.3 ∼Entries() . . . 122
7.24.3 Member Function Documentation . . . 122
7.24.3.1 resize() . . . 122
7.24.4 Member Data Documentation . . . 122
7.24.4.1 source . . . 122
7.24.4.2 target . . . 123
7.24.4.3 val . . . 123
7.25 Flock Class Reference . . . 123
7.25.1 Detailed Description . . . 124

7.25.2 Constructor & Destructor Documentation . . . 124
7.25.2.1 Flock() . . . 124
7.25.2.2 ∼Flock() . . . 124
7.25.3 Member Function Documentation . . . 124
7.25.3.1 add_data() . . . 124
7.25.3.2 colnames() . . . 125
7.25.3.3 get_counters() . . . 125
7.25.3.4 get_model() . . . 125
7.25.3.5 get_stats_support() . . . 125
7.25.3.6 get_stats_target() . . . 126
7.25.3.7 get_support_fun() . . . 126
7.25.3.8 init() . . . 126
7.25.3.9 likelihood_joint() . . . 126
7.25.3.10 nfuns() . . . 127
7.25.3.11 nleafs() . . . 127
7.25.3.12 nnodes() . . . 127
7.25.3.13 nterms() . . . 127
7.25.3.14 ntrees() . . . 127
7.25.3.15 operator()() . . . 127
7.25.3.16 parse_polytomies() . . . 128
7.25.3.17 print() . . . 128
7.25.3.18 set_seed() . . . 128
7.25.3.19 support_size() . . . 129
7.25.4 Member Data Documentation . . . 129
7.25.4.1 dat . . . 129
7.25.4.2 initialized . . . 129
7.25.4.3 model . . . 129
7.25.4.4 nfunctions . . . 129
7.25.4.5 rengine . . . 130
7.26 FreqTable< T > Class Template Reference . . . 130
7.26.1 Detailed Description . . . 130
7.26.2 Constructor & Destructor Documentation . . . 131
7.26.2.1 FreqTable() . . . 131
7.26.2.2 ∼FreqTable() . . . 131
7.26.3 Member Function Documentation . . . 131
7.26.3.1 add() . . . 131
7.26.3.2 as_vector() . . . 131
7.26.3.3 clear() . . . 132
7.26.3.4 get_data() . . . 132
7.26.3.5 get_index() . . . 132
7.26.3.6 make_hash() . . . 132
7.26.3.7 print() . . . 132

7.26.3.8 reserve() . . . 133
7.26.3.9 size() . . . 133
7.27 Geese Class Reference . . . 133
7.27.1 Detailed Description . . . 135
7.27.2 Constructor & Destructor Documentation . . . 136
7.27.2.1 Geese() [1/4] . . . 136
7.27.2.2 Geese() [2/4] . . . 136
7.27.2.3 Geese() [3/4] . . . 136
7.27.2.4 Geese() [4/4] . . . 136
7.27.2.5 ∼Geese() . . . 136
7.27.3 Member Function Documentation . . . 137
7.27.3.1 calc_reduced_sequence() . . . 137
7.27.3.2 calc_sequence() . . . 137
7.27.3.3 colnames() . . . 137
7.27.3.4 get_annotated_nodes() . . . 137
7.27.3.5 get_counters() . . . 137
7.27.3.6 get_model() . . . 138
7.27.3.7 get_probabilities() . . . 138
7.27.3.8 get_rengine() . . . 138
7.27.3.9 get_states() . . . 138
7.27.3.10 get_support_fun() . . . 138
7.27.3.11 inherit_support() . . . 139
7.27.3.12 init() . . . 139
7.27.3.13 init_node() . . . 139
7.27.3.14 likelihood() . . . 139
7.27.3.15 likelihood_exhaust() . . . 139
7.27.3.16 nannotations() . . . 140
7.27.3.17 nfuns() . . . 140
7.27.3.18 nleafs() . . . 140
7.27.3.19 nnodes() . . . 140
7.27.3.20 nterms() . . . 140
7.27.3.21 observed_counts() . . . 141
7.27.3.22 operator=() [1/2] . . . 141
7.27.3.23 operator=() [2/2] . . . 141
7.27.3.24 parse_polytomies() . . . 141
7.27.3.25 predict() . . . 141
7.27.3.26 predict_backend() . . . 142
7.27.3.27 predict_exhaust() . . . 142
7.27.3.28 predict_exhaust_backend() . . . 142
7.27.3.29 predict_sim() . . . 142
7.27.3.30 print() . . . 142
7.27.3.31 print_observed_counts() . . . 143

7.27.3.32 set_seed() . . . 143
7.27.3.33 simulate() . . . 143
7.27.3.34 support_size() . . . 143
7.27.3.35 update_annotations() . . . 143
7.27.4 Member Data Documentation . . . 143
7.27.4.1 delete_rengine . . . 144
7.27.4.2 delete_support . . . 144
7.27.4.3 initialized . . . 144
7.27.4.4 map_to_nodes . . . 144
7.27.4.5 nfunctions . . . 144
7.27.4.6 nodes . . . 144
7.27.4.7 pset_loc . . . 145
7.27.4.8 reduced_sequence . . . 145
7.27.4.9 sequence . . . 145
7.28 Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference . . . 145
7.28.1 Detailed Description . . . 148
7.28.2 Constructor & Destructor Documentation . . . 148
7.28.2.1 Model() [1/3] . . . 148
7.28.2.2 Model() [2/3] . . . 149
7.28.2.3 Model() [3/3] . . . 149
7.28.2.4 ~Model() . . . 149
7.28.3 Member Function Documentation . . . 149
7.28.3.1 add_array() . . . 149
7.28.3.2 add_counter() [1/2] . . . 150
7.28.3.3 add_counter() [2/2] . . . 150
7.28.3.4 add_rule() [1/2] . . . 150
7.28.3.5 add_rule() [2/2] . . . 150
7.28.3.6 add_rule_dyn() [1/2] . . . 150
7.28.3.7 add_rule_dyn() [2/2] . . . 151
7.28.3.8 colnames() . . . 151
7.28.3.9 conditional_prob() . . . 151
7.28.3.10 gen_key() . . . 152
7.28.3.11 get_arrays2support() . . . 152
7.28.3.12 get_counters() . . . 152
7.28.3.13 get_norm_const() . . . 152
7.28.3.14 get_pset() . . . 152
7.28.3.15 get_pset_arrays() . . . 153
7.28.3.16 get_pset_probs() . . . 153
7.28.3.17 get_pset_stats() [1/2] . . . 153
7.28.3.18 get_pset_stats() [2/2] . . . 153
7.28.3.19 get_rengine() . . . 153

7.28.3.20 get_rules() . . . 154
7.28.3.21 get_rules_dyn() . . . 154
7.28.3.22 get_stats_support() . . . 154
7.28.3.23 get_stats_target() . . . 154
7.28.3.24 get_support_fun() . . . 154
7.28.3.25 likelihood() [1/4] . . . 155
7.28.3.26 likelihood() [2/4] . . . 155
7.28.3.27 likelihood() [3/4] . . . 155
7.28.3.28 likelihood() [4/4] . . . 155
7.28.3.29 likelihood_total() . . . 156
7.28.3.30 nterms() . . . 156
7.28.3.31 operator=() . . . 156
7.28.3.32 print() . . . 156
7.28.3.33 print_stats() . . . 156
7.28.3.34 sample() [1/2] . . . 157
7.28.3.35 sample() [2/2] . . . 157
7.28.3.36 set_counters() . . . 157
7.28.3.37 set_keygen() . . . 157
7.28.3.38 set_rengine() . . . 157
7.28.3.39 set_rules() . . . 158
7.28.3.40 set_rules_dyn() . . . 158
7.28.3.41 set_seed() . . . 158
7.28.3.42 set_transform_model() . . . 158
7.28.3.43 size() . . . 159
7.28.3.44 size_unique() . . . 159
7.28.3.45 store_psets() . . . 159
7.28.3.46 support_size() . . . 159
7.28.3.47 transform_model() . . . 159
7.29 NetCounterData Class Reference . . . 160
7.29.1 Detailed Description . . . 160
7.29.2 Constructor & Destructor Documentation . . . 160
7.29.2.1 NetCounterData() [1/2] . . . 160
7.29.2.2 NetCounterData() [2/2] . . . 160
7.29.2.3 ~NetCounterData() . . . 161
7.29.3 Member Data Documentation . . . 161
7.29.3.1 indices . . . 161
7.29.3.2 numbers . . . 161
7.30 NetworkData Class Reference . . . 161
7.30.1 Detailed Description . . . 162
7.30.2 Constructor & Destructor Documentation . . . 162
7.30.2.1 NetworkData() [1/3] . . . 162
7.30.2.2 NetworkData() [2/3] . . . 162

7.30.2.3 NetworkData() [3/3] . . . 162
7.30.2.4 ~NetworkData() . . . 163
7.30.3 Member Data Documentation . . . 163
7.30.3.1 directed . . . 163
7.30.3.2 vertex_attr . . . 163
7.31 Node Class Reference . . . 163
7.31.1 Detailed Description . . . 164
7.31.2 Constructor & Destructor Documentation . . . 164
7.31.2.1 Node() [1/5] . . . 165
7.31.2.2 Node() [2/5] . . . 165
7.31.2.3 Node() [3/5] . . . 165
7.31.2.4 Node() [4/5] . . . 165
7.31.2.5 Node() [5/5] . . . 165
7.31.2.6 ~Node() . . . 166
7.31.3 Member Function Documentation . . . 166
7.31.3.1 get_parent() . . . 166
7.31.3.2 is_leaf() . . . 166
7.31.3.3 noffspring() . . . 166
7.31.4 Member Data Documentation . . . 166
7.31.4.1 annotations . . . 166
7.31.4.2 array . . . 167
7.31.4.3 arrays . . . 167
7.31.4.4 duplication . . . 167
7.31.4.5 id . . . 167
7.31.4.6 narray . . . 167
7.31.4.7 offspring . . . 168
7.31.4.8 ord . . . 168
7.31.4.9 parent . . . 168
7.31.4.10 probability . . . 168
7.31.4.11 subtree_prob . . . 168
7.31.4.12 visited . . . 169
7.32 NodeData Class Reference . . . 169
7.32.1 Detailed Description . . . 169
7.32.2 Constructor & Destructor Documentation . . . 169
7.32.2.1 NodeData() . . . 169
7.32.3 Member Data Documentation . . . 170
7.32.3.1 blengths . . . 170
7.32.3.2 duplication . . . 170
7.32.3.3 states . . . 170
7.33 PhyloCounterData Class Reference . . . 170
7.33.1 Detailed Description . . . 171
7.33.2 Constructor & Destructor Documentation . . . 171

7.33.2.1 PhyloCounterData() [1/2] . . . 171
7.33.2.2 PhyloCounterData() [2/2] . . . 171
7.33.3 Member Function Documentation . . . 171
7.33.3.1 at() . . . 171
7.33.3.2 begin() . . . 171
7.33.3.3 empty() . . . 172
7.33.3.4 end() . . . 172
7.33.3.5 get_counters() . . . 172
7.33.3.6 operator()() . . . 172
7.33.3.7 operator[]() . . . 172
7.33.3.8 push_back() . . . 172
7.33.3.9 reserve() . . . 173
7.33.3.10 shrink_to_fit() . . . 173
7.33.3.11 size() . . . 173
7.34 PhyloRuleDynData Class Reference . . . 173
7.34.1 Detailed Description . . . 173
7.34.2 Constructor & Destructor Documentation . . . 174
7.34.2.1 PhyloRuleDynData() . . . 174
7.34.2.2 ∼PhyloRuleDynData() . . . 174
7.34.3 Member Data Documentation . . . 174
7.34.3.1 counts . . . 174
7.34.3.2 duplication . . . 174
7.34.3.3 lb . . . 174
7.34.3.4 pos . . . 175
7.34.3.5 ub . . . 175
7.35 PowerSet< Array_Type, Data_Rule_Type > Class Template Reference . . . 175
7.35.1 Detailed Description . . . 176
7.35.2 Constructor & Destructor Documentation . . . 177
7.35.2.1 PowerSet() [1/3] . . . 177
7.35.2.2 PowerSet() [2/3] . . . 177
7.35.2.3 PowerSet() [3/3] . . . 177
7.35.2.4 ∼PowerSet() . . . 177
7.35.3 Member Function Documentation . . . 177
7.35.3.1 add_rule() [1/2] . . . 178
7.35.3.2 add_rule() [2/2] . . . 178
7.35.3.3 begin() . . . 178
7.35.3.4 calc() . . . 178
7.35.3.5 end() . . . 178
7.35.3.6 get_data() . . . 179
7.35.3.7 get_data_ptr() . . . 179
7.35.3.8 init_support() . . . 179
7.35.3.9 operator[]() . . . 179

7.35.3.10 reset() . . . 179
7.35.3.11 size() . . . 180
7.35.4 Member Data Documentation . . . 180
7.35.4.1 coordinates_free . . . 180
7.35.4.2 coordinates_locked . . . 180
7.35.4.3 data . . . 180
7.35.4.4 EmptyArray . . . 180
7.35.4.5 M . . . 181
7.35.4.6 N . . . 181
7.35.4.7 n_free . . . 181
7.35.4.8 n_locked . . . 181
7.35.4.9 rules . . . 181
7.35.4.10 rules_deleted . . . 182
7.36 Progress Class Reference . . . 182
7.36.1 Detailed Description . . . 182
7.36.2 Constructor & Destructor Documentation . . . 182
7.36.2.1 Progress() . . . 182
7.36.2.2 ∼Progress() . . . 183
7.36.3 Member Function Documentation . . . 183
7.36.3.1 end() . . . 183
7.36.3.2 next() . . . 183
7.37 Rule< Array_Type, Data_Type > Class Template Reference . . . 183
7.37.1 Detailed Description . . . 184
7.37.2 Constructor & Destructor Documentation . . . 184
7.37.2.1 Rule() [1/2] . . . 184
7.37.2.2 Rule() [2/2] . . . 184
7.37.2.3 ∼Rule() . . . 185
7.37.3 Member Function Documentation . . . 185
7.37.3.1 D() . . . 185
7.37.3.2 operator()() . . . 185
7.38 Rules< Array_Type, Data_Type > Class Template Reference . . . 185
7.38.1 Detailed Description . . . 186
7.38.2 Constructor & Destructor Documentation . . . 186
7.38.2.1 Rules() [1/2] . . . 186
7.38.2.2 Rules() [2/2] . . . 186
7.38.2.3 ∼Rules() . . . 187
7.38.3 Member Function Documentation . . . 187
7.38.3.1 add_rule() [1/2] . . . 187
7.38.3.2 add_rule() [2/2] . . . 187
7.38.3.3 get_seq() . . . 187
7.38.3.4 operator()() . . . 188
7.38.3.5 operator=() . . . 188

7.38.3.6 size() . . . 188
7.39 StatsCounter< Array_Type, Data_Type > Class Template Reference . . . 189
7.39.1 Detailed Description . . . 189
7.39.2 Constructor & Destructor Documentation . . . 189
7.39.2.1 StatsCounter() [1/3] . . . 189
7.39.2.2 StatsCounter() [2/3] . . . 190
7.39.2.3 StatsCounter() [3/3] . . . 190
7.39.2.4 ~StatsCounter() . . . 190
7.39.3 Member Function Documentation . . . 190
7.39.3.1 add_counter() . . . 190
7.39.3.2 count_all() . . . 191
7.39.3.3 count_current() . . . 191
7.39.3.4 count_init() . . . 191
7.39.3.5 get_counters() . . . 191
7.39.3.6 get_descriptions() . . . 191
7.39.3.7 get_names() . . . 191
7.39.3.8 reset_array() . . . 191
7.39.3.9 set_counters() . . . 192
7.39.3.10 size() . . . 192
7.40 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference . . . 192
7.40.1 Detailed Description . . . 194
7.40.2 Constructor & Destructor Documentation . . . 194
7.40.2.1 Support() [1/3] . . . 194
7.40.2.2 Support() [2/3] . . . 195
7.40.2.3 Support() [3/3] . . . 195
7.40.2.4 ~Support() . . . 195
7.40.3 Member Function Documentation . . . 195
7.40.3.1 add_counter() . . . 195
7.40.3.2 add_rule() [1/2] . . . 196
7.40.3.3 add_rule() [2/2] . . . 196
7.40.3.4 add_rule_dyn() [1/2] . . . 196
7.40.3.5 add_rule_dyn() [2/2] . . . 196
7.40.3.6 calc() . . . 196
7.40.3.7 eval_rules_dyn() . . . 197
7.40.3.8 get_counters() . . . 197
7.40.3.9 get_counts() . . . 197
7.40.3.10 get_current_stats() . . . 197
7.40.3.11 get_data() . . . 198
7.40.3.12 get_rules() . . . 198
7.40.3.13 get_rules_dyn() . . . 198
7.40.3.14 init_support() . . . 198

7.40.3.15 print() . . . 198
7.40.3.16 reset_array() [1/2] . . . 199
7.40.3.17 reset_array() [2/2] . . . 199
7.40.3.18 set_counters() . . . 199
7.40.3.19 set_rules() . . . 199
7.40.3.20 set_rules_dyn() . . . 199
7.40.4 Member Data Documentation . . . 199
7.40.4.1 change_stats . . . 200
7.40.4.2 coordiantes_n_free . . . 200
7.40.4.3 coordiantes_n_locked . . . 200
7.40.4.4 coordinates_free . . . 200
7.40.4.5 coordinates_locked . . . 200
7.40.4.6 current_stats . . . 201
7.40.4.7 delete_counters . . . 201
7.40.4.8 delete_rules . . . 201
7.40.4.9 delete_rules_dyn . . . 201
7.40.4.10 hashes . . . 201
7.40.4.11 hashes_initialized . . . 202
7.40.4.12 M . . . 202
7.40.4.13 max_num_elements . . . 202
7.40.4.14 N . . . 202
7.40.4.15 n_counters . . . 202
7.41 vecHasher< T > Struct Template Reference . . . 203
7.41.1 Detailed Description . . . 203
7.41.2 Member Function Documentation . . . 203
7.41.2.1 operator()() . . . 203

**8 File Documentation** **205**

8.1 include/barry/barray-bones.hpp File Reference . . . 205
8.2 include/barry/barray-iterator.hpp File Reference . . . 205
8.3 include/barry/barray-meat-operators.hpp File Reference . . . 206
8.3.1 Macro Definition Documentation . . . 206
8.3.1.1 BARRAY_TEMPLATE . . . 207
8.3.1.2 BARRAY_TEMPLATE_ARGS . . . 207
8.3.1.3 BARRAY_TYPE . . . 207
8.3.1.4 COL . . . 207
8.3.1.5 ROW . . . 207
8.3.2 Function Documentation . . . 207
8.3.2.1 BARRAY_TEMPLATE() [1/6] . . . 208
8.3.2.2 BARRAY_TEMPLATE() [2/6] . . . 208
8.3.2.3 BARRAY_TEMPLATE() [3/6] . . . 208
8.3.2.4 BARRAY_TEMPLATE() [4/6] . . . 208

8.3.2.5 BARRAY_TEMPLATE() [5/6] . . . 208
8.3.2.6 BARRAY_TEMPLATE() [6/6] . . . 209
8.3.2.7 BARRAY_TEMPLATE_ARGS() . . . 209
8.3.2.8 BARRAY_TYPE() . . . 209
8.3.2.9 for() . . . 209
8.3.2.10 operator()() . . . 209
8.3.3 Variable Documentation . . . 209
8.3.3.1 rhs . . . 210
8.3.3.2 this . . . 210
8.4 include/barry/barray-meat.hpp File Reference . . . 210
8.4.1 Macro Definition Documentation . . . 212
8.4.1.1 BARRAY_TEMPLATE . . . 212
8.4.1.2 BARRAY_TEMPLATE_ARGS . . . 212
8.4.1.3 BARRAY_TYPE . . . 213
8.4.1.4 COL . . . 213
8.4.1.5 ROW . . . 213
8.4.2 Function Documentation . . . 213
8.4.2.1 ans() . . . 213
8.4.2.2 BARRAY_TEMPLATE() [1/24] . . . 213
8.4.2.3 BARRAY_TEMPLATE() [2/24] . . . 214
8.4.2.4 BARRAY_TEMPLATE() [3/24] . . . 214
8.4.2.5 BARRAY_TEMPLATE() [4/24] . . . 214
8.4.2.6 BARRAY_TEMPLATE() [5/24] . . . 214
8.4.2.7 BARRAY_TEMPLATE() [6/24] . . . 214
8.4.2.8 BARRAY_TEMPLATE() [7/24] . . . 214
8.4.2.9 BARRAY_TEMPLATE() [8/24] . . . 215
8.4.2.10 BARRAY_TEMPLATE() [9/24] . . . 215
8.4.2.11 BARRAY_TEMPLATE() [10/24] . . . 215
8.4.2.12 BARRAY_TEMPLATE() [11/24] . . . 215
8.4.2.13 BARRAY_TEMPLATE() [12/24] . . . 215
8.4.2.14 BARRAY_TEMPLATE() [13/24] . . . 216
8.4.2.15 BARRAY_TEMPLATE() [14/24] . . . 216
8.4.2.16 BARRAY_TEMPLATE() [15/24] . . . 216
8.4.2.17 BARRAY_TEMPLATE() [16/24] . . . 216
8.4.2.18 BARRAY_TEMPLATE() [17/24] . . . 216
8.4.2.19 BARRAY_TEMPLATE() [18/24] . . . 216
8.4.2.20 BARRAY_TEMPLATE() [19/24] . . . 217
8.4.2.21 BARRAY_TEMPLATE() [20/24] . . . 217
8.4.2.22 BARRAY_TEMPLATE() [21/24] . . . 217
8.4.2.23 BARRAY_TEMPLATE() [22/24] . . . 217
8.4.2.24 BARRAY_TEMPLATE() [23/24] . . . 217
8.4.2.25 BARRAY_TEMPLATE() [24/24] . . . 217

8.4.2.26 COL() . . . 218
8.4.2.27 for() [1/3] . . . 218
8.4.2.28 for() [2/3] . . . 218
8.4.2.29 for() [3/3] . . . 218
8.4.2.30 if() [1/17] . . . 218
8.4.2.31 if() [2/17] . . . 218
8.4.2.32 if() [3/17] . . . 219
8.4.2.33 if() [4/17] . . . 219
8.4.2.34 if() [5/17] . . . 219
8.4.2.35 if() [6/17] . . . 219
8.4.2.36 if() [7/17] . . . 219
8.4.2.37 if() [8/17] . . . 219
8.4.2.38 if() [9/17] . . . 220
8.4.2.39 if() [10/17] . . . 220
8.4.2.40 if() [11/17] . . . 220
8.4.2.41 if() [12/17] . . . 220
8.4.2.42 if() [13/17] . . . 220
8.4.2.43 if() [14/17] . . . 220
8.4.2.44 if() [15/17] . . . 220
8.4.2.45 if() [16/17] . . . 221
8.4.2.46 if() [17/17] . . . 221
8.4.2.47 M() . . . 221
8.4.2.48 resize() [1/2] . . . 221
8.4.2.49 resize() [2/2] . . . 221
8.4.2.50 return() . . . 221
8.4.2.51 ROW() [1/2] . . . 222
8.4.2.52 ROW() [2/2] . . . 222
8.4.3 Variable Documentation . . . 222
8.4.3.1 add . . . 222
8.4.3.2 ans . . . 222
8.4.3.3 Array_ . . . 222
8.4.3.4 check_bounds . . . 223
8.4.3.5 check_exists . . . 223
8.4.3.6 col0 . . . 223
8.4.3.7 const . . . 223
8.4.3.8 copy_data . . . 224
8.4.3.9 data . . . 224
8.4.3.10 delete_data . . . 224
8.4.3.11 delete_data_ . . . 224
8.4.3.12 else . . . 224
8.4.3.13 false . . . 225
8.4.3.14 first . . . 225

8.4.3.15 i1 . . . 225
8.4.3.16 j . . . 225
8.4.3.17 j0 . . . 225
8.4.3.18 j1 . . . 225
8.4.3.19 M . . . 226
8.4.3.20 M_ . . . 226
8.4.3.21 N . . . 226
8.4.3.22 NCells . . . 226
8.4.3.23 report . . . 226
8.4.3.24 return . . . 227
8.4.3.25 row0 . . . 227
8.4.3.26 search . . . 227
8.4.3.27 source . . . 227
8.4.3.28 target . . . 227
8.4.3.29 v . . . 227
8.4.3.30 value . . . 228
8.5 include/barry/barraycell-bones.hpp File Reference . . . 228
8.6 include/barry/barraycell-meat.hpp File Reference . . . 228
8.7 include/barry/barraydense-bones.hpp File Reference . . . 229
8.8 include/barry/barraydense-meat-operators.hpp File Reference . . . 229
8.8.1 Macro Definition Documentation . . . 230
8.8.1.1 BDENSE_TEMPLATE . . . 230
8.8.1.2 BDENSE_TEMPLATE_ARGS . . . 230
8.8.1.3 BDENSE_TYPE . . . 230
8.8.1.4 COL . . . 230
8.8.1.5 POS . . . 231
8.8.1.6 POS_N . . . 231
8.8.1.7 ROW . . . 231
8.8.2 Function Documentation . . . 231
8.8.2.1 BDENSE_TEMPLATE() [1/4] . . . 231
8.8.2.2 BDENSE_TEMPLATE() [2/4] . . . 231
8.8.2.3 BDENSE_TEMPLATE() [3/4] . . . 232
8.8.2.4 BDENSE_TEMPLATE() [4/4] . . . 232
8.8.2.5 BDENSE_TEMPLATE_ARGS() . . . 232
8.8.2.6 BDENSE_TYPE() . . . 232
8.9 include/barry/barraydense-meat.hpp File Reference . . . 232
8.9.1 Macro Definition Documentation . . . 235
8.9.1.1 BDENSE_TEMPLATE . . . 235
8.9.1.2 BDENSE_TEMPLATE_ARGS . . . 235
8.9.1.3 BDENSE_TYPE . . . 235
8.9.1.4 COL . . . 235
8.9.1.5 POS . . . 235

8.9.1.6 POS_N . . . 236
8.9.1.7 ROW . . . 236
8.9.1.8 ZERO_CELL . . . 236
8.9.2 Function Documentation . . . 236
8.9.2.1 ans() . . . 236
8.9.2.2 BDENSE_TEMPLATE() [1/39] . . . 236
8.9.2.3 BDENSE_TEMPLATE() [2/39] . . . 237
8.9.2.4 BDENSE_TEMPLATE() [3/39] . . . 237
8.9.2.5 BDENSE_TEMPLATE() [4/39] . . . 237
8.9.2.6 BDENSE_TEMPLATE() [5/39] . . . 237
8.9.2.7 BDENSE_TEMPLATE() [6/39] . . . 237
8.9.2.8 BDENSE_TEMPLATE() [7/39] . . . 237
8.9.2.9 BDENSE_TEMPLATE() [8/39] . . . 238
8.9.2.10 BDENSE_TEMPLATE() [9/39] . . . 238
8.9.2.11 BDENSE_TEMPLATE() [10/39] . . . 238
8.9.2.12 BDENSE_TEMPLATE() [11/39] . . . 238
8.9.2.13 BDENSE_TEMPLATE() [12/39] . . . 238
8.9.2.14 BDENSE_TEMPLATE() [13/39] . . . 239
8.9.2.15 BDENSE_TEMPLATE() [14/39] . . . 239
8.9.2.16 BDENSE_TEMPLATE() [15/39] . . . 239
8.9.2.17 BDENSE_TEMPLATE() [16/39] . . . 239
8.9.2.18 BDENSE_TEMPLATE() [17/39] . . . 239
8.9.2.19 BDENSE_TEMPLATE() [18/39] . . . 240
8.9.2.20 BDENSE_TEMPLATE() [19/39] . . . 240
8.9.2.21 BDENSE_TEMPLATE() [20/39] . . . 240
8.9.2.22 BDENSE_TEMPLATE() [21/39] . . . 240
8.9.2.23 BDENSE_TEMPLATE() [22/39] . . . 240
8.9.2.24 BDENSE_TEMPLATE() [23/39] . . . 241
8.9.2.25 BDENSE_TEMPLATE() [24/39] . . . 241
8.9.2.26 BDENSE_TEMPLATE() [25/39] . . . 241
8.9.2.27 BDENSE_TEMPLATE() [26/39] . . . 241
8.9.2.28 BDENSE_TEMPLATE() [27/39] . . . 241
8.9.2.29 BDENSE_TEMPLATE() [28/39] . . . 242
8.9.2.30 BDENSE_TEMPLATE() [29/39] . . . 242
8.9.2.31 BDENSE_TEMPLATE() [30/39] . . . 242
8.9.2.32 BDENSE_TEMPLATE() [31/39] . . . 242
8.9.2.33 BDENSE_TEMPLATE() [32/39] . . . 242
8.9.2.34 BDENSE_TEMPLATE() [33/39] . . . 242
8.9.2.35 BDENSE_TEMPLATE() [34/39] . . . 243
8.9.2.36 BDENSE_TEMPLATE() [35/39] . . . 243
8.9.2.37 BDENSE_TEMPLATE() [36/39] . . . 243
8.9.2.38 BDENSE_TEMPLATE() [37/39] . . . 243

8.9.2.39 BDENSE_TEMPLATE() [38/39] . . . 243
8.9.2.40 BDENSE_TEMPLATE() [39/39] . . . 243
8.9.2.41 for() . . . 244
8.9.2.42 if() [1/4] . . . 244
8.9.2.43 if() [2/4] . . . 244
8.9.2.44 if() [3/4] . . . 244
8.9.2.45 if() [4/4] . . . 244
8.9.2.46 insert_cell() [1/2] . . . 244
8.9.2.47 insert_cell() [2/2] . . . 245
8.9.2.48 M() . . . 245
8.9.2.49 resize() [1/6] . . . 245
8.9.2.50 resize() [2/6] . . . 245
8.9.2.51 resize() [3/6] . . . 245
8.9.2.52 resize() [4/6] . . . 245
8.9.2.53 resize() [5/6] . . . 246
8.9.2.54 resize() [6/6] . . . 246
8.9.2.55 rm_cell() [1/3] . . . 246
8.9.2.56 rm_cell() [2/3] . . . 246
8.9.2.57 rm_cell() [3/3] . . . 246
8.9.2.58 va_end() . . . 246
8.9.2.59 va_start() . . . 247
8.9.2.60 vprintf() . . . 247
8.9.3 Variable Documentation . . . 247
8.9.3.1 add . . . 247
8.9.3.2 ans . . . 247
8.9.3.3 check_bounds . . . 247
8.9.3.4 check_exists . . . 248
8.9.3.5 col . . . 248
8.9.3.6 const . . . 248
8.9.3.7 copy_data . . . 248
8.9.3.8 data . . . 248
8.9.3.9 delete_data . . . 249
8.9.3.10 delete_data_ . . . 249
8.9.3.11 el . . . 249
8.9.3.12 el_colsums . . . 249
8.9.3.13 el_rowsums . . . 249
8.9.3.14 else . . . 250
8.9.3.15 false . . . 250
8.9.3.16 i1 . . . 250
8.9.3.17 j . . . 250
8.9.3.18 j0 . . . 250
8.9.3.19 j1 . . . 250

8.9.3.20 M . . . 251
8.9.3.21 M_ . . . 251
8.9.3.22 N . . . 251
8.9.3.23 report . . . 251
8.9.3.24 return . . . 251
8.9.3.25 source . . . 252
8.9.3.26 target . . . 252
8.9.3.27 v . . . 252
8.9.3.28 val0 . . . 252
8.9.3.29 val1 . . . 252
8.9.3.30 value . . . 252
8.10 include/barry/barraydensecell-bones.hpp File Reference . . . 253
8.10.1 Macro Definition Documentation . . . 253
8.10.1.1 POS . . . 253
8.11 include/barry/barraydensecell-meat.hpp File Reference . . . 254
8.11.1 Macro Definition Documentation . . . 254
8.11.1.1 POS . . . 254
8.12 include/barry/barraydensecol-bones.hpp File Reference . . . 254
8.12.1 Macro Definition Documentation . . . 255
8.12.1.1 POS . . . 255
8.12.1.2 POS_N . . . 255
8.12.1.3 ZERO_CELL . . . 255
8.13 include/barry/barraydenserow-bones.hpp File Reference . . . 256
8.13.1 Macro Definition Documentation . . . 256
8.13.1.1 POS . . . 256
8.13.1.2 POS_N . . . 257
8.13.1.3 ZERO_CELL . . . 257
8.14 include/barry/barrayrow-bones.hpp File Reference . . . 257
8.15 include/barry/barrayrow-meat.hpp File Reference . . . 257
8.15.1 Macro Definition Documentation . . . 257
8.15.1.1 BROW_TEMPLATE . . . 258
8.15.1.2 BROW_TEMPLATE_ARGS . . . 258
8.15.1.3 BROW_TYPE . . . 258
8.15.2 Function Documentation . . . 258
8.15.2.1 BROW_TEMPLATE() [1/5] . . . 258
8.15.2.2 BROW_TEMPLATE() [2/5] . . . 258
8.15.2.3 BROW_TEMPLATE() [3/5] . . . 259
8.15.2.4 BROW_TEMPLATE() [4/5] . . . 259
8.15.2.5 BROW_TEMPLATE() [5/5] . . . 259
8.16 include/barry/barrayvector-bones.hpp File Reference . . . 259
8.17 include/barry/barrayvector-meat.hpp File Reference . . . 260
8.18 include/barry/barry-configuration.hpp File Reference . . . 260

8.18.1 Macro Definition Documentation . . . 261
8.18.1.1 BARRY_CHECK_SUPPORT . . . 261
8.18.1.2 BARRY_ISFINITE . . . 261
8.18.1.3 BARRY_MAX_NUM_ELEMENTS . . . 261
8.18.1.4 BARRY_SAFE_EXP . . . 261
8.18.1.5 printf_barry . . . 261
8.18.2 Typedef Documentation . . . 261
8.18.2.1 Map . . . 262
8.19 include/barry/barry-debug.hpp File Reference . . . 262
8.19.1 Macro Definition Documentation . . . 262
8.19.1.1 BARRY_DEBUG_LEVEL . . . 262
8.20 include/barry/barry-macros.hpp File Reference . . . 263
8.20.1 Macro Definition Documentation . . . 263
8.20.1.1 BARRY_ONE . . . 263
8.20.1.2 BARRY_ONE_DENSE . . . 263
8.20.1.3 BARRY_UNUSED . . . 264
8.20.1.4 BARRY_ZERO . . . 264
8.20.1.5 BARRY_ZERO_DENSE . . . 264
8.21 include/barry/barry.hpp File Reference . . . 264
8.21.1 Macro Definition Documentation . . . 265
8.21.1.1 BARRY_HPP . . . 266
8.21.1.2 BARRY_VERSION . . . 266
8.21.1.3 BARRY_VERSION_MAYOR . . . 266
8.21.1.4 BARRY_VERSION_MINOR . . . 266
8.21.1.5 COUNTER_FUNCTION . . . 266
8.21.1.6 COUNTER_LAMBDA . . . 267
8.21.1.7 RULE_FUNCTION . . . 267
8.21.1.8 RULE_LAMBDA . . . 267
8.22 include/barry/cell-bones.hpp File Reference . . . 267
8.23 include/barry/cell-meat.hpp File Reference . . . 268
8.24 include/barry/col-bones.hpp File Reference . . . 268
8.25 include/barry/counters-bones.hpp File Reference . . . 268
8.26 include/barry/counters-meat.hpp File Reference . . . 269
8.26.1 Macro Definition Documentation . . . 270
8.26.1.1 COUNTER_TEMPLATE . . . 270
8.26.1.2 COUNTER_TEMPLATE_ARGS . . . 270
8.26.1.3 COUNTER_TYPE . . . 270
8.26.1.4 COUNTERS_TEMPLATE . . . 271
8.26.1.5 COUNTERS_TEMPLATE_ARGS . . . 271
8.26.1.6 COUNTERS_TYPE . . . 271
8.26.2 Function Documentation . . . 271
8.26.2.1 count_fun() . . . 271

8.26.2.2 COUNTER_TEMPLATE() [1/7] . . . 271
8.26.2.3 COUNTER_TEMPLATE() [2/7] . . . 272
8.26.2.4 COUNTER_TEMPLATE() [3/7] . . . 272
8.26.2.5 COUNTER_TEMPLATE() [4/7] . . . 272
8.26.2.6 COUNTER_TEMPLATE() [5/7] . . . 272
8.26.2.7 COUNTER_TEMPLATE() [6/7] . . . 272
8.26.2.8 COUNTER_TEMPLATE() [7/7] . . . 272
8.26.2.9 Counters() . . . 273
8.26.2.10 COUNTERS_TEMPLATE() [1/7] . . . 273
8.26.2.11 COUNTERS_TEMPLATE() [2/7] . . . 273
8.26.2.12 COUNTERS_TEMPLATE() [3/7] . . . 273
8.26.2.13 COUNTERS_TEMPLATE() [4/7] . . . 273
8.26.2.14 COUNTERS_TEMPLATE() [5/7] . . . 273
8.26.2.15 COUNTERS_TEMPLATE() [6/7] . . . 274
8.26.2.16 COUNTERS_TEMPLATE() [7/7] . . . 274
8.26.2.17 data() . . . 274
8.26.2.18 desc() . . . 274
8.26.2.19 init_fun() [1/3] . . . 274
8.26.2.20 init_fun() [2/3] . . . 275
8.26.2.21 init_fun() [3/3] . . . 275
8.26.2.22 name() . . . 275
8.26.3 Variable Documentation . . . 275
8.26.3.1 count_fun_ . . . 275
8.26.3.2 counter . . . 275
8.26.3.3 counter_ . . . 276
8.26.3.4 data_ . . . 276
8.26.3.5 desc_ . . . 276
8.26.3.6 i . . . 276
8.26.3.7 init_fun_ . . . 277
8.26.3.8 j . . . 277
8.26.3.9 name_ . . . 277
8.26.3.10 noexcept . . . 277
8.26.3.11 return . . . 277
8.26.3.12 this . . . 278
8.27 include/barry/counters/defm.hpp File Reference . . . 278
8.27.1 Macro Definition Documentation . . . 279
8.27.1.1 DEFM_COUNTER . . . 279
8.27.1.2 DEFM_COUNTER_LAMBDA . . . 280
8.27.1.3 DEFM_RULE . . . 280
8.27.1.4 DEFM_RULE_LAMBDA . . . 280
8.27.2 Typedef Documentation . . . 280
8.27.2.1 DEFMArray . . . 280

8.27.2.2 DEFMCounter . . . 281
8.27.2.3 DEFMCounters . . . 281
8.27.2.4 DEFMModel . . . 281
8.27.2.5 DEFMRule . . . 281
8.27.2.6 DEFMRules . . . 281
8.27.2.7 DEFMStatsCounter . . . 281
8.27.2.8 DEFMSupport . . . 282
8.28 include/barry/models/defm.hpp File Reference . . . 282
8.29 include/barry/counters/network-css.hpp File Reference . . . 282
8.29.1 Macro Definition Documentation . . . 284
8.29.1.1 CSS_APPEND . . . 284
8.29.1.2 CSS_CASE_ELSE . . . 284
8.29.1.3 CSS_CASE_PERCEIVED . . . 284
8.29.1.4 CSS_CASE_TRUTH . . . 285
8.29.1.5 CSS_CHECK_SIZE . . . 285
8.29.1.6 CSS_CHECK_SIZE_INIT . . . 285
8.29.1.7 CSS_NET_COUNTER_LAMBDA_INIT . . . 285
8.29.1.8 CSS_PERCEIVED_CELLS . . . 286
8.29.1.9 CSS_SIZE . . . 286
8.29.1.10 CSS_TRUE_CELLS . . . 286
8.29.2 Function Documentation . . . 286
8.29.2.1 counter_css_census01() . . . 286
8.29.2.2 counter_css_census02() . . . 287
8.29.2.3 counter_css_census03() . . . 287
8.29.2.4 counter_css_census04() . . . 287
8.29.2.5 counter_css_census05() . . . 287
8.29.2.6 counter_css_census06() . . . 288
8.29.2.7 counter_css_census07() . . . 288
8.29.2.8 counter_css_census08() . . . 288
8.29.2.9 counter_css_census09() . . . 288
8.29.2.10 counter_css_census10() . . . 289
8.29.2.11 counter_css_completely_false_recip_comiss() . . . 289
8.29.2.12 counter_css_completely_false_recip_omiss() . . . 289
8.29.2.13 counter_css_mixed_recip() . . . 289
8.29.2.14 counter_css_partially_false_recip_commi() . . . 289
8.29.2.15 counter_css_partially_false_recip_omiss() . . . 290
8.30 include/barry/counters/network.hpp File Reference . . . 290
8.30.1 Macro Definition Documentation . . . 293
8.30.1.1 BARRY_ZERO_NETWORK . . . 293
8.30.1.2 BARRY_ZERO_NETWORK_DENSE . . . 294
8.30.1.3 NET_C_DATA_IDX . . . 294
8.30.1.4 NET_C_DATA_NUM . . . 294

8.30.1.5 NETWORK_COUNTER . . . 294
8.30.1.6 NETWORK_COUNTER_LAMBDA . . . 294
8.30.1.7 NETWORK_RULE . . . 295
8.30.1.8 NETWORK_RULE_LAMBDA . . . 295
8.30.1.9 NETWORKDENSE_COUNTER_LAMBDA . . . 295
8.30.2 Typedef Documentation . . . 295
8.30.2.1 NetCounter . . . 295
8.30.2.2 NetCounters . . . 296
8.30.2.3 NetModel . . . 296
8.30.2.4 NetRule . . . 296
8.30.2.5 NetRules . . . 296
8.30.2.6 NetStatsCounter . . . 296
8.30.2.7 NetSupport . . . 296
8.30.2.8 Network . . . 297
8.30.2.9 NetworkDense . . . 297
8.30.3 Function Documentation . . . 297
8.30.3.1 rules_zerodiag() . . . 297
8.31 include/barry/counters/phylo.hpp File Reference . . . 297
8.31.1 Macro Definition Documentation . . . 300
8.31.1.1 DEFAULT_DUPLICATION . . . 300
8.31.1.2 DUPL_DUPL . . . 300
8.31.1.3 DUPL_EITH . . . 300
8.31.1.4 DUPL_SPEC . . . 300
8.31.1.5 IF_MATCHES . . . 301
8.31.1.6 IF_NOTMATCHES . . . 301
8.31.1.7 IS_DUPLICATION . . . 301
8.31.1.8 IS_EITHER . . . 301
8.31.1.9 IS_SPECIATION . . . 301
8.31.1.10 MAKE_DUPL_VARS . . . 302
8.31.1.11 PHYLO_CHECK_MISSING . . . 302
8.31.1.12 PHYLO_COUNTER_LAMBDA . . . 302
8.31.1.13 PHYLO_RULE_DYN_LAMBDA . . . 302
8.31.2 Typedef Documentation . . . 303
8.31.2.1 PhyloArray . . . 303
8.31.2.2 PhyloCounter . . . 303
8.31.2.3 PhyloCounters . . . 303
8.31.2.4 PhyloModel . . . 303
8.31.2.5 PhyloPowerSet . . . 303
8.31.2.6 PhyloRule . . . 304
8.31.2.7 PhyloRuleData . . . 304
8.31.2.8 PhyloRuleDyn . . . 304
8.31.2.9 PhyloRules . . . 304

8.31.2.10 PhyloRulesDyn . . . 304
8.31.2.11 PhyloStatsCounter . . . 304
8.31.2.12 PhyloSupport . . . 305
8.31.3 Function Documentation . . . 305
8.31.3.1 get_last_name() . . . 305
8.32 include/barry/freqtable.hpp File Reference . . . 305
8.33 include/barry/model-bones.hpp File Reference . . . 306
8.33.1 Function Documentation . . . 306
8.33.1.1 keygen_default() . . . 306
8.34 include/barry/model-meat.hpp File Reference . . . 307
8.34.1 Macro Definition Documentation . . . 307
8.34.1.1 MODEL_TEMPLATE . . . 307
8.34.1.2 MODEL_TEMPLATE_ARGS . . . 308
8.34.1.3 MODEL_TYPE . . . 308
8.34.2 Function Documentation . . . 308
8.34.2.1 likelihood_() . . . 308
8.34.2.2 MODEL_TEMPLATE() [1/2] . . . 308
8.34.2.3 MODEL_TEMPLATE() [2/2] . . . 309
8.34.2.4 update_normalizing_constant() . . . 309
8.35 include/barry/models/defm/defm-bones.hpp File Reference . . . 309
8.36 include/barry/models/defm/defm-meat.hpp File Reference . . . 310
8.36.1 Macro Definition Documentation . . . 310
8.36.1.1 DEFM_LOOP_ARRAYS . . . 310
8.36.1.2 DEFM_RANGES . . . 310
8.36.2 Function Documentation . . . 311
8.36.2.1 keygen_defm() . . . 311
8.37 include/barry/models/geese.hpp File Reference . . . 311
8.38 include/barry/models/geese/flock-bones.hpp File Reference . . . 311
8.39 include/barry/models/geese/flock-meat.hpp File Reference . . . 312
8.40 include/barry/models/geese/geese-bones.hpp File Reference . . . 312
8.40.1 Macro Definition Documentation . . . 313
8.40.1.1 INITIALIZED . . . 313
8.40.2 Function Documentation . . . 313
8.40.2.1 keygen_full() . . . 313
8.40.2.2 RULE_FUNCTION() . . . 314
8.40.2.3 vec_diff() . . . 314
8.40.2.4 vector_caster() . . . 314
8.41 include/barry/models/geese/geese-meat-constructors.hpp File Reference . . . 314
8.42 include/barry/models/geese/geese-meat-likelihood.hpp File Reference . . . 315
8.43 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference . . . 316
8.44 include/barry/models/geese/geese-meat-predict.hpp File Reference . . . 316
8.45 include/barry/models/geese/geese-meat-predict_exhaust.hpp File Reference . . . 317

