

barry: Your go-to motif accountant

0.0-1

Generated by Doxygen 1.9.1

1 Main Page	1
2 Module Index	3
2.1 Modules	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Module Documentation	9
5.1 Counting	9
5.1.1 Detailed Description	9
5.2 Statistical Models	9
5.2.1 Detailed Description	10
5.3 Network counters	10
5.3.1 Detailed Description	11
5.3.2 Macro Definition Documentation	11
5.3.2.1 CSS_APPEND	11
5.3.2.2 CSS_CASE_ELSE	12
5.3.2.3 CSS_CASE_PERCEIVED	12
5.3.2.4 CSS_CASE_TRUTH	12
5.3.2.5 CSS_CHECK_SIZE	12
5.3.2.6 CSS_CHECK