8.46 include/barry/models/geese/geese-meat-predict_sim.hpp File Reference . . . 317
8.47 include/barry/models/geese/geese-meat-simulate.hpp File Reference . . . 318
8.48 include/barry/models/geese/geese-meat.hpp File Reference . . . 318
8.49 include/barry/models/geese/geese-node-bones.hpp File Reference . . . 319
8.50 include/barry/powerset-bones.hpp File Reference . . . 319
8.51 include/barry/powerset-meat.hpp File Reference . . . 320
8.52 include/barry/progress.hpp File Reference . . . 320
8.52.1 Macro Definition Documentation . . . 321
8.52.1.1 BARRY_PROGRESS_BAR_WIDTH . . . 321
8.53 include/barry/rules-bones.hpp File Reference . . . 321
8.53.1 Function Documentation . . . 321
8.53.1.1 rule_fun_default() . . . 322
8.54 include/barry/rules-meat.hpp File Reference . . . 322
8.55 include/barry/statscounter-bones.hpp File Reference . . . 322
8.56 include/barry/statscounter-meat.hpp File Reference . . . 323
8.56.1 Macro Definition Documentation . . . 324
8.56.1.1 STATSCOUNTER_TEMPLATE . . . 324
8.56.1.2 STATSCOUNTER_TEMPLATE_ARGS . . . 324
8.56.1.3 STATSCOUNTER_TYPE . . . 324
8.56.2 Function Documentation . . . 324
8.56.2.1 clear() . . . 324
8.56.2.2 for() . . . 325
8.56.2.3 resize() . . . 325
8.56.2.4 STATSCOUNTER_TEMPLATE() [1/9] . . . 325
8.56.2.5 STATSCOUNTER_TEMPLATE() [2/9] . . . 325
8.56.2.6 STATSCOUNTER_TEMPLATE() [3/9] . . . 325
8.56.2.7 STATSCOUNTER_TEMPLATE() [4/9] . . . 325
8.56.2.8 STATSCOUNTER_TEMPLATE() [5/9] . . . 326
8.56.2.9 STATSCOUNTER_TEMPLATE() [6/9] . . . 326
8.56.2.10 STATSCOUNTER_TEMPLATE() [7/9] . . . 326
8.56.2.11 STATSCOUNTER_TEMPLATE() [8/9] . . . 326
8.56.2.12 STATSCOUNTER_TEMPLATE() [9/9] . . . 326
8.56.3 Variable Documentation . . . 326
8.56.3.1 counter . . . 327
8.56.3.2 counter_deleted . . . 327
8.56.3.3 counters . . . 327
8.56.3.4 counters_ . . . 327
8.56.3.5 current_stats . . . 327
8.56.3.6 EmptyArray . . . 328
8.56.3.7 f_ . . . 328
8.56.3.8 j . . . 328
8.56.3.9 return . . . 328

8.57 include/barry/support-bones.hpp File Reference . . . 328
8.58 include/barry/support-meat.hpp File Reference . . . 329
8.58.1 Macro Definition Documentation . . . 330
8.58.1.1 BARRY_SUPPORT_MEAT_HPP . . . 330
8.58.1.2 SUPPORT_TEMPLATE . . . 330
8.58.1.3 SUPPORT_TEMPLATE_ARGS . . . 331
8.58.1.4 SUPPORT_TYPE . . . 331
8.58.2 Function Documentation . . . 331
8.58.2.1 calc_backend_dense() . . . 331
8.58.2.2 calc_backend_sparse() . . . 331
8.58.2.3 for() . . . 331
8.58.2.4 if() [1/3] . . . 332
8.58.2.5 if() [2/3] . . . 332
8.58.2.6 if() [3/3] . . . 332
8.58.2.7 insert_cell() [1/2] . . . 332
8.58.2.8 insert_cell() [2/2] . . . 332
8.58.2.9 rm_cell() . . . 333
8.58.2.10 SUPPORT_TEMPLATE() [1/17] . . . 333
8.58.2.11 SUPPORT_TEMPLATE() [2/17] . . . 333
8.58.2.12 SUPPORT_TEMPLATE() [3/17] . . . 333
8.58.2.13 SUPPORT_TEMPLATE() [4/17] . . . 333
8.58.2.14 SUPPORT_TEMPLATE() [5/17] . . . 334
8.58.2.15 SUPPORT_TEMPLATE() [6/17] . . . 334
8.58.2.16 SUPPORT_TEMPLATE() [7/17] . . . 334
8.58.2.17 SUPPORT_TEMPLATE() [8/17] . . . 334
8.58.2.18 SUPPORT_TEMPLATE() [9/17] . . . 334
8.58.2.19 SUPPORT_TEMPLATE() [10/17] . . . 334
8.58.2.20 SUPPORT_TEMPLATE() [11/17] . . . 335
8.58.2.21 SUPPORT_TEMPLATE() [12/17] . . . 335
8.58.2.22 SUPPORT_TEMPLATE() [13/17] . . . 335
8.58.2.23 SUPPORT_TEMPLATE() [14/17] . . . 335
8.58.2.24 SUPPORT_TEMPLATE() [15/17] . . . 335
8.58.2.25 SUPPORT_TEMPLATE() [16/17] . . . 336
8.58.2.26 SUPPORT_TEMPLATE() [17/17] . . . 336
8.58.3 Variable Documentation . . . 336
8.58.3.1 array_bank . . . 336
8.58.3.2 change_stats_different . . . 336
8.58.3.3 coord_i . . . 336
8.58.3.4 coord_j . . . 336
8.58.3.5 counters . . . 337
8.58.3.6 counters_ . . . 337
8.58.3.7 delete_counters . . . 337

8.58.3.8 delete_rules . . . 337
8.58.3.9 delete_rules_dyn . . . 337
8.58.3.10 else . . . 338
8.58.3.11 f_ . . . 338
8.58.3.12 hashes . . . 338
8.58.3.13 return . . . 338
8.58.3.14 rules . . . 338
8.58.3.15 rules_ . . . 339
8.58.3.16 rules_dyn . . . 339
8.58.3.17 stats_bank . . . 339
8.58.3.18 tmp_chng . . . 339
8.59 include/barry/typedefs.hpp File Reference . . . 340
8.59.1 Typedef Documentation . . . 341
8.59.1.1 Col_type . . . 342
8.59.1.2 Counter_fun_type . . . 342
8.59.1.3 Counts_type . . . 342
8.59.1.4 MapVec_type . . . 342
8.59.1.5 Row_type . . . 343
8.59.1.6 Rule_fun_type . . . 343
8.59.1.7 uint . . . 343
8.59.2 Function Documentation . . . 343
8.59.2.1 vec_equal() . . . 343
8.59.2.2 vec_equal_approx() . . . 344
8.59.2.3 vec_inner_prod() [1/2] . . . 344
8.59.2.4 vec_inner_prod() [2/2] . . . 344
8.60 README.md File Reference . . . 344

**Index** **345**

# Chapter 1

# Main Page

## Barry: your to-go motif accountant

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

Among the key features included in barry, we have:

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

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

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

# Examples

## Counting statistics in a graph

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

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

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

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

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

  // Initializing the data. The program deals with freing the memory
  net.set_data(new netcounters::NetworkData, true);
  // Creating counter object for the network and adding stats to count
  netcounters::NetStatsCounter counter(&net);
  netcounters::counter_edges(counter.counters);
  netcounters::counter_ttriads(counter.counters);
  netcounters::counter_isolates(counter.counters);
  netcounters::counter_ctriads(counter.counters);
  netcounters::counter_mutual(counter.counters);

  // Counting and printing the results
  std::vector< double > counts = counter.count_all();

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

  return 0;
}
```

Compiling this program using g++

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

Yields the following output:

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

# Features

## Efficient memory usage

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

# Documentation

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

# Code of Conduct

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

# Chapter 2

# Module Index

## 2.1 Modules

Here is a list of all modules:


Counting . . . . . 11
Statistical Models . . . . . 11
DEFMArray counters . . . . . 12
Phylo counters . . . . . 21
Phylo rules . . . . . 29

# 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 . . . 35
BArrayCell< Cell_Type, Data_Type > . . . 49
BArrayCell_const< Cell_Type, Data_Type > . . . 52
BArrayDense< Cell_Type, Data_Type >
: Baseline class for binary arrays . . . 54
BArrayDenseCell< Cell_Type, Data_Type > . . . 69
BArrayDenseCell_const< Cell_Type, Data_Type > . . . 73
BArrayDenseCol< Cell_Type, Data_Type > . . . 73
BArrayDenseCol_const< Cell_Type, Data_Type > . . . 76
BArrayDenseRow< Cell_Type, Data_Type > . . . 78
BArrayDenseRow_const< Cell_Type, Data_Type > . . . 80
BArrayRow< Cell_Type, Data_Type > . . . 82
BArrayRow_const< Cell_Type, Data_Type > . . . 85
BArrayVector< Cell_Type, Data_Type >
: Row or column of a `BArray` . . . 87
BArrayVector_const< Cell_Type, Data_Type > . . . 91
Cell< Cell_Type >
: Entries in BArray. For now, it only has two members: . . . 94
Cell_const< Cell_Type > . . . 100
ConstBArrayRowIter< Cell_Type, Data_Type > . . . 100
Counter< Array_Type, Data_Type >
: A counter function based on change statistics . . . 102
Counters< Array_Type, Data_Type >
: Vector of counters . . . 107
DEFM . . . 111
DEFMCounterData
: Data class used to store arbitrary uint or double vectors . . . 113
DEFMData . . . 116
DEFMRuleData . . . 120
Entries< Cell_Type >
: A wrapper class to store `source`, `target`, `val` from a `BArray` object . . . 121
Flock
: A Flock is a group of Geese . . . 123
FreqTable< T >
: Frequency table of vectors . . . 130

Geese
Annotated Phylo Model . . . 133
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: . . . 145
NetCounterData
Data class used to store arbitrary uint or double vectors . . . 160
NetworkData
Data class for Networks . . . 161
Node
A single node for the model . . . 163
NodeData
Data definition for the PhyloArray class . . . 169
PhyloCounterData . . . 170
PhyloRuleDynData . . . 173
PowerSet< Array_Type, Data_Rule_Type >
Powerset of a binary array . . . 175
Progress
A simple progress bar . . . 182
Rule< Array_Type, Data_Type >
Rule for determining if a cell should be included in a sequence . . . 183
Rules< Array_Type, Data_Type >
Vector of objects of class Rule . . . 185
StatsCounter< Array_Type, Data_Type >
Count stats for a single Array . . . 189
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >
Compute the support of sufficient statistics . . . 192
vecHasher< T > . . . 203

# Chapter 4

# File Index

## 4.1 File List

Here is a list of all files with brief descriptions:


include/barry/barray-bones.hpp . . . 205
include/barry/barray-iterator.hpp . . . 205
include/barry/barray-meat-operators.hpp . . . 206
include/barry/barray-meat.hpp . . . 210
include/barry/barraycell-bones.hpp . . . 228
include/barry/barraycell-meat.hpp . . . 228
include/barry/barraydense-bones.hpp . . . 229
include/barry/barraydense-meat-operators.hpp . . . 229
include/barry/barraydense-meat.hpp . . . 232
include/barry/barraydensecell-bones.hpp . . . 253
include/barry/barraydensecell-meat.hpp . . . 254
include/barry/barraydensecol-bones.hpp . . . 254
include/barry/barraydenserow-bones.hpp . . . 256
include/barry/barrayrow-bones.hpp . . . 257
include/barry/barrayrow-meat.hpp . . . 257
include/barry/barrayvector-bones.hpp . . . 259
include/barry/barrayvector-meat.hpp . . . 260
include/barry/barry-configuration.hpp . . . 260
include/barry/barry-debug.hpp . . . 262
include/barry/barry-macros.hpp . . . 263
include/barry/barry.hpp . . . 264
include/barry/cell-bones.hpp . . . 267
include/barry/cell-meat.hpp . . . 268
include/barry/col-bones.hpp . . . 268
include/barry/counters-bones.hpp . . . 268
include/barry/counters-meat.hpp . . . 269
include/barry/freqtable.hpp . . . 305
include/barry/model-bones.hpp . . . 306
include/barry/model-meat.hpp . . . 307
include/barry/powerset-bones.hpp . . . 319
include/barry/powerset-meat.hpp . . . 320
include/barry/progress.hpp . . . 320
include/barry/rules-bones.hpp . . . 321
include/barry/rules-meat.hpp . . . 322
include/barry/statscounter-bones.hpp . . . 322

include/barry/statscounter-meat.hpp . . . 323
include/barry/support-bones.hpp . . . 328
include/barry/support-meat.hpp . . . 329
include/barry/typedefs.hpp . . . 340
include/barry/counters/defm.hpp . . . 278
include/barry/counters/network-css.hpp . . . 282
include/barry/counters/network.hpp . . . 290
include/barry/counters/phylo.hpp . . . 297
include/barry/models/defm.hpp . . . 282
include/barry/models/geese.hpp . . . 311
include/barry/models/defm/defm-bones.hpp . . . 309
include/barry/models/defm/defm-meat.hpp . . . 310
include/barry/models/geese/flock-bones.hpp . . . 311
include/barry/models/geese/flock-meat.hpp . . . 312
include/barry/models/geese/geese-bones.hpp . . . 312
include/barry/models/geese/geese-meat-constructors.hpp . . . 314
include/barry/models/geese/geese-meat-likelihood.hpp . . . 315
include/barry/models/geese/geese-meat-likelihood_exhaust.hpp . . . 316
include/barry/models/geese/geese-meat-predict.hpp . . . 316
include/barry/models/geese/geese-meat-predict_exhaust.hpp . . . 317
include/barry/models/geese/geese-meat-predict_sim.hpp . . . 317
include/barry/models/geese/geese-meat-simulate.hpp . . . 318
include/barry/models/geese/geese-meat.hpp . . . 318
include/barry/models/geese/geese-node-bones.hpp . . . 319

# 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 DEFMArray counters

Counters for network models.

### Functions

- void counter_ones (DEFMCounters *counters, int covar_index=-1)

  *Prevalence of ones.*
- void counter_transition (DEFMCounters *counters, std::vector< size_t > coords, int covar_index=-1)

  *Prevalence of ones.*
- void counter_fixed_effect (DEFMCounters *counters, int covar_index, double k)

  *Prevalence of ones.*
- template<typename Tnet = Network>
  void counter_edges (NetCounters< Tnet > *counters)

  *Number of edges.*
- template<typename Tnet = Network>
  void counter_isolates (NetCounters< Tnet > *counters)

  *Number of isolated vertices.*
- template<> void counter_isolates (NetCounters< NetworkDense > *counters)
- template<typename Tnet = Network>
  void counter_mutual (NetCounters< Tnet > *counters)

  *Number of mutual ties.*
- template<typename Tnet = Network>
  void counter_istar2 (NetCounters< Tnet > *counters)
- template<> void counter_istar2 (NetCounters< NetworkDense > *counters)
- template<typename Tnet = Network>
  void counter_ostar2 (NetCounters< Tnet > *counters)
- template<> void counter_ostar2 (NetCounters< NetworkDense > *counters)
- template<typename Tnet = Network>
  void counter_ttriads (NetCounters< Tnet > *counters)
- template<> void counter_ttriads (NetCounters< NetworkDense > *counters)
- template<typename Tnet = Network>
  void counter_ctriads (NetCounters< Tnet > *counters)
- template<> void counter_ctriads (NetCounters< NetworkDense > *counters)

- template<typename Tnet = Network>
  void counter_density (NetCounters< Tnet > *counters)
- template<typename Tnet = Network>
  void counter_idegree15 (NetCounters< Tnet > *counters)
- template<> void counter_idegree15 (NetCounters< NetworkDense > *counters)
- template<typename Tnet = Network>
  void counter_odegree15 (NetCounters< Tnet > *counters)
- template<> void counter_odegree15 (NetCounters< NetworkDense > *counters)
- template<typename Tnet = Network>
  void counter_absdiff (NetCounters< Tnet > *counters, uint attr_id, double alpha=1.0)

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

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

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

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

  *Counts number of vertices with a given out-degree.*

## Returns true if the cell is free

**Parameters**

| | |
|---|---|
| *rules* | A pointer to a `DEFMRules` object (`Rules<DEFMArray, bool>`). |

- void rules_markov_fixed (DEFMRules *rules, size_t markov_order)

  *Number of edges.*

### 5.3.1 Detailed Description

Counters for network models.

**Parameters**

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

### 5.3.2 Function Documentation

#### 5.3.2.1 counter_absdiff()

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

Sum of absolute attribute difference between ego and alter.

Definition at line 910 of file network.hpp.

#### 5.3.2.2 counter_ctriads() [1/2]

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

Definition at line 665 of file network.hpp.

#### 5.3.2.3 counter_ctriads() [2/2]

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

Definition at line 610 of file network.hpp.

#### 5.3.2.4 counter_degree()

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

Counts number of vertices with a given out-degree.

Definition at line 1328 of file network.hpp.

#### 5.3.2.5 counter_density()

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

Definition at line 731 of file network.hpp.

#### 5.3.2.6 counter_diff()

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

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

Definition at line 955 of file network.hpp.

#### 5.3.2.7 counter_edges()

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

Number of edges.

Definition at line 152 of file network.hpp.

#### 5.3.2.8 counter_fixed_effect()

```
void counter_fixed_effect (
            DEFMCounters * counters,
            int covar_index,
            double k )  [inline]
```

Prevalence of ones.

**Parameters**

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

Definition at line 356 of file defm.hpp.

#### 5.3.2.9 counter_idegree() [1/2]

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

Definition at line 1172 of file network.hpp.

#### 5.3.2.10 counter_idegree() [2/2]

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

Counts number of vertices with a given in-degree.

Definition at line 1125 of file network.hpp.

#### 5.3.2.11 counter_idegree15() [1/2]

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

Definition at line 787 of file network.hpp.

#### 5.3.2.12 counter_idegree15() [2/2]

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

Definition at line 759 of file network.hpp.

#### 5.3.2.13 counter_isolates() [1/2]

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

Definition at line 215 of file network.hpp.

#### 5.3.2.14 counter_isolates() [2/2]

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

Number of isolated vertices.

Definition at line 175 of file network.hpp.

#### 5.3.2.15 counter_istar2() [1/2]

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

Definition at line 338 of file network.hpp.

#### 5.3.2.16 counter_istar2() [2/2]

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

Definition at line 312 of file network.hpp.

#### 5.3.2.17 counter_mutual()

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

Number of mutual ties.

Definition at line 256 of file network.hpp.

#### 5.3.2.18 counter_nodecov()

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

Definition at line 1068 of file network.hpp.

#### 5.3.2.19 counter_nodeicov()

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

Definition at line 1018 of file network.hpp.

#### 5.3.2.20 counter_nodematch()

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

Definition at line 1093 of file network.hpp.

#### 5.3.2.21 counter_nodeocov()

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

Definition at line 1043 of file network.hpp.

#### 5.3.2.22 counter_odegree() [1/2]

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

Definition at line 1273 of file network.hpp.

#### 5.3.2.23 counter_odegree() [2/2]

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

Counts number of vertices with a given out-degree.

Definition at line 1225 of file network.hpp.

#### 5.3.2.24 counter_odegree15() [1/2]

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

Definition at line 864 of file network.hpp.

#### 5.3.2.25 counter_odegree15() [2/2]

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

Definition at line 836 of file network.hpp.

#### 5.3.2.26 counter_ones()

```
void counter_ones (
            DEFMCounters * counters,
            int covar_index = -1 )  [inline]
```

Prevalence of ones.

**Parameters**

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

Definition at line 187 of file defm.hpp.

#### 5.3.2.27 counter_ostar2() [1/2]

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

Definition at line 404 of file network.hpp.

#### 5.3.2.28 counter_ostar2() [2/2]

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

Definition at line 376 of file network.hpp.

#### 5.3.2.29 counter_transition()

```
void counter_transition (
            DEFMCounters * counters,
            std::vector< size_t > coords,
            int covar_index = -1 )  [inline]
```

Prevalence of ones.

**Parameters**

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

Definition at line 232 of file defm.hpp.

#### 5.3.2.30 counter_ttriads() [1/2]

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

Definition at line 531 of file network.hpp.

#### 5.3.2.31 counter_ttriads() [2/2]

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

Definition at line 441 of file network.hpp.

#### 5.3.2.32 NETWORK_COUNTER()

```
NETWORK_COUNTER (
            init_single_attr  )
```

Definition at line 999 of file network.hpp.

#### 5.3.2.33 rules_markov_fixed()

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

Number of edges.

Definition at line 390 of file defm.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_preserve_pseudogene (PhyloCounters *counters, unsigned int nfunA, unsigned int nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

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

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

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

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

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

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

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

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

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

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

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

  *Total number of neofunctionalization events sum_u sum_{w < u} [x(u,a)*(1 - x(w,a)) + (1 - x(u,a)) * x(w,a)] change stat: delta{x(u,a): 0->1} = 1 - 2 * x(w,a)*
- void counter_neofun_a2b (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

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

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

  *Indicator function. Equals to one if $k$ genes changed and zero otherwise.*
- void counter_less_than_p_prop_genes_changing (PhyloCounters *counters, double p, unsigned int duplication=DEFAULT_DUPLICATION)

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

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

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

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

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

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

### 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:

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

This algorithm implements the change statistic.

Definition at line 1393 of file phylo.hpp.

#### 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 888 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 193 of file phylo.hpp.

#### 5.4.2.4 counter_gains_from_0()

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

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

Needs to specify function a.

Definition at line 1727 of file phylo.hpp.

#### 5.4.2.5 counter_gains_k_offspring()

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

k genes gain function nfun

Definition at line 253 of file phylo.hpp.

#### 5.4.2.6 counter_genes_changing()

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

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

Definition at line 325 of file phylo.hpp.

#### 5.4.2.7 counter_k_genes_changing()

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

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

Definition at line 1491 of file phylo.hpp.

#### 5.4.2.8 counter_less_than_p_prop_genes_changing()

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

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

$<$ How many genes diverge the parent

Definition at line 1611 of file phylo.hpp.

#### 5.4.2.9 counter_longest()

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

Longest branch mutates (either by gain or by loss)

Definition at line 945 of file phylo.hpp.

#### 5.4.2.10 counter_loss()

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

Total count of losses for an specific function.

Definition at line 688 of file phylo.hpp.

#### 5.4.2.11 counter_maxfuns()

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

Cap the number of functions per gene.

Definition at line 626 of file phylo.hpp.

#### 5.4.2.12 counter_neofun()

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

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 1115 of file phylo.hpp.

#### 5.4.2.13 counter_neofun_a2b()

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

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 1260 of file phylo.hpp.

#### 5.4.2.14 counter_overall_changes()

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

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

Definition at line 740 of file phylo.hpp.

#### 5.4.2.15 counter_overall_gains()

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

Overall functional gains.

Total number of gains (irrespective of the function).

Definition at line 155 of file phylo.hpp.

#### 5.4.2.16 counter_overall_gains_from_0()

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

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

Needs to specify function a.

Definition at line 1793 of file phylo.hpp.

#### 5.4.2.17 counter_overall_loss()

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

Overall functional loss.

Definition at line 578 of file phylo.hpp.

#### 5.4.2.18 counter_pairwise_first_gain()

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

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

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

Definition at line 2045 of file phylo.hpp.

#### 5.4.2.19 counter_pairwise_neofun_singlefun()

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

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

Definition at line 1196 of file phylo.hpp.

#### 5.4.2.20 counter_pairwise_overall_change()

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

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

Needs to specify function a.

Definition at line 1841 of file phylo.hpp.

#### 5.4.2.21 counter_pairwise_preserving()

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

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

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

Definition at line 1906 of file phylo.hpp.

#### 5.4.2.22 counter_preserve_pseudogene()

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

Keeps track of how many pairs of genes preserve pseudostate.

Definition at line 394 of file phylo.hpp.

#### 5.4.2.23 counter_prop_genes_changing()

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

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

Definition at line 476 of file phylo.hpp.

#### 5.4.2.24 counter_subfun()

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

Total count of Sub-functionalization events.

It requires to specify data = {funA, funB}

Definition at line 799 of file phylo.hpp.

## 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 2177 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

- defm
- network
- phylo

### 6.2.1 Detailed Description

Tree class and TreeIterator class.

## 6.3 barry::counters::defm Namespace Reference

## 6.4 barry::counters::network Namespace Reference

## 6.5 barry::counters::phylo Namespace Reference

## 6.6 CHECK Namespace Reference

Integer constants used to specify which cell should be check.

### Variables

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

### 6.6.1 Detailed Description

Integer constants used to specify which cell should be check.

### 6.6.2 Variable Documentation

#### 6.6.2.1 BOTH

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

Definition at line 28 of file typedefs.hpp.

#### 6.6.2.2 NONE

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

Definition at line 29 of file typedefs.hpp.

#### 6.6.2.3 ONE

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

Definition at line 30 of file typedefs.hpp.

#### 6.6.2.4 TWO

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

Definition at line 31 of file typedefs.hpp.

## 6.7 EXISTS Namespace Reference

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

### Variables

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

### 6.7.1 Detailed Description

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

### 6.7.2 Variable Documentation

#### 6.7.2.1 AS_ONE

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

Definition at line 46 of file typedefs.hpp.

#### 6.7.2.2 AS_ZERO

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

Definition at line 45 of file typedefs.hpp.

#### 6.7.2.3 BOTH

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

Definition at line 39 of file typedefs.hpp.

#### 6.7.2.4 NONE

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

Definition at line 40 of file typedefs.hpp.

#### 6.7.2.5 ONE

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

Definition at line 41 of file typedefs.hpp.

#### 6.7.2.6 TWO

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

Definition at line 42 of file typedefs.hpp.

#### 6.7.2.7 UKNOWN

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

Definition at line 44 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
- bool is_dense () const noexcept

#### Constructors

***Parameters***

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

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

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

### Queries

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

***Parameters***

| | |
|---|---|
| i,j | *Coordinates* |
| check_bounds | *If `false` avoids checking bounds.* |

- bool is_empty (uint i, uint j, bool check_bounds=true) const
- uint nrow () const noexcept
- uint ncol () const noexcept
- uint nnozero () const noexcept
- Cell< Cell_Type > default_val () const

### Cell-wise insertion/deletion

***Parameters***

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

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

**Column/row wise interchange**

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

**Arithmetic operators**

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

## Public Attributes

- bool visited = false

## Friends

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

### 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 28 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 69 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 72 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 D_ptr() [1/2]

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

#### 7.1.3.6 D_ptr() [2/2]

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

#### 7.1.3.7 default_val()

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

#### 7.1.3.8 flush_data()

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

#### 7.1.3.9 get_cell()

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

#### 7.1.3.10 get_col_vec() [1/2]

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

#### 7.1.3.11 get_col_vec() [2/2]

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

#### 7.1.3.12 get_entries()

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

Get the edgelist.

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

**Returns**

Entries<Cell_Type>

#### 7.1.3.13 get_row_vec() [1/2]

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

#### 7.1.3.14 get_row_vec() [2/2]

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

#### 7.1.3.15 insert_cell() [1/3]

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

#### 7.1.3.16 insert_cell() [2/3]

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

#### 7.1.3.17 insert_cell() [3/3]

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

#### 7.1.3.18 is_dense()

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

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

#### 7.1.3.19 is_empty()

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

#### 7.1.3.20 ncol()

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

#### 7.1.3.21 nnozero()

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

#### 7.1.3.22 nrow()

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

#### 7.1.3.23 operator()() [1/2]

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

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

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

#### 7.1.3.25 operator*=()

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

#### 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-=() [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 )
```

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

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

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

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

#### 7.1.3.32 operator/=()

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

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

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

Move assignment.

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

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

Assignment constructor.

#### 7.1.3.35 operator==()

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

#### 7.1.3.36 out_of_range()

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

#### 7.1.3.37 print()

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

#### 7.1.3.38 reserve()

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

#### 7.1.3.39 resize()

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

#### 7.1.3.40 rm_cell()

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

#### 7.1.3.41 row()

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

#### 7.1.3.42 set_data()

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

Set the data object.

**Parameters**

| | |
|---|---|
| *data_* | |
| *delete_data_* | |

#### 7.1.3.43 swap_cells()

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

#### 7.1.3.44 swap_cols()

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

#### 7.1.3.45 swap_rows()

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

#### 7.1.3.46 toggle_cell()

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

#### 7.1.3.47 toggle_lock()

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

#### 7.1.3.48 transpose()

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

#### 7.1.3.49 zero_col()

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

#### 7.1.3.50 zero_row()

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

### 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 54 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/barray-bones.hpp
- include/barry/barraycell-bones.hpp
- include/barry/barraycell-meat.hpp
- include/barry/barrayrow-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/barray-bones.hpp
- include/barry/barraycell-bones.hpp
- include/barry/barraycell-meat.hpp
- include/barry/barrayrow-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
- BArrayDenseRow< Cell_Type, Data_Type > & row (uint i, bool check_bounds=true)
- const BArrayDenseRow_const< Cell_Type, Data_Type > row (uint i, bool check_bounds=true) const
- BArrayDenseCol< Cell_Type, Data_Type > & col (uint j, bool check_bounds=true)
- const BArrayDenseCol_const< Cell_Type, Data_Type > col (uint j, bool check_bounds=true) const
- Entries< Cell_Type > get_entries () const

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

#### Constructors

*Parameters*

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

- 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_ptr ()
- const Data_Type * D_ptr () const
- Data_Type & D ()
- const Data_Type & D () const

### Queries

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

*Parameters*

| | |
|---|---|
| i,j | *Coordinates* |
| check_bounds | *If `false` avoids checking bounds.* |

- bool is_empty (uint i, uint j, bool check_bounds=true) const
- uint nrow () const noexcept
- uint ncol () const noexcept
- uint nnozero () const noexcept
- Cell< Cell_Type > default_val () const

**Cell-wise insertion/deletion**

*Parameters*

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

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

**Column/row wise interchange**

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

**Arithmetic operators**

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

## Public Attributes

- bool visited = false

## Friends

- class BArrayDenseCell< Cell_Type, Data_Type >
- class BArrayDenseCol< Cell_Type, Data_Type >
- class BArrayDenseCol_const< Cell_Type, Data_Type >
- class BArrayDenseRow< Cell_Type, Data_Type >
- class BArrayDenseRow_const< Cell_Type, Data_Type >

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

**Template Parameters**

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

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

## 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 78 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 81 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() [1/2]

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

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

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

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

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

#### 7.4.3.4 colsum()

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

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

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

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

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

#### 7.4.3.7 D_ptr() [1/2]

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

#### 7.4.3.8 D_ptr() [2/2]

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

#### 7.4.3.9 default_val()

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

#### 7.4.3.10 get_cell()

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

#### 7.4.3.11 get_col_vec() [1/2]

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

#### 7.4.3.12 get_col_vec() [2/2]

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

#### 7.4.3.13 get_data()

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

#### 7.4.3.14 get_entries()

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

Get the edgelist.

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

**Returns**

Entries<Cell_Type>

#### 7.4.3.15 get_row_vec() [1/2]

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

#### 7.4.3.16 get_row_vec() [2/2]

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

#### 7.4.3.17 insert_cell() [1/2]

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

#### 7.4.3.18 insert_cell() [2/2]

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

#### 7.4.3.19 is_dense()

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

Definition at line 255 of file barraydense-bones.hpp.

#### 7.4.3.20 is_empty()

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

#### 7.4.3.21 ncol()

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

#### 7.4.3.22 nnozero()

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

#### 7.4.3.23 nrow()

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

#### 7.4.3.24 operator()() [1/2]

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

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

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

#### 7.4.3.26 operator*=()

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

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

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

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

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

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

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

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

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

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

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

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

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

#### 7.4.3.33 operator/=()

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

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

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

Move assignment.

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

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

Assignment constructor.

#### 7.4.3.36 operator==()

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

#### 7.4.3.37 out_of_range()

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

#### 7.4.3.38 print()

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

#### 7.4.3.39 reserve()

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

#### 7.4.3.40 resize()

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

#### 7.4.3.41 rm_cell()

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

#### 7.4.3.42 row() [1/2]

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

#### 7.4.3.43 row() [2/2]

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

#### 7.4.3.44 rowsum()

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

#### 7.4.3.45 set_data()

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

Set the data object.

**Parameters**

| | |
|---|---|
| *data_* | |
| *delete_↩ data_* | |

#### 7.4.3.46 swap_cells()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
void BArrayDense< Cell_Type, Data_Type >::swap_cells (
            uint i0,
```

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

#### 7.4.3.47 swap_cols()

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

#### 7.4.3.48 swap_rows()

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

#### 7.4.3.49 toggle_cell()

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

#### 7.4.3.50 toggle_lock()

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

#### 7.4.3.51 transpose()

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

#### 7.4.3.52 zero_col()

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

#### 7.4.3.53 zero_row()

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

### 7.4.4 Friends And Related Function Documentation

#### 7.4.4.1 BArrayDenseCell< Cell_Type, Data_Type >

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

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

#### 7.4.4.2 BArrayDenseCol< Cell_Type, Data_Type >

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

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

#### 7.4.4.3 BArrayDenseCol_const< Cell_Type, Data_Type >

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

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

#### 7.4.4.4 BArrayDenseRow< Cell_Type, Data_Type >

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

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

#### 7.4.4.5 BArrayDenseRow_const< Cell_Type, Data_Type >

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

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

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

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

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

## 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< Cell_Type, Data_Type > & operator= (const BArrayDenseCell< Cell_Type, Data_Type > &other)
- ~BArrayDenseCell ()
- void operator= (const Cell_Type &val)
- void operator+= (const Cell_Type &val)
- void operator-= (const Cell_Type &val)
- void operator*= (const Cell_Type &val)
- void operator/= (const Cell_Type &val)
- operator Cell_Type () const
- bool operator== (const Cell_Type &val) const

### Friends

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

## 7.5.1 Detailed Description

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

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

## 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 30 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 56 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 72 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 52 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 34 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 43 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 62 of file barraydensecell-meat.hpp.

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

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

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

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

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

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

#### 7.5.3.8 operator==()

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

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

### 7.5.4 Friends And Related Function Documentation

#### 7.5.4.1 BArrayDense< Cell_Type, Data_Type >

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

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

#### 7.5.4.2 BArrayDenseCol< Cell_Type, Data_Type >

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

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

#### 7.5.4.3 BArrayDenseCol_const< Cell_Type, Data_Type >

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

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

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

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

## 7.6 BArrayDenseCell_const< Cell_Type, Data_Type > Class Template Reference

### 7.6.1 Detailed Description

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

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

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

- include/barry/barraydense-bones.hpp

## 7.7 BArrayDenseCol< Cell_Type, Data_Type > Class Template Reference

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

### Public Member Functions

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

### Friends

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

### 7.7.1 Detailed Description

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

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

### 7.7.2 Constructor & Destructor Documentation

#### 7.7.2.1 BArrayDenseCol()

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

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

### 7.7.3 Member Function Documentation

#### 7.7.3.1 begin()

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

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

#### 7.7.3.2 end()

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

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

#### 7.7.3.3 operator()()

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

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

#### 7.7.3.4 size()

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

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

### 7.7.4 Friends And Related Function Documentation

#### 7.7.4.1 BArrayDense< Cell_Type, Data_Type >

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

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

#### 7.7.4.2 BArrayDenseCell< Cell_Type, Data_Type >

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

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

#### 7.7.4.3 BArrayDenseCell_const< Cell_Type, Data_Type >

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

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

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

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

# 7.8 BArrayDenseCol_const< Cell_Type, Data_Type > Class Template Reference

`#include <barraydensecol-bones.hpp>`

## Public Member Functions

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

## Friends

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

### 7.8.1 Detailed Description

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

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

### 7.8.2 Constructor & Destructor Documentation

#### 7.8.2.1 BArrayDenseCol_const()

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

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

### 7.8.3 Member Function Documentation

#### 7.8.3.1 begin()

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

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

#### 7.8.3.2 end()

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

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

#### 7.8.3.3 operator()()

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

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

#### 7.8.3.4 size()

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

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

### 7.8.4 Friends And Related Function Documentation

#### 7.8.4.1 BArrayDenseCell< Cell_Type, Data_Type >

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

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

#### 7.8.4.2 BArrayDenseCell_const< Cell_Type, Data_Type >

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

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

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

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

## 7.9 BArrayDenseRow< Cell_Type, Data_Type > Class Template Reference

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

### Public Member Functions

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

### Friends

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

### 7.9.1 Detailed Description

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

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

### 7.9.2 Constructor & Destructor Documentation

#### 7.9.2.1 BArrayDenseRow()

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

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

### 7.9.3 Member Function Documentation

#### 7.9.3.1 begin()

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

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

#### 7.9.3.2 end()

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

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

#### 7.9.3.3 operator()()

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

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

#### 7.9.3.4 size()

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

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

### 7.9.4 Friends And Related Function Documentation

#### 7.9.4.1 BArrayDense< Cell_Type, Data_Type >

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

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

#### 7.9.4.2 BArrayDenseCell< Cell_Type, Data_Type >

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

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

#### 7.9.4.3 BArrayDenseCell_const< Cell_Type, Data_Type >

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

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

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

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

## 7.10 BArrayDenseRow_const< Cell_Type, Data_Type > Class Template Reference

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

### Public Member Functions

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

## Friends

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

### 7.10.1 Detailed Description

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

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

### 7.10.2 Constructor & Destructor Documentation

#### 7.10.2.1 BArrayDenseRow_const()

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

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

### 7.10.3 Member Function Documentation

#### 7.10.3.1 begin()

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

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

#### 7.10.3.2 end()

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

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

#### 7.10.3.3 operator()()

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

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

#### 7.10.3.4 size()

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

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

### 7.10.4 Friends And Related Function Documentation

#### 7.10.4.1 BArrayDenseCell< Cell_Type, Data_Type >

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

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

#### 7.10.4.2 BArrayDenseCell_const< Cell_Type, Data_Type >

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

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

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

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

## 7.11 BArrayRow< Cell_Type, Data_Type > Class Template Reference

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

## Public Member Functions

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

### 7.11.1 Detailed Description

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

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

### 7.11.2 Constructor & Destructor Documentation

#### 7.11.2.1 BArrayRow()

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

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

#### 7.11.2.2 ~BArrayRow()

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

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

### 7.11.3 Member Function Documentation

#### 7.11.3.1 operator BArrayRow< Cell_Type, Data_Type >()

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

#### 7.11.3.2 operator*=()

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

#### 7.11.3.3 operator+=()

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

#### 7.11.3.4 operator-=()

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

#### 7.11.3.5 operator/=()

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

#### 7.11.3.6 operator=()

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

#### 7.11.3.7 operator==()

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

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

- include/barry/barrayrow-bones.hpp

## 7.12 BArrayRow_const< Cell_Type, Data_Type > Class Template Reference

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

### Public Member Functions

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

### 7.12.1 Detailed Description

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

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

### 7.12.2 Constructor & Destructor Documentation

#### 7.12.2.1 BArrayRow_const()

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

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

#### 7.12.2.2 ~BArrayRow_const()

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

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

### 7.12.3 Member Function Documentation

#### 7.12.3.1 operator BArrayRow_const< Cell_Type, Data_Type >()

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

#### 7.12.3.2 operator"!=()

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

#### 7.12.3.3 operator<()

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

#### 7.12.3.4 operator<=()

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

#### 7.12.3.5 operator==()

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

#### 7.12.3.6 operator>()

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

#### 7.12.3.7 operator>=()

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

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

- include/barry/barrayrow-bones.hpp

## 7.13 BArrayVector< Cell_Type, Data_Type > Class Template Reference

Row or column of a BArray

`#include <barrayvector-bones.hpp>`

### Public Member Functions

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

### 7.13.1 Detailed Description

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

Row or column of a BArray

**Template Parameters**

| | |
|---|---|
| *Cell_Type* | |
| *Data_Type* | |

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

### 7.13.2 Constructor & Destructor Documentation

#### 7.13.2.1 BArrayVector()

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

Construct a new BArrayVector object.

**Parameters**

| | |
|---|---|
| *Array_* | Pointer to a `BArray` object |
| *dim_* | Dimension. 0 means row and 1 means column. |
| *i_* | Element to point. |
| *check_bounds* | When `true`, check boundaries. |

Definition at line 32 of file barrayvector-bones.hpp.

#### 7.13.2.2 ~BArrayVector()

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

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

### 7.13.3 Member Function Documentation

#### 7.13.3.1 begin()

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

Definition at line 50 of file barrayvector-meat.hpp.

#### 7.13.3.2 end()

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

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

#### 7.13.3.3 is_col()

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

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

#### 7.13.3.4 is_row()

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

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

#### 7.13.3.5 operator std::vector< Cell_Type >()

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

Definition at line 175 of file barrayvector-meat.hpp.

#### 7.13.3.6 operator*=()

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

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

#### 7.13.3.7 operator+=()

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

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

#### 7.13.3.8 operator-=()

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

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

#### 7.13.3.9 operator/=()

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

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

#### 7.13.3.10 operator=()

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

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

#### 7.13.3.11 operator==()

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

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

#### 7.13.3.12 size()

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

Definition at line 39 of file barrayvector-meat.hpp.

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

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

## 7.14 BArrayVector_const< Cell_Type, Data_Type > Class Template Reference

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

### Public Member Functions

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

### 7.14.1 Detailed Description

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

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

## 7.14.2 Constructor & Destructor Documentation

### 7.14.2.1 BArrayVector_const()

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

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

### 7.14.2.2 ~BArrayVector_const()

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

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

## 7.14.3 Member Function Documentation

### 7.14.3.1 begin()

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

### 7.14.3.2 end()

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

### 7.14.3.3 is_col()

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

#### 7.14.3.4 is_row()

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

#### 7.14.3.5 operator std::vector< Cell_Type >()

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

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

#### 7.14.3.6 operator"!=()

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

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

#### 7.14.3.7 operator<()

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

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

#### 7.14.3.8 operator<=()

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

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

#### 7.14.3.9 operator==()

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

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

#### 7.14.3.10 operator>()

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

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

#### 7.14.3.11 operator>=()

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

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

#### 7.14.3.12 size()

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

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

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

## 7.15 Cell< Cell_Type > Class Template Reference

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

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

## Public Member Functions

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

## Public Attributes

- Cell_Type value
- bool visited
- bool active

### 7.15.1 Detailed Description

**template<class Cell_Type>**
**class Cell< Cell_Type >**

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

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

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

### 7.15.2 Constructor & Destructor Documentation

#### 7.15.2.1 Cell() [1/7]

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

#### 7.15.2.2 Cell() [2/7]

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

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

#### 7.15.2.3 ~Cell()

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

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

#### 7.15.2.4 Cell() [3/7]

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

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

#### 7.15.2.5 Cell() [4/7]

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

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

#### 7.15.2.6 Cell() [5/7]

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

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

#### 7.15.2.7 Cell() [6/7]

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

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

#### 7.15.2.8 Cell() [7/7]

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

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

### 7.15.3 Member Function Documentation

#### 7.15.3.1 add() [1/4]

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

#### 7.15.3.2 add() [2/4]

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

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

#### 7.15.3.3 add() [3/4]

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

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

#### 7.15.3.4 add() [4/4]

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

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

#### 7.15.3.5 operator Cell_Type()

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

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

#### 7.15.3.6 operator"!=()

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

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

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

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

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

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

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

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

#### 7.15.3.9 operator==()

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

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

### 7.15.4 Member Data Documentation

#### 7.15.4.1 active

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

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

#### 7.15.4.2 value

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

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

#### 7.15.4.3 visited

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

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

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

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

## 7.16 Cell_const< Cell_Type > Class Template Reference

### 7.16.1 Detailed Description

template<typename Cell_Type>
class Cell_const< Cell_Type >

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

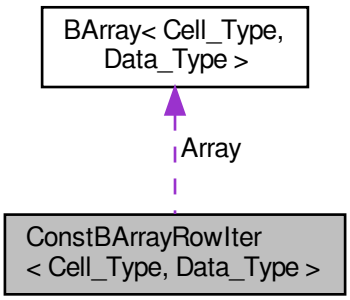
The documentation for this class was generated from the following file:

- include/barry/barray-meat.hpp

## 7.17 ConstBArrayRowIter< Cell_Type, Data_Type > Class Template Reference

`#include <barray-iterator.hpp>`

Collaboration diagram for ConstBArrayRowIter< Cell_Type, Data_Type >:




### Public Member Functions

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

### Public Attributes

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

### 7.17.1 Detailed Description

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

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

### 7.17.2 Constructor & Destructor Documentation

#### 7.17.2.1 ConstBArrayRowIter()

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

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

#### 7.17.2.2 ~ConstBArrayRowIter()

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

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

### 7.17.3 Member Data Documentation

#### 7.17.3.1 Array

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

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

#### 7.17.3.2 current_col

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

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

#### 7.17.3.3 current_row

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

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

#### 7.17.3.4 iter

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

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

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

- include/barry/barray-iterator.hpp

## 7.18 Counter< Array_Type, Data_Type > Class Template Reference

A counter function based on change statistics.

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

### Public Member Functions

- ~Counter ()
- double count (Array_Type &Array, uint i, uint j)
- double init (Array_Type &Array, uint i, uint j)
- std::string get_name () const
- std::string get_description () const

#### Creator passing a counter and an initializer

***Parameters***

| | |
|---|---|
| count_fun_ | *The main counter function.* |
| init_fun_ | *The initializer function can also be used to check if the `BArray` as the needed variables (see BArray::data).* |
| data_ | *Data to be used with the counter.* |
| delete_data_ | *When `true`, the destructor will delete the pointer in the main data.* |

- Counter ()

- Counter (Counter_fun_type< Array_Type, Data_Type > count_fun_, Counter_fun_type< Array_Type, Data_Type > init_fun_, Data_Type data_, std::string name_="", std::string desc_="")
- Counter (const Counter< Array_Type, Data_Type > &counter_)
  *Copy constructor.*
- Counter (Counter< Array_Type, Data_Type > &&counter_) noexcept
  *Move constructor.*
- Counter< Array_Type, Data_Type > operator= (const Counter< Array_Type, Data_Type > &counter_)
  *Copy assignment.*
- Counter< Array_Type, Data_Type > & operator= (Counter< Array_Type, Data_Type > &&counter_) noexcept
  *Move assignment.*

## Public Attributes

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

### 7.18.1 Detailed Description

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

A counter function based on change statistics.

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

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

### 7.18.2 Constructor & Destructor Documentation

#### 7.18.2.1 Counter() [1/4]

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

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

#### 7.18.2.2 Counter() [2/4]

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

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

#### 7.18.2.3 Counter() [3/4]

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

Copy constructor.

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

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

Move constructor.

#### 7.18.2.5 ~Counter()

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

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

### 7.18.3 Member Function Documentation

#### 7.18.3.1 count()

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

#### 7.18.3.2 get_description()

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

#### 7.18.3.3 get_name()

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

#### 7.18.3.4 init()

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

#### 7.18.3.5 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.

#### 7.18.3.6 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.

### 7.18.4 Member Data Documentation

#### 7.18.4.1 count_fun

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

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

#### 7.18.4.2 data

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

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

#### 7.18.4.3 desc

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

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

#### 7.18.4.4 init_fun

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

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

#### 7.18.4.5 name

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

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

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

- include/barry/counters-bones.hpp

# 7.19 Counters< Array_Type, Data_Type > Class Template Reference

Vector of counters.

`#include <counters-bones.hpp>`

## Public Member Functions

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

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

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

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

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

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

  *Number of counters in the set.*
- void add_counter (Counter< Array_Type, Data_Type > counter)
- void add_counter (Counter_fun_type< Array_Type, Data_Type > count_fun_, Counter_fun_type< Array_↩ Type, Data_Type > init_fun_, Data_Type data_, std::string name_="", std::string desc_="")
- std::vector< std::string > get_names () const
- std::vector< std::string > get_descriptions () const

### 7.19.1 Detailed Description

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

Vector of counters.

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

Definition at line 92 of file counters-bones.hpp.

### 7.19.2 Constructor & Destructor Documentation

#### 7.19.2.1 Counters() [1/3]

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

#### 7.19.2.2 ~Counters()

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

Definition at line 103 of file counters-bones.hpp.

#### 7.19.2.3 Counters() [2/3]

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

Copy constructor.

**Parameters**

| | |
|---|---|
| *counter_* | |

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

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

Move constructor.

**Parameters**

| | |
|---|---|
| *counters_* | |

### 7.19.3 Member Function Documentation

#### 7.19.3.1 add_counter() [1/2]

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

#### 7.19.3.2 add_counter() [2/2]

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

#### 7.19.3.3 get_descriptions()

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

#### 7.19.3.4 get_names()

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

#### 7.19.3.5 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>

#### 7.19.3.6 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>&

#### 7.19.3.7 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>*

#### 7.19.3.8 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 147 of file counters-bones.hpp.

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

- include/barry/counters-bones.hpp

## 7.20 DEFM Class Reference

`#include <defm-bones.hpp>`

### Public Member Functions

- DEFM (const int *id, const int *y, const double *x, size_t id_length, size_t y_ncol, size_t x_ncol, size_t m_order)
- ~DEFM ()
- defmcounters::DEFMModel & get_model ()
- void init ()
- double likelihood (std::vector< double > &par, bool as_log=false)
- void simulate (std::vector< double > par, int *y_out)
- size_t get_n_y () const
- size_t get_n_obs () const
- size_t get_n_covars () const
- size_t get_m_order () const
- size_t get_n_rows () const

### 7.20.1 Detailed Description

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

### 7.20.2 Constructor & Destructor Documentation

#### 7.20.2.1 DEFM()

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

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

#### 7.20.2.2 ~DEFM()

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

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

### 7.20.3 Member Function Documentation

#### 7.20.3.1 get_m_order()

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

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

#### 7.20.3.2 get_model()

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

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

#### 7.20.3.3 get_n_covars()

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

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

#### 7.20.3.4 get_n_obs()

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

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

#### 7.20.3.5 get_n_rows()

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

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

#### 7.20.3.6 get_n_y()

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

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

#### 7.20.3.7 init()

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

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

#### 7.20.3.8 likelihood()

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

#### 7.20.3.9 simulate()

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

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

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

- include/barry/models/defm/defm-bones.hpp
- include/barry/models/defm/defm-meat.hpp

## 7.21 DEFMCounterData Class Reference

Data class used to store arbitrary uint or double vectors.

`#include <defm.hpp>`

### Public Member Functions

- DEFMCounterData ()
- DEFMCounterData (const std::vector< size_t > indices_, const std::vector< double > numbers_, size_t markov_order_)
- size_t idx (size_t i)
- double num (size_t i)
- ~DEFMCounterData ()

### Public Attributes

- std::vector< size_t > indices
- std::vector< double > numbers
- size_t markov_order

  *Order of the markov process.*

## 7.21.1 Detailed Description

Data class used to store arbitrary uint or double vectors.

Definition at line 70 of file defm.hpp.

## 7.21.2 Constructor & Destructor Documentation

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

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

Definition at line 77 of file defm.hpp.

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

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

Definition at line 78 of file defm.hpp.

### 7.21.2.3 ~DEFMCounterData()

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

Definition at line 87 of file defm.hpp.

## 7.21.3 Member Function Documentation

#### 7.21.3.1 idx()

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

Definition at line 84 of file defm.hpp.

#### 7.21.3.2 num()

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

Definition at line 85 of file defm.hpp.

### 7.21.4 Member Data Documentation

#### 7.21.4.1 indices

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

Definition at line 73 of file defm.hpp.

#### 7.21.4.2 markov_order

```
size_t DEFMCounterData::markov_order
```

Order of the markov process.

Definition at line 75 of file defm.hpp.

#### 7.21.4.3 numbers

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

Definition at line 74 of file defm.hpp.

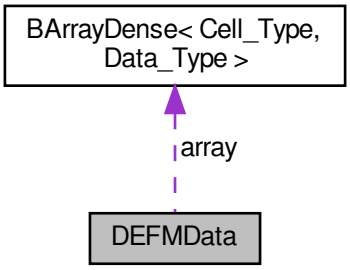
The documentation for this class was generated from the following file:

- include/barry/counters/defm.hpp

## 7.22 DEFMData Class Reference

`#include <defm.hpp>`

Collaboration diagram for DEFMData:




### Public Member Functions

- DEFMData ()
- DEFMData (DEFMArray *array_, const double *covariates_, size_t obs_start_, size_t X_ncol_, size_t X_nrow_)

  *Constructor.*
- double operator() (size_t i, size_t j) const

  *Access to the row (i) colum (j) data.*
- double at (size_t i, size_t j) const
- size_t ncol () const
- void print () const
- ~DEFMData ()

### Public Attributes

- DEFMArray * array
- const double * covariates

  *Vector of covariates (complete vector)*
- size_t obs_start

  *Index of the observation in the data.*
- size_t X_ncol

  *Number of covariates included in the model.*
- size_t X_nrow

  *Number of covariates included in the model.*

### 7.22.1 Detailed Description

Definition at line 25 of file defm.hpp.

## 7.22.2 Constructor & Destructor Documentation

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

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

Definition at line 34 of file defm.hpp.

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

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

Constructor.

**Parameters**

| | |
|---|---|
| *covariates_* | Pointer to the attribute data. |
| *obs_start_* | Location of the current observation in the covariates vector |
| *X_ncol_* | Number of columns (covariates.) |

Definition at line 43 of file defm.hpp.

### 7.22.2.3 ~DEFMData()

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

Definition at line 64 of file defm.hpp.

## 7.22.3 Member Function Documentation

#### 7.22.3.1 at()

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

#### 7.22.3.2 ncol()

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

Definition at line 128 of file defm.hpp.

#### 7.22.3.3 operator()()

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

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

**Parameters**

| | |
|---|---|
| *i* | |
| *j* | |

**Returns**

double

Definition at line 123 of file defm.hpp.

#### 7.22.3.4 print()

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

Definition at line 132 of file defm.hpp.

### 7.22.4 Member Data Documentation

#### 7.22.4.1 array

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

Definition at line 28 of file defm.hpp.

#### 7.22.4.2 covariates

```
const double* DEFMData::covariates
```

Vector of covariates (complete vector)

Definition at line 29 of file defm.hpp.

#### 7.22.4.3 obs_start

```
size_t DEFMData::obs_start
```

Index of the observation in the data.

Definition at line 30 of file defm.hpp.

#### 7.22.4.4 X_ncol

```
size_t DEFMData::X_ncol
```

Number of covariates included in the model.

Definition at line 31 of file defm.hpp.

#### 7.22.4.5 X_nrow

```
size_t DEFMData::X_nrow
```

Number of covariates included in the model.

Definition at line 32 of file defm.hpp.

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

- include/barry/counters/defm.hpp

## 7.23 DEFMRuleData Class Reference

`#include <defm.hpp>`

### Public Member Functions

- double num (size_t i)
- size_t idx (size_t i)
- DEFMRuleData ()
- DEFMRuleData (std::vector< double > numbers_, std::vector< size_t > indices_)

### 7.23.1 Detailed Description

Definition at line 91 of file defm.hpp.

### 7.23.2 Constructor & Destructor Documentation

#### 7.23.2.1 DEFMRuleData() [1/2]

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

Definition at line 101 of file defm.hpp.

#### 7.23.2.2 DEFMRuleData() [2/2]

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

Definition at line 103 of file defm.hpp.

### 7.23.3 Member Function Documentation

#### 7.23.3.1 idx()

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

Definition at line 99 of file defm.hpp.

#### 7.23.3.2 num()

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

Definition at line 98 of file defm.hpp.

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

- include/barry/counters/defm.hpp

## 7.24 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

### 7.24.1 Detailed Description

**template<typename Cell_Type>**
**class Entries< Cell_Type >**

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

**Template Parameters**

| | |
|---|---|
| *Cell_Type* | Any type |

Definition at line 79 of file typedefs.hpp.

### 7.24.2 Constructor & Destructor Documentation

#### 7.24.2.1 Entries() [1/2]

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

Definition at line 85 of file typedefs.hpp.

#### 7.24.2.2 Entries() [2/2]

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

Definition at line 86 of file typedefs.hpp.

#### 7.24.2.3 ~Entries()

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

Definition at line 93 of file typedefs.hpp.

### 7.24.3 Member Function Documentation

#### 7.24.3.1 resize()

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

Definition at line 95 of file typedefs.hpp.

### 7.24.4 Member Data Documentation

#### 7.24.4.1 source

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

Definition at line 81 of file typedefs.hpp.

#### 7.24.4.2 target

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

Definition at line 82 of file typedefs.hpp.

#### 7.24.4.3 val

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

Definition at line 83 of file typedefs.hpp.

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

- include/barry/typedefs.hpp

## 7.25 Flock Class Reference

A Flock is a group of Geese.

`#include <flock-bones.hpp>`

### Public Member Functions

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

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

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

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

  *Access the i-th geese element.*

#### Information about the model

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

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

### Public Attributes

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

## 7.25.1 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.

## 7.25.2 Constructor & Destructor Documentation

### 7.25.2.1 Flock()

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

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

### 7.25.2.2 ~Flock()

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

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

## 7.25.3 Member Function Documentation

### 7.25.3.1 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.

#### 7.25.3.2 colnames()

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

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

#### 7.25.3.3 get_counters()

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

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

#### 7.25.3.4 get_model()

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

Definition at line 131 of file flock-meat.hpp.

#### 7.25.3.5 get_stats_support()

```
std::vector< std::vector< double > > * Flock::get_stats_support ( )  [inline]
```

Definition at line 117 of file flock-meat.hpp.

#### 7.25.3.6 get_stats_target()

```
std::vector< std::vector< double > > * Flock::get_stats_target ( )  [inline]
```

Definition at line 124 of file flock-meat.hpp.

#### 7.25.3.7 get_support_fun()

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

Definition at line 110 of file flock-meat.hpp.

#### 7.25.3.8 init()

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

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

#### 7.25.3.9 likelihood_joint()

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

Returns the joint likelihood of the model.

**Parameters**

| | |
|---|---|
| *par* | Vector of model parameters. |
| *as_log* | When `true` it will return the value as log. |
| *use_reduced_sequence* | When `true` (default) will compute the likelihood using the reduced sequence, which is faster. |

**Returns**

double

Definition at line 138 of file flock-meat.hpp.

#### 7.25.3.10 nfuns()

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

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

#### 7.25.3.11 nleafs()

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

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

#### 7.25.3.12 nnodes()

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

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

#### 7.25.3.13 nterms()

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

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

#### 7.25.3.14 ntrees()

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

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

#### 7.25.3.15 operator()()

```
Geese * Flock::operator() (
            unsigned int i,
            bool check_bounds = true )  [inline]
```

Access the i-th geese element.

**Parameters**

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

**Returns**

Geese*

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

#### 7.25.3.16 parse_polytomies()

```
unsigned int Flock::parse_polytomies (
            bool verb = true,
            std::vector< size_t > * dist = nullptr ) const  [inline], [noexcept]
```

Check polytomies and return the largest.

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

#### 7.25.3.17 print()

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

Definition at line 258 of file flock-meat.hpp.

#### 7.25.3.18 set_seed()

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

Set the seed of the model.

**Parameters**

| | |
|---|---|
| *s* | Passed to the `rengine.seed()` member object. |

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

#### 7.25.3.19 support_size()

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

Definition at line 217 of file flock-meat.hpp.

### 7.25.4 Member Data Documentation

#### 7.25.4.1 dat

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

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

#### 7.25.4.2 initialized

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

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

#### 7.25.4.3 model

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

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

#### 7.25.4.4 nfunctions

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

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

#### 7.25.4.5 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

## 7.26 FreqTable< T > Class Template Reference

Frequency table of vectors.

`#include <freqtable.hpp>`

### Public Member Functions

- FreqTable ()
- ~FreqTable ()
- size_t add (const std::vector< T > &x, size_t *h_precomp)
- Counts_type as_vector () const
- const std::vector< double > & get_data () const
- const std::unordered_map< size_t, size_t > & get_index () const
- void clear ()
- void reserve (size_t n, size_t k)
- void print () const
- size_t size () const noexcept

  *Number of unique elements in the table. (.*
- size_t make_hash (const std::vector< double > &x) const

### 7.26.1 Detailed Description

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

Frequency table of vectors.

This is mostly used in `Support`. The main data is contained in the `data` double vector. The matrix is stored in a row-wise fashion, where the first element is the frequency with which the vector is observed.

For example, in a model with `k` terms the first k + 1 elements of `data` would be:

- weights
- term 1
- term 2
- ...
- term k

Definition at line 22 of file freqtable.hpp.

## 7.26.2 Constructor & Destructor Documentation

### 7.26.2.1 FreqTable()

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

Definition at line 34 of file freqtable.hpp.

### 7.26.2.2 ~FreqTable()

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

Definition at line 35 of file freqtable.hpp.

## 7.26.3 Member Function Documentation

### 7.26.3.1 add()

```
template<typename T >
size_t FreqTable< T >::add (
            const std::vector< T > & x,
            size_t * h_precomp )  [inline]
```

Definition at line 59 of file freqtable.hpp.

### 7.26.3.2 as_vector()

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

Definition at line 139 of file freqtable.hpp.

#### 7.26.3.3 clear()

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

Definition at line 168 of file freqtable.hpp.

#### 7.26.3.4 get_data()

```
template<typename T = double>
const std::vector< double >& FreqTable< T >::get_data ( ) const  [inline]
```

Definition at line 40 of file freqtable.hpp.

#### 7.26.3.5 get_index()

```
template<typename T = double>
const std::unordered_map<size_t,size_t>& FreqTable< T >::get_index ( ) const  [inline]
```

Definition at line 41 of file freqtable.hpp.

#### 7.26.3.6 make_hash()

```
template<typename T >
size_t FreqTable< T >::make_hash (
            const std::vector< double > & x ) const  [inline]
```

Definition at line 239 of file freqtable.hpp.

#### 7.26.3.7 print()

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

Definition at line 204 of file freqtable.hpp.

#### 7.26.3.8 reserve()

```
template<typename T >
void FreqTable< T >::reserve (
            size_t n,
            size_t k )  [inline]
```

Definition at line 182 of file freqtable.hpp.

#### 7.26.3.9 size()

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

Number of unique elements in the table. (.

**Returns**

size_t

Definition at line 231 of file freqtable.hpp.

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

- include/barry/freqtable.hpp

## 7.27 Geese Class Reference

Annotated Phylo Model.

`#include <geese-bones.hpp>`

### Public Member Functions

- ~Geese ()
- void init (unsigned int bar_width=BARRY_PROGRESS_BAR_WIDTH)
- void inherit_support (const Geese &model_, bool delete_support_=false)
- void calc_sequence (Node *n=nullptr)
- void calc_reduced_sequence ()
- double likelihood (const std::vector< double > &par, bool as_log=false, bool use_reduced_sequence=true)
- double likelihood_exhaust (const std::vector< double > &par)
- std::vector< double > get_probabilities () const
- void set_seed (const unsigned int &s)
- std::vector< std::vector< unsigned int > > simulate (const std::vector< double > &par)
- std::vector< std::vector< double > > observed_counts ()
- void print_observed_counts ()
- void print () const

  *Prints information about the GEESE.*
- void init_node (Node &n)
- void update_annotations (unsigned int nodeid, std::vector< unsigned int > newann)
- std::vector< std::vector< bool > > get_states () const

  *Powerset of a gene's possible states.*
- std::vector< unsigned int > get_annotated_nodes () const

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

#### Construct a new Geese object

*The model includes a total of `N + 1` nodes, the `+ 1` beign the root node.*

*Parameters*

| | |
|---|---|
| annotations | *A vector of vectors with annotations. It should be of length* `k` *(number of functions). Each vector should be of length* `N` *(equal to the number of nodes, including interior). Possible values are 0, 1, and 9.* |
| geneid | *Id of the gene. It should be of length* `N`*.* |
| parent | *Id of the parent gene. Also of length* `N` |
| duplication | *Logical scalar indicating the type of event (true: duplication, false: speciation.)* |

*The ordering of the entries does not matter. Passing the nodes in post order or not makes no difference to the constructor.*

- Geese ()
- Geese (std::vector< std::vector< unsigned int > > &annotations, std::vector< unsigned int > &geneid, std::vector< int > &parent, std::vector< bool > &duplication)
- Geese (const Geese &model_, bool copy_data=true)
- Geese (Geese &&x) noexcept
- Geese & operator= (const Geese &model_)=delete
- Geese & operator= (Geese &&model_) noexcept=delete

### Information about the model

*Parameters*

| | |
|---|---|
| verb | *When* `true` *it will print out information about the encountered polytomies.* |

- unsigned int nfuns () const noexcept
  *Number of functions analyzed.*
- unsigned int nnodes () const noexcept
  *Number of nodes (interior + leaf)*
- unsigned int nleafs () const noexcept
  *Number of leaf.*
- unsigned int nterms () const
  *Number of terms included.*
- unsigned int support_size () const noexcept
  *Number of unique sets of sufficient stats.*
- std::vector< unsigned int > nannotations () const noexcept
  *Number of annotations.*
- std::vector< std::string > colnames () const
  *Names of the terms in the model.*
- unsigned int parse_polytomies (bool verb=true, std::vector< size_t > *dist=nullptr) const noexcept
  *Check polytomies and return the largest.*

### Geese prediction

*Calculate the conditional probability*

*Parameters*

| | |
|---|---|
| par | *Vector of parameters (terms + root).* |
| res_prob | *Vector indicating each nodes' state probability.* |
| leave_one_out | *When* `true`*, it will compute the predictions using leave-one-out, thus the prediction will be repeated nleaf times.* |
| only_annotated | *When* `true`*, it will make the predictions only on the induced sub-tree with annotated leafs.* |
| use_reduced_sequence | *Passed to the* `likelihood` *method.* |
| preorder | *For the tree traversal.* |

*When `res_prob` is specified, the function will attach the member vector `probabilities` from the `Node`s objects. This contains the probability that the ith node has either of the possible states.*

***Returns***

*std::vector< double > Returns the posterior probability*

- std::vector< std::vector< double > > predict (const std::vector< double > &par, std::vector< std::vector< double > > *res_prob=nullptr, bool leave_one_out=false, bool only_annotated=false, bool use_reduced↩ _sequence=true)
- std::vector< std::vector< double > > predict_backend (const std::vector< double > &par, bool use_↩ reduced_sequence, const std::vector< uint > &preorder)
- std::vector< std::vector< double > > predict_exhaust_backend (const std::vector< double > &par, const std::vector< uint > &preorder)
- std::vector< std::vector< double > > predict_exhaust (const std::vector< double > &par)
- std::vector< std::vector< double > > predict_sim (const std::vector< double > &par, bool only_↩ annotated=false, unsigned int nsims=10000u)

**Non-const pointers to shared objects in <tt>Geese</tt>**

*These functions provide direct access to some member objects that are shared by the nodes within `Geese`.*

***Returns***

*`get_rengine()` returns the Pseudo-RNG engine used.*

*`get_counters()` returns the vector of counters used.*

*`get_model()` returns the `Model` object used.*

*`get_support_fun()` returns the computed support of the model.*

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

## Public Attributes

- unsigned int nfunctions
- std::map< unsigned int, Node > nodes
- barry::MapVec_type< unsigned int > map_to_nodes
- std::vector< std::vector< std::vector< size_t > > > pset_loc

  *Locations of columns.*
- std::vector< unsigned int > sequence
- std::vector< unsigned int > reduced_sequence
- bool initialized = false
- bool delete_rengine = false
- bool delete_support = false

### 7.27.1 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.

### 7.27.2 Constructor & Destructor Documentation

#### 7.27.2.1 Geese() [1/4]

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

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

#### 7.27.2.2 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.

#### 7.27.2.3 Geese() [3/4]

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

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

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

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

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

#### 7.27.2.5 ~Geese()

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

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

## 7.27.3 Member Function Documentation

### 7.27.3.1 calc_reduced_sequence()

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

Definition at line 331 of file geese-meat.hpp.

### 7.27.3.2 calc_sequence()

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

Definition at line 287 of file geese-meat.hpp.

### 7.27.3.3 colnames()

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

Names of the terms in the model.

Definition at line 453 of file geese-meat.hpp.

### 7.27.3.4 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 669 of file geese-meat.hpp.

### 7.27.3.5 get_counters()

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

Definition at line 652 of file geese-meat.hpp.

#### 7.27.3.6 get_model()

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

Definition at line 657 of file geese-meat.hpp.

#### 7.27.3.7 get_probabilities()

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

Definition at line 379 of file geese-meat.hpp.

#### 7.27.3.8 get_rengine()

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

Definition at line 647 of file geese-meat.hpp.

#### 7.27.3.9 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 665 of file geese-meat.hpp.

#### 7.27.3.10 get_support_fun()

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

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

#### 7.27.3.11 inherit_support()

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

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

#### 7.27.3.12 init()

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

Definition at line 103 of file geese-meat.hpp.

#### 7.27.3.13 init_node()

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

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

#### 7.27.3.14 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.

#### 7.27.3.15 likelihood_exhaust()

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

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

#### 7.27.3.16 nannotations()

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

Number of annotations.

Definition at line 444 of file geese-meat.hpp.

#### 7.27.3.17 nfuns()

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

Number of functions analyzed.

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

#### 7.27.3.18 nleafs()

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

Number of leaf.

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

#### 7.27.3.19 nnodes()

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

Number of nodes (interior + leaf)

Definition at line 407 of file geese-meat.hpp.

#### 7.27.3.20 nterms()

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

Number of terms included.

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

#### 7.27.3.21 observed_counts()

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

Definition at line 495 of file geese-meat.hpp.

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

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

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

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

#### 7.27.3.24 parse_polytomies()

```
unsigned int Geese::parse_polytomies (
            bool verb = true,
            std::vector< size_t > * dist = nullptr ) const  [inline], [noexcept]
```

Check polytomies and return the largest.

Definition at line 460 of file geese-meat.hpp.

#### 7.27.3.25 predict()

```
std::vector< std::vector< double > > Geese::predict (
            const std::vector< double > & par,
            std::vector< std::vector< double > > * res_prob = nullptr,
            bool leave_one_out = false,
            bool only_annotated = false,
            bool use_reduced_sequence = true )  [inline]
```

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

#### 7.27.3.26 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.

#### 7.27.3.27 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.

#### 7.27.3.28 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.

#### 7.27.3.29 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.

#### 7.27.3.30 print()

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

Prints information about the GEESE.

Definition at line 629 of file geese-meat.hpp.

#### 7.27.3.31 print_observed_counts()

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

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

#### 7.27.3.32 set_seed()

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

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

#### 7.27.3.33 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.

#### 7.27.3.34 support_size()

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

Number of unique sets of sufficient stats.

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

#### 7.27.3.35 update_annotations()

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

Definition at line 258 of file geese-meat.hpp.

### 7.27.4 Member Data Documentation

#### 7.27.4.1 delete_rengine

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

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

#### 7.27.4.2 delete_support

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

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

#### 7.27.4.3 initialized

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

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

#### 7.27.4.4 map_to_nodes

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

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

#### 7.27.4.5 nfunctions

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

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

#### 7.27.4.6 nodes

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

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

#### 7.27.4.7 pset_loc

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

Locations of columns.

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

#### 7.27.4.8 reduced_sequence

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

Definition at line 114 of file geese-bones.hpp.

#### 7.27.4.9 sequence

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

Definition at line 113 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

## 7.28 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_fun ()

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

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

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

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

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

- void add_rule (Rule< Array_Type, Data_Rule_Type > &rule)
- void add_rule (Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_, Data_Rule_Type data_)
- void set_rules (Rules< Array_Type, Data_Rule_Type > *rules_)
- void add_rule_dyn (Rule< Array_Type, Data_Rule_Dyn_Type > &rule)
- void add_rule_dyn (Rule_fun_type< Array_Type, Data_Rule_Dyn_Type > count_fun_, Data_Rule_Dyn_Type data_)
- void set_rules_dyn (Rules< Array_Type, Data_Rule_Dyn_Type > *rules_)

### Likelihood functions.

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

***Parameters***

| | |
|---|---|
| params | *Vector of parameters* |
| as_log | *When `true`, the function returns the log-likelihood.* |

- double likelihood (const std::vector< double > &params, const uint &i, bool as_log=false)
- double likelihood (const std::vector< double > &params, const Array_Type &Array_, int i=-1, bool as_log=false)
- double likelihood (const std::vector< double > &params, const std::vector< double > &target_, const uint &i, bool as_log=false)
- double likelihood (const std::vector< double > &params, const double *target_, const uint &i, bool as_log=false)
- double likelihood_total (const std::vector< double > &params, bool as_log=false)

### Extract elements by index

***Parameters***

| | |
|---|---|
| i | *Index relative to the array in the model.* |
| params | *A new vector of model parameters to compute the normalizing constant.* |
| as_log | *When `true` returns the logged version of the normalizing constant.* |

- double get_norm_const (const std::vector< double > &params, const uint &i, bool as_log=false)
- const std::vector< Array_Type > * get_pset (const uint &i)
- const std::vector< double > * get_pset_stats (const uint &i)

### Size of the model

*Number of different supports included in the model*

*This will return the size of `stats_target`.*

***Returns***

*`size()` returns the number of arrays in the model.*

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

*`nterms()` returns the number of terms in the model.*

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

- std::vector< std::vector< double > > * get_stats_target ()

  *Raw pointers to the support and target statistics.*
- std::vector< std::vector< double > > * get_stats_support ()
- std::vector< unsigned int > * get_arrays2support ()
- std::vector< std::vector< Array_Type > > * get_pset_arrays ()
- std::vector< std::vector< double > > * get_pset_stats ()

  *Statistics of the support(s)*
- std::vector< std::vector< double > > * get_pset_probs ()

- void set_transform_model (std::function< std::vector< double >(double *, unsigned int)> fun, std::vector< std::string > names)

  *Set the transform_model_fun object.*
- std::vector< double > transform_model (double *data, unsigned int k)

### 7.28.1 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:

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

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 44 of file model-bones.hpp.

### 7.28.2 Constructor & Destructor Documentation

#### 7.28.2.1 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 ( )
```

#### 7.28.2.2 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_ )
```

#### 7.28.2.3 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_ )
```

#### 7.28.2.4 ~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 173 of file model-bones.hpp.

### 7.28.3 Member Function Documentation

#### 7.28.3.1 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.

#### 7.28.3.2 add_counter() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_counter (
            Counter< Array_Type, Data_Counter_Type > & counter )
```

#### 7.28.3.3 add_counter() [2/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_counter (
            Counter_fun_type< Array_Type, Data_Counter_Type > count_fun_,
            Counter_fun_type< Array_Type, Data_Counter_Type > init_fun_ = nullptr,
            Data_Counter_Type data_ = nullptr )
```

#### 7.28.3.4 add_rule() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > & rule )
```

#### 7.28.3.5 add_rule() [2/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
            Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_,
            Data_Rule_Type data_ )
```

#### 7.28.3.6 add_rule_dyn() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_dyn
(
            Rule< Array_Type, Data_Rule_Dyn_Type > & rule )
```

#### 7.28.3.7 add_rule_dyn() [2/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_dyn
(
            Rule_fun_type< Array_Type, Data_Rule_Dyn_Type > count_fun_,
            Data_Rule_Dyn_Type data_ )
```

#### 7.28.3.8 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
```

#### 7.28.3.9 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

#### 7.28.3.10 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_ )
```

#### 7.28.3.11 get_arrays2support()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< unsigned int >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_↩
Rule_Dyn_Type >::get_arrays2support ( )
```

#### 7.28.3.12 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 ( )
```

#### 7.28.3.13 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 )
```

#### 7.28.3.14 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 )
```

#### 7.28.3.15 get_pset_arrays()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector< Array_Type > >* Model< Array_Type, Data_Counter_Type, Data_Rule_↩
Type, Data_Rule_Dyn_Type >::get_pset_arrays ( )
```

#### 7.28.3.16 get_pset_probs()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector<double> >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_pset_probs ( )
```

#### 7.28.3.17 get_pset_stats() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector<double> >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_pset_stats ( )
```

Statistics of the support(s)

#### 7.28.3.18 get_pset_stats() [2/2]

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

#### 7.28.3.19 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
```

#### 7.28.3.20 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 ( )
```

#### 7.28.3.21 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 ( )
```

#### 7.28.3.22 get_stats_support()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector< double > >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_stats_support ( )
```

#### 7.28.3.23 get_stats_target()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector< double > >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_stats_target ( )
```

Raw pointers to the support and target statistics.

The support of the model is stored as a vector of vector<double>. Each element of it contains the support for an specific type of array included. It represents an array of size `(k + 1) x n unique elements`, with the data stored by-row. The last element of each entry corresponds to the weights, i.e., the frequency with which such sufficient statistics are observed in the support.

#### 7.28.3.24 get_support_fun()

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

#### 7.28.3.25 likelihood() [1/4]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::likelihood
(
            const std::vector< double > & params,
            const Array_Type & Array_,
            int i = -1,
            bool as_log = false )
```

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

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::likelihood
(
            const std::vector< double > & params,
            const double * target_,
            const uint & i,
            bool as_log = false )
```

#### 7.28.3.27 likelihood() [3/4]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::likelihood
(
            const std::vector< double > & params,
            const std::vector< double > & target_,
            const uint & i,
            bool as_log = false )
```

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

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::likelihood
(
            const std::vector< double > & params,
            const uint & i,
            bool as_log = false )
```

#### 7.28.3.29 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 )
```

#### 7.28.3.30 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]
```

#### 7.28.3.31 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_ )
```

#### 7.28.3.32 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.

#### 7.28.3.33 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
```

#### 7.28.3.34 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 = {} )
```

#### 7.28.3.35 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 )
```

#### 7.28.3.36 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_ )
```

#### 7.28.3.37 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_ )
```

#### 7.28.3.38 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 143 of file model-bones.hpp.

#### 7.28.3.39 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_ )
```

#### 7.28.3.40 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_ )
```

#### 7.28.3.41 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 153 of file model-bones.hpp.

#### 7.28.3.42 set_transform_model()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_transform↩
_model (
            std::function< std::vector< double >(double *, unsigned int)> fun,
            std::vector< std::string > names )
```

Set the transform_model_fun object.

The transform_model function is used to transform the data

**Parameters**

| | |
|---|---|
| *data* | |
| *target* | |
| *n_arrays* | |
| *arrays2support* | |

#### 7.28.3.43 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]
```

#### 7.28.3.44 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]
```

#### 7.28.3.45 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]
```

#### 7.28.3.46 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]
```

#### 7.28.3.47 transform_model()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↩
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector<double> Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type
>::transform_model (
            double * data,
            unsigned int k )
```

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

- include/barry/model-bones.hpp

## 7.29 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

### 7.29.1 Detailed Description

Data class used to store arbitrary uint or double vectors.

Definition at line 56 of file network.hpp.

### 7.29.2 Constructor & Destructor Documentation

#### 7.29.2.1 NetCounterData() [1/2]

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

Definition at line 62 of file network.hpp.

#### 7.29.2.2 NetCounterData() [2/2]

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

Definition at line 63 of file network.hpp.

#### 7.29.2.3 ~NetCounterData()

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

Definition at line 68 of file network.hpp.

### 7.29.3 Member Data Documentation

#### 7.29.3.1 indices

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

Definition at line 59 of file network.hpp.

#### 7.29.3.2 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

## 7.30 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

### 7.30.1 Detailed Description

Data class for Networks.

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

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

Definition at line 19 of file network.hpp.

### 7.30.2 Constructor & Destructor Documentation

#### 7.30.2.1 NetworkData() [1/3]

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

Definition at line 25 of file network.hpp.

#### 7.30.2.2 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.

#### 7.30.2.3 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.

#### 7.30.2.4 ~NetworkData()

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

Definition at line 51 of file network.hpp.

### 7.30.3 Member Data Documentation

#### 7.30.3.1 directed

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

Definition at line 22 of file network.hpp.

#### 7.30.3.2 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:

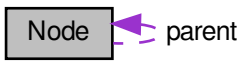
- include/barry/counters/network.hpp

## 7.31 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.*

### 7.31.1 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.

### 7.31.2 Constructor & Destructor Documentation

#### 7.31.2.1 Node() [1/5]

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

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

#### 7.31.2.2 Node() [2/5]

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

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

#### 7.31.2.3 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.

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

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

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

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

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

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

#### 7.31.2.6 ∼Node()

```
Node::∼Node ( )  [inline]
```

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

### 7.31.3 Member Function Documentation

#### 7.31.3.1 get_parent()

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

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

#### 7.31.3.2 is_leaf()

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

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

#### 7.31.3.3 noffspring()

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

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

### 7.31.4 Member Data Documentation

#### 7.31.4.1 annotations

```
std::vector< unsigned int > Node::annotations
```

Observed annotations (only defined for Geese)

Definition at line 18 of file geese-node-bones.hpp.

#### 7.31.4.2 array

```
phylocounters::PhyloArray Node::array
```

Definition at line 17 of file geese-node-bones.hpp.

#### 7.31.4.3 arrays

```
std::vector< phylocounters::PhyloArray > Node::arrays = {}
```

Arrays given all possible states.

Definition at line 21 of file geese-node-bones.hpp.

#### 7.31.4.4 duplication

```
bool Node::duplication
```

Definition at line 19 of file geese-node-bones.hpp.

#### 7.31.4.5 id

```
unsigned int Node::id
```

Id of the node (as specified in the input)

Definition at line 14 of file geese-node-bones.hpp.

#### 7.31.4.6 narray

```
std::vector< unsigned int > Node::narray = {}
```

ID of the array in the model.

Definition at line 24 of file geese-node-bones.hpp.

#### 7.31.4.7 offspring

```
std::vector< Node* > Node::offspring = {}
```

Offspring nodes.

Definition at line 23 of file geese-node-bones.hpp.

#### 7.31.4.8 ord

```
unsigned int Node::ord
```

Order in which the node was created.

Definition at line 15 of file geese-node-bones.hpp.

#### 7.31.4.9 parent

```
Node* Node::parent = nullptr
```

Parent node.

Definition at line 22 of file geese-node-bones.hpp.

#### 7.31.4.10 probability

```
std::vector< double > Node::probability
```

The probability of observing each state.

Definition at line 28 of file geese-node-bones.hpp.

#### 7.31.4.11 subtree_prob

```
std::vector< double > Node::subtree_prob
```

Induced subtree probabilities.

Definition at line 27 of file geese-node-bones.hpp.

#### 7.31.4.12 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

## 7.32 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

### 7.32.1 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.

### 7.32.2 Constructor & Destructor Documentation

#### 7.32.2.1 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.

### 7.32.3 Member Data Documentation

#### 7.32.3.1 blengths

```
std::vector< double > NodeData::blengths = {}
```

Branch length.

Definition at line 44 of file phylo.hpp.

#### 7.32.3.2 duplication

```
bool NodeData::duplication = true
```

Definition at line 54 of file phylo.hpp.

#### 7.32.3.3 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

## 7.33 PhyloCounterData Class Reference

```
#include <phylo.hpp>
```

### Public Member Functions

- PhyloCounterData (std::vector< uint > data_, std::vector< double > *counters_=nullptr)
- PhyloCounterData ()
- uint at (uint d)
- uint operator() (uint d)
- uint operator[ ] (uint d)
- void reserve (uint x)
- void push_back (uint x)
- void shrink_to_fit ()
- uint size ()
- std::vector< uint >::iterator begin ()
- std::vector< uint >::iterator end ()
- bool empty ()
- std::vector< double > * get_counters ()

### 7.33.1 Detailed Description

Definition at line 69 of file phylo.hpp.

### 7.33.2 Constructor & Destructor Documentation

#### 7.33.2.1 PhyloCounterData() [1/2]

```
PhyloCounterData::PhyloCounterData (
            std::vector< uint > data_,
            std::vector< double > * counters_ = nullptr )  [inline]
```

Definition at line 75 of file phylo.hpp.

#### 7.33.2.2 PhyloCounterData() [2/2]

```
PhyloCounterData::PhyloCounterData ( )  [inline]
```

Definition at line 80 of file phylo.hpp.

### 7.33.3 Member Function Documentation

#### 7.33.3.1 at()

```
uint PhyloCounterData::at (
            uint d )  [inline]
```

Definition at line 82 of file phylo.hpp.

#### 7.33.3.2 begin()

```
std::vector< uint >::iterator PhyloCounterData::begin ( )  [inline]
```

Definition at line 90 of file phylo.hpp.

#### 7.33.3.3 empty()

```
bool PhyloCounterData::empty ( )  [inline]
```

Definition at line 93 of file phylo.hpp.

#### 7.33.3.4 end()

```
std::vector< uint >::iterator PhyloCounterData::end ( )  [inline]
```

Definition at line 91 of file phylo.hpp.

#### 7.33.3.5 get_counters()

```
std::vector< double >* PhyloCounterData::get_counters ( )  [inline]
```

Definition at line 94 of file phylo.hpp.

#### 7.33.3.6 operator()()

```
uint PhyloCounterData::operator() (
            uint d )  [inline]
```

Definition at line 83 of file phylo.hpp.

#### 7.33.3.7 operator[]()

```
uint PhyloCounterData::operator[] (
            uint d )  [inline]
```

Definition at line 84 of file phylo.hpp.

#### 7.33.3.8 push_back()

```
void PhyloCounterData::push_back (
            uint x )  [inline]
```

Definition at line 86 of file phylo.hpp.

#### 7.33.3.9 reserve()

```
void PhyloCounterData::reserve (
            uint x )  [inline]
```

Definition at line 85 of file phylo.hpp.

#### 7.33.3.10 shrink_to_fit()

```
void PhyloCounterData::shrink_to_fit ( )  [inline]
```

Definition at line 87 of file phylo.hpp.

#### 7.33.3.11 size()

```
uint PhyloCounterData::size ( )  [inline]
```

Definition at line 88 of file phylo.hpp.

The documentation for this class was generated from the following file:

- include/barry/counters/phylo.hpp

## 7.34 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

### 7.34.1 Detailed Description

Definition at line 2147 of file phylo.hpp.

## 7.34.2 Constructor & Destructor Documentation

### 7.34.2.1 PhyloRuleDynData()

```
PhyloRuleDynData::PhyloRuleDynData (
            const std::vector< double > * counts_,
            uint pos_,
            uint lb_,
            uint ub_,
            uint duplication_ )  [inline]
```

Definition at line 2155 of file phylo.hpp.

### 7.34.2.2 ~PhyloRuleDynData()

```
PhyloRuleDynData::~PhyloRuleDynData ( )  [inline]
```

Definition at line 2164 of file phylo.hpp.

## 7.34.3 Member Data Documentation

### 7.34.3.1 counts

```
const std::vector< double >* PhyloRuleDynData::counts
```

Definition at line 2149 of file phylo.hpp.

### 7.34.3.2 duplication

```
uint PhyloRuleDynData::duplication
```

Definition at line 2153 of file phylo.hpp.

### 7.34.3.3 lb

```
uint PhyloRuleDynData::lb
```

Definition at line 2151 of file phylo.hpp.

#### 7.34.3.4 pos

```
uint PhyloRuleDynData::pos
```

Definition at line 2150 of file phylo.hpp.

#### 7.34.3.5 ub

```
uint PhyloRuleDynData::ub
```

Definition at line 2152 of file phylo.hpp.

The documentation for this class was generated from the following file:

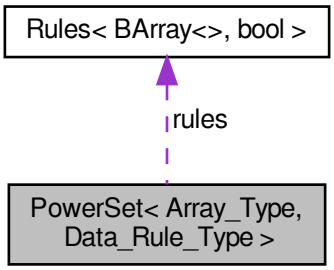
- include/barry/counters/phylo.hpp

## 7.35 PowerSet< Array_Type, Data_Rule_Type > Class Template Reference

Powerset of a binary array.

```
#include <powerset-bones.hpp>
```

Collaboration diagram for PowerSet< Array_Type, Data_Rule_Type >:



## Public Member Functions

- void init_support ()
- void calc ()
- void reset (uint N_, uint M_)

### Construct and destroy a PowerSet object

- PowerSet ()
- PowerSet (uint N_, uint M_)
- PowerSet (const Array_Type &array)
- ~PowerSet ()

### Wrappers for the <tt>Rules</tt> member.

*These will add rules to the model, which are shared by the support and the actual counter function.*

- void add_rule (Rule< Array_Type, Data_Rule_Type > rule)
- void add_rule (Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_, Data_Rule_Type data_)

### Getter functions

- const std::vector< Array_Type > * get_data_ptr () const
- std::vector< Array_Type > get_data () const
- std::vector< Array_Type >::iterator begin ()
- std::vector< Array_Type >::iterator end ()
- std::size_t size () const noexcept
- const Array_Type & operator[ ] (const unsigned int &i) const

## Public Attributes

- Array_Type EmptyArray
- std::vector< Array_Type > data
- Rules< Array_Type, Data_Rule_Type > * rules
- uint N
- uint M
- bool rules_deleted = false
- std::vector< size_t > coordinates_free
- std::vector< size_t > coordinates_locked
- size_t n_free
- size_t n_locked

## 7.35.1 Detailed Description

**template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>**
**class PowerSet< Array_Type, Data_Rule_Type >**

Powerset of a binary array.

**Template Parameters**

| | |
|---|---|
| *Array_Type* | |
| *Data_Rule_Type* | |

Definition at line 11 of file powerset-bones.hpp.

## 7.35.2 Constructor & Destructor Documentation

### 7.35.2.1 PowerSet() [1/3]

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
PowerSet< Array_Type, Data_Rule_Type >::PowerSet ( )  [inline]
```

Definition at line 36 of file powerset-bones.hpp.

### 7.35.2.2 PowerSet() [2/3]

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
PowerSet< Array_Type, Data_Rule_Type >::PowerSet (
            uint N_,
            uint M_ )  [inline]
```

Definition at line 38 of file powerset-bones.hpp.

### 7.35.2.3 PowerSet() [3/3]

```
template<typename Array_Type , typename Data_Rule_Type >
PowerSet< Array_Type, Data_Rule_Type >::PowerSet (
            const Array_Type & array )  [inline]
```

Definition at line 5 of file powerset-meat.hpp.

### 7.35.2.4 ~PowerSet()

```
template<typename Array_Type , typename Data_Rule_Type >
PowerSet< Array_Type, Data_Rule_Type >::~PowerSet  [inline]
```

Definition at line 13 of file powerset-meat.hpp.

## 7.35.3 Member Function Documentation

#### 7.35.3.1 add_rule() [1/2]

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > rule )  [inline]
```

Definition at line 173 of file powerset-meat.hpp.

#### 7.35.3.2 add_rule() [2/2]

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::add_rule (
            Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_,
            Data_Rule_Type data_ )  [inline]
```

Definition at line 182 of file powerset-meat.hpp.

#### 7.35.3.3 begin()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type >::iterator PowerSet< Array_Type, Data_Rule_Type >::begin ( )  [inline]
```

Definition at line 68 of file powerset-bones.hpp.

#### 7.35.3.4 calc()

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::calc  [inline]
```

Definition at line 144 of file powerset-meat.hpp.

#### 7.35.3.5 end()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type >::iterator PowerSet< Array_Type, Data_Rule_Type >::end ( )  [inline]
```

Definition at line 69 of file powerset-bones.hpp.

#### 7.35.3.6 get_data()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type > PowerSet< Array_Type, Data_Rule_Type >::get_data ( ) const  [inline]
```

Definition at line 67 of file powerset-bones.hpp.

#### 7.35.3.7 get_data_ptr()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
const std::vector< Array_Type >* PowerSet< Array_Type, Data_Rule_Type >::get_data_ptr ( )
const  [inline]
```

Definition at line 66 of file powerset-bones.hpp.

#### 7.35.3.8 init_support()

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::init_support  [inline]
```

Definition at line 19 of file powerset-meat.hpp.

#### 7.35.3.9 operator[]()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
const Array_Type& PowerSet< Array_Type, Data_Rule_Type >::operator[] (
            const unsigned int & i ) const  [inline]
```

Definition at line 71 of file powerset-bones.hpp.

#### 7.35.3.10 reset()

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::reset (
            uint N_,
            uint M_ )  [inline]
```

Definition at line 160 of file powerset-meat.hpp.

#### 7.35.3.11 size()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::size_t PowerSet< Array_Type, Data_Rule_Type >::size ( ) const  [inline], [noexcept]
```

Definition at line 70 of file powerset-bones.hpp.

### 7.35.4 Member Data Documentation

#### 7.35.4.1 coordinates_free

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< size_t > PowerSet< Array_Type, Data_Rule_Type >::coordinates_free
```

Definition at line 26 of file powerset-bones.hpp.

#### 7.35.4.2 coordinates_locked

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< size_t > PowerSet< Array_Type, Data_Rule_Type >::coordinates_locked
```

Definition at line 27 of file powerset-bones.hpp.

#### 7.35.4.3 data

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type > PowerSet< Array_Type, Data_Rule_Type >::data
```

Definition at line 19 of file powerset-bones.hpp.

#### 7.35.4.4 EmptyArray

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
Array_Type PowerSet< Array_Type, Data_Rule_Type >::EmptyArray
```

Definition at line 18 of file powerset-bones.hpp.

#### 7.35.4.5 M

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
uint PowerSet< Array_Type, Data_Rule_Type >::M
```

Definition at line 22 of file powerset-bones.hpp.

#### 7.35.4.6 N

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
uint PowerSet< Array_Type, Data_Rule_Type >::N
```

Definition at line 22 of file powerset-bones.hpp.

#### 7.35.4.7 n_free

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
size_t PowerSet< Array_Type, Data_Rule_Type >::n_free
```

Definition at line 28 of file powerset-bones.hpp.

#### 7.35.4.8 n_locked

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
size_t PowerSet< Array_Type, Data_Rule_Type >::n_locked
```

Definition at line 29 of file powerset-bones.hpp.

#### 7.35.4.9 rules

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
Rules<Array_Type,Data_Rule_Type>* PowerSet< Array_Type, Data_Rule_Type >::rules
```

Definition at line 20 of file powerset-bones.hpp.

#### 7.35.4.10 rules_deleted

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
bool PowerSet< Array_Type, Data_Rule_Type >::rules_deleted = false
```

Definition at line 23 of file powerset-bones.hpp.

The documentation for this class was generated from the following files:

- include/barry/powerset-bones.hpp
- include/barry/powerset-meat.hpp

## 7.36 Progress Class Reference

A simple progress bar.

```
#include <progress.hpp>
```

### Public Member Functions

- Progress (int n_, int width_)
- ~Progress ()
- void next ()
- void end ()

### 7.36.1 Detailed Description

A simple progress bar.

Definition at line 11 of file progress.hpp.

### 7.36.2 Constructor & Destructor Documentation

#### 7.36.2.1 Progress()

```
Progress::Progress (
            int n_,
            int width_ )  [inline]
```

Definition at line 30 of file progress.hpp.

#### 7.36.2.2 ~Progress()

```
Progress::~Progress ( )  [inline]
```

Definition at line 23 of file progress.hpp.

### 7.36.3 Member Function Documentation

#### 7.36.3.1 end()

```
void Progress::end ( )  [inline]
```

Definition at line 52 of file progress.hpp.

#### 7.36.3.2 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

## 7.37 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_)

## 7.37.1 Detailed Description

**template<typename Array_Type = BArray<>, typename Data_Type = bool>**
**class Rule< Array_Type, Data_Type >**

Rule for determining if a cell should be included in a sequence.

Rules can be used together with `Support` and `PowerSet` to determine which cells should be included when enumerating all possible realizations of a binary array.

**Template Parameters**

| | |
|---|---|
| *Array_Type* | An object of class `BArray`. |
| *Data_Type* | Any type. |

Definition at line 20 of file rules-bones.hpp.

## 7.37.2 Constructor & Destructor Documentation

### 7.37.2.1 Rule() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Rule< Array_Type, Data_Type >::Rule ( )  [inline]
```

Definition at line 38 of file rules-bones.hpp.

### 7.37.2.2 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_ )  [inline]
```

Definition at line 39 of file rules-bones.hpp.

#### 7.37.2.3 ~Rule()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Rule< Array_Type, Data_Type >::~Rule ( )  [inline]
```

Definition at line 45 of file rules-bones.hpp.

### 7.37.3 Member Function Documentation

#### 7.37.3.1 D()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Data_Type& Rule< Array_Type, Data_Type >::D ( )
```

Read/Write access to the data.

#### 7.37.3.2 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 37 of file rules-meat.hpp.

The documentation for this class was generated from the following files:

- include/barry/rules-bones.hpp
- include/barry/rules-meat.hpp

## 7.38 Rules< Array_Type, Data_Type > Class Template Reference

Vector of objects of class Rule.

```
#include <rules-bones.hpp>
```

### Public Member Functions

- Rules ()
- Rules (const Rules< Array_Type, Data_Type > &rules_)
- Rules< Array_Type, Data_Type > operator= (const Rules< Array_Type, Data_Type > &rules_)
- ~Rules ()
- uint size () const noexcept
- bool operator() (const Array_Type &a, uint i, uint j)

  *Check whether a given cell is free or locked.*
- void get_seq (const Array_Type &a, std::vector< size_t > *free, std::vector< size_t > *locked=nullptr)

  *Computes the sequence of free and locked cells in an BArray.*

**Rule adding**

*Parameters*

| | |
|---|---|
| rule | |

- void add_rule (Rule< Array_Type, Data_Type > rule)
- void add_rule (Rule_fun_type< Array_Type, Data_Type > rule_, Data_Type data_)

### 7.38.1 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 60 of file rules-bones.hpp.

### 7.38.2 Constructor & Destructor Documentation

#### 7.38.2.1 Rules() [1/2]

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::Rules ( )  [inline]
```

Definition at line 66 of file rules-bones.hpp.

#### 7.38.2.2 Rules() [2/2]

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::Rules (
            const Rules< Array_Type, Data_Type > & rules_ )  [inline]
```

Definition at line 5 of file rules-meat.hpp.

#### 7.38.2.3 ~Rules()

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::~Rules ( )  [inline]
```

Definition at line 71 of file rules-bones.hpp.

### 7.38.3 Member Function Documentation

#### 7.38.3.1 add_rule() [1/2]

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::add_rule (
            Rule< Array_Type, Data_Type > rule )  [inline]
```

Definition at line 42 of file rules-meat.hpp.

#### 7.38.3.2 add_rule() [2/2]

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::add_rule (
            Rule_fun_type< Array_Type, Data_Type > rule_,
            Data_Type data_ )  [inline]
```

Definition at line 52 of file rules-meat.hpp.

#### 7.38.3.3 get_seq()

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::get_seq (
            const Array_Type & a,
            std::vector< size_t > * free,
            std::vector< size_t > * locked = nullptr )  [inline]
```

Computes the sequence of free and locked cells in an BArray.

**Parameters**

| | |
|---|---|
| *a* | An object of class BArray. |
| *free* | Pointer to a vector of pairs (i, j) listing the free cells. |
| *locked* | (optional) Pointer to a vector of pairs (i, j) listing the locked cells. |

**Returns**

Nothing.

Definition at line 83 of file rules-meat.hpp.

#### 7.38.3.4 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 67 of file rules-meat.hpp.

#### 7.38.3.5 operator=()

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type > Rules< Array_Type, Data_Type >::operator= (
            const Rules< Array_Type, Data_Type > & rules_ )
```

Definition at line 19 of file rules-meat.hpp.

#### 7.38.3.6 size()

```
template<typename Array_Type , typename Data_Type >
uint Rules< Array_Type, Data_Type >::size ( ) const  [inline], [noexcept]
```

Definition at line 73 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

# 7.39 StatsCounter< Array_Type, Data_Type > Class Template Reference

Count stats for a single Array.

`#include <statscounter-bones.hpp>`

## Public Member Functions

- StatsCounter (const Array_Type *Array_)

  *Creator of a `StatsCounter`*
- StatsCounter (const StatsCounter< Array_Type, Data_Type > &counter)

  *Copy constructor.*
- StatsCounter ()

  *Can be created without setting the array.*
- ~StatsCounter ()
- void reset_array (const Array_Type *Array_)

  *Changes the reference array for the counting.*
- void add_counter (Counter< Array_Type, Data_Type > f_)
- void set_counters (Counters< Array_Type, Data_Type > *counters_)
- void count_init (uint i, uint j)

  *Counter functions This function recurses through the entries of `Array` and at each step of adding a new cell it uses the functions to list the statistics.*
- void count_current (uint i, uint j)
- std::vector< double > count_all ()
- Counters< Array_Type, Data_Type > * get_counters ()
- std::vector< std::string > get_names () const
- std::vector< std::string > get_descriptions () const
- size_t size () const

## 7.39.1 Detailed Description

**template<typename Array_Type, typename Data_Type>**
**class StatsCounter< Array_Type, Data_Type >**

Count stats for a single Array.

Users can a list of functions that can be used with this. The baseline set of arguments is a pointer to a binary array and a dataset to add the counts to.

Definition at line 14 of file statscounter-bones.hpp.

## 7.39.2 Constructor & Destructor Documentation

### 7.39.2.1 StatsCounter() [1/3]

```
template<typename Array_Type , typename Data_Type >
StatsCounter< Array_Type, Data_Type >::StatsCounter (
            const Array_Type * Array_ )  [inline]
```

Creator of a `StatsCounter`

**Parameters**

| | |
|---|---|
| *Array↩_* | A const pointer to a BArray. |

Definition at line 37 of file statscounter-bones.hpp.

#### 7.39.2.2 StatsCounter() [2/3]

```
template<typename Array_Type , typename Data_Type >
StatsCounter< Array_Type, Data_Type >::StatsCounter (
            const StatsCounter< Array_Type, Data_Type > & counter )
```

Copy constructor.

**Parameters**

| | |
|---|---|
| *counter* | |

#### 7.39.2.3 StatsCounter() [3/3]

```
template<typename Array_Type , typename Data_Type >
StatsCounter< Array_Type, Data_Type >::StatsCounter ( )  [inline]
```

Can be created without setting the array.

Definition at line 59 of file statscounter-bones.hpp.

#### 7.39.2.4 ~StatsCounter()

```
template<typename Array_Type , typename Data_Type >
StatsCounter< Array_Type, Data_Type >::~StatsCounter ( )
```

### 7.39.3 Member Function Documentation

#### 7.39.3.1 add_counter()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::add_counter (
            Counter< Array_Type, Data_Type > f_ )
```

#### 7.39.3.2 count_all()

```
template<typename Array_Type , typename Data_Type >
std::vector< double > StatsCounter< Array_Type, Data_Type >::count_all  [inline]
```

Definition at line 99 of file statscounter-meat.hpp.

#### 7.39.3.3 count_current()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::count_current (
            uint i,
            uint j )
```

#### 7.39.3.4 count_init()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::count_init (
            uint i,
            uint j )
```

Counter functions This function recurses through the entries of `Array` and at each step of adding a new cell it uses the functions to list the statistics.

#### 7.39.3.5 get_counters()

```
template<typename Array_Type , typename Data_Type >
Counters<Array_Type,Data_Type>* StatsCounter< Array_Type, Data_Type >::get_counters ( )
```

#### 7.39.3.6 get_descriptions()

```
template<typename Array_Type , typename Data_Type >
std::vector< std::string > StatsCounter< Array_Type, Data_Type >::get_descriptions ( ) const
```

#### 7.39.3.7 get_names()

```
template<typename Array_Type , typename Data_Type >
std::vector< std::string > StatsCounter< Array_Type, Data_Type >::get_names ( ) const
```

#### 7.39.3.8 reset_array()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::reset_array (
            const Array_Type * Array_ )
```

Changes the reference array for the counting.

**Parameters**

| | |
|---|---|
| *Array↔_* | A pointer to an array of class `Array_Type`. |

#### 7.39.3.9 set_counters()

```
template<typename Array_Type , typename Data_Type >
void StatsCounter< Array_Type, Data_Type >::set_counters (
            Counters< Array_Type, Data_Type > * counters_ )
```

#### 7.39.3.10 size()

```
template<typename Array_Type , typename Data_Type >
size_t StatsCounter< Array_Type, Data_Type >::size ( ) const  [inline]
```

Definition at line 86 of file statscounter-bones.hpp.

The documentation for this class was generated from the following files:

- include/barry/statscounter-bones.hpp
- include/barry/statscounter-meat.hpp

## 7.40 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference

Compute the support of sufficient statistics.

```
#include <support-bones.hpp>
```

### Public Member Functions

- Support (const Array_Type &Array_)

  *Constructor passing a reference Array.*
- Support (uint N_, uint M_)

  *Constructor specifying the dimensions of the array (empty).*
- Support ()
- ~Support ()
- void init_support (std::vector< Array_Type > *array_bank=nullptr, std::vector< double > *stats_bank=nullptr)
- void calc (std::vector< Array_Type > *array_bank=nullptr, std::vector< double > *stats_bank=nullptr, unsigned int max_num_elements_=0u)

  *Computes the entire support.*
- std::vector< double > get_counts () const

- std::vector< double > * get_current_stats ()

  *List current statistics.*
- void print () const
- const FreqTable< double > & get_data () const
- Counters< Array_Type, Data_Counter_Type > * get_counters ()

  *Vector of couter functions.*
- Rules< Array_Type, Data_Rule_Type > * get_rules ()

  *Vector of static rules (cells to iterate).*
- Rules< Array_Type, Data_Rule_Dyn_Type > * get_rules_dyn ()

  *Vector of dynamic rules (to include/exclude a realizaton).*

### Resets the support calculator

*If needed, the counters of a support object can be reused.*

***Parameters***

| | |
|---|---|
| Array↩_ | *New array over which the support will be computed.* |

- void reset_array ()
- void reset_array (const Array_Type &Array_)

### Manage counters

***Parameters***

| | |
|---|---|
| f_ | *A counter to be added.* |
| counters↩_ | *A vector of counters to be added.* |

- void add_counter (Counter< Array_Type, Data_Counter_Type > f_)
- void set_counters (Counters< Array_Type, Data_Counter_Type > *counters_)

### Manage rules

***Parameters***

| | |
|---|---|
| f_ | *A rule to be added.* |
| counters↩_ | *A vector of rules to be added.* |

- void add_rule (Rule< Array_Type, Data_Rule_Type > *f_)
- void add_rule (Rule< Array_Type, Data_Rule_Type > f_)
- void set_rules (Rules< Array_Type, Data_Rule_Type > *rules_)
- void add_rule_dyn (Rule< Array_Type, Data_Rule_Dyn_Type > *f_)
- void add_rule_dyn (Rule< Array_Type, Data_Rule_Dyn_Type > f_)
- void set_rules_dyn (Rules< Array_Type, Data_Rule_Dyn_Type > *rules_)
- bool eval_rules_dyn (const std::vector< double > &counts, const uint &i, const uint &j)

## Public Attributes

- uint N

- uint M
- bool delete_counters = true
- bool delete_rules = true
- bool delete_rules_dyn = true
- uint max_num_elements = BARRY_MAX_NUM_ELEMENTS
- std::vector< double > current_stats
- std::vector< size_t > coordinates_free
- std::vector< size_t > coordinates_locked
- size_t coordiantes_n_free
- size_t coordiantes_n_locked
- std::vector< double > change_stats
- std::vector< size_t > hashes
- std::vector< bool > hashes_initialized
- size_t n_counters

### 7.40.1 Detailed Description

**template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>**
**class Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >**

Compute the support of sufficient statistics.

Given an array and a set of counters, this object iterates throughout the support set of the Array while at the same time computing the support of the sufficient statitics.

The members `rule` and `rule_dyn` allow constraining the support. The first will establish which cells of the array will be used to iterate, for example, in the case of social networks, self-loops are not allowed, so the entire diagonal would be fixed to zero, reducing the size of the support.

In the case of `rule_dyn`, the function will stablish dynamically whether the current state will be included in the counts or not. For example, this set of rules can be used to constrain the support to networks that have a prescribed degree sequence.

Definition at line 42 of file support-bones.hpp.

### 7.40.2 Constructor & Destructor Documentation

#### 7.40.2.1 Support() [1/3]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Support (
            const Array_Type & Array_ )  [inline]
```

Constructor passing a reference Array.

Definition at line 87 of file support-bones.hpp.

#### 7.40.2.2 Support() [2/3]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Support (
            uint N_,
            uint M_ )  [inline]
```

Constructor specifying the dimensions of the array (empty).

Definition at line 96 of file support-bones.hpp.

#### 7.40.2.3 Support() [3/3]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Support ( )
[inline]
```

Definition at line 103 of file support-bones.hpp.

#### 7.40.2.4 ~Support()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::~Support ( )
[inline]
```

Definition at line 110 of file support-bones.hpp.

### 7.40.3 Member Function Documentation

#### 7.40.3.1 add_counter()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_counter
(
            Counter< Array_Type, Data_Counter_Type > f_ )
```

#### 7.40.3.2 add_rule() [1/2]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > * f_ )
```

#### 7.40.3.3 add_rule() [2/2]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
            Rule< Array_Type, Data_Rule_Type > f_ )
```

#### 7.40.3.4 add_rule_dyn() [1/2]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_↩
dyn (
            Rule< Array_Type, Data_Rule_Dyn_Type > * f_ )
```

#### 7.40.3.5 add_rule_dyn() [2/2]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_↩
dyn (
            Rule< Array_Type, Data_Rule_Dyn_Type > f_ )
```

#### 7.40.3.6 calc()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::calc (
            std::vector< Array_Type > * array_bank = nullptr,
            std::vector< double > * stats_bank = nullptr,
            unsigned int max_num_elements_ = 0u )
```

Computes the entire support.

Not to be used by the user. Sets the starting point in the array (column-major).

**Parameters**

| | |
|---|---|
| *array_bank* | If specified, the counter will add to the vector each possible state of the array, as it counts. |
| *stats_bank* | If specified, the counter will add to the vector each possible set of statistics, as it counts. |

#### 7.40.3.7 eval_rules_dyn()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::eval_↩
rules_dyn (
            const std::vector< double > & counts,
            const uint & i,
            const uint & j )
```

#### 7.40.3.8 get_counters()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Counters<Array_Type,Data_Counter_Type>* Support< Array_Type, Data_Counter_Type, Data_Rule_↩
Type, Data_Rule_Dyn_Type >::get_counters ( )
```

Vector of couter functions.

#### 7.40.3.9 get_counts()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< double > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn↩
_Type >::get_counts ( ) const
```

#### 7.40.3.10 get_current_stats()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< double >* Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_↩
Dyn_Type >::get_current_stats ( )
```

List current statistics.

#### 7.40.3.11 get_data()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
const FreqTable< double >& Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_↩
Rule_Dyn_Type >::get_data ( ) const
```

#### 7.40.3.12 get_rules()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Rules<Array_Type,Data_Rule_Type>* Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_rules ( )
```

Vector of static rules (cells to iterate).

#### 7.40.3.13 get_rules_dyn()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Rules<Array_Type,Data_Rule_Dyn_Type>* Support< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_rules_dyn ( )
```

Vector of dynamic rules (to include/exclude a realizaton).

#### 7.40.3.14 init_support()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::init_↩
support (
            std::vector< Array_Type > * array_bank = nullptr,
            std::vector< double > * stats_bank = nullptr )
```

#### 7.40.3.15 print()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::print ( )
const
```

#### 7.40.3.16 reset_array() [1/2]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::reset_array
( )
```

#### 7.40.3.17 reset_array() [2/2]

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::reset_array
(
            const Array_Type & Array_ )
```

#### 7.40.3.18 set_counters()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_↩
counters (
            Counters< Array_Type, Data_Counter_Type > * counters_ )
```

#### 7.40.3.19 set_rules()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_rules (
            Rules< Array_Type, Data_Rule_Type > * rules_ )
```

#### 7.40.3.20 set_rules_dyn()

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_rules↩
_dyn (
            Rules< Array_Type, Data_Rule_Dyn_Type > * rules_ )
```

### 7.40.4 Member Data Documentation

#### 7.40.4.1 change_stats

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< double > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn↩
_Type >::change_stats
```

Definition at line 80 of file support-bones.hpp.

#### 7.40.4.2 coordiantes_n_free

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::coordiantes↩
_n_free
```

Definition at line 78 of file support-bones.hpp.

#### 7.40.4.3 coordiantes_n_locked

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::coordiantes↩
_n_locked
```

Definition at line 79 of file support-bones.hpp.

#### 7.40.4.4 coordinates_free

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< size_t > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn↩
_Type >::coordinates_free
```

Definition at line 76 of file support-bones.hpp.

#### 7.40.4.5 coordinates_locked

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< size_t > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn↩
_Type >::coordinates_locked
```

Definition at line 77 of file support-bones.hpp.

#### 7.40.4.6 current_stats

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< double > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn↩
_Type >::current_stats
```

Definition at line 75 of file support-bones.hpp.

#### 7.40.4.7 delete_counters

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↩
counters = true
```

Definition at line 69 of file support-bones.hpp.

#### 7.40.4.8 delete_rules

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↩
rules = true
```

Definition at line 70 of file support-bones.hpp.

#### 7.40.4.9 delete_rules_dyn

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↩
rules_dyn = true
```

Definition at line 71 of file support-bones.hpp.

#### 7.40.4.10 hashes

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< size_t > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn↩
_Type >::hashes
```

Definition at line 81 of file support-bones.hpp.

#### 7.40.4.11 hashes_initialized

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< bool > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_↩
Type >::hashes_initialized
```

Definition at line 82 of file support-bones.hpp.

#### 7.40.4.12 M

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::M
```

Definition at line 68 of file support-bones.hpp.

#### 7.40.4.13 max_num_elements

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::max_num_↩
elements = BARRY_MAX_NUM_ELEMENTS
```

Definition at line 72 of file support-bones.hpp.

#### 7.40.4.14 N

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::N
```

Definition at line 68 of file support-bones.hpp.

#### 7.40.4.15 n_counters

```
template<typename Array_Type = BArray<bool, bool>, typename Data_Counter_Type = bool, typename
Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::n_↩
counters
```

Definition at line 83 of file support-bones.hpp.

The documentation for this class was generated from the following file:

- include/barry/support-bones.hpp

# 7.41 vecHasher< T > Struct Template Reference

`#include <typedefs.hpp>`

## Public Member Functions

- std::size_t operator() (std::vector< T > const &dat) const noexcept

## 7.41.1 Detailed Description

**template<typename T>**
**struct vecHasher< T >**

Definition at line 106 of file typedefs.hpp.

## 7.41.2 Member Function Documentation

### 7.41.2.1 operator()()

```
template<typename T >
std::size_t vecHasher< T >::operator() (
            std::vector< T > const & dat ) const  [inline], [noexcept]
```

Definition at line 109 of file typedefs.hpp.

The documentation for this struct was generated from the following file:

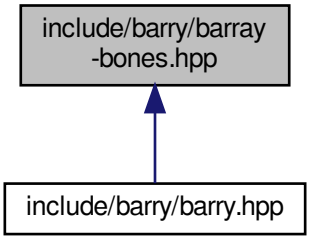
- include/barry/typedefs.hpp

# Chapter 8

# File Documentation

## 8.1 include/barry/barray-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:




### Classes

- class BArray< Cell_Type, Data_Type >

  *Baseline class for binary arrays.*

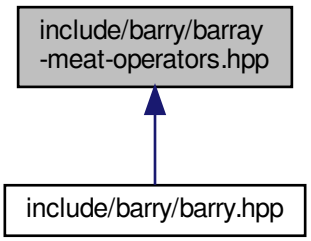
## 8.2 include/barry/barray-iterator.hpp File Reference

### Classes

- class ConstBArrayRowIter< Cell_Type, Data_Type >

## 8.3 include/barry/barray-meat-operators.hpp File Reference

This graph shows which files directly or indirectly include this file:




### Macros

- #define BARRAY_TYPE() BArray<Cell_Type, Data_Type>
- #define BARRAY_TEMPLATE_ARGS() <typename Cell_Type, typename Data_Type>
- #define BARRAY_TEMPLATE(a, b) template BARRAY_TEMPLATE_ARGS() inline a BARRAY_TYPE()::b
- #define ROW(a) this->el_ij[a]
- #define COL(a) this->el_ji[a]

### Functions

- template BARRAY_TEMPLATE_ARGS () inline void checkdim_(const BARRAY_TYPE() &lhs
- template const BARRAY_TYPE () &rhs)
- BARRAY_TEMPLATE (BARRAY_TYPE()&, operator+=)(const BArray< Cell_Type
- for (uint i=0u;i< nrow();++i) for(uint j=0u = el[POS(i, j)]
- j< ncol();++j) this-> operator() (i, j)+
- BARRAY_TEMPLATE (BARRAY_TYPE()&, operator+=)(const Cell_Type &rhs)
- BARRAY_TEMPLATE (BARRAY_TYPE()&, operator-=)(const BArray< Cell_Type
- BARRAY_TEMPLATE (BARRAY_TYPE()&, operator-=)(const Cell_Type &rhs)
- BARRAY_TEMPLATE (BARRAY_TYPE()&, operator*=)(const Cell_Type &rhs)
- BARRAY_TEMPLATE (BARRAY_TYPE()&, operator/=)(const Cell_Type &rhs)

### Variables

- Data_Type & rhs
- return * this

### 8.3.1 Macro Definition Documentation

#### 8.3.1.1 BARRAY_TEMPLATE

```
#define BARRAY_TEMPLATE(
              a,
              b )  template BARRAY_TEMPLATE_ARGS() inline a BARRAY_TYPE()::b
```

Definition at line 11 of file barray-meat-operators.hpp.

#### 8.3.1.2 BARRAY_TEMPLATE_ARGS

```
template BARRAY_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

Definition at line 9 of file barray-meat-operators.hpp.

#### 8.3.1.3 BARRAY_TYPE

```
template Data_Type BARRAY_TYPE( ) BArray<Cell_Type, Data_Type>
```

Definition at line 7 of file barray-meat-operators.hpp.

#### 8.3.1.4 COL

```
#define COL(
              a ) this->el_ji[a]
```

Definition at line 15 of file barray-meat-operators.hpp.

#### 8.3.1.5 ROW

```
#define ROW(
              a ) this->el_ij[a]
```

Definition at line 14 of file barray-meat-operators.hpp.

### 8.3.2 Function Documentation

#### 8.3.2.1 BARRAY_TEMPLATE() [1/6]

```
BARRAY_TEMPLATE (
            BARRAY_TYPE()& ,
            operator*  ) const &
```

Definition at line 88 of file barray-meat-operators.hpp.

#### 8.3.2.2 BARRAY_TEMPLATE() [2/6]

```
BARRAY_TEMPLATE (
            BARRAY_TYPE()& ,
            operator+  ) const
```

#### 8.3.2.3 BARRAY_TEMPLATE() [3/6]

```
BARRAY_TEMPLATE (
            BARRAY_TYPE()& ,
            operator+  ) const &
```

Definition at line 46 of file barray-meat-operators.hpp.

#### 8.3.2.4 BARRAY_TEMPLATE() [4/6]

```
BARRAY_TEMPLATE (
            BARRAY_TYPE()& ,
            operator-  ) const
```

#### 8.3.2.5 BARRAY_TEMPLATE() [5/6]

```
BARRAY_TEMPLATE (
            BARRAY_TYPE()& ,
            operator-  ) const &
```

Definition at line 75 of file barray-meat-operators.hpp.

#### 8.3.2.6 BARRAY_TEMPLATE() [6/6]

```
BARRAY_TEMPLATE (
              BARRAY_TYPE()& ,
              operator/  ) const &
```

Definition at line 105 of file barray-meat-operators.hpp.

#### 8.3.2.7 BARRAY_TEMPLATE_ARGS()

```
template BARRAY_TEMPLATE_ARGS ( ) const &
```

#### 8.3.2.8 BARRAY_TYPE()

```
template const BARRAY_TYPE ( ) &
```

Definition at line 20 of file barray-meat-operators.hpp.

#### 8.3.2.9 for()

```
for ( ) = el[POS(i, j)]  [pure virtual]
```

Definition at line 66 of file barray-meat-operators.hpp.

#### 8.3.2.10 operator()()

```
j< ncol(); ++j) this-> operator() (
              i ,
              j  )
```

### 8.3.3 Variable Documentation

#### 8.3.3.1 rhs

```
Data_Type & rhs
```

**Initial value:**

```
{
    checkdim_(*this, rhs)
```

Definition at line 33 of file barray-meat-operators.hpp.

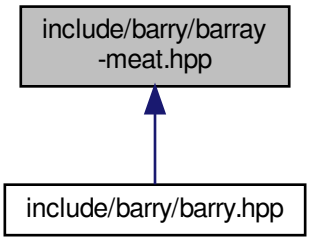
#### 8.3.3.2 this

```
return * this
```

Definition at line 43 of file barray-meat-operators.hpp.

## 8.4 include/barry/barray-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:




### Macros

- #define BARRAY_TYPE() BArray<Cell_Type, Data_Type>
- #define BARRAY_TEMPLATE_ARGS() <typename Cell_Type, typename Data_Type>
- #define BARRAY_TEMPLATE(a, b) template BARRAY_TEMPLATE_ARGS() inline a BARRAY_TYPE()::b
- #define ROW(a) this->el_ij[a]
- #define COL(a) this->el_ji[a]

## Functions

- BARRAY_TEMPLATE (, BArray)(uint N_
- el_ij resize (N)
- el_ji resize (M)
- for (uint i=0u;i< source.size();++i)
- Data_Type bool M (Array_.M)
- BARRAY_TEMPLATE (BARRAY_TYPE() &, operator=)(const BArray< Cell_Type
- BARRAY_TEMPLATE (, BArray)(BARRAY_TYPE() &&x) noexcept
- BARRAY_TEMPLATE (BARRAY_TYPE() &, operator=)(BARRAY_TYPE() &&x) noexcept
- BARRAY_TEMPLATE (bool, operator==)(const BARRAY_TYPE() &Array_)
- BARRAY_TEMPLATE (,~BArray)()
- BARRAY_TEMPLATE (void, set_data)(Data_Type *data_
- BARRAY_TEMPLATE (Data_Type *, D_ptr)()
- BARRAY_TEMPLATE (Data_Type &, D)()
- BARRAY_TEMPLATE (void, out_of_range)(uint i
- BARRAY_TEMPLATE (Cell_Type, get_cell)(uint i
- if (ROW(i).size()==0u) return(Cell_Type) 0.0
- if (search !=ROW(i).end()) return search -> second.value
- return (Cell_Type) 0.0
- BARRAY_TEMPLATE (std::vector< Cell_Type >, get_row_vec)(uint i
- std::vector< Cell_Type > ans (ncol(),(Cell_Type) false)
- for (const auto &iter :row(i, false)) ans[iter.first]
- BARRAY_TEMPLATE (void, get_row_vec)(std
- BARRAY_TEMPLATE (BARRAY_TYPE() &, operator-=)(const std
- BARRAY_TEMPLATE (void, insert_cell)(uint i
- if (check_exists)
- COL (j).emplace(i
- & ROW (i)[j])
- BARRAY_TEMPLATE (void, swap_cells)(uint i0
- if (report !=nullptr)(*report)
- if (check0 &check1)
- else if (!check0 &check1)
- else if (check0 &!check1)
- BARRAY_TEMPLATE (void, toggle_cell)(uint i
- BARRAY_TEMPLATE (void, swap_rows)(uint i0
- if (ROW(i0).size()==0u) move0
- if (ROW(i1).size()==0u) move1
- if (!move0 &&!move1) return
- ROW (i0).swap(ROW(i1))
- BARRAY_TEMPLATE (void, swap_cols)(uint j0
- if (COL(j0).size()==0u) check0
- if (COL(j1).size()==0u) check1
- if (check0 &&check1)
- else if (check0 &&!check1)
- else if (!check0 &&check1)
- BARRAY_TEMPLATE (void, zero_row)(uint i
- for (auto row=row0.begin();row !=row0.end();++row) rm_cell(i
- BARRAY_TEMPLATE (void, zero_col)(uint j
- if (COL(j).size()==0u) return
- BARRAY_TEMPLATE (void, transpose)()
- BARRAY_TEMPLATE (void, clear)(bool hard)
- BARRAY_TEMPLATE (void, resize)(uint N_
- if (M_< M) for(uint j = N_

## Variables

- uint M_
- uint const std::vector< uint > & source
- uint const std::vector< uint > const std::vector< uint > & target
- uint const std::vector< uint > const std::vector< uint > const std::vector< Cell_Type > & value
- uint const std::vector< uint > const std::vector< uint > const std::vector< Cell_Type > bool add
- if(source.size() !=value.size()) throw std N = N_
- M = M_
- return
- Data_Type & Array_
- Data_Type bool copy_data
- bool delete_data_
- data = data_
- delete_data = delete_data_
- uint j const
- uint j
- auto search = ROW(i).find(j)
- return ans
- uint const Cell< Cell_Type > & v
- uint const Cell< Cell_Type > bool check_bounds
- uint const Cell< Cell_Type > bool bool check_exists
- else
- NCells
- uint j0
- uint uint i1
- uint uint uint j1
- uint uint uint bool int int * report
- auto row0 = ROW(i)
- row first
- row false
- auto col0 = COL(j)

### 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 17 of file barray-meat.hpp.

#### 8.4.1.2 BARRAY_TEMPLATE_ARGS

```
#define BARRAY_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

Definition at line 15 of file barray-meat.hpp.

#### 8.4.1.3 BARRAY_TYPE

```
#define BARRAY_TYPE( ) BArray<Cell_Type, Data_Type>
```

Definition at line 13 of file barray-meat.hpp.

#### 8.4.1.4 COL

```
#define COL(
              a ) this->el_ji[a]
```

Definition at line 21 of file barray-meat.hpp.

#### 8.4.1.5 ROW

```
#define ROW(
              a ) this->el_ij[a]
```

Definition at line 20 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() [1/24]

```
BARRAY_TEMPLATE (
             BArray  ) &&  [noexcept]
```

Definition at line 230 of file barray-meat.hpp.

#### 8.4.2.3 BARRAY_TEMPLATE() [2/24]

```
BARRAY_TEMPLATE (
            BArray  )
```

#### 8.4.2.4 BARRAY_TEMPLATE() [3/24]

```
BARRAY_TEMPLATE (
            ~ BArray )
```

Definition at line 339 of file barray-meat.hpp.

#### 8.4.2.5 BARRAY_TEMPLATE() [4/24]

```
BARRAY_TEMPLATE (
            BARRAY_TYPE() & ,
            operator-  ) const
```

Definition at line 597 of file barray-meat.hpp.

#### 8.4.2.6 BARRAY_TEMPLATE() [5/24]

```
BARRAY_TEMPLATE (
            BARRAY_TYPE() & ,
            operator  ) &&  [noexcept]
```

Definition at line 272 of file barray-meat.hpp.

#### 8.4.2.7 BARRAY_TEMPLATE() [6/24]

```
BARRAY_TEMPLATE (
            BARRAY_TYPE() & ,
            operator  ) const
```

#### 8.4.2.8 BARRAY_TEMPLATE() [7/24]

```
BARRAY_TEMPLATE (
            bool ,
            operator  = = ) const &
```

Definition at line 321 of file barray-meat.hpp.

#### 8.4.2.9 BARRAY_TEMPLATE() [8/24]

```
BARRAY_TEMPLATE (
            Cell_Type ,
            get_cell  )
```

#### 8.4.2.10 BARRAY_TEMPLATE() [9/24]

```
BARRAY_TEMPLATE (
            Data_Type & ,
            D  )
```

Definition at line 372 of file barray-meat.hpp.

#### 8.4.2.11 BARRAY_TEMPLATE() [10/24]

```
BARRAY_TEMPLATE (
            Data_Type * ,
            D_ptr  )
```

Definition at line 361 of file barray-meat.hpp.

#### 8.4.2.12 BARRAY_TEMPLATE() [11/24]

```
BARRAY_TEMPLATE (
            std::vector< Cell_Type > ,
            get_row_vec  )
```

#### 8.4.2.13 BARRAY_TEMPLATE() [12/24]

```
BARRAY_TEMPLATE (
            void ,
            clear  )
```

Definition at line 1130 of file barray-meat.hpp.

#### 8.4.2.14 BARRAY_TEMPLATE() [13/24]

```
BARRAY_TEMPLATE (
            void ,
            get_row_vec  )
```

Definition at line 452 of file barray-meat.hpp.

#### 8.4.2.15 BARRAY_TEMPLATE() [14/24]

```
BARRAY_TEMPLATE (
            void ,
            insert_cell  )
```

#### 8.4.2.16 BARRAY_TEMPLATE() [15/24]

```
BARRAY_TEMPLATE (
            void ,
            out_of_range  )
```

#### 8.4.2.17 BARRAY_TEMPLATE() [16/24]

```
BARRAY_TEMPLATE (
            void ,
            resize  )
```

#### 8.4.2.18 BARRAY_TEMPLATE() [17/24]

```
BARRAY_TEMPLATE (
            void ,
            set_data  )
```

#### 8.4.2.19 BARRAY_TEMPLATE() [18/24]

```
BARRAY_TEMPLATE (
            void ,
            swap_cells  )
```

#### 8.4.2.20 BARRAY_TEMPLATE() [19/24]

```
BARRAY_TEMPLATE (
            void ,
            swap_cols  )
```

#### 8.4.2.21 BARRAY_TEMPLATE() [20/24]

```
BARRAY_TEMPLATE (
            void ,
            swap_rows  )
```

#### 8.4.2.22 BARRAY_TEMPLATE() [21/24]

```
BARRAY_TEMPLATE (
            void ,
            toggle_cell  )
```

#### 8.4.2.23 BARRAY_TEMPLATE() [22/24]

```
BARRAY_TEMPLATE (
            void ,
            transpose  )
```

Definition at line 1069 of file barray-meat.hpp.

#### 8.4.2.24 BARRAY_TEMPLATE() [23/24]

```
BARRAY_TEMPLATE (
            void ,
            zero_col  )
```

#### 8.4.2.25 BARRAY_TEMPLATE() [24/24]

```
BARRAY_TEMPLATE (
            void ,
            zero_row  )
```

#### 8.4.2.26 COL()

```
COL (
            j  )
```

#### 8.4.2.27 for() [1/3]

```
for (
            auto row = row0.begin();row !=row0.end();++row )
```

#### 8.4.2.28 for() [2/3]

```
for (
            const auto &iter  :rowi, false )
```

#### 8.4.2.29 for() [3/3]

```
for ( )
```

Definition at line 51 of file barray-meat.hpp.

#### 8.4.2.30 if() [1/17]

```
else if (
            !check0 && check1 )
```

Definition at line 1008 of file barray-meat.hpp.

#### 8.4.2.31 if() [2/17]

```
else if (
            !check0 & check1 )
```

Definition at line 856 of file barray-meat.hpp.

#### 8.4.2.32 if() [3/17]

```
if (
              !move0 &&!  move1 )
```

#### 8.4.2.33 if() [4/17]

```
else if (
              check0 &!  check1 )
```

Definition at line 864 of file barray-meat.hpp.

#### 8.4.2.34 if() [5/17]

```
else if (
              check0 &&!  check1 )
```

Definition at line 999 of file barray-meat.hpp.

#### 8.4.2.35 if() [6/17]

```
if (
              check0 && check1 )
```

Definition at line 972 of file barray-meat.hpp.

#### 8.4.2.36 if() [7/17]

```
if (
              check0 & check1 )
```

Definition at line 838 of file barray-meat.hpp.

#### 8.4.2.37 if() [8/17]

```
else if (
              check_exists  = = CHECK::BOTH )
```

Definition at line 679 of file barray-meat.hpp.

#### 8.4.2.38 if() [9/17]

```
if (
             COL(j).size()  = =0u )
```

#### 8.4.2.39 if() [10/17]

```
if (
             COL(j0).size()  = =0u )
```

#### 8.4.2.40 if() [11/17]

```
if (
             COL(j1).size()  = =0u )
```

#### 8.4.2.41 if() [12/17]

```
else if ( ) = N_
```

Definition at line 86 of file barray-meat.hpp.

#### 8.4.2.42 if() [13/17]

```
if (
             report !    = nullptr )
```

#### 8.4.2.43 if() [14/17]

```
if (
             ROW(i).size()  = =0u )
```

#### 8.4.2.44 if() [15/17]

```
if (
             ROW(i0).size()  = =0u )
```

#### 8.4.2.45 if() [16/17]

```
if (
             ROW(i1).size()   = =0u )
```

#### 8.4.2.46 if() [17/17]

```
if (
             search !     = ROW(i).end() ) -> second.value
```

#### 8.4.2.47 M()

```
Data_Type bool M (
             Array_.  M )
```

Definition at line 136 of file barray-meat.hpp.

#### 8.4.2.48 resize() [1/2]

```
el_ji resize (
             M  )
```

#### 8.4.2.49 resize() [2/2]

```
el_ij resize (
             N  )
```

#### 8.4.2.50 return()

```
return (
             Cell_Type  )
```

#### 8.4.2.51 ROW() [1/2]

```
& ROW (
              i  )
```

#### 8.4.2.52 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 34 of file barray-meat.hpp.

#### 8.4.3.2 ans

```
return ans
```

Definition at line 449 of file barray-meat.hpp.

#### 8.4.3.3 Array_

```
Data_Type & Array_
```

Definition at line 134 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 672 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 673 of file barray-meat.hpp.

#### 8.4.3.6 col0

```
auto col0 = COL(j)
```

Definition at line 1061 of file barray-meat.hpp.

#### 8.4.3.7 const

```
uint bool check_bounds const
```

**Initial value:**

```
{
    if (i >= N)
        throw std::range_error("The row is out of range.")
```

Definition at line 402 of file barray-meat.hpp.

#### 8.4.3.8 copy_data

```
Data_Type bool copy_data
```

Definition at line 135 of file barray-meat.hpp.

#### 8.4.3.9 data

```
data = data_
```

Definition at line 354 of file barray-meat.hpp.

#### 8.4.3.10 delete_data

```
delete_data = delete_data_
```

Definition at line 355 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 348 of file barray-meat.hpp.

#### 8.4.3.12 else

```
else (
             void  )
```

**Initial value:**

```
{
        ROW(i).insert(std::pair< uint, Cell<Cell_Type>>(j, v))
```

Definition at line 703 of file barray-meat.hpp.

#### 8.4.3.13 false

```
row false
```

Definition at line 1042 of file barray-meat.hpp.

#### 8.4.3.14 first

```
row first
```

Definition at line 1042 of file barray-meat.hpp.

#### 8.4.3.15 i1

```
uint i1
```

Definition at line 776 of file barray-meat.hpp.

#### 8.4.3.16 j

```
uint j
```

**Initial value:**

```
{
    if (init_fun == nullptr)
        return 0.0
```

Definition at line 414 of file barray-meat.hpp.

#### 8.4.3.17 j0

```
uint j0
```

Definition at line 775 of file barray-meat.hpp.

#### 8.4.3.18 j1

```
uint j1
```

Definition at line 776 of file barray-meat.hpp.

#### 8.4.3.19 M

```
M = M_
```

Definition at line 44 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 30 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 43 of file barray-meat.hpp.

#### 8.4.3.22 NCells

```
NCells
```

Definition at line 707 of file barray-meat.hpp.

#### 8.4.3.23 report

```
uint uint uint bool int int* report
```

Definition at line 779 of file barray-meat.hpp.

#### 8.4.3.24 return

```
return
```

Definition at line 66 of file barray-meat.hpp.

#### 8.4.3.25 row0

```
auto row0 = ROW(i)
```

Definition at line 1040 of file barray-meat.hpp.

#### 8.4.3.26 search

```
auto search = ROW(i).find(j)
```

Definition at line 426 of file barray-meat.hpp.

#### 8.4.3.27 source

```
uint const std::vector< uint > & source
```

Definition at line 31 of file barray-meat.hpp.

#### 8.4.3.28 target

```
uint const std::vector< uint > const std::vector< uint > & target
```

Definition at line 32 of file barray-meat.hpp.

#### 8.4.3.29 v

```
uint Cell_Type v
```

Definition at line 671 of file barray-meat.hpp.

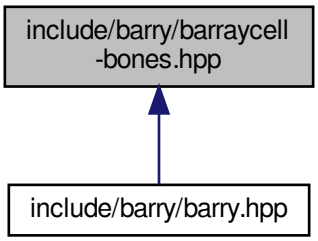
#### 8.4.3.30 value

```
uint const std::vector< uint > const std::vector< uint > const std::vector< Cell_Type >&
value
```

Definition at line 33 of file barray-meat.hpp.

## 8.5 include/barry/barraycell-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:

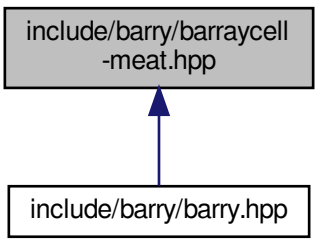



### Classes

- class BArrayCell< Cell_Type, Data_Type >
- class BArrayCell_const< Cell_Type, Data_Type >

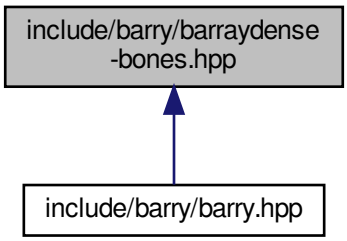
## 8.6 include/barry/barraycell-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



## 8.7 include/barry/barraydense-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



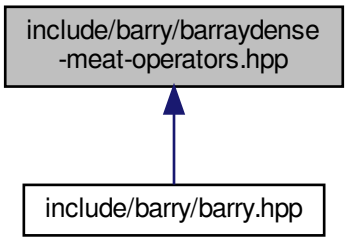

### Classes

- class BArrayDense< Cell_Type, Data_Type >

  *Baseline class for binary arrays.*

## 8.8 include/barry/barraydense-meat-operators.hpp File Reference

This graph shows which files directly or indirectly include this file:




### Macros

- #define BDENSE_TYPE() BArrayDense<Cell_Type, Data_Type>
- #define BDENSE_TEMPLATE_ARGS() <typename Cell_Type, typename Data_Type>
- #define BDENSE_TEMPLATE(a, b) template BDENSE_TEMPLATE_ARGS() inline a BDENSE_TYPE()::b
- #define ROW(a) this->el_ij[a]
- #define COL(a) this->el_ji[a]
- #define POS(a, b) (b)*N + (a)
- #define POS_N(a, b, c) (b)*(c) + (a)

## Functions

- template BDENSE_TEMPLATE_ARGS () inline void checkdim_(const BDENSE_TYPE() &lhs
- template const BDENSE_TYPE () &rhs)
- BDENSE_TEMPLATE (BDENSE_TYPE()&, operator+=)(const BDENSE_TYPE() &rhs)
- BDENSE_TEMPLATE (BDENSE_TYPE()&, operator-=)(const BDENSE_TYPE() &rhs)
- BDENSE_TEMPLATE (BDENSE_TYPE()&, operator∗=)(const Cell_Type &rhs)
- BDENSE_TEMPLATE (BDENSE_TYPE()&, operator/=)(const Cell_Type &rhs)

### 8.8.1 Macro Definition Documentation

#### 8.8.1.1 BDENSE_TEMPLATE

```
#define BDENSE_TEMPLATE(
              a,
              b )  template BDENSE_TEMPLATE_ARGS() inline a BDENSE_TYPE()::b
```

Definition at line 11 of file barraydense-meat-operators.hpp.

#### 8.8.1.2 BDENSE_TEMPLATE_ARGS

```
template BDENSE_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

Definition at line 9 of file barraydense-meat-operators.hpp.

#### 8.8.1.3 BDENSE_TYPE

```
template Data_Type BDENSE_TYPE( ) BArrayDense<Cell_Type, Data_Type>
```

Definition at line 7 of file barraydense-meat-operators.hpp.

#### 8.8.1.4 COL

```
#define COL(
              a ) this->el_ji[a]
```

Definition at line 15 of file barraydense-meat-operators.hpp.

#### 8.8.1.5 POS

```
#define POS(
              a,
              b ) (b)*N + (a)
```

Definition at line 16 of file barraydense-meat-operators.hpp.

#### 8.8.1.6 POS_N

```
#define POS_N(
              a,
              b,
              c ) (b)*(c) + (a)
```

Definition at line 17 of file barraydense-meat-operators.hpp.

#### 8.8.1.7 ROW

```
#define ROW(
              a ) this->el_ij[a]
```

Definition at line 14 of file barraydense-meat-operators.hpp.

### 8.8.2 Function Documentation

#### 8.8.2.1 BDENSE_TEMPLATE() [1/4]

```
BDENSE_TEMPLATE (
             BDENSE_TYPE()& ,
             operator*  ) const &
```

Definition at line 90 of file barraydense-meat-operators.hpp.

#### 8.8.2.2 BDENSE_TEMPLATE() [2/4]

```
BDENSE_TEMPLATE (
             BDENSE_TYPE()& ,
             operator+  ) const &
```

Definition at line 34 of file barraydense-meat-operators.hpp.

#### 8.8.2.3 BDENSE_TEMPLATE() [3/4]

```
BDENSE_TEMPLATE (
            BDENSE_TYPE()& ,
            operator-  ) const &
```

Definition at line 61 of file barraydense-meat-operators.hpp.

#### 8.8.2.4 BDENSE_TEMPLATE() [4/4]

```
BDENSE_TEMPLATE (
            BDENSE_TYPE()& ,
            operator/  ) const &
```

Definition at line 101 of file barraydense-meat-operators.hpp.

#### 8.8.2.5 BDENSE_TEMPLATE_ARGS()

```
template BDENSE_TEMPLATE_ARGS ( ) const &
```

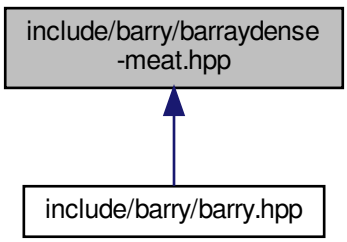
#### 8.8.2.6 BDENSE_TYPE()

```
template const BDENSE_TYPE ( ) &
```

Definition at line 22 of file barraydense-meat-operators.hpp.

## 8.9 include/barry/barraydense-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



## Macros

- #define BDENSE_TYPE() BArrayDense<Cell_Type, Data_Type>
- #define BDENSE_TEMPLATE_ARGS() <typename Cell_Type, typename Data_Type>
- #define BDENSE_TEMPLATE(a, b) template BDENSE_TEMPLATE_ARGS() inline a BDENSE_TYPE()::b
- #define ROW(a) this->el_ij[a]
- #define COL(a) this->el_ji[a]
- #define POS(a, b) (b)*N + (a)
- #define POS_N(a, b, c) (b)*(c) + (a)
- #define ZERO_CELL static_cast<Cell_Type>(0.0)

## Functions

- BDENSE_TEMPLATE (, BArrayDense)(uint N_
- el resize (N *M, ZERO_CELL)
- el_rowsums resize (N, ZERO_CELL)
- el_colsums resize (M, ZERO_CELL)
- for (uint i=0u;i< source.size();++i)
- BDENSE_TEMPLATE (, BArrayDense)(const BDENSE_TYPE() &Array_
- bool M (Array_.M)
- BDENSE_TEMPLATE (BDENSE_TYPE() &, operator=)(const BDENSE_TYPE() &Array_)
- BDENSE_TEMPLATE (, BArrayDense)(BDENSE_TYPE() &&x) noexcept
- BDENSE_TEMPLATE (BDENSE_TYPE() &, operator=)(BDENSE_TYPE() &&x) noexcept
- BDENSE_TEMPLATE (bool, operator==)(const BDENSE_TYPE() &Array_)
- BDENSE_TEMPLATE (, ~BArrayDense)()
- BDENSE_TEMPLATE (void, set_data)(Data_Type *data_
- BDENSE_TEMPLATE (Data_Type *, D_ptr)()
- BDENSE_TEMPLATE (const Data_Type *, D_ptr)() const
- BDENSE_TEMPLATE (Data_Type &, D)()
- BDENSE_TEMPLATE (const Data_Type &, D)() const
- BDENSE_TEMPLATE (void, out_of_range)(uint i
- BDENSE_TEMPLATE (Cell_Type, get_cell)(uint i
- BDENSE_TEMPLATE (std::vector< Cell_Type >, get_row_vec)(uint i
- std::vector< Cell_Type > ans (ncol(), static_cast< Cell_Type >(false))
- BDENSE_TEMPLATE (void, get_row_vec)(std
- BDENSE_TEMPLATE (Entries< Cell_Type >, get_entries)() const
- BDENSE_TEMPLATE (bool, is_empty)(uint i
- BDENSE_TEMPLATE (unsigned int, nrow)() const noexcept
- BDENSE_TEMPLATE (unsigned int, ncol)() const noexcept
- BDENSE_TEMPLATE (unsigned int, nnozero)() const noexcept
- BDENSE_TEMPLATE (Cell< Cell_Type >, default_val)() const
- BDENSE_TEMPLATE (BDENSE_TYPE() &, operator+=)(const std
- BDENSE_TEMPLATE (BDENSE_TYPE() &, operator-=)(const std
- BDENSE_TEMPLATE (void, insert_cell)(uint i
- if (el[POS(i, j)]==BARRY_ZERO_DENSE)
- BDENSE_TEMPLATE (void, swap_cells)(uint i0
- if ((i0==i1) &&(j0==j1)) return
- rm_cell (i0, j0, false, false)
- rm_cell (i1, j1, false, false)
- insert_cell (i0, j0, val1, false, false)
- insert_cell (i1, j1, val0, false, false)
- BDENSE_TEMPLATE (void, toggle_cell)(uint i
- else rm_cell (i, j, false, false)
- BDENSE_TEMPLATE (void, swap_rows)(uint i0

- BDENSE_TEMPLATE (void, swap_cols)(uint j0
- BDENSE_TEMPLATE (void, zero_row)(uint i
- if (el_rowsums[i]==ZERO_CELL) return
- BDENSE_TEMPLATE (void, zero_col)(uint j
- if (el_colsums[j]==ZERO_CELL) return
- BDENSE_TEMPLATE (void, transpose)()
- BDENSE_TEMPLATE (void, clear)(bool hard)
- BDENSE_TEMPLATE (void, resize)(uint N_
- el resize (N_ ∗M_, ZERO_CELL)
- el_rowsums resize (N_, ZERO_CELL)
- el_colsums resize (M_, ZERO_CELL)
- BDENSE_TEMPLATE (void, reserve)()
- BDENSE_TEMPLATE (void, print)(const char ∗fmt
- va_start (args, fmt)
- vprintf (fmt, args)
- va_end (args)
- BDENSE_TEMPLATE (const std::vector< Cell_Type > &, get_data)() const
- BDENSE_TEMPLATE (const Cell_Type, rowsum)(unsigned int i) const
- BDENSE_TEMPLATE (const Cell_Type, colsum)(unsigned int j) const

## Variables

- uint M_
- uint const std::vector< uint > & source
- uint const std::vector< uint > const std::vector< uint > & target
- uint const std::vector< uint > const std::vector< uint > const std::vector< Cell_Type > & value
- uint const std::vector< uint > const std::vector< uint > const std::vector< Cell_Type > bool add
- if(source.size() !=value.size()) throw std N = N_
- M = M_
- return
- bool copy_data
- bool delete_data_
- data = data_
- delete_data = delete_data_
- uint j const
- uint j
- return el [POS(i, j)] == ZERO_CELL
- return ans
- uint const Cell< Cell_Type > & v
- uint const Cell< Cell_Type > bool check_bounds
- uint const Cell< Cell_Type > bool bool check_exists
- else
- el_rowsums [i] = (v.value - old)
- el_colsums [j] = (v.value - old)
- uint j0
- uint uint i1
- uint uint uint j1
- uint uint uint bool int int ∗ report
- Cell_Type val0 = el[POS(i0,j0)]
- Cell_Type val1 = el[POS(i1,j1)]
- false
- col

## 8.9.1 Macro Definition Documentation

### 8.9.1.1 BDENSE_TEMPLATE

```
#define BDENSE_TEMPLATE(
            a,
            b )  template BDENSE_TEMPLATE_ARGS() inline a BDENSE_TYPE()::b
```

Definition at line 27 of file barraydense-meat.hpp.

### 8.9.1.2 BDENSE_TEMPLATE_ARGS

```
#define BDENSE_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

Definition at line 25 of file barraydense-meat.hpp.

### 8.9.1.3 BDENSE_TYPE

```
#define BDENSE_TYPE( ) BArrayDense<Cell_Type, Data_Type>
```

Definition at line 23 of file barraydense-meat.hpp.

### 8.9.1.4 COL

```
#define COL(
            a ) this->el_ji[a]
```

Definition at line 31 of file barraydense-meat.hpp.

### 8.9.1.5 POS

```
#define POS(
            a,
            b ) (b)*N + (a)
```

Definition at line 32 of file barraydense-meat.hpp.

#### 8.9.1.6 POS_N

```
#define POS_N(
              a,
              b,
              c ) (b)*(c) + (a)
```

Definition at line 33 of file barraydense-meat.hpp.

#### 8.9.1.7 ROW

```
#define ROW(
              a ) this->el_ij[a]
```

Definition at line 30 of file barraydense-meat.hpp.

#### 8.9.1.8 ZERO_CELL

```
#define ZERO_CELL static_cast<Cell_Type>(0.0)
```

Definition at line 38 of file barraydense-meat.hpp.

### 8.9.2 Function Documentation

#### 8.9.2.1 ans()

```
std::vector< Cell_Type > ans (
             ncol() ,
             static_cast< Cell_Type > false )
```

#### 8.9.2.2 BDENSE_TEMPLATE() [1/39]

```
BDENSE_TEMPLATE (
             BArrayDense  ) &&  [noexcept]
```

Definition at line 240 of file barraydense-meat.hpp.

#### 8.9.2.3 BDENSE_TEMPLATE() [2/39]

```
BDENSE_TEMPLATE (
            BArrayDense  ) const &
```

#### 8.9.2.4 BDENSE_TEMPLATE() [3/39]

```
BDENSE_TEMPLATE (
            BArrayDense  )
```

#### 8.9.2.5 BDENSE_TEMPLATE() [4/39]

```
BDENSE_TEMPLATE (
            ~ BArrayDense )
```

Definition at line 318 of file barraydense-meat.hpp.

#### 8.9.2.6 BDENSE_TEMPLATE() [5/39]

```
BDENSE_TEMPLATE (
            BDENSE_TYPE() & ,
            operator+  ) const
```

Definition at line 566 of file barraydense-meat.hpp.

#### 8.9.2.7 BDENSE_TEMPLATE() [6/39]

```
BDENSE_TEMPLATE (
            BDENSE_TYPE() & ,
            operator-  ) const
```

Definition at line 584 of file barraydense-meat.hpp.

#### 8.9.2.8 BDENSE_TEMPLATE() [7/39]

```
BDENSE_TEMPLATE (
            BDENSE_TYPE() & ,
            operator  ) &&  [noexcept]
```

Definition at line 257 of file barraydense-meat.hpp.

#### 8.9.2.9 BDENSE_TEMPLATE() [8/39]

```
BDENSE_TEMPLATE (
            BDENSE_TYPE() & ,
            operator  ) const &
```

Definition at line 194 of file barraydense-meat.hpp.

#### 8.9.2.10 BDENSE_TEMPLATE() [9/39]

```
BDENSE_TEMPLATE (
            bool ,
            is_empty  )
```

#### 8.9.2.11 BDENSE_TEMPLATE() [10/39]

```
BDENSE_TEMPLATE (
            bool ,
            operator  = = ) const &
```

Definition at line 300 of file barraydense-meat.hpp.

#### 8.9.2.12 BDENSE_TEMPLATE() [11/39]

```
BDENSE_TEMPLATE (
            Cell< Cell_Type > ,
            default_val  ) const
```

Definition at line 562 of file barraydense-meat.hpp.

#### 8.9.2.13 BDENSE_TEMPLATE() [12/39]

```
BDENSE_TEMPLATE (
            Cell_Type ,
            get_cell  )
```

#### 8.9.2.14 BDENSE_TEMPLATE() [13/39]

```
BDENSE_TEMPLATE (
            const Cell_Type,
            colsum  ) const
```

Definition at line 999 of file barraydense-meat.hpp.

#### 8.9.2.15 BDENSE_TEMPLATE() [14/39]

```
BDENSE_TEMPLATE (
            const Cell_Type,
            rowsum  ) const
```

Definition at line 994 of file barraydense-meat.hpp.

#### 8.9.2.16 BDENSE_TEMPLATE() [15/39]

```
BDENSE_TEMPLATE (
            const Data_Type & ,
            D  ) const
```

Definition at line 353 of file barraydense-meat.hpp.

#### 8.9.2.17 BDENSE_TEMPLATE() [16/39]

```
BDENSE_TEMPLATE (
            const Data_Type * ,
            D_ptr  ) const
```

Definition at line 345 of file barraydense-meat.hpp.

#### 8.9.2.18 BDENSE_TEMPLATE() [17/39]

```
BDENSE_TEMPLATE (
            const std::vector< Cell_Type > & ,
            get_data  ) const
```

Definition at line 989 of file barraydense-meat.hpp.

#### 8.9.2.19 BDENSE_TEMPLATE() [18/39]

```
BDENSE_TEMPLATE (
            Data_Type & ,
            D  )
```

Definition at line 349 of file barraydense-meat.hpp.

#### 8.9.2.20 BDENSE_TEMPLATE() [19/39]

```
BDENSE_TEMPLATE (
            Data_Type * ,
            D_ptr  )
```

Definition at line 341 of file barraydense-meat.hpp.

#### 8.9.2.21 BDENSE_TEMPLATE() [20/39]

```
BDENSE_TEMPLATE (
            Entries< Cell_Type > ,
            get_entries  ) const
```

Definition at line 502 of file barraydense-meat.hpp.

#### 8.9.2.22 BDENSE_TEMPLATE() [21/39]

```
BDENSE_TEMPLATE (
            std::vector< Cell_Type > ,
            get_row_vec  )
```

#### 8.9.2.23 BDENSE_TEMPLATE() [22/39]

```
BDENSE_TEMPLATE (
            unsigned int ,
            ncol  ) const  [noexcept]
```

Definition at line 548 of file barraydense-meat.hpp.

#### 8.9.2.24 BDENSE_TEMPLATE() [23/39]

```
BDENSE_TEMPLATE (
            unsigned int ,
            nnozero  ) const  [noexcept]
```

Definition at line 552 of file barraydense-meat.hpp.

#### 8.9.2.25 BDENSE_TEMPLATE() [24/39]

```
BDENSE_TEMPLATE (
            unsigned int ,
            nrow  ) const  [noexcept]
```

Definition at line 544 of file barraydense-meat.hpp.

#### 8.9.2.26 BDENSE_TEMPLATE() [25/39]

```
BDENSE_TEMPLATE (
            void ,
            clear  )
```

Definition at line 896 of file barraydense-meat.hpp.

#### 8.9.2.27 BDENSE_TEMPLATE() [26/39]

```
BDENSE_TEMPLATE (
            void ,
            get_row_vec  )
```

Definition at line 402 of file barraydense-meat.hpp.

#### 8.9.2.28 BDENSE_TEMPLATE() [27/39]

```
BDENSE_TEMPLATE (
            void ,
            insert_cell  )
```

#### 8.9.2.29 BDENSE_TEMPLATE() [28/39]

```
BDENSE_TEMPLATE (
             void ,
             out_of_range  )
```

#### 8.9.2.30 BDENSE_TEMPLATE() [29/39]

```
BDENSE_TEMPLATE (
             void ,
             print  ) const
```

#### 8.9.2.31 BDENSE_TEMPLATE() [30/39]

```
BDENSE_TEMPLATE (
             void ,
             reserve  )
```

Definition at line 946 of file barraydense-meat.hpp.

#### 8.9.2.32 BDENSE_TEMPLATE() [31/39]

```
BDENSE_TEMPLATE (
             void ,
             resize  )
```

#### 8.9.2.33 BDENSE_TEMPLATE() [32/39]

```
BDENSE_TEMPLATE (
             void ,
             set_data  )
```

#### 8.9.2.34 BDENSE_TEMPLATE() [33/39]

```
BDENSE_TEMPLATE (
             void ,
             swap_cells  )
```

#### 8.9.2.35 BDENSE_TEMPLATE() [34/39]

```
BDENSE_TEMPLATE (
            void ,
            swap_cols  )
```

#### 8.9.2.36 BDENSE_TEMPLATE() [35/39]

```
BDENSE_TEMPLATE (
            void ,
            swap_rows  )
```

#### 8.9.2.37 BDENSE_TEMPLATE() [36/39]

```
BDENSE_TEMPLATE (
            void ,
            toggle_cell  )
```

#### 8.9.2.38 BDENSE_TEMPLATE() [37/39]

```
BDENSE_TEMPLATE (
            void ,
            transpose  )
```

Definition at line 868 of file barraydense-meat.hpp.

#### 8.9.2.39 BDENSE_TEMPLATE() [38/39]

```
BDENSE_TEMPLATE (
            void ,
            zero_col  )
```

#### 8.9.2.40 BDENSE_TEMPLATE() [39/39]

```
BDENSE_TEMPLATE (
            void ,
            zero_row  )
```

#### 8.9.2.41 for()

```
for ( )
```

Definition at line 64 of file barraydense-meat.hpp.

#### 8.9.2.42 if() [1/4]

```
if (
              (i0==i1) &&(j0==j1)  )
```

#### 8.9.2.43 if() [2/4]

```
if (
              el [POS(i, j)] = = BARRY_ZERO_DENSE )
```

Definition at line 663 of file barraydense-meat.hpp.

#### 8.9.2.44 if() [3/4]

```
if (
              el_colsums [j] = =ZERO_CELL )
```

#### 8.9.2.45 if() [4/4]

```
if (
              el_rowsums [i] = =ZERO_CELL )
```

#### 8.9.2.46 insert_cell() [1/2]

```
insert_cell (
              i0 ,
              j0 ,
              val1 ,
              false ,
              false  )
```

#### 8.9.2.47 insert_cell() [2/2]

```
insert_cell (
            i1 ,
            j1 ,
            val0 ,
            false ,
            false  )
```

#### 8.9.2.48 M()

```
bool M (
            Array_.  M )
```

Definition at line 157 of file barraydense-meat.hpp.

#### 8.9.2.49 resize() [1/6]

```
el_colsums resize (
            M ,
            ZERO_CELL  )
```

#### 8.9.2.50 resize() [2/6]

```
el_colsums resize (
            M_ ,
            ZERO_CELL  )
```

#### 8.9.2.51 resize() [3/6]

```
el resize (
            N * M,
            ZERO_CELL  )
```

#### 8.9.2.52 resize() [4/6]

```
el_rowsums resize (
            N ,
            ZERO_CELL  )
```

#### 8.9.2.53 resize() [5/6]

```
el resize (
            N_ * M_,
            ZERO_CELL  )
```

#### 8.9.2.54 resize() [6/6]

```
el_rowsums resize (
            N_ ,
            ZERO_CELL  )
```

#### 8.9.2.55 rm_cell() [1/3]

```
else rm_cell (
            i ,
            j ,
            false ,
            false  )
```

#### 8.9.2.56 rm_cell() [2/3]

```
rm_cell (
            i0 ,
            j0 ,
            false ,
            false  )
```

#### 8.9.2.57 rm_cell() [3/3]

```
rm_cell (
            i1 ,
            j1 ,
            false ,
            false  )
```

#### 8.9.2.58 va_end()

```
va_end (
            args  )
```

#### 8.9.2.59 va_start()

```
va_start (
             args ,
             fmt  )
```

#### 8.9.2.60 vprintf()

```
vprintf (
             fmt ,
             args  )
```

### 8.9.3 Variable Documentation

#### 8.9.3.1 add

```
uint const std::vector< uint > const std::vector< uint > bool add
```

**Initial value:**

```
{

    if (source.size() != target.size())
        throw std::length_error("-source- and -target- don't match on length.")
```

Definition at line 47 of file barraydense-meat.hpp.

#### 8.9.3.2 ans

```
return ans
```

Definition at line 398 of file barraydense-meat.hpp.

#### 8.9.3.3 check_bounds

```
bool check_bounds
```

**Initial value:**

```
{

    if (check_bounds)
    {
        out_of_range(i0,0u);
        out_of_range(i1,0u);
    }




    for (uint j = 0u; j < M; ++j)
        std::swap(el[POS(i0, j)], el[POS(i1, j)])
```

Definition at line 654 of file barraydense-meat.hpp.

#### 8.9.3.4 check_exists

```
uint bool int check_exists
```

**Initial value:**

```
{

    if (check_bounds)
        out_of_range(i,j)
```

Definition at line 655 of file barraydense-meat.hpp.

#### 8.9.3.5 col

```
col
```

Definition at line 843 of file barraydense-meat.hpp.

#### 8.9.3.6 const

```
const
```

**Initial value:**

```
{
    if (i >= N)
        throw std::range_error("The row is out of range.")
```

Definition at line 360 of file barraydense-meat.hpp.

#### 8.9.3.7 copy_data

```
bool copy_data
```

Definition at line 156 of file barraydense-meat.hpp.

#### 8.9.3.8 data

```
data = data_
```

Definition at line 334 of file barraydense-meat.hpp.

#### 8.9.3.9 delete_data

```
delete_data = delete_data_
```

Definition at line 335 of file barraydense-meat.hpp.

#### 8.9.3.10 delete_data_

```
bool delete_data_
```

**Initial value:**

```
{
    if ((data != nullptr) && delete_data)
        delete data
```

Definition at line 328 of file barraydense-meat.hpp.

#### 8.9.3.11 el

```
return el == ZERO_CELL
```

Definition at line 381 of file barraydense-meat.hpp.

#### 8.9.3.12 el_colsums

```
el_colsums[j] = (v.value - old)
```

Definition at line 675 of file barraydense-meat.hpp.

#### 8.9.3.13 el_rowsums

```
el_rowsums[i] = (v.value - old)
```

Definition at line 674 of file barraydense-meat.hpp.

#### 8.9.3.14 else

```
else (
              void  )
```

**Initial value:**

```
{
        Cell_Type old = el[POS(i,j)]
```

Definition at line 670 of file barraydense-meat.hpp.

#### 8.9.3.15 false

```
false
```

Definition at line 767 of file barraydense-meat.hpp.

#### 8.9.3.16 i1

```
uint i1
```

Definition at line 721 of file barraydense-meat.hpp.

#### 8.9.3.17 j

```
j
```

Definition at line 373 of file barraydense-meat.hpp.

#### 8.9.3.18 j0

```
uint j0
```

Definition at line 720 of file barraydense-meat.hpp.

#### 8.9.3.19 j1

```
uint j1
```

Definition at line 721 of file barraydense-meat.hpp.

#### 8.9.3.20 M

```
M = M_
```

Definition at line 57 of file barraydense-meat.hpp.

#### 8.9.3.21 M_

```
uint M_
```

**Initial value:**

```
{
    std::vector< Cell_Type > el_tmp(el)
```

Definition at line 43 of file barraydense-meat.hpp.

#### 8.9.3.22 N

```
N = N_
```

Definition at line 56 of file barraydense-meat.hpp.

#### 8.9.3.23 report

```
uint uint uint bool int int* report
```

**Initial value:**

```
{
    if (check_bounds) {
        out_of_range(i0,j0);
        out_of_range(i1,j1);
    }


    if (report != nullptr)
        (*report) = EXISTS::BOTH
```

Definition at line 724 of file barraydense-meat.hpp.

#### 8.9.3.24 return

```
return
```

Definition at line 94 of file barraydense-meat.hpp.

#### 8.9.3.25 source

```
uint const std::vector< uint >& source
```

Definition at line 44 of file barraydense-meat.hpp.

#### 8.9.3.26 target

```
uint const std::vector< uint > const std::vector< uint >& target
```

Definition at line 45 of file barraydense-meat.hpp.

#### 8.9.3.27 v

```
uint Cell_Type v
```

Definition at line 653 of file barraydense-meat.hpp.

#### 8.9.3.28 val0

```
Cell_Type val0 = el[POS(i0,j0)]
```

Definition at line 742 of file barraydense-meat.hpp.

#### 8.9.3.29 val1

```
Cell_Type val1 = el[POS(i1,j1)]
```

Definition at line 743 of file barraydense-meat.hpp.

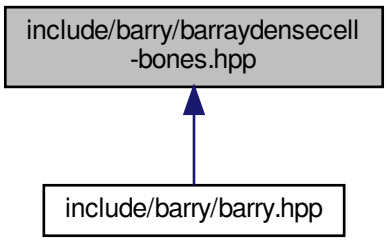
#### 8.9.3.30 value

```
uint const std::vector< uint > const std::vector< uint > const std::vector< Cell_Type >&
value
```

Definition at line 46 of file barraydense-meat.hpp.

# 8.10 include/barry/barraydensecell-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:




## Classes

- class BArrayDenseCell< Cell_Type, Data_Type >

## Macros

- #define POS(a, b) (a) + (b) * N

## 8.10.1 Macro Definition Documentation

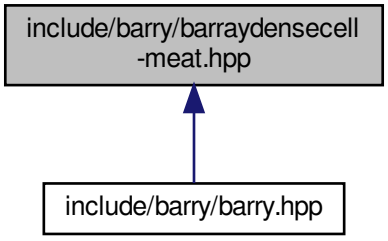
### 8.10.1.1 POS

```
#define POS(
              a,
              b ) (a) + (b) * N
```

Definition at line 6 of file barraydensecell-bones.hpp.

## 8.11 include/barry/barraydensecell-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:




### Macros

- #define POS(a, b) (a) + (b) ∗ dat->N

### 8.11.1 Macro Definition Documentation

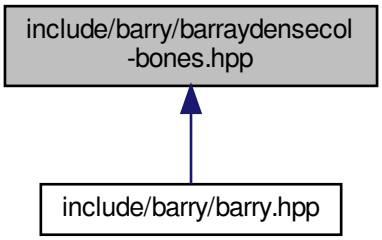
#### 8.11.1.1 POS

```
#define POS(
              a,
              b ) (a) + (b) * dat->N
```

Definition at line 6 of file barraydensecell-meat.hpp.

## 8.12 include/barry/barraydensecol-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



## Classes

- class BArrayDenseCol< Cell_Type, Data_Type >
- class BArrayDenseCol_const< Cell_Type, Data_Type >

## Macros

- #define POS(a, b) (b)*N + (a)
- #define POS_N(a, b, c) (b)*(c) + (a)
- #define ZERO_CELL static_cast<Cell_Type>(0.0)

### 8.12.1 Macro Definition Documentation

#### 8.12.1.1 POS

```
#define POS(
              a,
              b ) (b)*N + (a)
```

Definition at line 4 of file barraydensecol-bones.hpp.

#### 8.12.1.2 POS_N

```
#define POS_N(
              a,
              b,
              c ) (b)*(c) + (a)
```

Definition at line 5 of file barraydensecol-bones.hpp.

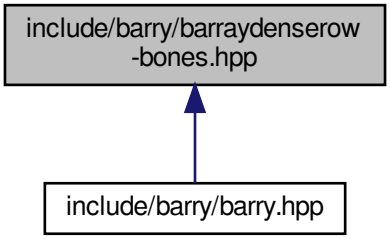
#### 8.12.1.3 ZERO_CELL

```
#define ZERO_CELL static_cast<Cell_Type>(0.0)
```

Definition at line 6 of file barraydensecol-bones.hpp.

## 8.13 include/barry/barraydenserow-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:




### Classes

- class BArrayDenseRow< Cell_Type, Data_Type >
- class BArrayDenseRow_const< Cell_Type, Data_Type >

### Macros

- #define POS(a, b) (b) * N + (a)
- #define POS_N(a, b, c) (b)*(c) + (a)
- #define ZERO_CELL static_cast< Cell_Type >(0.0)

### 8.13.1 Macro Definition Documentation

#### 8.13.1.1 POS

```
#define POS(
              a,
              b ) (b) * N + (a)
```

Definition at line 4 of file barraydenserow-bones.hpp.

#### 8.13.1.2 POS_N

```
#define POS_N(
              a,
              b,
              c ) (b)*(c) + (a)
```

Definition at line 5 of file barraydenserow-bones.hpp.

#### 8.13.1.3 ZERO_CELL

```
#define ZERO_CELL static_cast< Cell_Type >(0.0)
```

Definition at line 6 of file barraydenserow-bones.hpp.

## 8.14 include/barry/barrayrow-bones.hpp File Reference

### Classes

- class BArrayRow< Cell_Type, Data_Type >
- class BArrayRow_const< Cell_Type, Data_Type >

## 8.15 include/barry/barrayrow-meat.hpp File Reference

### Macros

- #define BROW_TYPE() BArrayRow<Cell_Type, Data_Type>
- #define BROW_TEMPLATE_ARGS() <typename Cell_Type, typename Data_Type>
- #define BROW_TEMPLATE(a, b) template BROW_TEMPLATE_ARGS() inline a BROW_TYPE()::b

### Functions

- BROW_TEMPLATE (void, operator=)(const BROW_TYPE() &val)
- BROW_TEMPLATE (void, operator+=)(const BROW_TYPE() &val)
- BROW_TEMPLATE (void, operator-=)(const BROW_TYPE() &val)
- BROW_TEMPLATE (void, operator*=)(const BROW_TYPE() &val)
- BROW_TEMPLATE (void, operator/=)(const BROW_TYPE() &val)

### 8.15.1 Macro Definition Documentation

#### 8.15.1.1 BROW_TEMPLATE

```
#define BROW_TEMPLATE(
              a,
              b )  template BROW_TEMPLATE_ARGS() inline a BROW_TYPE()::b
```

Definition at line 8 of file barrayrow-meat.hpp.

#### 8.15.1.2 BROW_TEMPLATE_ARGS

```
#define BROW_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

Definition at line 6 of file barrayrow-meat.hpp.

#### 8.15.1.3 BROW_TYPE

```
#define BROW_TYPE( ) BArrayRow<Cell_Type, Data_Type>
```

Definition at line 4 of file barrayrow-meat.hpp.

### 8.15.2 Function Documentation

#### 8.15.2.1 BROW_TEMPLATE() [1/5]

```
BROW_TEMPLATE (
             void ,
             operator*  ) const &
```

Definition at line 45 of file barrayrow-meat.hpp.

#### 8.15.2.2 BROW_TEMPLATE() [2/5]

```
BROW_TEMPLATE (
             void ,
             operator+  ) const &
```

Definition at line 25 of file barrayrow-meat.hpp.

#### 8.15.2.3 BROW_TEMPLATE() [3/5]

```
BROW_TEMPLATE (
            void ,
            operator-  ) const &
```

Definition at line 34 of file barrayrow-meat.hpp.

#### 8.15.2.4 BROW_TEMPLATE() [4/5]

```
BROW_TEMPLATE (
            void ,
            operator/  ) const &
```

Definition at line 55 of file barrayrow-meat.hpp.

#### 8.15.2.5 BROW_TEMPLATE() [5/5]

```
BROW_TEMPLATE (
            void ,
            operator  ) const &
```

Definition at line 11 of file barrayrow-meat.hpp.

## 8.16 include/barry/barrayvector-bones.hpp File Reference

### Classes

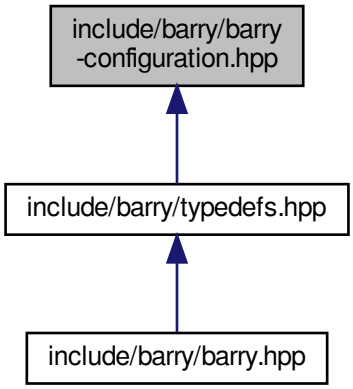
- class BArrayVector< Cell_Type, Data_Type >

  *Row or column of a BArray*
- class BArrayVector_const< Cell_Type, Data_Type >

## 8.17 include/barry/barrayvector-meat.hpp File Reference

## 8.18 include/barry/barry-configuration.hpp File Reference

This graph shows which files directly or indirectly include this file:




### Configuration MACROS

These are mostly related to performance. The definitions follow:

- `BARRY_USE_UNORDERED_MAP` If specified, then barry is compiled using `std::unordered_map`. Otherwise it will use `std::map` for the arrays.
- `BARRY_USE_SAFE_EXP` When specified, it will multiply all likelihoods in `Model` by (1/-100)/(1/-100) so that numerical overflows are avoided.
- `BARRY_USE_ISFINITE` When specified, it will introduce a macro that checks whether the likelihood is finite or not.
- `printf_barry` If not specified, will be defined as `printf`.
- `BARRY_DEBUG_LEVEL`, when defined, will make things verbose.

- #define BARRY_SAFE_EXP -100.0
- #define BARRY_ISFINITE(a)
- #define BARRY_CHECK_SUPPORT(x, maxs)
- #define printf_barry printf
- #define BARRY_MAX_NUM_ELEMENTS static_cast< size_t >(UINT_MAX/2u)
- template<typename Ta , typename Tb >
  using Map = std::map< Ta, Tb >

## 8.18.1 Macro Definition Documentation

### 8.18.1.1 BARRY_CHECK_SUPPORT

```
#define BARRY_CHECK_SUPPORT(
              x,
              maxs )
```

Definition at line 47 of file barry-configuration.hpp.

### 8.18.1.2 BARRY_ISFINITE

```
#define BARRY_ISFINITE(
              a )
```

Definition at line 40 of file barry-configuration.hpp.

### 8.18.1.3 BARRY_MAX_NUM_ELEMENTS

```
#define BARRY_MAX_NUM_ELEMENTS static_cast< size_t >(UINT_MAX/2u)
```

Definition at line 55 of file barry-configuration.hpp.

### 8.18.1.4 BARRY_SAFE_EXP

```
#define BARRY_SAFE_EXP -100.0
```

Definition at line 33 of file barry-configuration.hpp.

### 8.18.1.5 printf_barry

```
#define printf_barry printf
```

Definition at line 51 of file barry-configuration.hpp.

## 8.18.2 Typedef Documentation

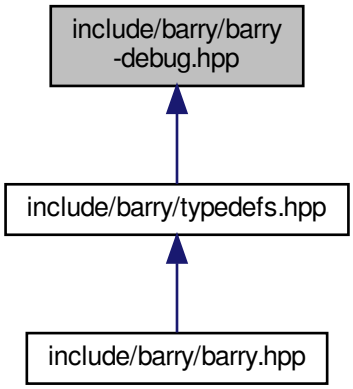
#### 8.18.2.1 Map

```
template<typename Ta , typename Tb >
using Map = std::map<Ta,Tb>
```

Definition at line 27 of file barry-configuration.hpp.

## 8.19 include/barry/barry-debug.hpp File Reference

This graph shows which files directly or indirectly include this file:




### Macros

- #define BARRY_DEBUG_LEVEL 0

### 8.19.1 Macro Definition Documentation

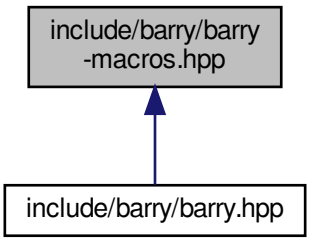
#### 8.19.1.1 BARRY_DEBUG_LEVEL

```
#define BARRY_DEBUG_LEVEL 0
```

Definition at line 5 of file barry-debug.hpp.

# 8.20 include/barry/barry-macros.hpp File Reference

This graph shows which files directly or indirectly include this file:




## Macros

- #define BARRY_ZERO Cell<Cell_Type>(0.0)
- #define BARRY_ZERO_DENSE static_cast<Cell_Type>(0.0)
- #define BARRY_ONE Cell<Cell_Type>(1.0)
- #define BARRY_ONE_DENSE static_cast<Cell_Type>(1.0)
- #define BARRY_UNUSED(expr) do { (void)(expr); } while (0);

## 8.20.1 Macro Definition Documentation

### 8.20.1.1 BARRY_ONE

```
#define BARRY_ONE Cell<Cell_Type>(1.0)
```

Definition at line 7 of file barry-macros.hpp.

### 8.20.1.2 BARRY_ONE_DENSE

```
#define BARRY_ONE_DENSE static_cast<Cell_Type>(1.0)
```

Definition at line 8 of file barry-macros.hpp.

#### 8.20.1.3 BARRY_UNUSED

```
#define BARRY_UNUSED(
             expr ) do { (void)(expr); } while (0);
```

Definition at line 10 of file barry-macros.hpp.

#### 8.20.1.4 BARRY_ZERO

```
#define BARRY_ZERO Cell<Cell_Type>(0.0)
```

Definition at line 4 of file barry-macros.hpp.

#### 8.20.1.5 BARRY_ZERO_DENSE

```
#define BARRY_ZERO_DENSE static_cast<Cell_Type>(0.0)
```

Definition at line 5 of file barry-macros.hpp.

## 8.21 include/barry/barry.hpp File Reference

```
#include <iostream>
#include <cstdarg>
#include <vector>
#include <unordered_map>
#include <functional>
#include <stdexcept>
#include <cmath>
#include <map>
#include <algorithm>
#include <utility>
#include <random>
#include <climits>
#include <cfloat>
#include <string>
#include <cstdint>
#include <memory>
#include "typedefs.hpp"
#include "barry-macros.hpp"
#include "freqtable.hpp"
#include "cell-bones.hpp"
#include "cell-meat.hpp"
#include "barray-bones.hpp"
#include "barraycell-bones.hpp"
#include "barray-meat.hpp"
#include "barraycell-meat.hpp"
#include "barray-meat-operators.hpp"
#include "barraydense-bones.hpp"
```

```
#include "barraydensecell-bones.hpp"
#include "barraydenserow-bones.hpp"
#include "barraydensecol-bones.hpp"
#include "barraydense-meat.hpp"
#include "barraydensecell-meat.hpp"
#include "barraydense-meat-operators.hpp"
#include "counters-bones.hpp"
#include "counters-meat.hpp"
#include "statscounter-bones.hpp"
#include "statscounter-meat.hpp"
#include "support-bones.hpp"
#include "support-meat.hpp"
#include "powerset-bones.hpp"
#include "powerset-meat.hpp"
#include "model-bones.hpp"
#include "model-meat.hpp"
#include "rules-bones.hpp"
#include "rules-meat.hpp"
#include "counters/network.hpp"
#include "counters/phylo.hpp"
#include "counters/defm.hpp"
```

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
- barry::counters::defm

## Macros

- #define BARRY_HPP
- #define BARRY_VERSION_MAYOR 0
- #define BARRY_VERSION_MINOR 1
- #define BARRY_VERSION BARRY_VERSION_MAYOR ## . ## BARRY_VERSION_MINOR
- #define COUNTER_FUNCTION(a)
- #define COUNTER_LAMBDA(a)
- #define RULE_FUNCTION(a)
- #define RULE_LAMBDA(a)

### 8.21.1 Macro Definition Documentation

#### 8.21.1.1 BARRY_HPP

```
#define BARRY_HPP
```

Definition at line 23 of file barry.hpp.

#### 8.21.1.2 BARRY_VERSION

```
#define BARRY_VERSION BARRY_VERSION_MAYOR ## .  ## BARRY_VERSION_MINOR
```

Definition at line 27 of file barry.hpp.

#### 8.21.1.3 BARRY_VERSION_MAYOR

```
#define BARRY_VERSION_MAYOR 0
```

Definition at line 25 of file barry.hpp.

#### 8.21.1.4 BARRY_VERSION_MINOR

```
#define BARRY_VERSION_MINOR 1
```

Definition at line 26 of file barry.hpp.

#### 8.21.1.5 COUNTER_FUNCTION

```
#define COUNTER_FUNCTION(
              a )
```

**Value:**

```
    template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
    inline double (a) (const Array_Type & Array, uint i, uint j, Data_Type & data)\
```

Definition at line 94 of file barry.hpp.

#### 8.21.1.6 COUNTER_LAMBDA

```
#define COUNTER_LAMBDA(
                a )
```

**Value:**

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
Counter_fun_type<Array_Type, Data_Type> a = \
[](const Array_Type & Array, uint i, uint j, Data_Type & data)
```

Definition at line 97 of file barry.hpp.

#### 8.21.1.7 RULE_FUNCTION

```
#define RULE_FUNCTION(
                a )
```

**Value:**

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
inline bool (a) (const Array_Type & Array, uint i, uint j, Data_Type & data)\
```

Definition at line 101 of file barry.hpp.

#### 8.21.1.8 RULE_LAMBDA

```
#define RULE_LAMBDA(
                a )
```

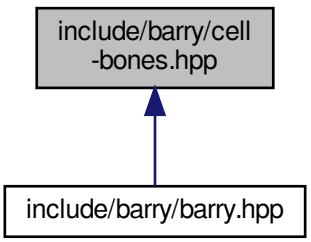
**Value:**

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
Rule_fun_type<Array_Type, Data_Type> a = \
[](const Array_Type & Array, uint i, uint j, Data_Type & data)
```

Definition at line 104 of file barry.hpp.

## 8.22 include/barry/cell-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



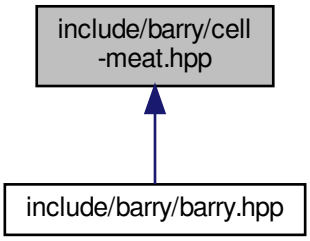
### Classes

- class Cell< Cell_Type >

  *Entries* in *BArray*. *For now, it only has two members:*

## 8.23 include/barry/cell-meat.hpp File Reference

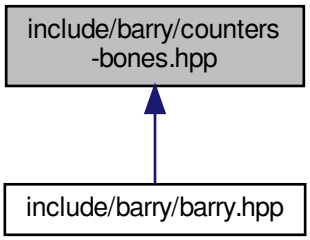
This graph shows which files directly or indirectly include this file:




## 8.24 include/barry/col-bones.hpp File Reference

## 8.25 include/barry/counters-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:




### Classes

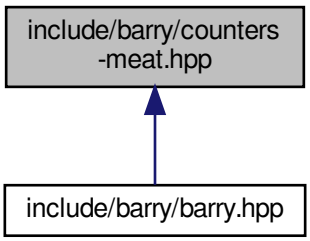
- class Counter< Array_Type, Data_Type >

  *A counter function based on change statistics.*
- class Counters< Array_Type, Data_Type >

  *Vector of counters.*

# 8.26 include/barry/counters-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:




## Macros

- #define COUNTER_TYPE() Counter<Array_Type,Data_Type>
- #define COUNTER_TEMPLATE_ARGS() <typename Array_Type, typename Data_Type>
- #define COUNTER_TEMPLATE(a, b) template COUNTER_TEMPLATE_ARGS() inline a COUNTER_TYPE()↩ ::b
- #define 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_ 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 Counters (Counters< Array_Type, Data_Type > &&counters_) noexcept
- COUNTERS_TEMPLATE (COUNTERS_TYPE(), operator=)(const Counters< Array_Type
- COUNTERS_TEMPLATE (COUNTERS_TYPE() &, operator=)(Counters< Array_Type
- COUNTERS_TEMPLATE (void, add_counter)(Counter< Array_Type
- COUNTERS_TEMPLATE (std::vector< std::string >, get_names)() const
- COUNTERS_TEMPLATE (std::vector< std::string >, get_descriptions)() const

## Variables

- Data_Type & counter_
- Data_Type &&counter_ noexcept
- uint i
- uint uint j
- return ∗ this
- 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 std::string name_
- Data_Type Counter_fun_type< Array_Type, Data_Type > Data_Type std::string std::string desc_

### 8.26.1 Macro Definition Documentation

#### 8.26.1.1 COUNTER_TEMPLATE

```
#define COUNTER_TEMPLATE(
              a,
              b )  template COUNTER_TEMPLATE_ARGS() inline a COUNTER_TYPE()::b
```

Definition at line 8 of file counters-meat.hpp.

#### 8.26.1.2 COUNTER_TEMPLATE_ARGS

```
#define COUNTER_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

Definition at line 6 of file counters-meat.hpp.

#### 8.26.1.3 COUNTER_TYPE

```
#define COUNTER_TYPE( ) Counter<Array_Type,Data_Type>
```

Definition at line 4 of file counters-meat.hpp.

#### 8.26.1.4 COUNTERS_TEMPLATE

```
#define COUNTERS_TEMPLATE(
              a,
              b )  template COUNTERS_TEMPLATE_ARGS() inline a COUNTERS_TYPE()::b
```

Definition at line 116 of file counters-meat.hpp.

#### 8.26.1.5 COUNTERS_TEMPLATE_ARGS

```
#define COUNTERS_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

Definition at line 114 of file counters-meat.hpp.

#### 8.26.1.6 COUNTERS_TYPE

```
#define COUNTERS_TYPE( ) Counters<Array_Type,Data_Type>
```

Definition at line 112 of file counters-meat.hpp.

### 8.26.2 Function Documentation

#### 8.26.2.1 count_fun()

```
return count_fun (
             Array ,
             i ,
             j ,
             data  )
```

#### 8.26.2.2 COUNTER_TEMPLATE() [1/7]

```
COUNTER_TEMPLATE (
             Counter  ) const
```

#### 8.26.2.3 COUNTER_TEMPLATE() [2/7]

```
COUNTER_TEMPLATE (
            COUNTER_TYPE() & ,
            operator  )
```

#### 8.26.2.4 COUNTER_TEMPLATE() [3/7]

```
COUNTER_TEMPLATE (
            COUNTER_TYPE() ,
            operator  ) const
```

#### 8.26.2.5 COUNTER_TEMPLATE() [4/7]

```
COUNTER_TEMPLATE (
            double ,
            count  ) &
```

< Move assignment

#### 8.26.2.6 COUNTER_TEMPLATE() [5/7]

```
COUNTER_TEMPLATE (
            double ,
            init  ) &
```

#### 8.26.2.7 COUNTER_TEMPLATE() [6/7]

```
COUNTER_TEMPLATE (
            std::string ,
            get_description  ) const
```

Definition at line 104 of file counters-meat.hpp.

#### 8.26.2.8 COUNTER_TEMPLATE() [7/7]

```
COUNTER_TEMPLATE (
            std::string ,
            get_name  ) const
```

Definition at line 100 of file counters-meat.hpp.

#### 8.26.2.9 Counters()

```
Data_Type Counters (
            Counters< Array_Type, Data_Type > && counters_ )  [noexcept]
```

Definition at line 130 of file counters-meat.hpp.

#### 8.26.2.10 COUNTERS_TEMPLATE() [1/7]

```
COUNTERS_TEMPLATE (
            Counters  )
```

Definition at line 119 of file counters-meat.hpp.

#### 8.26.2.11 COUNTERS_TEMPLATE() [2/7]

```
COUNTERS_TEMPLATE (
            COUNTER_TYPE() & ,
            operator [] )
```

Definition at line 121 of file counters-meat.hpp.

#### 8.26.2.12 COUNTERS_TEMPLATE() [3/7]

```
COUNTERS_TEMPLATE (
            COUNTERS_TYPE() & ,
            operator  )
```

#### 8.26.2.13 COUNTERS_TEMPLATE() [4/7]

```
COUNTERS_TEMPLATE (
            COUNTERS_TYPE() ,
            operator  ) const
```

#### 8.26.2.14 COUNTERS_TEMPLATE() [5/7]

```
COUNTERS_TEMPLATE (
            std::vector< std::string > ,
            get_descriptions  ) const
```

Definition at line 192 of file counters-meat.hpp.

#### 8.26.2.15 COUNTERS_TEMPLATE() [6/7]

```
COUNTERS_TEMPLATE (
            std::vector< std::string > ,
            get_names  ) const
```

Definition at line 181 of file counters-meat.hpp.

#### 8.26.2.16 COUNTERS_TEMPLATE() [7/7]

```
COUNTERS_TEMPLATE (
            void ,
            add_counter  )
```

#### 8.26.2.17 data()

```
Data_Type&& counter_ data (
            std::move(counter_.data)  )
```

#### 8.26.2.18 desc()

```
Data_Type&& counter_ desc (
            std::move(counter_.desc)  )
```

Move constructor.

Definition at line 31 of file counters-meat.hpp.

#### 8.26.2.19 init_fun() [1/3]

```
return init_fun (
            Array ,
            i ,
            j ,
            data  )
```

#### 8.26.2.20 init_fun() [2/3]

```
Data_Type init_fun (
             counter_.  init_fun )
```

Definition at line 13 of file counters-meat.hpp.

#### 8.26.2.21 init_fun() [3/3]

```
Data_Type&& counter_ init_fun (
             std::move(counter_.init_fun)  )
```

#### 8.26.2.22 name()

```
Data_Type&& counter_ name (
             std::move(counter_.name)  )
```

### 8.26.3 Variable Documentation

#### 8.26.3.1 count_fun_

```
Data_Type count_fun_
```

Definition at line 161 of file counters-meat.hpp.

#### 8.26.3.2 counter

```
Data_Type counter
```

**Initial value:**

```
{
    data.push_back(counter)
```

Definition at line 152 of file counters-meat.hpp.

#### 8.26.3.3 counter_

```
Data_Type & counter_
```

**Initial value:**

```
{
    if (this != &counter_) {
        this->count_fun = counter_.count_fun;
        this->init_fun = counter_.init_fun;

        this->data = counter_.data;
        this->name = counter_.name;
        this->desc = counter_.desc;
    }
    return *this
```

Definition at line 12 of file counters-meat.hpp.

#### 8.26.3.4 data_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> Data_Type data_
```

Definition at line 163 of file counters-meat.hpp.

#### 8.26.3.5 desc_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> Data_Type std::string std::string desc_
```

**Initial value:**

```
{

    data.push_back(Counter<Array_Type,Data_Type>(
        count_fun_,
        init_fun_,
        data_,
        name_,
        desc_
    ))
```

Definition at line 165 of file counters-meat.hpp.

#### 8.26.3.6 i

```
uint i
```

Definition at line 80 of file counters-meat.hpp.

#### 8.26.3.7 init_fun_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> init_fun_
```

Definition at line 162 of file counters-meat.hpp.

#### 8.26.3.8 j

```
uint uint j
```

**Initial value:**

```
{
    if (count_fun == nullptr)
        return 0.0
```

Definition at line 80 of file counters-meat.hpp.

#### 8.26.3.9 name_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> Data_Type std::string name_
```

Definition at line 164 of file counters-meat.hpp.

#### 8.26.3.10 noexcept

```
Data_Type &&counters_ noexcept
```

**Initial value:**

```
{
    if (this != &counter_)
    {
        this->data = std::move(counter_.data);

        this->count_fun = std::move(counter_.count_fun);
        this->init_fun = std::move(counter_.init_fun);

        this->name = std::move(counter_.name);
        this->desc = std::move(counter_.desc);
    }
    return *this
```

Definition at line 26 of file counters-meat.hpp.

#### 8.26.3.11 return

```
return
```

Definition at line 157 of file counters-meat.hpp.

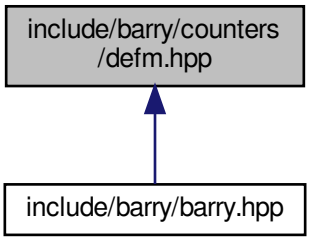
#### 8.26.3.12 this

```
return* this
```

Definition at line 138 of file counters-meat.hpp.

## 8.27 include/barry/counters/defm.hpp File Reference

This graph shows which files directly or indirectly include this file:




### Classes

- class DEFMData
- class DEFMCounterData

  *Data class used to store arbitrary uint or double vectors.*
- class DEFMRuleData

### Macros

#### Macros for defining counters

- #define DEFM_COUNTER(a) inline double (a) (const DEFMArray & Array, uint i, uint j, DEFMCounterData & data)
- #define DEFM_COUNTER_LAMBDA(a)

#### Macros for defining rules

- #define DEFM_RULE(a) inline bool (a) (const DEFMArray & Array, uint i, uint j, bool & data)
- #define DEFM_RULE_LAMBDA(a)

## Typedefs

- typedef BArrayDense< int, DEFMData > DEFMArray

**Convenient typedefs for network objects.**

- typedef Counter< DEFMArray, DEFMCounterData > DEFMCounter
- typedef Counters< DEFMArray, DEFMCounterData > DEFMCounters
- typedef Support< DEFMArray, DEFMCounterData, DEFMRuleData > DEFMSupport
- typedef StatsCounter< DEFMArray, DEFMCounterData > DEFMStatsCounter
- typedef Model< DEFMArray, DEFMCounterData, DEFMRuleData, DEFMRuleData > DEFMModel
- typedef Rule< DEFMArray, DEFMRuleData > DEFMRule
- typedef Rules< DEFMArray, DEFMRuleData > DEFMRules

## Functions

- void counter_ones (DEFMCounters *counters, int covar_index=-1)

  *Prevalence of ones.*
- void counter_transition (DEFMCounters *counters, std::vector< size_t > coords, int covar_index=-1)

  *Prevalence of ones.*
- void counter_fixed_effect (DEFMCounters *counters, int covar_index, double k)

  *Prevalence of ones.*

**Returns true if the cell is free**

***Parameters***

| | |
|---|---|
| rules | *A pointer to a* `DEFMRules` *object (*`Rules<DEFMArray, bool>`*).* |

- void rules_markov_fixed (DEFMRules *rules, size_t markov_order)

  *Number of edges.*

### 8.27.1 Macro Definition Documentation

#### 8.27.1.1 DEFM_COUNTER

```
#define DEFM_COUNTER(
              a ) inline double (a) (const DEFMArray & Array, uint i, uint j, DEFMCounterData
& data)
```

Function for definition of a network counter function

Definition at line 150 of file defm.hpp.

#### 8.27.1.2 DEFM_COUNTER_LAMBDA

```
#define DEFM_COUNTER_LAMBDA(
              a )
```

**Value:**
```
Counter_fun_type<DEFMArray, DEFMCounterData> a = \
    [](const DEFMArray & Array, uint i, uint j, DEFMCounterData & data)
```

Lambda function for definition of a network counter function

Definition at line 154 of file defm.hpp.

#### 8.27.1.3 DEFM_RULE

```
#define DEFM_RULE(
              a ) inline bool (a) (const DEFMArray & Array, uint i, uint j, bool & data)
```

Function for definition of a network counter function

Definition at line 165 of file defm.hpp.

#### 8.27.1.4 DEFM_RULE_LAMBDA

```
#define DEFM_RULE_LAMBDA(
              a )
```

**Value:**
```
Rule_fun_type<DEFMArray, DEFMRuleData> a = \
[](const DEFMArray & Array, uint i, uint j, DEFMRuleData & data)
```

Lambda function for definition of a network counter function

Definition at line 169 of file defm.hpp.

### 8.27.2 Typedef Documentation

#### 8.27.2.1 DEFMArray

```
typedef BArrayDense<int, DEFMData> DEFMArray
```

Definition at line 23 of file defm.hpp.

#### 8.27.2.2 DEFMCounter

```
typedef Counter<DEFMArray, DEFMCounterData > DEFMCounter
```

Definition at line 114 of file defm.hpp.

#### 8.27.2.3 DEFMCounters

```
typedef Counters<DEFMArray, DEFMCounterData> DEFMCounters
```

Definition at line 115 of file defm.hpp.

#### 8.27.2.4 DEFMModel

```
typedef Model<DEFMArray, DEFMCounterData,DEFMRuleData,DEFMRuleData> DEFMModel
```

Definition at line 118 of file defm.hpp.

#### 8.27.2.5 DEFMRule

```
typedef Rule<DEFMArray, DEFMRuleData> DEFMRule
```

Definition at line 119 of file defm.hpp.

#### 8.27.2.6 DEFMRules

```
typedef Rules<DEFMArray, DEFMRuleData> DEFMRules
```

Definition at line 120 of file defm.hpp.

#### 8.27.2.7 DEFMStatsCounter

```
typedef StatsCounter<DEFMArray, DEFMCounterData> DEFMStatsCounter
```

Definition at line 117 of file defm.hpp.

#### 8.27.2.8 DEFMSupport

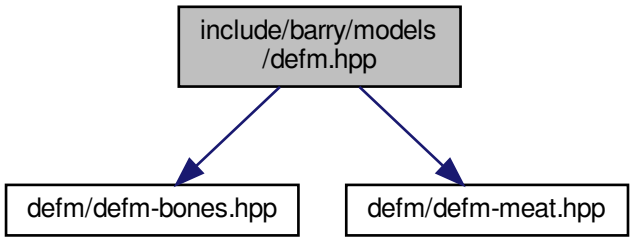
```
typedef Support<DEFMArray, DEFMCounterData, DEFMRuleData> DEFMSupport
```

Definition at line 116 of file defm.hpp.

## 8.28 include/barry/models/defm.hpp File Reference

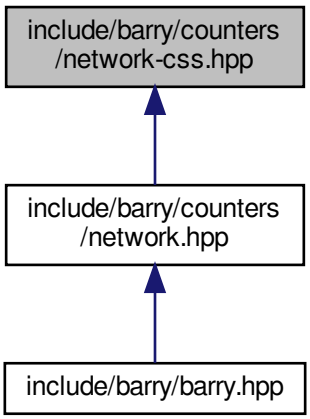
```
#include "defm/defm-bones.hpp"
#include "defm/defm-meat.hpp"
```

Include dependency graph for defm.hpp:




## 8.29 include/barry/counters/network-css.hpp File Reference

This graph shows which files directly or indirectly include this file:



## Macros

- #define CSS_SIZE()
- #define CSS_CASE_TRUTH() if ((i < n) && (j < n))
- #define CSS_TRUE_CELLS()
- #define CSS_CASE_PERCEIVED() else if (((i >= s) && (i < e)) & ((j >= s) && (j < e)))
- #define CSS_PERCEIVED_CELLS()
- #define CSS_CASE_ELSE()
- #define CSS_CHECK_SIZE_INIT()
- #define CSS_CHECK_SIZE()
- #define CSS_APPEND(name)
- #define CSS_NET_COUNTER_LAMBDA_INIT()

## Functions

- template<typename Tnet = Network>
  void counter_css_partially_false_recip_commi (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)

  *Counts errors of commission.*
- template<typename Tnet = Network>
  void counter_css_partially_false_recip_omiss (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)

  *Counts errors of omission.*
- template<typename Tnet = Network>
  void counter_css_completely_false_recip_comiss (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)

  *Counts completely false reciprocity (comission)*
- template<typename Tnet = Network>
  void counter_css_completely_false_recip_omiss (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)

  *Counts completely false reciprocity (omission)*
- template<typename Tnet = Network>
  void counter_css_mixed_recip (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)

  *Counts mixed reciprocity errors.*
- template<typename Tnet = Network>
  void counter_css_census01 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)
- template<typename Tnet = Network>
  void counter_css_census02 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)
- template<typename Tnet = Network>
  void counter_css_census03 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)
- template<typename Tnet = Network>
  void counter_css_census04 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)
- template<typename Tnet = Network>
  void counter_css_census05 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)
- template<typename Tnet = Network>
  void counter_css_census06 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)

- template<typename Tnet = Network>
  void counter_css_census07 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end↩
  _)
- template<typename Tnet = Network>
  void counter_css_census08 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end↩
  _)
- template<typename Tnet = Network>
  void counter_css_census09 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end↩
  _)
- template<typename Tnet = Network>
  void counter_css_census10 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end↩
  _)

## 8.29.1 Macro Definition Documentation

### 8.29.1.1 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,\
            NetCounterData({netsize, i == 0u ? netsize : end_[i-1], end_[i]}, {}),\
            tmpname);}
```

Definition at line 42 of file network-css.hpp.

### 8.29.1.2 CSS_CASE_ELSE

```
#define CSS_CASE_ELSE( )
```

Definition at line 27 of file network-css.hpp.

### 8.29.1.3 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.

#### 8.29.1.4 CSS_CASE_TRUTH

```
#define CSS_CASE_TRUTH( ) if ((i < n) && (j < n))
```

Definition at line 13 of file network-css.hpp.

#### 8.29.1.5 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.

#### 8.29.1.6 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.

#### 8.29.1.7 CSS_NET_COUNTER_LAMBDA_INIT

```
#define CSS_NET_COUNTER_LAMBDA_INIT( )
```

**Value:**

```
        NETWORK_COUNTER_LAMBDA(tmp_init) {\
        CSS_CHECK_SIZE_INIT() \
        return 0.0; \
    };
```

Definition at line 49 of file network-css.hpp.

#### 8.29.1.8 CSS_PERCEIVED_CELLS

```
#define CSS_PERCEIVED_CELLS( )
```

**Value:**

```
    double tji = static_cast<double>(Array(j - s, i - s, false)); \
    double pji = static_cast<double>(Array(j, i, false)); \
    double tij = static_cast<double>(Array(i - s, j - s, false));
```

Definition at line 21 of file network-css.hpp.

#### 8.29.1.9 CSS_SIZE

```
#define CSS_SIZE( )
```

**Value:**

```
    uint n = data.indices[0u]; \
    uint s = data.indices[1u]; \
    uint e = data.indices[2u];
```

Definition at line 7 of file network-css.hpp.

#### 8.29.1.10 CSS_TRUE_CELLS

```
#define CSS_TRUE_CELLS( )
```

**Value:**

```
    double tji = static_cast<double>(Array(j, i, false)); \
    double pij = static_cast<double>(Array(i + s, j + s, false)); \
    double pji = static_cast<double>(Array(j + s, i + s, false));
```

Definition at line 14 of file network-css.hpp.

### 8.29.2 Function Documentation

#### 8.29.2.1 counter_css_census01()

```
template<typename Tnet = Network>
void counter_css_census01 (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Definition at line 275 of file network-css.hpp.

#### 8.29.2.2 counter_css_census02()

```
template<typename Tnet = Network>
void counter_css_census02 (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Definition at line 325 of file network-css.hpp.

#### 8.29.2.3 counter_css_census03()

```
template<typename Tnet = Network>
void counter_css_census03 (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Definition at line 364 of file network-css.hpp.

#### 8.29.2.4 counter_css_census04()

```
template<typename Tnet = Network>
void counter_css_census04 (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Definition at line 403 of file network-css.hpp.

#### 8.29.2.5 counter_css_census05()

```
template<typename Tnet = Network>
void counter_css_census05 (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Definition at line 442 of file network-css.hpp.

#### 8.29.2.6 counter_css_census06()

```
template<typename Tnet = Network>
void counter_css_census06 (
             NetCounters< Tnet > * counters,
             uint netsize,
             const std::vector< uint > & end_ )  [inline]
```

Definition at line 481 of file network-css.hpp.

#### 8.29.2.7 counter_css_census07()

```
template<typename Tnet = Network>
void counter_css_census07 (
             NetCounters< Tnet > * counters,
             uint netsize,
             const std::vector< uint > & end_ )  [inline]
```

Definition at line 520 of file network-css.hpp.

#### 8.29.2.8 counter_css_census08()

```
template<typename Tnet = Network>
void counter_css_census08 (
             NetCounters< Tnet > * counters,
             uint netsize,
             const std::vector< uint > & end_ )  [inline]
```

Definition at line 559 of file network-css.hpp.

#### 8.29.2.9 counter_css_census09()

```
template<typename Tnet = Network>
void counter_css_census09 (
             NetCounters< Tnet > * counters,
             uint netsize,
             const std::vector< uint > & end_ )  [inline]
```

Definition at line 598 of file network-css.hpp.

#### 8.29.2.10 counter_css_census10()

```
template<typename Tnet = Network>
void counter_css_census10 (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Definition at line 637 of file network-css.hpp.

#### 8.29.2.11 counter_css_completely_false_recip_comiss()

```
template<typename Tnet = Network>
void counter_css_completely_false_recip_comiss (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Counts completely false reciprocity (comission)

Definition at line 154 of file network-css.hpp.

#### 8.29.2.12 counter_css_completely_false_recip_omiss()

```
template<typename Tnet = Network>
void counter_css_completely_false_recip_omiss (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Counts completely false reciprocity (omission)

Definition at line 194 of file network-css.hpp.

#### 8.29.2.13 counter_css_mixed_recip()

```
template<typename Tnet = Network>
void counter_css_mixed_recip (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Counts mixed reciprocity errors.

Definition at line 234 of file network-css.hpp.

#### 8.29.2.14 counter_css_partially_false_recip_commi()

```
template<typename Tnet = Network>
void counter_css_partially_false_recip_commi (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

Counts errors of commission.

**Parameters**

| | |
|---|---|
| *netsize* | Size of the reference (true) network |
| *end_* | Vector indicating one past the ending index of each network. (see details) |

The `end_` parameter should be of length `N of networks` - 1. It is assumed that the first network ends at `netsize`.

Definition at line 63 of file network-css.hpp.

#### 8.29.2.15 counter_css_partially_false_recip_omiss()

```
template<typename Tnet = Network>
void counter_css_partially_false_recip_omiss (
            NetCounters< Tnet > * counters,
            uint netsize,
            const std::vector< uint > & end_ )  [inline]
```

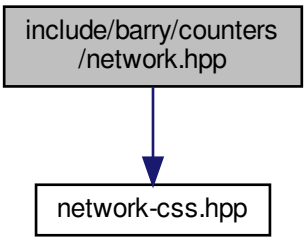
Counts errors of omission.

Definition at line 110 of file network-css.hpp.

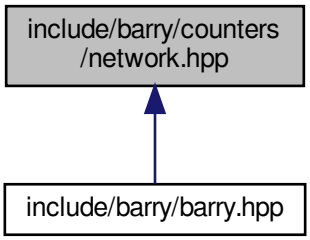
## 8.30 include/barry/counters/network.hpp File Reference

```
#include "network-css.hpp"
```

Include dependency graph for network.hpp:



This graph shows which files directly or indirectly include this file:




## Classes

- class NetworkData

  *Data class for Networks.*
- class NetCounterData

  *Data class used to store arbitrary uint or double vectors.*

## Macros

- #define NET_C_DATA_IDX(i) (data.indices[i])
- #define NET_C_DATA_NUM(i) (data.numbers[i])

### Macros for defining counters

- #define NETWORK_COUNTER(a)
- #define NETWORK_COUNTER_LAMBDA(a)
- #define NETWORKDENSE_COUNTER_LAMBDA(a)

### Macros for defining rules

- #define NETWORK_RULE(a)
- #define NETWORK_RULE_LAMBDA(a)

## Functions

- template<typename Tnet = Network>
  void counter_edges (NetCounters< Tnet > *counters)

  *Number of edges.*
- template<typename Tnet = Network>
  void counter_isolates (NetCounters< Tnet > *counters)

  *Number of isolated vertices.*
- template<> void counter_isolates (NetCounters< NetworkDense > *counters)
- template<typename Tnet = Network>
  void counter_mutual (NetCounters< Tnet > *counters)

  *Number of mutual ties.*
- template<typename Tnet = Network>
  void counter_istar2 (NetCounters< Tnet > *counters)
- template<> void counter_istar2 (NetCounters< NetworkDense > *counters)
- template<typename Tnet = Network>
  void counter_ostar2 (NetCounters< Tnet > *counters)
- template<> void counter_ostar2 (NetCounters< NetworkDense > *counters)
- template<typename Tnet = Network>
  void counter_ttriads (NetCounters< Tnet > *counters)
- template<> void counter_ttriads (NetCounters< NetworkDense > *counters)
- template<typename Tnet = Network>
  void counter_ctriads (NetCounters< Tnet > *counters)
- template<> void counter_ctriads (NetCounters< NetworkDense > *counters)
- template<typename Tnet = Network>
  void counter_density (NetCounters< Tnet > *counters)
- template<typename Tnet = Network>
  void counter_idegree15 (NetCounters< Tnet > *counters)
- template<> void counter_idegree15 (NetCounters< NetworkDense > *counters)
- template<typename Tnet = Network>
  void counter_odegree15 (NetCounters< Tnet > *counters)
- template<> void counter_odegree15 (NetCounters< NetworkDense > *counters)
- template<typename Tnet = Network>
  void counter_absdiff (NetCounters< Tnet > *counters, uint attr_id, double alpha=1.0)

  *Sum of absolute attribute difference between ego and alter.*
- template<typename Tnet = Network>
  void counter_diff (NetCounters< Tnet > *counters, uint attr_id, double alpha=1.0, double tail_head=true)

  *Sum of attribute difference between ego and alter to pow(alpha)*
- NETWORK_COUNTER (init_single_attr)
- template<typename Tnet = Network>
  void counter_nodeicov (NetCounters< Tnet > *counters, uint attr_id)
- template<typename Tnet = Network>
  void counter_nodeocov (NetCounters< Tnet > *counters, uint attr_id)
- template<typename Tnet = Network>
  void counter_nodecov (NetCounters< Tnet > *counters, uint attr_id)
- template<typename Tnet = Network>
  void counter_nodematch (NetCounters< Tnet > *counters, uint attr_id)
- template<typename Tnet = Network>
  void counter_idegree (NetCounters< Tnet > *counters, std::vector< uint > d)

  *Counts number of vertices with a given in-degree.*
- template<> void counter_idegree (NetCounters< NetworkDense > *counters, std::vector< uint > d)
- template<typename Tnet = Network>
  void counter_odegree (NetCounters< Tnet > *counters, std::vector< uint > d)

  *Counts number of vertices with a given out-degree.*
- template<> void counter_odegree (NetCounters< NetworkDense > *counters, std::vector< uint > d)

- template<typename Tnet = Network>
  void counter_degree (NetCounters< Tnet > *counters, std::vector< uint > d)

  *Counts number of vertices with a given out-degree.*

**Rules for network models**

***Parameters***

| | |
|---|---|
| rules | *A pointer to a* `NetRules` *object (*`Rules<Network, bool>`*).* |

- template<typename Tnet = Network>
  void rules_zerodiag (NetRules< Tnet > *rules)

  *Number of edges.*

## Convenient typedefs for network objects.

- #define BARRY_ZERO_NETWORK 0.0
- #define BARRY_ZERO_NETWORK_DENSE 0
- typedef BArray< double, NetworkData > Network
- typedef BArrayDense< int, NetworkData > NetworkDense
- template<typename Tnet = Network>
  using NetCounter = Counter< Tnet, NetCounterData >
- template<typename Tnet = Network>
  using NetCounters = Counters< Tnet, NetCounterData >
- template<typename Tnet = Network>
  using NetSupport = Support< Tnet, NetCounterData >
- template<typename Tnet = Network>
  using NetStatsCounter = StatsCounter< Tnet, NetCounterData >
- template<typename Tnet >
  using NetModel = Model< Tnet, NetCounterData >
- template<typename Tnet = Network>
  using NetRule = Rule< Tnet, bool >
- template<typename Tnet = Network>
  using NetRules = Rules< Tnet, bool >

### 8.30.1 Macro Definition Documentation

#### 8.30.1.1 BARRY_ZERO_NETWORK

```
#define BARRY_ZERO_NETWORK 0.0
```

Definition at line 85 of file network.hpp.

#### 8.30.1.2 BARRY_ZERO_NETWORK_DENSE

```
#define BARRY_ZERO_NETWORK_DENSE 0
```

Definition at line 86 of file network.hpp.

#### 8.30.1.3 NET_C_DATA_IDX

```
#define NET_C_DATA_IDX(
              i ) (data.indices[i])
```

Definition at line 74 of file network.hpp.

#### 8.30.1.4 NET_C_DATA_NUM

```
#define NET_C_DATA_NUM(
              i ) (data.numbers[i])
```

Definition at line 75 of file network.hpp.

#### 8.30.1.5 NETWORK_COUNTER

```
#define NETWORK_COUNTER(
              a )
```

**Value:**

```
template<typename Tnet = Network>\
inline double (a) (const Tnet & Array, uint i, uint j, NetCounterData & data)
```

Function for definition of a network counter function

Definition at line 114 of file network.hpp.

#### 8.30.1.6 NETWORK_COUNTER_LAMBDA

```
#define NETWORK_COUNTER_LAMBDA(
              a )
```

**Value:**

```
Counter_fun_type<Tnet, NetCounterData> a = \
    [](const Tnet & Array, uint i, uint j, NetCounterData & data)
```

Lambda function for definition of a network counter function

Definition at line 119 of file network.hpp.

#### 8.30.1.7 NETWORK_RULE

```
#define NETWORK_RULE(
                a )
```

**Value:**
```
template<typename Tnet = Network>\
inline bool (a) (const Tnet & Array, uint i, uint j, bool & data)
```

Function for definition of a network counter function

Definition at line 133 of file network.hpp.

#### 8.30.1.8 NETWORK_RULE_LAMBDA

```
#define NETWORK_RULE_LAMBDA(
                a )
```

**Value:**
```
Rule_fun_type<Tnet, bool> a = \
[](const Tnet & Array, uint i, uint j, bool & data)
```

Lambda function for definition of a network counter function

Definition at line 138 of file network.hpp.

#### 8.30.1.9 NETWORKDENSE_COUNTER_LAMBDA

```
#define NETWORKDENSE_COUNTER_LAMBDA(
                a )
```

**Value:**
```
Counter_fun_type<NetworkDense, NetCounterData> a = \
    [](const NetworkDense & Array, uint i, uint j, NetCounterData & data)
```

Definition at line 123 of file network.hpp.

### 8.30.2 Typedef Documentation

#### 8.30.2.1 NetCounter

```
template<typename Tnet = Network>
using NetCounter = Counter<Tnet, NetCounterData >
```

Definition at line 89 of file network.hpp.

#### 8.30.2.2 NetCounters

```
template<typename Tnet = Network>
using NetCounters = Counters<Tnet, NetCounterData>
```

Definition at line 92 of file network.hpp.

#### 8.30.2.3 NetModel

```
template<typename Tnet >
using NetModel = Model<Tnet, NetCounterData>
```

Definition at line 101 of file network.hpp.

#### 8.30.2.4 NetRule

```
template<typename Tnet = Network>
using NetRule = Rule<Tnet, bool>
```

Definition at line 104 of file network.hpp.

#### 8.30.2.5 NetRules

```
template<typename Tnet = Network>
using NetRules = Rules<Tnet, bool>
```

Definition at line 107 of file network.hpp.

#### 8.30.2.6 NetStatsCounter

```
template<typename Tnet = Network>
using NetStatsCounter = StatsCounter<Tnet, NetCounterData>
```

Definition at line 98 of file network.hpp.

#### 8.30.2.7 NetSupport

```
template<typename Tnet = Network>
using NetSupport = Support<Tnet, NetCounterData >
```

Definition at line 95 of file network.hpp.

#### 8.30.2.8 Network

```
typedef BArray<double, NetworkData> Network
```

Definition at line 82 of file network.hpp.

#### 8.30.2.9 NetworkDense

```
typedef BArrayDense<int, NetworkData> NetworkDense
```

Definition at line 83 of file network.hpp.

### 8.30.3 Function Documentation

#### 8.30.3.1 rules_zerodiag()

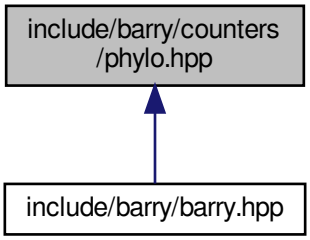
```
template<typename Tnet = Network>
void rules_zerodiag (
            NetRules< Tnet > * rules )  [inline]
```

Number of edges.

Definition at line 1383 of file network.hpp.

## 8.31 include/barry/counters/phylo.hpp File Reference

This graph shows which files directly or indirectly include this file:



## Classes

- class NodeData

  *Data definition for the `PhyloArray` class.*
- class PhyloCounterData
- class PhyloRuleDynData

## Macros

- #define DEFAULT_DUPLICATION 1u
- #define DUPL_SPEC 0u
- #define DUPL_DUPL 1u
- #define DUPL_EITH 2u
- #define MAKE_DUPL_VARS()
- #define IS_EITHER() (DATA_AT == DUPL_EITH)
- #define IS_DUPLICATION() ((DATA_AT == DUPL_DUPL) & (DPL))
- #define IS_SPECIATION() ((DATA_AT == DUPL_SPEC) & (!DPL))
- #define IF_MATCHES()
- #define IF_NOTMATCHES()
- #define PHYLO_COUNTER_LAMBDA(a)

  *Extension of a simple counter.*
- #define PHYLO_RULE_DYN_LAMBDA(a)
- #define PHYLO_CHECK_MISSING()

## Typedefs

- typedef std::vector< std::pair< uint, uint > > PhyloRuleData

**Convenient typedefs for Node objects.**

- typedef BArrayDense< uint, NodeData > PhyloArray
- typedef Counter< PhyloArray, PhyloCounterData > PhyloCounter
- typedef Counters< PhyloArray, PhyloCounterData > PhyloCounters
- typedef Rule< PhyloArray, PhyloRuleData > PhyloRule
- typedef Rules< PhyloArray, PhyloRuleData > PhyloRules
- typedef Rule< PhyloArray, PhyloRuleDynData > PhyloRuleDyn
- typedef Rules< PhyloArray, PhyloRuleDynData > PhyloRulesDyn
- typedef Support< PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport
- typedef StatsCounter< PhyloArray, PhyloCounterData > PhyloStatsCounter
- typedef Model< PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel
- typedef PowerSet< PhyloArray, PhyloRuleData > PhyloPowerSet

## Functions

- std::string get_last_name (unsigned int d)
- void counter_overall_gains (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  *Overall functional gains.*
- void counter_gains (PhyloCounters *counters, std::vector< uint > nfun, unsigned int duplication=DEFAULT_DUPLICATION)

  *Functional gains for a specific function (`nfun`).*
- void counter_gains_k_offspring (PhyloCounters *counters, std::vector< uint > nfun, uint k=1u, unsigned int duplication=DEFAULT_DUPLICATION)

  *k genes gain function nfun*
- void counter_genes_changing (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  *Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)*
- void counter_preserve_pseudogene (PhyloCounters *counters, unsigned int nfunA, unsigned int nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  *Keeps track of how many pairs of genes preserve pseudostate.*
- void counter_prop_genes_changing (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  *Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)*
- void counter_overall_loss (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  *Overall functional loss.*
- void counter_maxfuns (PhyloCounters *counters, uint lb, uint ub, unsigned int duplication=DEFAULT_DUPLICATION)

  *Cap the number of functions per gene.*
- void counter_loss (PhyloCounters *counters, std::vector< uint > nfun, unsigned int duplication=DEFAULT_DUPLICATION)

  *Total count of losses for an specific function.*
- void counter_overall_changes (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  *Total number of changes. Use this statistic to account for "preservation".*
- void counter_subfun (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  *Total count of Sub-functionalization events.*
- void counter_cogain (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  *Co-evolution (joint gain or loss)*
- void counter_longest (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  *Longest branch mutates (either by gain or by loss)*
- void counter_neofun (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  *Total number of neofunctionalization events.*
- void counter_pairwise_neofun_singlefun (PhyloCounters *counters, uint nfunA, unsigned int duplication=DEFAULT_DUPLICATI

  *Total number of neofunctionalization events sum_u sum_{w < u} [x(u,a)*(1 - x(w,a)) + (1 - x(u,a)) * x(w,a)] change stat: delta{x(u,a): 0->1} = 1 - 2 * x(w,a)*
- void counter_neofun_a2b (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  *Total number of neofunctionalization events.*
- void counter_co_opt (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  *Function co-opting.*
- void counter_k_genes_changing (PhyloCounters *counters, unsigned int k, unsigned int duplication=DEFAULT_DUPLICATION)

  *Indicator function. Equals to one if $k$ genes changed and zero otherwise.*
- void counter_less_than_p_prop_genes_changing (PhyloCounters *counters, double p, unsigned int duplication=DEFAULT_DUPLICATION)

  *Indicator function. Equals to one if $k$ genes changed and zero otherwise.*
- void counter_gains_from_0 (PhyloCounters *counters, std::vector< uint > nfun, unsigned int duplication=DEFAULT_DUPLICAT

  *Used when all the functions are in 0 (like the root node prob.)*
- void counter_overall_gains_from_0 (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  *Used when all the functions are in 0 (like the root node prob.)*
- void counter_pairwise_overall_change (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)

  *Used when all the functions are in 0 (like the root node prob.)*

- void counter_pairwise_preserving (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  *Used when all the functions are in 0 (like the root node prob.)*
- void counter_pairwise_first_gain (PhyloCounters *counters, uint nfunA, uint nfunB, unsigned int duplication=DEFAULT_DUPLICATION)

  *Used when all the functions are in 0 (like the root node prob.)*
- void rule_dyn_limit_changes (PhyloSupport *support, uint pos, uint lb, uint ub, unsigned int duplication=DEFAULT_DUPLICATIO

  *Overall functional gains.*

## 8.31.1 Macro Definition Documentation

### 8.31.1.1 DEFAULT_DUPLICATION

```
#define DEFAULT_DUPLICATION 1u
```

Definition at line 5 of file phylo.hpp.

### 8.31.1.2 DUPL_DUPL

```
#define DUPL_DUPL 1u
```

Definition at line 7 of file phylo.hpp.

### 8.31.1.3 DUPL_EITH

```
#define DUPL_EITH 2u
```

Definition at line 8 of file phylo.hpp.

### 8.31.1.4 DUPL_SPEC

```
#define DUPL_SPEC 0u
```

Definition at line 6 of file phylo.hpp.

#### 8.31.1.5 IF_MATCHES

```
#define IF_MATCHES( )
```

**Value:**

```
    MAKE_DUPL_VARS() \
    if (IS_EITHER() | IS_DUPLICATION() | IS_SPECIATION())
```

Definition at line 19 of file phylo.hpp.

#### 8.31.1.6 IF_NOTMATCHES

```
#define IF_NOTMATCHES( )
```

**Value:**

```
    MAKE_DUPL_VARS() \
    if (!IS_EITHER() & !IS_DUPLICATION() & !IS_SPECIATION())
```

Definition at line 21 of file phylo.hpp.

#### 8.31.1.7 IS_DUPLICATION

```
#define IS_DUPLICATION( ) ((DATA_AT == DUPL_DUPL) & (DPL))
```

Definition at line 16 of file phylo.hpp.

#### 8.31.1.8 IS_EITHER

```
#define IS_EITHER( ) (DATA_AT == DUPL_EITH)
```

Definition at line 15 of file phylo.hpp.

#### 8.31.1.9 IS_SPECIATION

```
#define IS_SPECIATION( ) ((DATA_AT == DUPL_SPEC) & (!DPL))
```

Definition at line 17 of file phylo.hpp.

#### 8.31.1.10 MAKE_DUPL_VARS

```
#define MAKE_DUPL_VARS( )
```

**Value:**

```
    bool DPL = Array.D_ptr()->duplication; \
    unsigned int DATA_AT = data[0u];
```

Definition at line 11 of file phylo.hpp.

#### 8.31.1.11 PHYLO_CHECK_MISSING

```
#define PHYLO_CHECK_MISSING( )
```

**Value:**

```
    if (Array.D_ptr() == nullptr) \
    throw std::logic_error("The array data is nullptr."); \
```

Definition at line 139 of file phylo.hpp.

#### 8.31.1.12 PHYLO_COUNTER_LAMBDA

```
#define PHYLO_COUNTER_LAMBDA(
                a )
```

**Value:**

```
    Counter_fun_type<PhyloArray, PhyloCounterData> a = \
    [](const PhyloArray & Array, uint i, uint j, PhyloCounterData & data)
```

Extension of a simple counter.

It allows specifying extra arguments, in particular, the corresponding sets of rows to which this statistic may be relevant. This could be important in the case of, for example, counting correlation type statistics between function 1 and 2, and between function 1 and 3.

Definition at line 133 of file phylo.hpp.

#### 8.31.1.13 PHYLO_RULE_DYN_LAMBDA

```
#define PHYLO_RULE_DYN_LAMBDA(
                a )
```

**Value:**

```
    Rule_fun_type<PhyloArray, PhyloRuleDynData> a = \
    [](const PhyloArray & Array, uint i, uint j, PhyloRuleDynData & data)
```

Definition at line 136 of file phylo.hpp.

## 8.31.2 Typedef Documentation

### 8.31.2.1 PhyloArray

```
typedef BArrayDense<uint, NodeData> PhyloArray
```

Definition at line 106 of file phylo.hpp.

### 8.31.2.2 PhyloCounter

```
typedef Counter<PhyloArray, PhyloCounterData > PhyloCounter
```

Definition at line 107 of file phylo.hpp.

### 8.31.2.3 PhyloCounters

```
typedef Counters< PhyloArray, PhyloCounterData> PhyloCounters
```

Definition at line 108 of file phylo.hpp.

### 8.31.2.4 PhyloModel

```
typedef Model<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel
```

Definition at line 118 of file phylo.hpp.

### 8.31.2.5 PhyloPowerSet

```
typedef PowerSet<PhyloArray, PhyloRuleData> PhyloPowerSet
```

Definition at line 119 of file phylo.hpp.

#### 8.31.2.6 PhyloRule

```
typedef Rule<PhyloArray,PhyloRuleData> PhyloRule
```

Definition at line 110 of file phylo.hpp.

#### 8.31.2.7 PhyloRuleData

```
typedef std::vector< std::pair< uint, uint > > PhyloRuleData
```

Definition at line 99 of file phylo.hpp.

#### 8.31.2.8 PhyloRuleDyn

```
typedef Rule<PhyloArray,PhyloRuleDynData> PhyloRuleDyn
```

Definition at line 113 of file phylo.hpp.

#### 8.31.2.9 PhyloRules

```
typedef Rules<PhyloArray,PhyloRuleData> PhyloRules
```

Definition at line 111 of file phylo.hpp.

#### 8.31.2.10 PhyloRulesDyn

```
typedef Rules<PhyloArray,PhyloRuleDynData> PhyloRulesDyn
```

Definition at line 114 of file phylo.hpp.

#### 8.31.2.11 PhyloStatsCounter

```
typedef StatsCounter<PhyloArray, PhyloCounterData> PhyloStatsCounter
```

Definition at line 117 of file phylo.hpp.

#### 8.31.2.12 PhyloSupport

```
typedef Support<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport
```

Definition at line 116 of file phylo.hpp.

### 8.31.3 Function Documentation

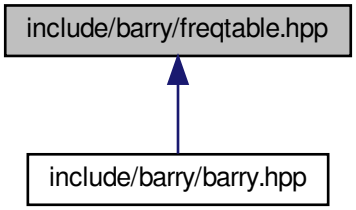
#### 8.31.3.1 get_last_name()

```
std::string get_last_name (
            unsigned int d )  [inline]
```

Definition at line 142 of file phylo.hpp.

## 8.32 include/barry/freqtable.hpp File Reference

This graph shows which files directly or indirectly include this file:



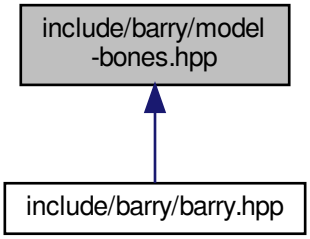

### Classes

- class FreqTable< T >

  *Frequency table of vectors.*

## 8.33 include/barry/model-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:




### Classes

- class Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >

  *General framework for discrete exponential models. This class allows generating discrete exponential models in the form of a linear exponential model:*

### Functions

- template<typename Array_Type >
  std::vector< double > keygen_default (const Array_Type &Array_)

  *Array Hasher class (used for computing support)*

### 8.33.1 Function Documentation

#### 8.33.1.1 keygen_default()

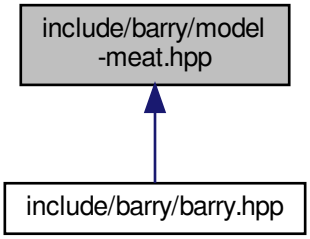
```
template<typename Array_Type >
std::vector< double > keygen_default (
            const Array_Type & Array_ )  [inline]
```

Array Hasher class (used for computing support)

Definition at line 9 of file model-bones.hpp.

# 8.34 include/barry/model-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:




## Macros

- #define MODEL_TYPE()
- #define MODEL_TEMPLATE_ARGS()
- #define MODEL_TEMPLATE(a, b) template MODEL_TEMPLATE_ARGS() inline a MODEL_TYPE()::b

## Functions

- double update_normalizing_constant (const double *params, const double *support, size_t k, size_t n)
- double likelihood_ (const double *stats_target, const std::vector< double > &params, const double normalizing_constant, size_t n_params, bool log_=false)
- MODEL_TEMPLATE (, Model)()
- MODEL_TEMPLATE (, Model)(const MODEL_TYPE() &Model_)

### 8.34.1 Macro Definition Documentation

#### 8.34.1.1 MODEL_TEMPLATE

```
#define MODEL_TEMPLATE(
              a,
              b )  template MODEL_TEMPLATE_ARGS() inline a MODEL_TYPE()::b
```

Definition at line 87 of file model-meat.hpp.

#### 8.34.1.2 MODEL_TEMPLATE_ARGS

```
#define MODEL_TEMPLATE_ARGS( )
```

**Value:**

```
    <typename Array_Type, typename Data_Counter_Type,\
    typename Data_Rule_Type, typename Data_Rule_Dyn_Type>
```

Definition at line 84 of file model-meat.hpp.

#### 8.34.1.3 MODEL_TYPE

```
#define MODEL_TYPE( )
```

**Value:**

```
    Model<Array_Type,Data_Counter_Type,Data_Rule_Type,\
    Data_Rule_Dyn_Type>
```

Definition at line 81 of file model-meat.hpp.

### 8.34.2 Function Documentation

#### 8.34.2.1 likelihood_()

```
double likelihood_ (
            const double * stats_target,
            const std::vector< double > & params,
            const double normalizing_constant,
            size_t n_params,
            bool log_ = false )  [inline]
```

Definition at line 45 of file model-meat.hpp.

#### 8.34.2.2 MODEL_TEMPLATE() [1/2]

```
MODEL_TEMPLATE (
            Model  )
```

Definition at line 91 of file model-meat.hpp.

#### 8.34.2.3 MODEL_TEMPLATE() [2/2]

```
MODEL_TEMPLATE (
            Model  ) const &
```

Definition at line 155 of file model-meat.hpp.

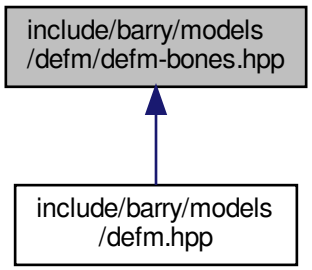
#### 8.34.2.4 update_normalizing_constant()

```
double update_normalizing_constant (
            const double * params,
            const double * support,
            size_t k,
            size_t n )  [inline]
```

Definition at line 9 of file model-meat.hpp.

## 8.35 include/barry/models/defm/defm-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:

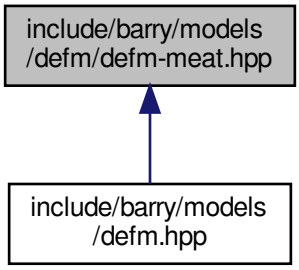



### Classes

- class DEFM

## 8.36 include/barry/models/defm/defm-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:




### Macros

- #define DEFM_RANGES(a)
- #define DEFM_LOOP_ARRAYS(a) for (size_t a = 0u; a < (nobs_i - M_order); ++a)

### Functions

- std::vector< double > keygen_defm (const defmcounters::DEFMArray &Array_)

### 8.36.1 Macro Definition Documentation

#### 8.36.1.1 DEFM_LOOP_ARRAYS

```
#define DEFM_LOOP_ARRAYS(
              a )  for (size_t a = 0u; a < (nobs_i - M_order); ++a)
```

Definition at line 41 of file defm-meat.hpp.

#### 8.36.1.2 DEFM_RANGES

```
#define DEFM_RANGES(
              a )
```

**Value:**

```
    size_t __CONCAT(start_,a) = start_end[a * 2u];\
    size_t __CONCAT(end_,a)   = start_end[a * 2u + 1u];\
    size_t __CONCAT(nobs_,a)  = __CONCAT(end_,i) - __CONCAT(start_,i) + 1u;
```

Definition at line 36 of file defm-meat.hpp.

### 8.36.2 Function Documentation

#### 8.36.2.1 keygen_defm()

```
std::vector< double > keygen_defm (
            const defmcounters::DEFMArray & Array_ )  [inline]
```

Definition at line 4 of file defm-meat.hpp.

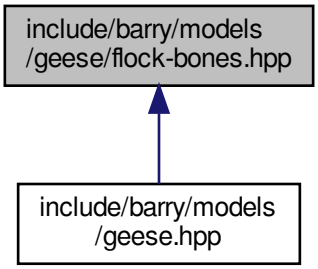
## 8.37 include/barry/models/geese.hpp File Reference

```
#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:

## 8.38 include/barry/models/geese/flock-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



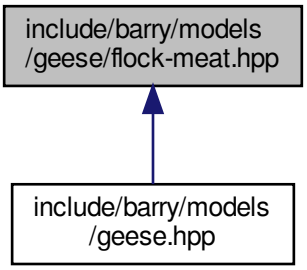
### Classes

- class Flock

  *A Flock is a group of Geese.*

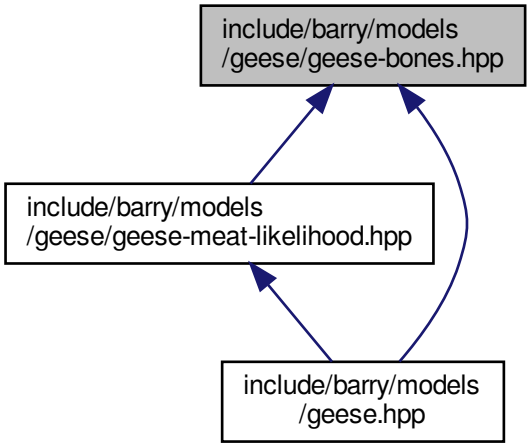
## 8.39 include/barry/models/geese/flock-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:




## 8.40 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)

### 8.40.1 Macro Definition Documentation

#### 8.40.1.1 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.

### 8.40.2 Function Documentation

#### 8.40.2.1 keygen_full()

```
std::vector< double > keygen_full (
            const phylocounters::PhyloArray & array )  [inline]
```

Definition at line 35 of file geese-bones.hpp.

#### 8.40.2.2 RULE_FUNCTION()

```
RULE_FUNCTION (
             rule_empty_free  )
```

Definition at line 26 of file geese-bones.hpp.

#### 8.40.2.3 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.

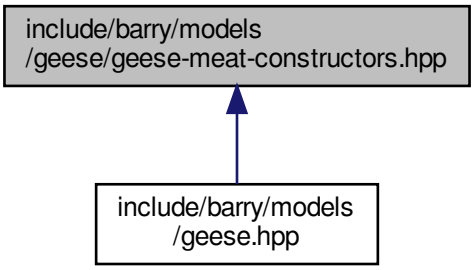
#### 8.40.2.4 vector_caster()

```
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.

## 8.41 include/barry/models/geese/geese-meat-constructors.hpp File Reference

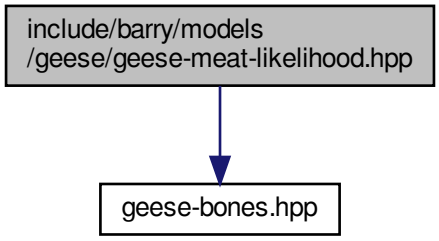
This graph shows which files directly or indirectly include this file:



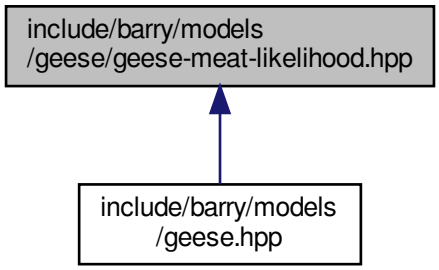
## 8.42 include/barry/models/geese/geese-meat-likelihood.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat-likelihood.hpp:

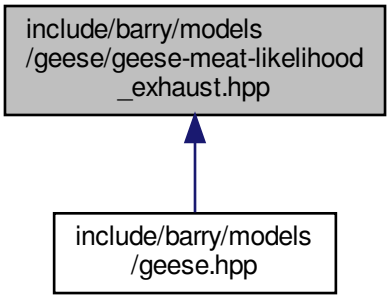



This graph shows which files directly or indirectly include this file:



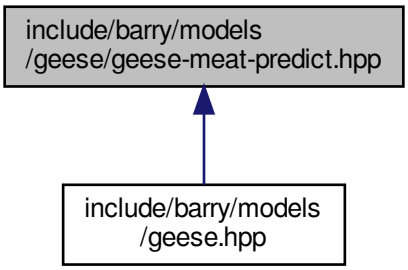
## 8.43 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference

This graph shows which files directly or indirectly include this file:



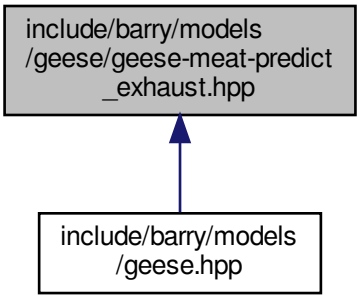

## 8.44 include/barry/models/geese/geese-meat-predict.hpp File Reference

This graph shows which files directly or indirectly include this file:



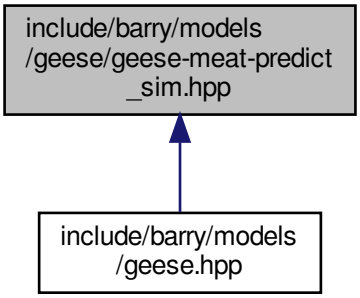
## 8.45 include/barry/models/geese/geese-meat-predict_exhaust.hpp File Reference

This graph shows which files directly or indirectly include this file:



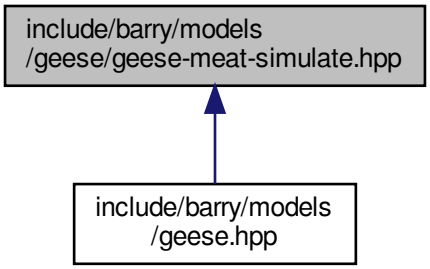

## 8.46 include/barry/models/geese/geese-meat-predict_sim.hpp File Reference

This graph shows which files directly or indirectly include this file:



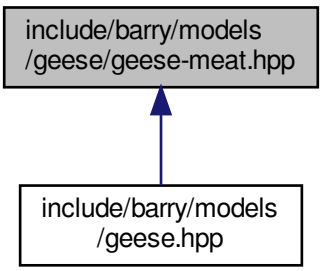
## 8.47 include/barry/models/geese/geese-meat-simulate.hpp File Reference

This graph shows which files directly or indirectly include this file:



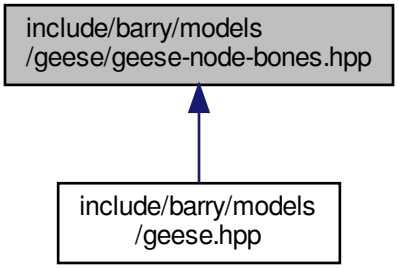

## 8.48 include/barry/models/geese/geese-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



## 8.49 include/barry/models/geese/geese-node-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



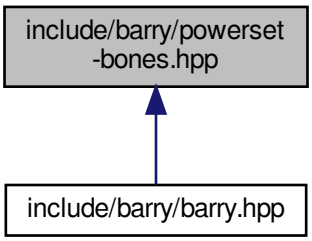

### Classes

- class Node

  *A single node for the model.*

## 8.50 include/barry/powerset-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



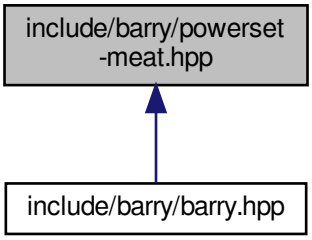

### Classes

- class PowerSet< Array_Type, Data_Rule_Type >

  *Powerset of a binary array.*

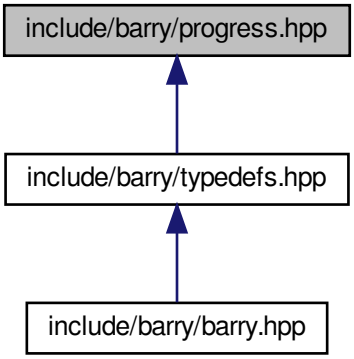
## 8.51 include/barry/powerset-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:




## 8.52 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

### 8.52.1 Macro Definition Documentation

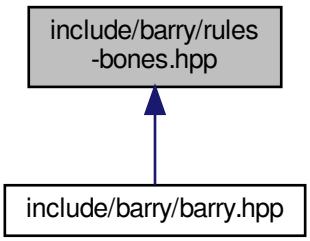
#### 8.52.1.1 BARRY_PROGRESS_BAR_WIDTH

```
#define BARRY_PROGRESS_BAR_WIDTH 80
```

Definition at line 5 of file progress.hpp.

## 8.53 include/barry/rules-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:




### Classes

- class Rule< Array_Type, Data_Type >

  *Rule for determining if a cell should be included in a sequence.*
- class Rules< Array_Type, Data_Type >

  *Vector of objects of class Rule.*

### Functions

- template<typename Array_Type , typename Data_Type >
  bool rule_fun_default (const Array_Type *array, uint i, uint j, Data_Type *dat)

### 8.53.1 Function Documentation

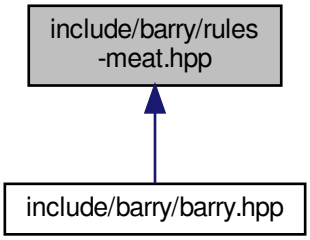
#### 8.53.1.1 rule_fun_default()

```
template<typename Array_Type , typename Data_Type >
bool rule_fun_default (
             const Array_Type * array,
             uint i,
             uint j,
             Data_Type * dat )
```

Definition at line 5 of file rules-bones.hpp.

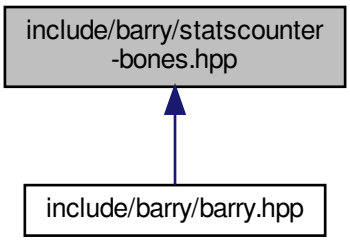
## 8.54 include/barry/rules-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:




## 8.55 include/barry/statscounter-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



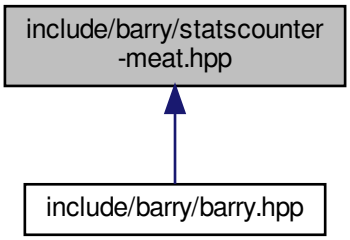
### Classes

- class StatsCounter< Array_Type, Data_Type >

  *Count stats for a single Array.*

## 8.56 include/barry/statscounter-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:




### Macros

- #define STATSCOUNTER_TYPE() StatsCounter<Array_Type,Data_Type>
- #define STATSCOUNTER_TEMPLATE_ARGS() <typename Array_Type, typename Data_Type>
- #define STATSCOUNTER_TEMPLATE(a, b) template STATSCOUNTER_TEMPLATE_ARGS() inline a STATSCOUNTER_TYPE()::b

### Functions

- STATSCOUNTER_TEMPLATE (, StatsCounter)(const StatsCounter< Array_Type
- EmptyArray clear ()
- STATSCOUNTER_TEMPLATE (,~StatsCounter)()
- STATSCOUNTER_TEMPLATE (void, reset_array)(const Array_Type *Array_)
- STATSCOUNTER_TEMPLATE (void, add_counter)(Counter< Array_Type
- STATSCOUNTER_TEMPLATE (void, set_counters)(Counters< Array_Type
- STATSCOUNTER_TEMPLATE (void, count_init)(uint i
- current_stats resize (counters->size(), 0.0)
- for (uint n=0u;n< counters->size();++n) current_stats[n]
- STATSCOUNTER_TEMPLATE (void, count_current)(uint i
- STATSCOUNTER_TEMPLATE (std::vector< std::string >, get_names)() const
- STATSCOUNTER_TEMPLATE (std::vector< std::string >, get_descriptions)() const

## Variables

- Data_Type & counter
- EmptyArray = ∗Array
- current_stats = counter.current_stats
- counters = new Counters<Array_Type,Data_Type>((∗counter.counters))
- counter_deleted = false
- Data_Type f_
- return
- Data_Type ∗ counters_
- uint j

### 8.56.1 Macro Definition Documentation

#### 8.56.1.1 STATSCOUNTER_TEMPLATE

```
#define STATSCOUNTER_TEMPLATE(
              a,
              b )  template STATSCOUNTER_TEMPLATE_ARGS() inline a STATSCOUNTER_TYPE()::b
```

Definition at line 8 of file statscounter-meat.hpp.

#### 8.56.1.2 STATSCOUNTER_TEMPLATE_ARGS

```
template STATSCOUNTER_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

Definition at line 6 of file statscounter-meat.hpp.

#### 8.56.1.3 STATSCOUNTER_TYPE

```
template Data_Type * STATSCOUNTER_TYPE( ) StatsCounter<Array_Type,Data_Type>
```

Definition at line 4 of file statscounter-meat.hpp.

### 8.56.2 Function Documentation

#### 8.56.2.1 clear()

```
EmptyArray clear ( )
```

#### 8.56.2.2 for()

```
for (
            uint n = 0u;n< counters->size();++n )
```

#### 8.56.2.3 resize()

```
current_stats resize (
            counters-> size(),
            0.  0 )
```

#### 8.56.2.4 STATSCOUNTER_TEMPLATE() [1/9]

```
STATSCOUNTER_TEMPLATE (
            StatsCounter  ) const
```

#### 8.56.2.5 STATSCOUNTER_TEMPLATE() [2/9]

```
STATSCOUNTER_TEMPLATE (
            ~ StatsCounter )
```

Definition at line 27 of file statscounter-meat.hpp.

#### 8.56.2.6 STATSCOUNTER_TEMPLATE() [3/9]

```
STATSCOUNTER_TEMPLATE (
            std::vector< std::string > ,
            get_descriptions  ) const
```

Definition at line 256 of file statscounter-meat.hpp.

#### 8.56.2.7 STATSCOUNTER_TEMPLATE() [4/9]

```
STATSCOUNTER_TEMPLATE (
            std::vector< std::string > ,
            get_names  ) const
```

Definition at line 251 of file statscounter-meat.hpp.

#### 8.56.2.8 STATSCOUNTER_TEMPLATE() [5/9]

```
STATSCOUNTER_TEMPLATE (
            void ,
            add_counter  )
```

#### 8.56.2.9 STATSCOUNTER_TEMPLATE() [6/9]

```
STATSCOUNTER_TEMPLATE (
            void ,
            count_current  )
```

#### 8.56.2.10 STATSCOUNTER_TEMPLATE() [7/9]

```
STATSCOUNTER_TEMPLATE (
            void ,
            count_init  )
```

#### 8.56.2.11 STATSCOUNTER_TEMPLATE() [8/9]

```
STATSCOUNTER_TEMPLATE (
            void ,
            reset_array  ) const
```

Definition at line 34 of file statscounter-meat.hpp.

#### 8.56.2.12 STATSCOUNTER_TEMPLATE() [9/9]

```
STATSCOUNTER_TEMPLATE (
            void ,
            set_counters  )
```

### 8.56.3 Variable Documentation

#### 8.56.3.1 counter

```
Data_Type& counter
```

**Initial value:**

```
{
    Array      = counter.Array
```

Definition at line 12 of file statscounter-meat.hpp.

#### 8.56.3.2 counter_deleted

```
counter_deleted = false
```

Definition at line 23 of file statscounter-meat.hpp.

#### 8.56.3.3 counters

```
counters = new Counters<Array_Type,Data_Type>((*counter.counters))
```

Definition at line 22 of file statscounter-meat.hpp.

#### 8.56.3.4 counters_

```
Data_Type* counters_
```

**Initial value:**

```
{

    if (!counter_deleted)
        delete counters
```

Definition at line 53 of file statscounter-meat.hpp.

#### 8.56.3.5 current_stats

```
current_stats = counter.current_stats
```

Definition at line 19 of file statscounter-meat.hpp.

#### 8.56.3.6 EmptyArray

```
EmptyArray = *Array
```

Definition at line 17 of file statscounter-meat.hpp.

#### 8.56.3.7 f_

```
Data_Rule_Dyn_Type f_
```

**Initial value:**

```
{

    counters->add_counter(f_)
```

Definition at line 44 of file statscounter-meat.hpp.

#### 8.56.3.8 j

```
uint j
```

**Initial value:**

```
{


    if (counters->size() == 0u)
        throw std::logic_error("No counters added: Cannot count without knowning what to count!")
```

Definition at line 66 of file statscounter-meat.hpp.

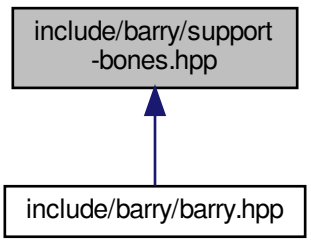
#### 8.56.3.9 return

```
return
```

Definition at line 49 of file statscounter-meat.hpp.

## 8.57 include/barry/support-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



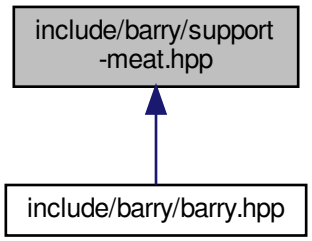
### Classes

- class Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >

  *Compute the support of sufficient statistics.*

## 8.58 include/barry/support-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:




### Macros

- #define BARRY_SUPPORT_MEAT_HPP 1
- #define SUPPORT_TEMPLATE_ARGS()
- #define SUPPORT_TYPE()
- #define SUPPORT_TEMPLATE(a, b)

### Functions

- SUPPORT_TEMPLATE (void, init_support)(std
- SUPPORT_TEMPLATE (void, reset_array)()
- SUPPORT_TEMPLATE (void, reset_array)(const Array_Type &Array_)
- SUPPORT_TEMPLATE (void, calc_backend_sparse)(uint pos
- calc_backend_sparse (pos+1u, array_bank, stats_bank)
- EmptyArray insert_cell (coord_i, coord_j, EmptyArray.default_val().value, false, false)
- for (uint n=0u;n< n_counters;++n)
- if (rules_dyn->size() > 0u)
- if (array_bank !=nullptr) array_bank -> push_back(EmptyArray)
- EmptyArray rm_cell (coord_i, coord_j, false, false)
- if (change_stats_different > 0u)
- SUPPORT_TEMPLATE (void, calc_backend_dense)(uint pos
- calc_backend_dense (pos+1u, array_bank, stats_bank)
- EmptyArray insert_cell (coord_i, coord_j, 1, false, false)
- SUPPORT_TEMPLATE (void, calc)(std
- SUPPORT_TEMPLATE (void, add_counter)(Counter< Array_Type
- SUPPORT_TEMPLATE (void, set_counters)(Counters< Array_Type

- SUPPORT_TEMPLATE (void, add_rule)(Rule< Array_Type
- SUPPORT_TEMPLATE (void, set_rules)(Rules< Array_Type
- SUPPORT_TEMPLATE (void, add_rule_dyn)(Rule< Array_Type
- SUPPORT_TEMPLATE (void, set_rules_dyn)(Rules< Array_Type
- SUPPORT_TEMPLATE (bool, eval_rules_dyn)(const std
- SUPPORT_TEMPLATE (std::vector< double >, get_counts)() const
- SUPPORT_TEMPLATE (std::vector< double > *, get_current_stats)()
- SUPPORT_TEMPLATE (void, print)() const
- SUPPORT_TEMPLATE (const FreqTable< double > &, get_data)() const

## Variables

- std::vector< Array_Type > * array_bank
- std::vector< Array_Type > std::vector< double > * stats_bank
- const size_t & coord_i = coordinates_free[pos * 2u]
- const size_t & coord_j = coordinates_free[pos * 2u + 1u]
- double tmp_chng
- unsigned int change_stats_different = hashes_initialized[pos] ? 0u : 1u
- else
- & hashes [pos]
- return
- Data_Counter_Type f_
- Data_Counter_Type * counters_
- delete_counters = false
- counters = counters_
- Data_Rule_Type * rules_
- delete_rules = false
- rules = rules_
- delete_rules_dyn = false
- rules_dyn = rules_

### 8.58.1 Macro Definition Documentation

#### 8.58.1.1 BARRY_SUPPORT_MEAT_HPP

```
#define BARRY_SUPPORT_MEAT_HPP 1
```

Definition at line 2 of file support-meat.hpp.

#### 8.58.1.2 SUPPORT_TEMPLATE

```
#define SUPPORT_TEMPLATE(
              a,
              b )
```

**Value:**

```
template SUPPORT_TEMPLATE_ARGS() \
inline a SUPPORT_TYPE()::b
```

Definition at line 10 of file support-meat.hpp.

#### 8.58.1.3 SUPPORT_TEMPLATE_ARGS

```
template SUPPORT_TEMPLATE_ARGS( )
```

**Value:**

```
    <typename Array_Type, typename \
    Data_Counter_Type, typename Data_Rule_Type, typename Data_Rule_Dyn_Type>
```

Definition at line 4 of file support-meat.hpp.

#### 8.58.1.4 SUPPORT_TYPE

```
template Data_Rule_Dyn_Type * SUPPORT_TYPE( )
```

**Value:**

```
    Support<Array_Type,Data_Counter_Type,Data_Rule_Type,\
    Data_Rule_Dyn_Type>
```

Definition at line 7 of file support-meat.hpp.

### 8.58.2 Function Documentation

#### 8.58.2.1 calc_backend_dense()

```
calc_backend_dense (
            pos+ 1u,
            array_bank ,
            stats_bank  )
```

#### 8.58.2.2 calc_backend_sparse()

```
calc_backend_sparse (
            pos+ 1u,
            array_bank ,
            stats_bank  )
```

#### 8.58.2.3 for()

```
for ( )
```

Definition at line 159 of file support-meat.hpp.

#### 8.58.2.4 if() [1/3]

```
if (
             array_bank !     = nullptr ) -> push_back(EmptyArray)
```

#### 8.58.2.5 if() [2/3]

```
if (
             change_stats_different ,
             0u  )
```

Definition at line 239 of file support-meat.hpp.

#### 8.58.2.6 if() [3/3]

```
if (
             rules_dyn-> size(),
             0u  )
```

Definition at line 187 of file support-meat.hpp.

#### 8.58.2.7 insert_cell() [1/2]

```
EmptyArray insert_cell (
             coord_i ,
             coord_j ,
             1 ,
             false ,
             false  )
```

#### 8.58.2.8 insert_cell() [2/2]

```
EmptyArray insert_cell (
             coord_i ,
             coord_j ,
             EmptyArray.default_val().  value,
             false ,
             false  )
```

#### 8.58.2.9 rm_cell()

```
EmptyArray rm_cell (
            coord_i ,
            coord_j ,
            false ,
            false  )
```

#### 8.58.2.10 SUPPORT_TEMPLATE() [1/17]

```
SUPPORT_TEMPLATE (
            bool ,
            eval_rules_dyn  ) const
```

Definition at line 488 of file support-meat.hpp.

#### 8.58.2.11 SUPPORT_TEMPLATE() [2/17]

```
SUPPORT_TEMPLATE (
            const FreqTable< double > & ,
            get_data  ) const
```

Definition at line 557 of file support-meat.hpp.

#### 8.58.2.12 SUPPORT_TEMPLATE() [3/17]

```
SUPPORT_TEMPLATE (
            std::vector< double > * ,
            get_current_stats  )
```

Definition at line 542 of file support-meat.hpp.

#### 8.58.2.13 SUPPORT_TEMPLATE() [4/17]

```
SUPPORT_TEMPLATE (
            std::vector< double > ,
            get_counts  ) const
```

Definition at line 530 of file support-meat.hpp.

#### 8.58.2.14 SUPPORT_TEMPLATE() [5/17]

```
SUPPORT_TEMPLATE (
            void ,
            add_counter  )
```

#### 8.58.2.15 SUPPORT_TEMPLATE() [6/17]

```
SUPPORT_TEMPLATE (
            void ,
            add_rule  )
```

#### 8.58.2.16 SUPPORT_TEMPLATE() [7/17]

```
SUPPORT_TEMPLATE (
            void ,
            add_rule_dyn  )
```

#### 8.58.2.17 SUPPORT_TEMPLATE() [8/17]

```
SUPPORT_TEMPLATE (
            void ,
            calc  )
```

Definition at line 371 of file support-meat.hpp.

#### 8.58.2.18 SUPPORT_TEMPLATE() [9/17]

```
SUPPORT_TEMPLATE (
            void ,
            calc_backend_dense  )
```

#### 8.58.2.19 SUPPORT_TEMPLATE() [10/17]

```
SUPPORT_TEMPLATE (
            void ,
            calc_backend_sparse  )
```

#### 8.58.2.20 SUPPORT_TEMPLATE() [11/17]

```
SUPPORT_TEMPLATE (
            void ,
            init_support  )
```

Definition at line 13 of file support-meat.hpp.

#### 8.58.2.21 SUPPORT_TEMPLATE() [12/17]

```
SUPPORT_TEMPLATE (
            void ,
            print  ) const
```

Definition at line 546 of file support-meat.hpp.

#### 8.58.2.22 SUPPORT_TEMPLATE() [13/17]

```
SUPPORT_TEMPLATE (
            void ,
            reset_array  )
```

Definition at line 114 of file support-meat.hpp.

#### 8.58.2.23 SUPPORT_TEMPLATE() [14/17]

```
SUPPORT_TEMPLATE (
            void ,
            reset_array  ) const &
```

Definition at line 120 of file support-meat.hpp.

#### 8.58.2.24 SUPPORT_TEMPLATE() [15/17]

```
SUPPORT_TEMPLATE (
            void ,
            set_counters  )
```

#### 8.58.2.25 SUPPORT_TEMPLATE() [16/17]

```
SUPPORT_TEMPLATE (
            void ,
            set_rules  )
```

#### 8.58.2.26 SUPPORT_TEMPLATE() [17/17]

```
SUPPORT_TEMPLATE (
            void ,
            set_rules_dyn  )
```

### 8.58.3 Variable Documentation

#### 8.58.3.1 array_bank

```
std::vector< Array_Type > * array_bank
```

Definition at line 131 of file support-meat.hpp.

#### 8.58.3.2 change_stats_different

```
unsigned int change_stats_different = hashes_initialized[pos] ?  0u :  1u
```

Definition at line 158 of file support-meat.hpp.

#### 8.58.3.3 coord_i

```
const size_t & coord_i = coordinates_free[pos * 2u]
```

Definition at line 144 of file support-meat.hpp.

#### 8.58.3.4 coord_j

```
const size_t & coord_j = coordinates_free[pos * 2u + 1u]
```

Definition at line 145 of file support-meat.hpp.

#### 8.58.3.5 counters

```
counters = counters_
```

Definition at line 416 of file support-meat.hpp.

#### 8.58.3.6 counters_

```
Data_Counter_Type* counters_
```

**Initial value:**

```
{

    if (delete_counters)
        delete counters
```

Definition at line 409 of file support-meat.hpp.

#### 8.58.3.7 delete_counters

```
delete_counters = false
```

Definition at line 415 of file support-meat.hpp.

#### 8.58.3.8 delete_rules

```
delete_rules = false
```

Definition at line 449 of file support-meat.hpp.

#### 8.58.3.9 delete_rules_dyn

```
delete_rules_dyn = false
```

Definition at line 481 of file support-meat.hpp.

#### 8.58.3.10 else

```
else (
              void  )
```

**Initial value:**

```
{
        if (change_stats_different > 0u)
            hashes[pos] = data.add(current_stats, nullptr)
```

Definition at line 212 of file support-meat.hpp.

#### 8.58.3.11 f_

```
Data_Rule_Dyn_Type f_
```

**Initial value:**

```
{

    counters->add_counter(f_)
```

Definition at line 400 of file support-meat.hpp.

#### 8.58.3.12 hashes

```
& hashes
```

Definition at line 217 of file support-meat.hpp.

#### 8.58.3.13 return

```
return
```

Definition at line 251 of file support-meat.hpp.

#### 8.58.3.14 rules

```
rules = rules_
```

Definition at line 450 of file support-meat.hpp.

#### 8.58.3.15 rules_

```
Data_Rule_Dyn_Type * rules_
```

**Initial value:**
```
{

    if (delete_rules)
        delete rules
```

Definition at line 443 of file support-meat.hpp.

#### 8.58.3.16 rules_dyn

```
rules_dyn = rules_
```

Definition at line 482 of file support-meat.hpp.

#### 8.58.3.17 stats_bank

```
std::vector< Array_Type > std::vector< double > * stats_bank
```

**Initial value:**
```
{

    if (pos >= coordiantes_n_free)
        return
```

Definition at line 132 of file support-meat.hpp.

#### 8.58.3.18 tmp_chng

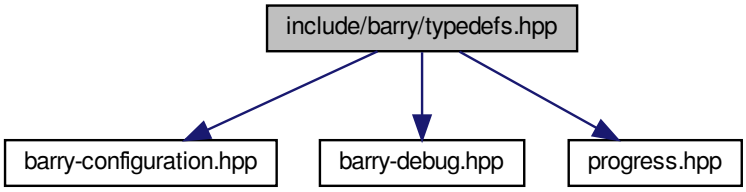
```
double tmp_chng
```

Definition at line 157 of file support-meat.hpp.

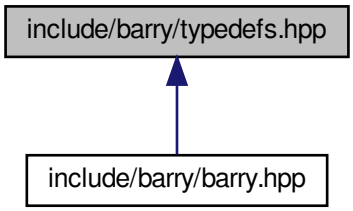
## 8.59 include/barry/typedefs.hpp File Reference

```
#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 T *a, const T *b, size_t n)
- template<> double vec_inner_prod (const double *a, const double *b, size_t n)

- template<typename T >
  bool vec_equal (const std::vector< T > &a, const std::vector< T > &b)

  *Compares if -a- and -b- are equal.*
- template<typename T >
  bool vec_equal_approx (const std::vector< T > &a, const std::vector< T > &b, double eps=1e-100)

## Variables

- const int CHECK::BOTH = -1
- const int CHECK::NONE = 0
- const int CHECK::ONE = 1
- const int CHECK::TWO = 2
- const int EXISTS::BOTH = -1
- const int EXISTS::NONE = 0
- const int EXISTS::ONE = 1
- const int EXISTS::TWO = 1
- const int EXISTS::UKNOWN = -1
- const int EXISTS::AS_ZERO = 0
- const int EXISTS::AS_ONE = 1

### 8.59.1 Typedef Documentation

#### 8.59.1.1 Col_type

```
template<typename Cell_Type >
using Col_type = Map< uint, Cell<Cell_Type>* >
```

Definition at line 71 of file typedefs.hpp.

#### 8.59.1.2 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 148 of file typedefs.hpp.

#### 8.59.1.3 Counts_type

```
typedef std::vector< std::pair< std::vector<double>, uint > > Counts_type
```

Definition at line 52 of file typedefs.hpp.

#### 8.59.1.4 MapVec_type

```
template<typename Ta = double, typename Tb = uint>
using MapVec_type = std::unordered_map< std::vector< Ta >, Tb, vecHasher<Ta> >
```

Definition at line 129 of file typedefs.hpp.

#### 8.59.1.5 Row_type

```
template<typename Cell_Type >
using Row_type = Map< uint, Cell<Cell_Type> >
```

Definition at line 68 of file typedefs.hpp.

#### 8.59.1.6 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 151 of file typedefs.hpp.

#### 8.59.1.7 uint

```
typedef unsigned int uint
```

Definition at line 18 of file typedefs.hpp.

### 8.59.2 Function Documentation

#### 8.59.2.1 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 162 of file typedefs.hpp.

#### 8.59.2.2 vec_equal_approx()

```
template<typename T >
bool vec_equal_approx (
             const std::vector< T > & a,
             const std::vector< T > & b,
             double eps = 1e-100 )  [inline]
```

Definition at line 180 of file typedefs.hpp.

#### 8.59.2.3 vec_inner_prod() [1/2]

```
template<>
double vec_inner_prod (
             const double * a,
             const double * b,
             size_t n )  [inline]
```

Definition at line 226 of file typedefs.hpp.

#### 8.59.2.4 vec_inner_prod() [2/2]

```
template<typename T >
T vec_inner_prod (
             const T * a,
             const T * b,
             size_t n )  [inline]
```

Definition at line 203 of file typedefs.hpp.

## 8.60 README.md File Reference

# Index


~BArray
BArray< Cell_Type, Data_Type >, 39
~BArrayCell
BArrayCell< Cell_Type, Data_Type >, 50
~BArrayCell_const
BArrayCell_const< Cell_Type, Data_Type >, 52
~BArrayDense
BArrayDense< Cell_Type, Data_Type >, 58
~BArrayDenseCell
BArrayDenseCell< Cell_Type, Data_Type >, 70
~BArrayRow
BArrayRow< Cell_Type, Data_Type >, 83
~BArrayRow_const
BArrayRow_const< Cell_Type, Data_Type >, 85
~BArrayVector
BArrayVector< Cell_Type, Data_Type >, 88
~BArrayVector_const
BArrayVector_const< Cell_Type, Data_Type >, 92
~Cell
Cell< Cell_Type >, 96
~ConstBArrayRowIter
ConstBArrayRowIter< Cell_Type, Data_Type >, 101
~Counter
Counter< Array_Type, Data_Type >, 104
~Counters
Counters< Array_Type, Data_Type >, 108
~DEFM
DEFM, 111
~DEFMCounterData
DEFMCounterData, 114
~DEFMData
DEFMData, 117
~Entries
Entries< Cell_Type >, 122
~Flock
Flock, 124
~FreqTable
FreqTable< T >, 131
~Geese
Geese, 136
~Model
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 149
~NetCounterData
NetCounterData, 160
~NetworkData
NetworkData, 163
~Node
Node, 165
~PhyloRuleDynData
PhyloRuleDynData, 174
~PowerSet
PowerSet< Array_Type, Data_Rule_Type >, 177
~Progress
Progress, 182
~Rule
Rule< Array_Type, Data_Type >, 184
~Rules
Rules< Array_Type, Data_Type >, 186
~StatsCounter
StatsCounter< Array_Type, Data_Type >, 190
~Support
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 195

active
Cell< Cell_Type >, 99
add
barray-meat.hpp, 222
barraydense-meat.hpp, 247
Cell< Cell_Type >, 97
FreqTable< T >, 131
add_array
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 149
add_counter
Counters< Array_Type, Data_Type >, 108, 109
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 150
StatsCounter< Array_Type, Data_Type >, 190
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 195
add_data
Flock, 124
add_rule
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 150
PowerSet< Array_Type, Data_Rule_Type >, 177, 178
Rules< Array_Type, Data_Type >, 187
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >,

195, 196
add_rule_dyn
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 150
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 196
annotations
Node, 166
ans
barray-meat.hpp, 213, 222
barraydense-meat.hpp, 236, 247
Array
ConstBArrayRowIter< Cell_Type, Data_Type >, 101
array
DEFMData, 118
Node, 166
Array_
barray-meat.hpp, 222
array_bank
support-meat.hpp, 336
arrays
Node, 167
AS_ONE
EXISTS, 33
as_vector
FreqTable< T >, 131
AS_ZERO
EXISTS, 33
at
DEFMData, 117
PhyloCounterData, 171

BArray
BArray< Cell_Type, Data_Type >, 38, 39
BArray< Cell_Type, Data_Type >, 35
∼BArray, 39
BArray, 38, 39
BArrayCell< Cell_Type, Data_Type >, 48
BArrayCell_const< Cell_Type, Data_Type >, 48
clear, 39
col, 39
D, 40
D_ptr, 40
default_val, 40
flush_data, 40
get_cell, 40
get_col_vec, 41
get_entries, 41
get_row_vec, 41
insert_cell, 42
is_dense, 42
is_empty, 42
ncol, 43
nnozero, 43
nrow, 43
operator∗=, 43
operator(), 43
operator+=, 44
operator-=, 44
operator/=, 45
operator=, 45
operator==, 45
out_of_range, 45
print, 45
reserve, 46
resize, 46
rm_cell, 46
row, 46
set_data, 46
swap_cells, 47
swap_cols, 47
swap_rows, 47
toggle_cell, 47
toggle_lock, 47
transpose, 48
visited, 49
zero_col, 48
zero_row, 48
barray-meat-operators.hpp
BARRAY_TEMPLATE, 206–208
BARRAY_TEMPLATE_ARGS, 207, 209
BARRAY_TYPE, 207, 209
COL, 207
for, 209
operator(), 209
rhs, 209
ROW, 207
this, 210
barray-meat.hpp
add, 222
ans, 213, 222
Array_, 222
BARRAY_TEMPLATE, 212–217
BARRAY_TEMPLATE_ARGS, 212
BARRAY_TYPE, 212
check_bounds, 222
check_exists, 223
COL, 213, 217
col0, 223
const, 223
copy_data, 223
data, 224
delete_data, 224
delete_data_, 224
else, 224
false, 224
first, 225
for, 218
i1, 225
if, 218–221
j, 225
j0, 225
j1, 225
M, 221, 225

M_, 226
N, 226
NCells, 226
report, 226
resize, 221
return, 221, 226
ROW, 213, 221, 222
row0, 227
search, 227
source, 227
target, 227
v, 227
value, 227
BARRAY_TEMPLATE
barray-meat-operators.hpp, 206–208
barray-meat.hpp, 212–217
BARRAY_TEMPLATE_ARGS
barray-meat-operators.hpp, 207, 209
barray-meat.hpp, 212
BARRAY_TYPE
barray-meat-operators.hpp, 207, 209
barray-meat.hpp, 212
BArrayCell
BArrayCell< Cell_Type, Data_Type >, 50
BArrayCell< Cell_Type, Data_Type >, 49
~BArrayCell, 50
BArray< Cell_Type, Data_Type >, 48
BArrayCell, 50
operator Cell_Type, 50
operator*=, 50
operator+=, 50
operator-=, 51
operator/=, 51
operator=, 51
operator==, 51
BArrayCell_const
BArrayCell_const< Cell_Type, Data_Type >, 52
BArrayCell_const< Cell_Type, Data_Type >, 52
~BArrayCell_const, 52
BArray< Cell_Type, Data_Type >, 48
BArrayCell_const, 52
operator Cell_Type, 53
operator!=, 53
operator<, 53
operator<=, 53
operator>, 53
operator>=, 54
operator==, 53
BArrayDense
BArrayDense< Cell_Type, Data_Type >, 57, 58
BArrayDense< Cell_Type, Data_Type >, 54
~BArrayDense, 58
BArrayDense, 57, 58
BArrayDenseCell< Cell_Type, Data_Type >, 68, 72
BArrayDenseCol< Cell_Type, Data_Type >, 68, 75
BArrayDenseCol_const< Cell_Type, Data_Type >, 68
BArrayDenseRow< Cell_Type, Data_Type >, 69, 80
BArrayDenseRow_const< Cell_Type, Data_Type >, 69
clear, 58
col, 58, 59
colsum, 59
D, 59
D_ptr, 59
default_val, 60
get_cell, 60
get_col_vec, 60
get_data, 60
get_entries, 60
get_row_vec, 61
insert_cell, 61
is_dense, 62
is_empty, 62
ncol, 62
nnozero, 62
nrow, 62
operator*=, 63
operator(), 62, 63
operator+=, 63
operator-=, 63, 64
operator/=, 64
operator=, 64
operator==, 64
out_of_range, 65
print, 65
reserve, 65
resize, 65
rm_cell, 65
row, 65, 66
rowsum, 66
set_data, 66
swap_cells, 66
swap_cols, 67
swap_rows, 67
toggle_cell, 67
toggle_lock, 67
transpose, 67
visited, 69
zero_col, 68
zero_row, 68
barraydense-meat-operators.hpp
BDENSE_TEMPLATE, 230–232
BDENSE_TEMPLATE_ARGS, 230, 232
BDENSE_TYPE, 230, 232
COL, 230
POS, 230
POS_N, 231
ROW, 231
barraydense-meat.hpp
add, 247
ans, 236, 247
BDENSE_TEMPLATE, 235–243
BDENSE_TEMPLATE_ARGS, 235

BDENSE_TYPE, 235
check_bounds, 247
check_exists, 247
COL, 235
col, 248
const, 248
copy_data, 248
data, 248
delete_data, 248
delete_data_, 249
el, 249
el_colsums, 249
el_rowsums, 249
else, 249
false, 250
for, 243
i1, 250
if, 244
insert_cell, 244
j, 250
j0, 250
j1, 250
M, 245, 250
M_, 251
N, 251
POS, 235
POS_N, 235
report, 251
resize, 245, 246
return, 251
rm_cell, 246
ROW, 236
source, 251
target, 252
v, 252
va_end, 246
va_start, 246
val0, 252
val1, 252
value, 252
vprintf, 247
ZERO_CELL, 236
BArrayDenseCell
BArrayDenseCell< Cell_Type, Data_Type >, 70
BArrayDenseCell< Cell_Type, Data_Type >, 69
~BArrayDenseCell, 70
BArrayDense< Cell_Type, Data_Type >, 68, 72
BArrayDenseCell, 70
BArrayDenseCol< Cell_Type, Data_Type >, 72, 75
BArrayDenseCol_const< Cell_Type, Data_Type >, 72, 77
BArrayDenseRow< Cell_Type, Data_Type >, 80
BArrayDenseRow_const< Cell_Type, Data_Type >, 82
operator Cell_Type, 71
operator*=, 71
operator+=, 71
operator-=, 71
operator/=, 71
operator=, 71, 72
operator==, 72
barraydensecell-bones.hpp
POS, 253
barraydensecell-meat.hpp
POS, 254
BArrayDenseCell_const< Cell_Type, Data_Type >, 73
BArrayDenseCol< Cell_Type, Data_Type >, 75
BArrayDenseCol_const< Cell_Type, Data_Type >, 77
BArrayDenseRow< Cell_Type, Data_Type >, 80
BArrayDenseRow_const< Cell_Type, Data_Type >, 82
BArrayDenseCol
BArrayDenseCol< Cell_Type, Data_Type >, 74
BArrayDenseCol< Cell_Type, Data_Type >, 73
BArrayDense< Cell_Type, Data_Type >, 68, 75
BArrayDenseCell< Cell_Type, Data_Type >, 72, 75
BArrayDenseCell_const< Cell_Type, Data_Type >, 75
BArrayDenseCol, 74
begin, 74
end, 74
operator(), 74
size, 75
barraydensecol-bones.hpp
POS, 255
POS_N, 255
ZERO_CELL, 255
BArrayDenseCol_const
BArrayDenseCol_const< Cell_Type, Data_Type >, 76
BArrayDenseCol_const< Cell_Type, Data_Type >, 76
BArrayDense< Cell_Type, Data_Type >, 68
BArrayDenseCell< Cell_Type, Data_Type >, 72, 77
BArrayDenseCell_const< Cell_Type, Data_Type >, 77
BArrayDenseCol_const, 76
begin, 76
end, 77
operator(), 77
size, 77
BArrayDenseRow
BArrayDenseRow< Cell_Type, Data_Type >, 78
BArrayDenseRow< Cell_Type, Data_Type >, 78
BArrayDense< Cell_Type, Data_Type >, 69, 80
BArrayDenseCell< Cell_Type, Data_Type >, 80
BArrayDenseCell_const< Cell_Type, Data_Type >, 80
BArrayDenseRow, 78
begin, 79
end, 79
operator(), 79
size, 79
barraydenserow-bones.hpp

POS, 256
POS_N, 256
ZERO_CELL, 257
BArrayDenseRow_const
BArrayDenseRow_const< Cell_Type, Data_Type >, 81
BArrayDenseRow_const< Cell_Type, Data_Type >, 80
BArrayDense< Cell_Type, Data_Type >, 69
BArrayDenseCell< Cell_Type, Data_Type >, 82
BArrayDenseCell_const< Cell_Type, Data_Type >, 82
BArrayDenseRow_const, 81
begin, 81
end, 81
operator(), 81
size, 82
BArrayRow
BArrayRow< Cell_Type, Data_Type >, 83
BArrayRow< Cell_Type, Data_Type >, 82
~BArrayRow, 83
BArrayRow, 83
operator BArrayRow< Cell_Type, Data_Type >, 83
operator*=, 84
operator+=, 84
operator-=, 84
operator/=, 84
operator=, 84
operator==, 84
barrayrow-meat.hpp
BROW_TEMPLATE, 257–259
BROW_TEMPLATE_ARGS, 258
BROW_TYPE, 258
BArrayRow_const
BArrayRow_const< Cell_Type, Data_Type >, 85
BArrayRow_const< Cell_Type, Data_Type >, 85
~BArrayRow_const, 85
BArrayRow_const, 85
operator BArrayRow_const< Cell_Type, Data_Type >, 86
operator!=, 86
operator<, 86
operator<=, 86
operator>, 86
operator>=, 87
operator==, 86
BArrayVector
BArrayVector< Cell_Type, Data_Type >, 88
BArrayVector< Cell_Type, Data_Type >, 87
~BArrayVector, 88
BArrayVector, 88
begin, 88
end, 89
is_col, 89
is_row, 89
operator std::vector< Cell_Type >, 89
operator*=, 89
operator+=, 90
operator-=, 90
operator/=, 90
operator=, 90
operator==, 90
size, 91
BArrayVector_const
BArrayVector_const< Cell_Type, Data_Type >, 92
BArrayVector_const< Cell_Type, Data_Type >, 91
~BArrayVector_const, 92
BArrayVector_const, 92
begin, 92
end, 92
is_col, 92
is_row, 92
operator std::vector< Cell_Type >, 93
operator!=, 93
operator<, 93
operator<=, 93
operator>, 94
operator>=, 94
operator==, 93
size, 94
barry, 31
barry-configuration.hpp
BARRY_CHECK_SUPPORT, 261
BARRY_ISFINITE, 261
BARRY_MAX_NUM_ELEMENTS, 261
BARRY_SAFE_EXP, 261
Map, 261
printf_barry, 261
barry-debug.hpp
BARRY_DEBUG_LEVEL, 262
barry-macros.hpp
BARRY_ONE, 263
BARRY_ONE_DENSE, 263
BARRY_UNUSED, 263
BARRY_ZERO, 264
BARRY_ZERO_DENSE, 264
barry.hpp
BARRY_HPP, 265
BARRY_VERSION, 266
BARRY_VERSION_MAYOR, 266
BARRY_VERSION_MINOR, 266
COUNTER_FUNCTION, 266
COUNTER_LAMBDA, 266
RULE_FUNCTION, 267
RULE_LAMBDA, 267
barry::counters, 31
barry::counters::defm, 32
barry::counters::network, 32
barry::counters::phylo, 32
BARRY_CHECK_SUPPORT
barry-configuration.hpp, 261
BARRY_DEBUG_LEVEL
barry-debug.hpp, 262
BARRY_HPP
barry.hpp, 265
BARRY_ISFINITE
barry-configuration.hpp, 261

BARRY_MAX_NUM_ELEMENTS
barry-configuration.hpp, 261
BARRY_ONE
barry-macros.hpp, 263
BARRY_ONE_DENSE
barry-macros.hpp, 263
BARRY_PROGRESS_BAR_WIDTH
progress.hpp, 321
BARRY_SAFE_EXP
barry-configuration.hpp, 261
BARRY_SUPPORT_MEAT_HPP
support-meat.hpp, 330
BARRY_UNUSED
barry-macros.hpp, 263
BARRY_VERSION
barry.hpp, 266
BARRY_VERSION_MAYOR
barry.hpp, 266
BARRY_VERSION_MINOR
barry.hpp, 266
BARRY_ZERO
barry-macros.hpp, 264
BARRY_ZERO_DENSE
barry-macros.hpp, 264
BARRY_ZERO_NETWORK
network.hpp, 293
BARRY_ZERO_NETWORK_DENSE
network.hpp, 293
BDENSE_TEMPLATE
barraydense-meat-operators.hpp, 230–232
barraydense-meat.hpp, 235–243
BDENSE_TEMPLATE_ARGS
barraydense-meat-operators.hpp, 230, 232
barraydense-meat.hpp, 235
BDENSE_TYPE
barraydense-meat-operators.hpp, 230, 232
barraydense-meat.hpp, 235
begin
BArrayDenseCol< Cell_Type, Data_Type >, 74
BArrayDenseCol_const< Cell_Type, Data_Type >, 76
BArrayDenseRow< Cell_Type, Data_Type >, 79
BArrayDenseRow_const< Cell_Type, Data_Type >, 81
BArrayVector< Cell_Type, Data_Type >, 88
BArrayVector_const< Cell_Type, Data_Type >, 92
PhyloCounterData, 171
PowerSet< Array_Type, Data_Rule_Type >, 178
blengths
NodeData, 170
BOTH
CHECK, 32
EXISTS, 34
BROW_TEMPLATE
barrayrow-meat.hpp, 257–259
BROW_TEMPLATE_ARGS
barrayrow-meat.hpp, 258
BROW_TYPE
barrayrow-meat.hpp, 258
calc
PowerSet< Array_Type, Data_Rule_Type >, 178
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 196
calc_backend_dense
support-meat.hpp, 331
calc_backend_sparse
support-meat.hpp, 331
calc_reduced_sequence
Geese, 137
calc_sequence
Geese, 137
Cell
Cell< Cell_Type >, 95–97
Cell< Cell_Type >, 94
~Cell, 96
active, 99
add, 97
Cell, 95–97
operator Cell_Type, 98
operator!=, 98
operator=, 98
operator==, 98
value, 99
visited, 99
Cell_const< Cell_Type >, 100
change_stats
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 199
change_stats_different
support-meat.hpp, 336
CHECK, 32
BOTH, 32
NONE, 32
ONE, 32
TWO, 33
check_bounds
barray-meat.hpp, 222
barraydense-meat.hpp, 247
check_exists
barray-meat.hpp, 223
barraydense-meat.hpp, 247
clear
BArray< Cell_Type, Data_Type >, 39
BArrayDense< Cell_Type, Data_Type >, 58
FreqTable< T >, 131
statscounter-meat.hpp, 324
COL
barray-meat-operators.hpp, 207
barray-meat.hpp, 213, 217
barraydense-meat-operators.hpp, 230
barraydense-meat.hpp, 235
col
BArray< Cell_Type, Data_Type >, 39
BArrayDense< Cell_Type, Data_Type >, 58, 59

barraydense-meat.hpp, 248
col0
barray-meat.hpp, 223
Col_type
typedefs.hpp, 341
colnames
Flock, 125
Geese, 137
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 151
colsum
BArrayDense< Cell_Type, Data_Type >, 59
conditional_prob
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 151
const
barray-meat.hpp, 223
barraydense-meat.hpp, 248
ConstBArrayRowIter
ConstBArrayRowIter< Cell_Type, Data_Type >, 101
ConstBArrayRowIter< Cell_Type, Data_Type >, 100
~ConstBArrayRowIter, 101
Array, 101
ConstBArrayRowIter, 101
current_col, 101
current_row, 101
iter, 102
coord_i
support-meat.hpp, 336
coord_j
support-meat.hpp, 336
coordiantes_n_free
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 200
coordiantes_n_locked
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 200
coordinates_free
PowerSet< Array_Type, Data_Rule_Type >, 180
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 200
coordinates_locked
PowerSet< Array_Type, Data_Rule_Type >, 180
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 200
copy_data
barray-meat.hpp, 223
barraydense-meat.hpp, 248
count
Counter< Array_Type, Data_Type >, 104
count_all
StatsCounter< Array_Type, Data_Type >, 190
count_current
StatsCounter< Array_Type, Data_Type >, 191
count_fun
Counter< Array_Type, Data_Type >, 106
counters-meat.hpp, 271
count_fun_
counters-meat.hpp, 275
count_init
StatsCounter< Array_Type, Data_Type >, 191
Counter
Counter< Array_Type, Data_Type >, 103, 104
counter
counters-meat.hpp, 275
statscounter-meat.hpp, 326
Counter< Array_Type, Data_Type >, 102
~Counter, 104
count, 104
count_fun, 106
Counter, 103, 104
data, 106
desc, 106
get_description, 105
get_name, 105
init, 105
init_fun, 106
name, 106
operator=, 105
counter_
counters-meat.hpp, 275
counter_absdiff
DEFMArray counters, 14
counter_co_opt
Phylo counters, 23
counter_cogain
Phylo counters, 23
counter_css_census01
network-css.hpp, 286
counter_css_census02
network-css.hpp, 286
counter_css_census03
network-css.hpp, 287
counter_css_census04
network-css.hpp, 287
counter_css_census05
network-css.hpp, 287
counter_css_census06
network-css.hpp, 287
counter_css_census07
network-css.hpp, 288
counter_css_census08
network-css.hpp, 288
counter_css_census09
network-css.hpp, 288
counter_css_census10
network-css.hpp, 288
counter_css_completely_false_recip_comiss
network-css.hpp, 289

counter_css_completely_false_recip_omiss
network-css.hpp, 289
counter_css_mixed_recip
network-css.hpp, 289
counter_css_partially_false_recip_commi
network-css.hpp, 289
counter_css_partially_false_recip_omiss
network-css.hpp, 290
counter_ctriads
DEFMArray counters, 14
counter_degree
DEFMArray counters, 14
counter_deleted
statscounter-meat.hpp, 327
counter_density
DEFMArray counters, 14
counter_diff
DEFMArray counters, 15
counter_edges
DEFMArray counters, 15
counter_fixed_effect
DEFMArray counters, 15
Counter_fun_type
typedefs.hpp, 342
COUNTER_FUNCTION
barry.hpp, 266
counter_gains
Phylo counters, 23
counter_gains_from_0
Phylo counters, 23
counter_gains_k_offspring
Phylo counters, 24
counter_genes_changing
Phylo counters, 24
counter_idegree
DEFMArray counters, 16
counter_idegree15
DEFMArray counters, 16
counter_isolates
DEFMArray counters, 16, 17
counter_istar2
DEFMArray counters, 17
counter_k_genes_changing
Phylo counters, 24
COUNTER_LAMBDA
barry.hpp, 266
counter_less_than_p_prop_genes_changing
Phylo counters, 24
counter_longest
Phylo counters, 25
counter_loss
Phylo counters, 25
counter_maxfuns
Phylo counters, 25
counter_mutual
DEFMArray counters, 17
counter_neofun
Phylo counters, 25
counter_neofun_a2b
Phylo counters, 26
counter_nodecov
DEFMArray counters, 17
counter_nodeicov
DEFMArray counters, 18
counter_nodematch
DEFMArray counters, 18
counter_nodeocov
DEFMArray counters, 18
counter_odegree
DEFMArray counters, 18
counter_odegree15
DEFMArray counters, 19
counter_ones
DEFMArray counters, 19
counter_ostar2
DEFMArray counters, 19, 20
counter_overall_changes
Phylo counters, 26
counter_overall_gains
Phylo counters, 26
counter_overall_gains_from_0
Phylo counters, 26
counter_overall_loss
Phylo counters, 27
counter_pairwise_first_gain
Phylo counters, 27
counter_pairwise_neofun_singlefun
Phylo counters, 27
counter_pairwise_overall_change
Phylo counters, 27
counter_pairwise_preserving
Phylo counters, 28
counter_preserve_pseudogene
Phylo counters, 28
counter_prop_genes_changing
Phylo counters, 28
counter_subfun
Phylo counters, 28
COUNTER_TEMPLATE
counters-meat.hpp, 270–272
COUNTER_TEMPLATE_ARGS
counters-meat.hpp, 270
counter_transition
DEFMArray counters, 20
counter_ttriads
DEFMArray counters, 20
COUNTER_TYPE
counters-meat.hpp, 270
Counters
Counters< Array_Type, Data_Type >, 107, 108
counters-meat.hpp, 272
counters
statscounter-meat.hpp, 327
support-meat.hpp, 336
Counters< Array_Type, Data_Type >, 107
∼Counters, 108

add_counter, 108, 109
Counters, 107, 108
get_descriptions, 109
get_names, 109
operator=, 109, 110
operator[], 110
size, 110
counters-meat.hpp
count_fun, 271
count_fun_, 275
counter, 275
counter_, 275
COUNTER_TEMPLATE, 270–272
COUNTER_TEMPLATE_ARGS, 270
COUNTER_TYPE, 270
Counters, 272
COUNTERS_TEMPLATE, 270, 273, 274
COUNTERS_TEMPLATE_ARGS, 271
COUNTERS_TYPE, 271
data, 274
data_, 276
desc, 274
desc_, 276
i, 276
init_fun, 274, 275
init_fun_, 276
j, 277
name, 275
name_, 277
noexcept, 277
return, 277
this, 277
counters_
statscounter-meat.hpp, 327
support-meat.hpp, 337
COUNTERS_TEMPLATE
counters-meat.hpp, 270, 273, 274
COUNTERS_TEMPLATE_ARGS
counters-meat.hpp, 271
COUNTERS_TYPE
counters-meat.hpp, 271
Counting, 11
counts
PhyloRuleDynData, 174
Counts_type
typedefs.hpp, 342
covariates
DEFMData, 119
CSS_APPEND
network-css.hpp, 284
CSS_CASE_ELSE
network-css.hpp, 284
CSS_CASE_PERCEIVED
network-css.hpp, 284
CSS_CASE_TRUTH
network-css.hpp, 284
CSS_CHECK_SIZE
network-css.hpp, 285
CSS_CHECK_SIZE_INIT
network-css.hpp, 285
CSS_NET_COUNTER_LAMBDA_INIT
network-css.hpp, 285
CSS_PERCEIVED_CELLS
network-css.hpp, 285
CSS_SIZE
network-css.hpp, 286
CSS_TRUE_CELLS
network-css.hpp, 286
current_col
ConstBArrayRowIter< Cell_Type, Data_Type >, 101
current_row
ConstBArrayRowIter< Cell_Type, Data_Type >, 101
current_stats
statscounter-meat.hpp, 327
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 200

D
BArray< Cell_Type, Data_Type >, 40
BArrayDense< Cell_Type, Data_Type >, 59
Rule< Array_Type, Data_Type >, 185
D_ptr
BArray< Cell_Type, Data_Type >, 40
BArrayDense< Cell_Type, Data_Type >, 59
dat
Flock, 129
data
barray-meat.hpp, 224
barraydense-meat.hpp, 248
Counter< Array_Type, Data_Type >, 106
counters-meat.hpp, 274
PowerSet< Array_Type, Data_Rule_Type >, 180
data_
counters-meat.hpp, 276
DEFAULT_DUPLICATION
phylo.hpp, 300
default_val
BArray< Cell_Type, Data_Type >, 40
BArrayDense< Cell_Type, Data_Type >, 60
DEFM, 111
~DEFM, 111
DEFM, 111
get_m_order, 112
get_model, 112
get_n_covars, 112
get_n_obs, 112
get_n_rows, 112
get_n_y, 112
init, 113
likelihood, 113
simulate, 113
defm-meat.hpp
DEFM_LOOP_ARRAYS, 310
DEFM_RANGES, 310

keygen_defm, 311
defm.hpp
DEFM_COUNTER, 279
DEFM_COUNTER_LAMBDA, 279
DEFM_RULE, 280
DEFM_RULE_LAMBDA, 280
DEFMArray, 280
DEFMCounter, 280
DEFMCounters, 281
DEFMModel, 281
DEFMRule, 281
DEFMRules, 281
DEFMStatsCounter, 281
DEFMSupport, 281
DEFM_COUNTER
defm.hpp, 279
DEFM_COUNTER_LAMBDA
defm.hpp, 279
DEFM_LOOP_ARRAYS
defm-meat.hpp, 310
DEFM_RANGES
defm-meat.hpp, 310
DEFM_RULE
defm.hpp, 280
DEFM_RULE_LAMBDA
defm.hpp, 280
DEFMArray
defm.hpp, 280
DEFMArray counters, 12
counter_absdiff, 14
counter_ctriads, 14
counter_degree, 14
counter_density, 14
counter_diff, 15
counter_edges, 15
counter_fixed_effect, 15
counter_idegree, 16
counter_idegree15, 16
counter_isolates, 16, 17
counter_istar2, 17
counter_mutual, 17
counter_nodecov, 17
counter_nodeicov, 18
counter_nodematch, 18
counter_nodeocov, 18
counter_odegree, 18
counter_odegree15, 19
counter_ones, 19
counter_ostar2, 19, 20
counter_transition, 20
counter_ttriads, 20
NETWORK_COUNTER, 21
rules_markov_fixed, 21
DEFMCounter
defm.hpp, 280
DEFMCounterData, 113
∼DEFMCounterData, 114
DEFMCounterData, 114
idx, 114
indices, 115
markov_order, 115
num, 115
numbers, 115
DEFMCounters
defm.hpp, 281
DEFMData, 116
∼DEFMData, 117
array, 118
at, 117
covariates, 119
DEFMData, 117
ncol, 118
obs_start, 119
operator(), 118
print, 118
X_ncol, 119
X_nrow, 119
DEFMModel
defm.hpp, 281
DEFMRule
defm.hpp, 281
DEFMRuleData, 120
DEFMRuleData, 120
idx, 120
num, 120
DEFMRules
defm.hpp, 281
DEFMStatsCounter
defm.hpp, 281
DEFMSupport
defm.hpp, 281
delete_counters
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 201
support-meat.hpp, 337
delete_data
barray-meat.hpp, 224
barraydense-meat.hpp, 248
delete_data_
barray-meat.hpp, 224
barraydense-meat.hpp, 249
delete_rengine
Geese, 143
delete_rules
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 201
support-meat.hpp, 337
delete_rules_dyn
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 201
support-meat.hpp, 337
delete_support
Geese, 144

desc
Counter< Array_Type, Data_Type >, 106
counters-meat.hpp, 274
desc_
counters-meat.hpp, 276
directed
NetworkData, 163
DUPL_DUPL
phylo.hpp, 300
DUPL_EITH
phylo.hpp, 300
DUPL_SPEC
phylo.hpp, 300
duplication
Node, 167
NodeData, 170
PhyloRuleDynData, 174

el
barraydense-meat.hpp, 249
el_colsums
barraydense-meat.hpp, 249
el_rowsums
barraydense-meat.hpp, 249
else
barray-meat.hpp, 224
barraydense-meat.hpp, 249
support-meat.hpp, 337
empty
PhyloCounterData, 171
EmptyArray
PowerSet< Array_Type, Data_Rule_Type >, 180
statscounter-meat.hpp, 327
end
BArrayDenseCol< Cell_Type, Data_Type >, 74
BArrayDenseCol_const< Cell_Type, Data_Type >, 77
BArrayDenseRow< Cell_Type, Data_Type >, 79
BArrayDenseRow_const< Cell_Type, Data_Type >, 81
BArrayVector< Cell_Type, Data_Type >, 89
BArrayVector_const< Cell_Type, Data_Type >, 92
PhyloCounterData, 172
PowerSet< Array_Type, Data_Rule_Type >, 178
Progress, 183
Entries
Entries< Cell_Type >, 121, 122
Entries< Cell_Type >, 121
~Entries, 122
Entries, 121, 122
resize, 122
source, 122
target, 122
val, 123
eval_rules_dyn
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 197
EXISTS, 33
AS_ONE, 33
AS_ZERO, 33
BOTH, 34
NONE, 34
ONE, 34
TWO, 34
UKNOWN, 34

f_
statscounter-meat.hpp, 328
support-meat.hpp, 338
false
barray-meat.hpp, 224
barraydense-meat.hpp, 250
first
barray-meat.hpp, 225
Flock, 123
~Flock, 124
add_data, 124
colnames, 125
dat, 129
Flock, 124
get_counters, 125
get_model, 125
get_stats_support, 125
get_stats_target, 125
get_support_fun, 126
init, 126
initialized, 129
likelihood_joint, 126
model, 129
nfunctions, 129
nfuns, 126
nleafs, 127
nnodes, 127
nterms, 127
ntrees, 127
operator(), 127
parse_polytomies, 128
print, 128
rengine, 129
set_seed, 128
support_size, 128
flush_data
BArray< Cell_Type, Data_Type >, 40
for
barray-meat-operators.hpp, 209
barray-meat.hpp, 218
barraydense-meat.hpp, 243
statscounter-meat.hpp, 324
support-meat.hpp, 331
FreqTable
FreqTable< T >, 131
FreqTable< T >, 130
~FreqTable, 131
add, 131
as_vector, 131
clear, 131
FreqTable, 131

get_data, 132
get_index, 132
make_hash, 132
print, 132
reserve, 132
size, 133

Geese, 133
~Geese, 136
calc_reduced_sequence, 137
calc_sequence, 137
colnames, 137
delete_rengine, 143
delete_support, 144
Geese, 136
get_annotated_nodes, 137
get_counters, 137
get_model, 137
get_probabilities, 138
get_rengine, 138
get_states, 138
get_support_fun, 138
inherit_support, 138
init, 139
init_node, 139
initialized, 144
likelihood, 139
likelihood_exhaust, 139
map_to_nodes, 144
nannotations, 139
nfunctions, 144
nfuns, 140
nleafs, 140
nnodes, 140
nodes, 144
nterms, 140
observed_counts, 140
operator=, 141
parse_polytomies, 141
predict, 141
predict_backend, 141
predict_exhaust, 142
predict_exhaust_backend, 142
predict_sim, 142
print, 142
print_observed_counts, 142
pset_loc, 144
reduced_sequence, 145
sequence, 145
set_seed, 143
simulate, 143
support_size, 143
update_annotations, 143
geese-bones.hpp
INITIALIZED, 313
keygen_full, 313
RULE_FUNCTION, 313
vec_diff, 314
vector_caster, 314

gen_key
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 151
get_annotated_nodes
Geese, 137
get_arrays2support
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 152
get_cell
BArray< Cell_Type, Data_Type >, 40
BArrayDense< Cell_Type, Data_Type >, 60
get_col_vec
BArray< Cell_Type, Data_Type >, 41
BArrayDense< Cell_Type, Data_Type >, 60
get_counters
Flock, 125
Geese, 137
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 152
PhyloCounterData, 172
StatsCounter< Array_Type, Data_Type >, 191
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 197
get_counts
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 197
get_current_stats
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 197
get_data
BArrayDense< Cell_Type, Data_Type >, 60
FreqTable< T >, 132
PowerSet< Array_Type, Data_Rule_Type >, 178
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 197
get_data_ptr
PowerSet< Array_Type, Data_Rule_Type >, 179
get_description
Counter< Array_Type, Data_Type >, 105
get_descriptions
Counters< Array_Type, Data_Type >, 109
StatsCounter< Array_Type, Data_Type >, 191
get_entries
BArray< Cell_Type, Data_Type >, 41
BArrayDense< Cell_Type, Data_Type >, 60
get_index
FreqTable< T >, 132
get_last_name
phylo.hpp, 305
get_m_order
DEFM, 112

get_model
DEFM, 112
Flock, 125
Geese, 137
get_n_covars
DEFM, 112
get_n_obs
DEFM, 112
get_n_rows
DEFM, 112
get_n_y
DEFM, 112
get_name
Counter< Array_Type, Data_Type >, 105
get_names
Counters< Array_Type, Data_Type >, 109
StatsCounter< Array_Type, Data_Type >, 191
get_norm_const
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 152
get_parent
Node, 166
get_probabilities
Geese, 138
get_pset
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 152
get_pset_arrays
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 152
get_pset_probs
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 153
get_pset_stats
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 153
get_rengine
Geese, 138
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 153
get_row_vec
BArray< Cell_Type, Data_Type >, 41
BArrayDense< Cell_Type, Data_Type >, 61
get_rules
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 153
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 198
get_rules_dyn
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 154
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 198
get_seq
Rules< Array_Type, Data_Type >, 187
get_states
Geese, 138
get_stats_support
Flock, 125
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 154
get_stats_target
Flock, 125
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 154
get_support_fun
Flock, 126
Geese, 138
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 154

hashes
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 201
support-meat.hpp, 338
hashes_initialized
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 201

i
counters-meat.hpp, 276
i1
barray-meat.hpp, 225
barraydense-meat.hpp, 250
id
Node, 167
idx
DEFMCounterData, 114
DEFMRuleData, 120
if
barray-meat.hpp, 218–221
barraydense-meat.hpp, 244
support-meat.hpp, 331, 332
IF_MATCHES
phylo.hpp, 300
IF_NOTMATCHES
phylo.hpp, 301
include/barry/barray-bones.hpp, 205
include/barry/barray-iterator.hpp, 205
include/barry/barray-meat-operators.hpp, 206
include/barry/barray-meat.hpp, 210
include/barry/barraycell-bones.hpp, 228

include/barry/barraycell-meat.hpp, 228
include/barry/barraydense-bones.hpp, 229
include/barry/barraydense-meat-operators.hpp, 229
include/barry/barraydense-meat.hpp, 232
include/barry/barraydensecell-bones.hpp, 253
include/barry/barraydensecell-meat.hpp, 254
include/barry/barraydensecol-bones.hpp, 254
include/barry/barraydenserow-bones.hpp, 256
include/barry/barrayrow-bones.hpp, 257
include/barry/barrayrow-meat.hpp, 257
include/barry/barrayvector-bones.hpp, 259
include/barry/barrayvector-meat.hpp, 260
include/barry/barry-configuration.hpp, 260
include/barry/barry-debug.hpp, 262
include/barry/barry-macros.hpp, 263
include/barry/barry.hpp, 264
include/barry/cell-bones.hpp, 267
include/barry/cell-meat.hpp, 268
include/barry/col-bones.hpp, 268
include/barry/counters-bones.hpp, 268
include/barry/counters-meat.hpp, 269
include/barry/counters/defm.hpp, 278
include/barry/counters/network-css.hpp, 282
include/barry/counters/network.hpp, 290
include/barry/counters/phylo.hpp, 297
include/barry/freqtable.hpp, 305
include/barry/model-bones.hpp, 306
include/barry/model-meat.hpp, 307
include/barry/models/defm.hpp, 282
include/barry/models/defm/defm-bones.hpp, 309
include/barry/models/defm/defm-meat.hpp, 310
include/barry/models/geese.hpp, 311
include/barry/models/geese/flock-bones.hpp, 311
include/barry/models/geese/flock-meat.hpp, 312
include/barry/models/geese/geese-bones.hpp, 312
include/barry/models/geese/geese-meat-constructors.hpp, 314
include/barry/models/geese/geese-meat-likelihood.hpp, 315
include/barry/models/geese/geese-meat-likelihood_exhaust.hpp, 316
include/barry/models/geese/geese-meat-predict.hpp, 316
include/barry/models/geese/geese-meat-predict_exhaust.hpp, 317
include/barry/models/geese/geese-meat-predict_sim.hpp, 317
include/barry/models/geese/geese-meat-simulate.hpp, 318
include/barry/models/geese/geese-meat.hpp, 318
include/barry/models/geese/geese-node-bones.hpp, 319
include/barry/powerset-bones.hpp, 319
include/barry/powerset-meat.hpp, 320
include/barry/progress.hpp, 320
include/barry/rules-bones.hpp, 321
include/barry/rules-meat.hpp, 322
include/barry/statscounter-bones.hpp, 322
include/barry/statscounter-meat.hpp, 323
include/barry/support-bones.hpp, 328
include/barry/support-meat.hpp, 329
include/barry/typedefs.hpp, 340
indices
    DEFMCounterData, 115
    NetCounterData, 161
inherit_support
    Geese, 138
init
    Counter< Array_Type, Data_Type >, 105
    DEFM, 113
    Flock, 126
    Geese, 139
init_fun
    Counter< Array_Type, Data_Type >, 106
    counters-meat.hpp, 274, 275
init_fun_
    counters-meat.hpp, 276
init_node
    Geese, 139
init_support
    PowerSet< Array_Type, Data_Rule_Type >, 179
    Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 198
INITIALIZED
    geese-bones.hpp, 313
initialized
    Flock, 129
    Geese, 144
insert_cell
    BArray< Cell_Type, Data_Type >, 42
    BArrayDense< Cell_Type, Data_Type >, 61
    barraydense-meat.hpp, 244
    support-meat.hpp, 332
is_col
    BArrayVector< Cell_Type, Data_Type >, 89
    BArrayVector_const< Cell_Type, Data_Type >, 92
is_dense
    BArray< Cell_Type, Data_Type >, 42
    BArrayDense< Cell_Type, Data_Type >, 62
IS_DUPLICATION
    phylo.hpp, 301
IS_EITHER
    phylo.hpp, 301
is_empty
    BArray< Cell_Type, Data_Type >, 42
    BArrayDense< Cell_Type, Data_Type >, 62
is_leaf
    Node, 166
is_row
    BArrayVector< Cell_Type, Data_Type >, 89
    BArrayVector_const< Cell_Type, Data_Type >, 92
IS_SPECIATION
    phylo.hpp, 301
iter

ConstBArrayRowIter< Cell_Type, Data_Type >, 102

j
barray-meat.hpp, 225
barraydense-meat.hpp, 250
counters-meat.hpp, 277
statscounter-meat.hpp, 328
j0
barray-meat.hpp, 225
barraydense-meat.hpp, 250
j1
barray-meat.hpp, 225
barraydense-meat.hpp, 250

keygen_default
model-bones.hpp, 306
keygen_defm
defm-meat.hpp, 311
keygen_full
geese-bones.hpp, 313

lb
PhyloRuleDynData, 174
likelihood
DEFM, 113
Geese, 139
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 154, 155
likelihood_
model-meat.hpp, 308
likelihood_exhaust
Geese, 139
likelihood_joint
Flock, 126
likelihood_total
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 155

M
barray-meat.hpp, 221, 225
barraydense-meat.hpp, 245, 250
PowerSet< Array_Type, Data_Rule_Type >, 180
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 202
M_
barray-meat.hpp, 226
barraydense-meat.hpp, 251
MAKE_DUPL_VARS
phylo.hpp, 301
make_hash
FreqTable< T >, 132
Map
barry-configuration.hpp, 261
map_to_nodes
Geese, 144
MapVec_type
typedefs.hpp, 342
markov_order
DEFMCounterData, 115
max_num_elements
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 202
Model
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 148, 149
model
Flock, 129
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 145
~Model, 149
add_array, 149
add_counter, 150
add_rule, 150
add_rule_dyn, 150
colnames, 151
conditional_prob, 151
gen_key, 151
get_arrays2support, 152
get_counters, 152
get_norm_const, 152
get_pset, 152
get_pset_arrays, 152
get_pset_probs, 153
get_pset_stats, 153
get_rengine, 153
get_rules, 153
get_rules_dyn, 154
get_stats_support, 154
get_stats_target, 154
get_support_fun, 154
likelihood, 154, 155
likelihood_total, 155
Model, 148, 149
nterms, 156
operator=, 156
print, 156
print_stats, 156
sample, 156, 157
set_counters, 157
set_keygen, 157
set_rengine, 157
set_rules, 157
set_rules_dyn, 158
set_seed, 158
set_transform_model, 158
size, 159
size_unique, 159
store_psets, 159
support_size, 159
transform_model, 159
model-bones.hpp

keygen_default, 306
model-meat.hpp
likelihood_, 308
MODEL_TEMPLATE, 307, 308
MODEL_TEMPLATE_ARGS, 307
MODEL_TYPE, 308
update_normalizing_constant, 309
MODEL_TEMPLATE
model-meat.hpp, 307, 308
MODEL_TEMPLATE_ARGS
model-meat.hpp, 307
MODEL_TYPE
model-meat.hpp, 308

N
barray-meat.hpp, 226
barraydense-meat.hpp, 251
PowerSet< Array_Type, Data_Rule_Type >, 181
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 202
n_counters
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 202
n_free
PowerSet< Array_Type, Data_Rule_Type >, 181
n_locked
PowerSet< Array_Type, Data_Rule_Type >, 181
name
Counter< Array_Type, Data_Type >, 106
counters-meat.hpp, 275
name_
counters-meat.hpp, 277
nannotations
Geese, 139
narray
Node, 167
NCells
barray-meat.hpp, 226
ncol
BArray< Cell_Type, Data_Type >, 43
BArrayDense< Cell_Type, Data_Type >, 62
DEFMData, 118
NET_C_DATA_IDX
network.hpp, 294
NET_C_DATA_NUM
network.hpp, 294
NetCounter
network.hpp, 295
NetCounterData, 160
~NetCounterData, 160
indices, 161
NetCounterData, 160
numbers, 161
NetCounters
network.hpp, 295
NetModel
network.hpp, 296
NetRule
network.hpp, 296
NetRules
network.hpp, 296
NetStatsCounter
network.hpp, 296
NetSupport
network.hpp, 296
Network
network.hpp, 296
network-css.hpp
counter_css_census01, 286
counter_css_census02, 286
counter_css_census03, 287
counter_css_census04, 287
counter_css_census05, 287
counter_css_census06, 287
counter_css_census07, 288
counter_css_census08, 288
counter_css_census09, 288
counter_css_census10, 288
counter_css_completely_false_recip_comiss, 289
counter_css_completely_false_recip_omiss, 289
counter_css_mixed_recip, 289
counter_css_partially_false_recip_commi, 289
counter_css_partially_false_recip_omiss, 290
CSS_APPEND, 284
CSS_CASE_ELSE, 284
CSS_CASE_PERCEIVED, 284
CSS_CASE_TRUTH, 284
CSS_CHECK_SIZE, 285
CSS_CHECK_SIZE_INIT, 285
CSS_NET_COUNTER_LAMBDA_INIT, 285
CSS_PERCEIVED_CELLS, 285
CSS_SIZE, 286
CSS_TRUE_CELLS, 286
network.hpp
BARRY_ZERO_NETWORK, 293
BARRY_ZERO_NETWORK_DENSE, 293
NET_C_DATA_IDX, 294
NET_C_DATA_NUM, 294
NetCounter, 295
NetCounters, 295
NetModel, 296
NetRule, 296
NetRules, 296
NetStatsCounter, 296
NetSupport, 296
Network, 296
NETWORK_COUNTER, 294
NETWORK_COUNTER_LAMBDA, 294
NETWORK_RULE, 294
NETWORK_RULE_LAMBDA, 295
NetworkDense, 297
NETWORKDENSE_COUNTER_LAMBDA, 295
rules_zerodiag, 297
NETWORK_COUNTER
DEFMArray counters, 21

network.hpp, 294
NETWORK_COUNTER_LAMBDA
network.hpp, 294
NETWORK_RULE
network.hpp, 294
NETWORK_RULE_LAMBDA
network.hpp, 295
NetworkData, 161
~NetworkData, 163
directed, 163
NetworkData, 162
vertex_attr, 163
NetworkDense
network.hpp, 297
NETWORKDENSE_COUNTER_LAMBDA
network.hpp, 295
next
Progress, 183
nfunctions
Flock, 129
Geese, 144
nfuns
Flock, 126
Geese, 140
nleafs
Flock, 127
Geese, 140
nnodes
Flock, 127
Geese, 140
nnozero
BArray< Cell_Type, Data_Type >, 43
BArrayDense< Cell_Type, Data_Type >, 62
Node, 163
~Node, 165
annotations, 166
array, 166
arrays, 167
duplication, 167
get_parent, 166
id, 167
is_leaf, 166
narray, 167
Node, 164, 165
noffspring, 166
offspring, 167
ord, 168
parent, 168
probability, 168
subtree_prob, 168
visited, 168
NodeData, 169
blengths, 170
duplication, 170
NodeData, 169
states, 170
nodes
Geese, 144
noexcept
counters-meat.hpp, 277
noffspring
Node, 166
NONE
CHECK, 32
EXISTS, 34
nrow
BArray< Cell_Type, Data_Type >, 43
BArrayDense< Cell_Type, Data_Type >, 62
nterms
Flock, 127
Geese, 140
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 156
ntrees
Flock, 127
num
DEFMCounterData, 115
DEFMRuleData, 120
numbers
DEFMCounterData, 115
NetCounterData, 161

obs_start
DEFMData, 119
observed_counts
Geese, 140
offspring
Node, 167
ONE
CHECK, 32
EXISTS, 34
operator BArrayRow< Cell_Type, Data_Type >
BArrayRow< Cell_Type, Data_Type >, 83
operator BArrayRow_const< Cell_Type, Data_Type >
BArrayRow_const< Cell_Type, Data_Type >, 86
operator Cell_Type
BArrayCell< Cell_Type, Data_Type >, 50
BArrayCell_const< Cell_Type, Data_Type >, 53
BArrayDenseCell< Cell_Type, Data_Type >, 71
Cell< Cell_Type >, 98
operator std::vector< Cell_Type >
BArrayVector< Cell_Type, Data_Type >, 89
BArrayVector_const< Cell_Type, Data_Type >, 93
operator!=
BArrayCell_const< Cell_Type, Data_Type >, 53
BArrayRow_const< Cell_Type, Data_Type >, 86
BArrayVector_const< Cell_Type, Data_Type >, 93
Cell< Cell_Type >, 98
operator<
BArrayCell_const< Cell_Type, Data_Type >, 53
BArrayRow_const< Cell_Type, Data_Type >, 86
BArrayVector_const< Cell_Type, Data_Type >, 93
operator<=
BArrayCell_const< Cell_Type, Data_Type >, 53
BArrayRow_const< Cell_Type, Data_Type >, 86
BArrayVector_const< Cell_Type, Data_Type >, 93

operator>
BArrayCell_const< Cell_Type, Data_Type >, 53
BArrayRow_const< Cell_Type, Data_Type >, 86
BArrayVector_const< Cell_Type, Data_Type >, 94
operator>=
BArrayCell_const< Cell_Type, Data_Type >, 54
BArrayRow_const< Cell_Type, Data_Type >, 87
BArrayVector_const< Cell_Type, Data_Type >, 94
operator*=
BArray< Cell_Type, Data_Type >, 43
BArrayCell< Cell_Type, Data_Type >, 50
BArrayDense< Cell_Type, Data_Type >, 63
BArrayDenseCell< Cell_Type, Data_Type >, 71
BArrayRow< Cell_Type, Data_Type >, 84
BArrayVector< Cell_Type, Data_Type >, 89
operator()
BArray< Cell_Type, Data_Type >, 43
barray-meat-operators.hpp, 209
BArrayDense< Cell_Type, Data_Type >, 62, 63
BArrayDenseCol< Cell_Type, Data_Type >, 74
BArrayDenseCol_const< Cell_Type, Data_Type >, 77
BArrayDenseRow< Cell_Type, Data_Type >, 79
BArrayDenseRow_const< Cell_Type, Data_Type >, 81
DEFMData, 118
Flock, 127
PhyloCounterData, 172
Rule< Array_Type, Data_Type >, 185
Rules< Array_Type, Data_Type >, 188
vecHasher< T >, 203
operator+=
BArray< Cell_Type, Data_Type >, 44
BArrayCell< Cell_Type, Data_Type >, 50
BArrayDense< Cell_Type, Data_Type >, 63
BArrayDenseCell< Cell_Type, Data_Type >, 71
BArrayRow< Cell_Type, Data_Type >, 84
BArrayVector< Cell_Type, Data_Type >, 90
operator-=
BArray< Cell_Type, Data_Type >, 44
BArrayCell< Cell_Type, Data_Type >, 51
BArrayDense< Cell_Type, Data_Type >, 63, 64
BArrayDenseCell< Cell_Type, Data_Type >, 71
BArrayRow< Cell_Type, Data_Type >, 84
BArrayVector< Cell_Type, Data_Type >, 90
operator/=
BArray< Cell_Type, Data_Type >, 45
BArrayCell< Cell_Type, Data_Type >, 51
BArrayDense< Cell_Type, Data_Type >, 64
BArrayDenseCell< Cell_Type, Data_Type >, 71
BArrayRow< Cell_Type, Data_Type >, 84
BArrayVector< Cell_Type, Data_Type >, 90
operator=
BArray< Cell_Type, Data_Type >, 45
BArrayCell< Cell_Type, Data_Type >, 51
BArrayDense< Cell_Type, Data_Type >, 64
BArrayDenseCell< Cell_Type, Data_Type >, 71, 72
BArrayRow< Cell_Type, Data_Type >, 84
BArrayVector< Cell_Type, Data_Type >, 90
Cell< Cell_Type >, 98
Counter< Array_Type, Data_Type >, 105
Counters< Array_Type, Data_Type >, 109, 110
Geese, 141
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 156
Rules< Array_Type, Data_Type >, 188
operator==
BArray< Cell_Type, Data_Type >, 45
BArrayCell< Cell_Type, Data_Type >, 51
BArrayCell_const< Cell_Type, Data_Type >, 53
BArrayDense< Cell_Type, Data_Type >, 64
BArrayDenseCell< Cell_Type, Data_Type >, 72
BArrayRow< Cell_Type, Data_Type >, 84
BArrayRow_const< Cell_Type, Data_Type >, 86
BArrayVector< Cell_Type, Data_Type >, 90
BArrayVector_const< Cell_Type, Data_Type >, 93
Cell< Cell_Type >, 98
operator[]
Counters< Array_Type, Data_Type >, 110
PhyloCounterData, 172
PowerSet< Array_Type, Data_Rule_Type >, 179
ord
Node, 168
out_of_range
BArray< Cell_Type, Data_Type >, 45
BArrayDense< Cell_Type, Data_Type >, 65

parent
Node, 168
parse_polytomies
Flock, 128
Geese, 141
Phylo counters, 21
counter_co_opt, 23
counter_cogain, 23
counter_gains, 23
counter_gains_from_0, 23
counter_gains_k_offspring, 24
counter_genes_changing, 24
counter_k_genes_changing, 24
counter_less_than_p_prop_genes_changing, 24
counter_longest, 25
counter_loss, 25
counter_maxfuns, 25
counter_neofun, 25
counter_neofun_a2b, 26
counter_overall_changes, 26
counter_overall_gains, 26
counter_overall_gains_from_0, 26
counter_overall_loss, 27
counter_pairwise_first_gain, 27
counter_pairwise_neofun_singlefun, 27
counter_pairwise_overall_change, 27
counter_pairwise_preserving, 28
counter_preserve_pseudogene, 28

counter_prop_genes_changing, 28
counter_subfun, 28
Phylo rules, 29
rule_dyn_limit_changes, 29
phylo.hpp
DEFAULT_DUPLICATION, 300
DUPL_DUPL, 300
DUPL_EITH, 300
DUPL_SPEC, 300
get_last_name, 305
IF_MATCHES, 300
IF_NOTMATCHES, 301
IS_DUPLICATION, 301
IS_EITHER, 301
IS_SPECIATION, 301
MAKE_DUPL_VARS, 301
PHYLO_CHECK_MISSING, 302
PHYLO_COUNTER_LAMBDA, 302
PHYLO_RULE_DYN_LAMBDA, 302
PhyloArray, 303
PhyloCounter, 303
PhyloCounters, 303
PhyloModel, 303
PhyloPowerSet, 303
PhyloRule, 303
PhyloRuleData, 304
PhyloRuleDyn, 304
PhyloRules, 304
PhyloRulesDyn, 304
PhyloStatsCounter, 304
PhyloSupport, 304
PHYLO_CHECK_MISSING
phylo.hpp, 302
PHYLO_COUNTER_LAMBDA
phylo.hpp, 302
PHYLO_RULE_DYN_LAMBDA
phylo.hpp, 302
PhyloArray
phylo.hpp, 303
PhyloCounter
phylo.hpp, 303
PhyloCounterData, 170
at, 171
begin, 171
empty, 171
end, 172
get_counters, 172
operator(), 172
operator[], 172
PhyloCounterData, 171
push_back, 172
reserve, 172
shrink_to_fit, 173
size, 173
PhyloCounters
phylo.hpp, 303
PhyloModel
phylo.hpp, 303
PhyloPowerSet
phylo.hpp, 303
PhyloRule
phylo.hpp, 303
PhyloRuleData
phylo.hpp, 304
PhyloRuleDyn
phylo.hpp, 304
PhyloRuleDynData, 173
∼PhyloRuleDynData, 174
counts, 174
duplication, 174
lb, 174
PhyloRuleDynData, 174
pos, 174
ub, 175
PhyloRules
phylo.hpp, 304
PhyloRulesDyn
phylo.hpp, 304
PhyloStatsCounter
phylo.hpp, 304
PhyloSupport
phylo.hpp, 304
POS
barraydense-meat-operators.hpp, 230
barraydense-meat.hpp, 235
barraydensecell-bones.hpp, 253
barraydensecell-meat.hpp, 254
barraydensecol-bones.hpp, 255
barraydenserow-bones.hpp, 256
pos
PhyloRuleDynData, 174
POS_N
barraydense-meat-operators.hpp, 231
barraydense-meat.hpp, 235
barraydensecol-bones.hpp, 255
barraydenserow-bones.hpp, 256
PowerSet
PowerSet< Array_Type, Data_Rule_Type >, 177
PowerSet< Array_Type, Data_Rule_Type >, 175
∼PowerSet, 177
add_rule, 177, 178
begin, 178
calc, 178
coordinates_free, 180
coordinates_locked, 180
data, 180
EmptyArray, 180
end, 178
get_data, 178
get_data_ptr, 179
init_support, 179
M, 180
N, 181
n_free, 181
n_locked, 181
operator[], 179

PowerSet, 177
reset, 179
rules, 181
rules_deleted, 181
size, 179
predict
Geese, 141
predict_backend
Geese, 141
predict_exhaust
Geese, 142
predict_exhaust_backend
Geese, 142
predict_sim
Geese, 142
print
BArray< Cell_Type, Data_Type >, 45
BArrayDense< Cell_Type, Data_Type >, 65
DEFMData, 118
Flock, 128
FreqTable< T >, 132
Geese, 142
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 156
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 198
print_observed_counts
Geese, 142
print_stats
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 156
printf_barry
barry-configuration.hpp, 261
probability
Node, 168
Progress, 182
~Progress, 182
end, 183
next, 183
Progress, 182
progress.hpp
BARRY_PROGRESS_BAR_WIDTH, 321
pset_loc
Geese, 144
push_back
PhyloCounterData, 172

README.md, 344
reduced_sequence
Geese, 145
rengine
Flock, 129
report
barray-meat.hpp, 226
barraydense-meat.hpp, 251
reserve
BArray< Cell_Type, Data_Type >, 46
BArrayDense< Cell_Type, Data_Type >, 65
FreqTable< T >, 132
PhyloCounterData, 172
reset
PowerSet< Array_Type, Data_Rule_Type >, 179
reset_array
StatsCounter< Array_Type, Data_Type >, 191
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 198, 199
resize
BArray< Cell_Type, Data_Type >, 46
barray-meat.hpp, 221
BArrayDense< Cell_Type, Data_Type >, 65
barraydense-meat.hpp, 245, 246
Entries< Cell_Type >, 122
statscounter-meat.hpp, 325
return
barray-meat.hpp, 221, 226
barraydense-meat.hpp, 251
counters-meat.hpp, 277
statscounter-meat.hpp, 328
support-meat.hpp, 338
rhs
barray-meat-operators.hpp, 209
rm_cell
BArray< Cell_Type, Data_Type >, 46
BArrayDense< Cell_Type, Data_Type >, 65
barraydense-meat.hpp, 246
support-meat.hpp, 332
ROW
barray-meat-operators.hpp, 207
barray-meat.hpp, 213, 221, 222
barraydense-meat-operators.hpp, 231
barraydense-meat.hpp, 236
row
BArray< Cell_Type, Data_Type >, 46
BArrayDense< Cell_Type, Data_Type >, 65, 66
row0
barray-meat.hpp, 227
Row_type
typedefs.hpp, 342
rowsum
BArrayDense< Cell_Type, Data_Type >, 66
Rule
Rule< Array_Type, Data_Type >, 184
Rule< Array_Type, Data_Type >, 183
~Rule, 184
D, 185
operator(), 185
Rule, 184
rule_dyn_limit_changes
Phylo rules, 29
rule_fun_default
rules-bones.hpp, 321
Rule_fun_type
typedefs.hpp, 343

RULE_FUNCTION
barry.hpp, 267
geese-bones.hpp, 313
RULE_LAMBDA
barry.hpp, 267
Rules
Rules< Array_Type, Data_Type >, 186
rules
PowerSet< Array_Type, Data_Rule_Type >, 181
support-meat.hpp, 338
Rules< Array_Type, Data_Type >, 185
∼Rules, 186
add_rule, 187
get_seq, 187
operator(), 188
operator=, 188
Rules, 186
size, 188
rules-bones.hpp
rule_fun_default, 321
rules_
support-meat.hpp, 338
rules_deleted
PowerSet< Array_Type, Data_Rule_Type >, 181
rules_dyn
support-meat.hpp, 339
rules_markov_fixed
DEFMArray counters, 21
rules_zerodiag
network.hpp, 297

sample
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 156, 157
search
barray-meat.hpp, 227
sequence
Geese, 145
set_counters
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 157
StatsCounter< Array_Type, Data_Type >, 192
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 199
set_data
BArray< Cell_Type, Data_Type >, 46
BArrayDense< Cell_Type, Data_Type >, 66
set_keygen
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 157
set_rengine
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 157
set_rules
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 157
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 199
set_rules_dyn
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 158
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 199
set_seed
Flock, 128
Geese, 143
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 158
set_transform_model
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 158
shrink_to_fit
PhyloCounterData, 173
simulate
DEFM, 113
Geese, 143
size
BArrayDenseCol< Cell_Type, Data_Type >, 75
BArrayDenseCol_const< Cell_Type, Data_Type >, 77
BArrayDenseRow< Cell_Type, Data_Type >, 79
BArrayDenseRow_const< Cell_Type, Data_Type >, 82
BArrayVector< Cell_Type, Data_Type >, 91
BArrayVector_const< Cell_Type, Data_Type >, 94
Counters< Array_Type, Data_Type >, 110
FreqTable< T >, 133
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 159
PhyloCounterData, 173
PowerSet< Array_Type, Data_Rule_Type >, 179
Rules< Array_Type, Data_Type >, 188
StatsCounter< Array_Type, Data_Type >, 192
size_unique
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 159
source
barray-meat.hpp, 227
barraydense-meat.hpp, 251
Entries< Cell_Type >, 122
states
NodeData, 170
Statistical Models, 11
stats_bank

support-meat.hpp, 339
StatsCounter
StatsCounter< Array_Type, Data_Type >, 189, 190
StatsCounter< Array_Type, Data_Type >, 189
~StatsCounter, 190
add_counter, 190
count_all, 190
count_current, 191
count_init, 191
get_counters, 191
get_descriptions, 191
get_names, 191
reset_array, 191
set_counters, 192
size, 192
StatsCounter, 189, 190
statscounter-meat.hpp
clear, 324
counter, 326
counter_deleted, 327
counters, 327
counters_, 327
current_stats, 327
EmptyArray, 327
f_, 328
for, 324
j, 328
resize, 325
return, 328
STATSCOUNTER_TEMPLATE, 324–326
STATSCOUNTER_TEMPLATE_ARGS, 324
STATSCOUNTER_TYPE, 324
STATSCOUNTER_TEMPLATE
statscounter-meat.hpp, 324–326
STATSCOUNTER_TEMPLATE_ARGS
statscounter-meat.hpp, 324
STATSCOUNTER_TYPE
statscounter-meat.hpp, 324
store_psets
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 159
subtree_prob
Node, 168
Support
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 194, 195
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 192
~Support, 195
add_counter, 195
add_rule, 195, 196
add_rule_dyn, 196
calc, 196
change_stats, 199
coordiantes_n_free, 200
coordiantes_n_locked, 200
coordinates_free, 200
coordinates_locked, 200
current_stats, 200
delete_counters, 201
delete_rules, 201
delete_rules_dyn, 201
eval_rules_dyn, 197
get_counters, 197
get_counts, 197
get_current_stats, 197
get_data, 197
get_rules, 198
get_rules_dyn, 198
hashes, 201
hashes_initialized, 201
init_support, 198
M, 202
max_num_elements, 202
N, 202
n_counters, 202
print, 198
reset_array, 198, 199
set_counters, 199
set_rules, 199
set_rules_dyn, 199
Support, 194, 195
support-meat.hpp
array_bank, 336
BARRY_SUPPORT_MEAT_HPP, 330
calc_backend_dense, 331
calc_backend_sparse, 331
change_stats_different, 336
coord_i, 336
coord_j, 336
counters, 336
counters_, 337
delete_counters, 337
delete_rules, 337
delete_rules_dyn, 337
else, 337
f_, 338
for, 331
hashes, 338
if, 331, 332
insert_cell, 332
return, 338
rm_cell, 332
rules, 338
rules_, 338
rules_dyn, 339
stats_bank, 339
SUPPORT_TEMPLATE, 330, 333–336
SUPPORT_TEMPLATE_ARGS, 330
SUPPORT_TYPE, 331
tmp_chng, 339
support_size
Flock, 128

Geese, 143
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 159
SUPPORT_TEMPLATE
support-meat.hpp, 330, 333–336
SUPPORT_TEMPLATE_ARGS
support-meat.hpp, 330
SUPPORT_TYPE
support-meat.hpp, 331
swap_cells
BArray< Cell_Type, Data_Type >, 47
BArrayDense< Cell_Type, Data_Type >, 66
swap_cols
BArray< Cell_Type, Data_Type >, 47
BArrayDense< Cell_Type, Data_Type >, 67
swap_rows
BArray< Cell_Type, Data_Type >, 47
BArrayDense< Cell_Type, Data_Type >, 67

target
barray-meat.hpp, 227
barraydense-meat.hpp, 252
Entries< Cell_Type >, 122
this
barray-meat-operators.hpp, 210
counters-meat.hpp, 277
tmp_chng
support-meat.hpp, 339
toggle_cell
BArray< Cell_Type, Data_Type >, 47
BArrayDense< Cell_Type, Data_Type >, 67
toggle_lock
BArray< Cell_Type, Data_Type >, 47
BArrayDense< Cell_Type, Data_Type >, 67
transform_model
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 159
transpose
BArray< Cell_Type, Data_Type >, 48
BArrayDense< Cell_Type, Data_Type >, 67
TWO
CHECK, 33
EXISTS, 34
typedefs.hpp
Col_type, 341
Counter_fun_type, 342
Counts_type, 342
MapVec_type, 342
Row_type, 342
Rule_fun_type, 343
uint, 343
vec_equal, 343
vec_equal_approx, 343
vec_inner_prod, 344

ub
PhyloRuleDynData, 175
uint
typedefs.hpp, 343
UKNOWN
EXISTS, 34
update_annotations
Geese, 143
update_normalizing_constant
model-meat.hpp, 309

v
barray-meat.hpp, 227
barraydense-meat.hpp, 252
va_end
barraydense-meat.hpp, 246
va_start
barraydense-meat.hpp, 246
val
Entries< Cell_Type >, 123
val0
barraydense-meat.hpp, 252
val1
barraydense-meat.hpp, 252
value
barray-meat.hpp, 227
barraydense-meat.hpp, 252
Cell< Cell_Type >, 99
vec_diff
geese-bones.hpp, 314
vec_equal
typedefs.hpp, 343
vec_equal_approx
typedefs.hpp, 343
vec_inner_prod
typedefs.hpp, 344
vecHasher< T >, 203
operator(), 203
vector_caster
geese-bones.hpp, 314
vertex_attr
NetworkData, 163
visited
BArray< Cell_Type, Data_Type >, 49
BArrayDense< Cell_Type, Data_Type >, 69
Cell< Cell_Type >, 99
Node, 168
vprintf
barraydense-meat.hpp, 247

X_ncol
DEFMData, 119
X_nrow
DEFMData, 119

ZERO_CELL
barraydense-meat.hpp, 236
barraydensecol-bones.hpp, 255
barraydenserow-bones.hpp, 257
zero_col
BArray< Cell_Type, Data_Type >, 48

BArrayDense< Cell_Type, Data_Type >, 68
zero_row
BArray< Cell_Type, Data_Type >, 48
BArrayDense< Cell_Type, Data_Type >, 68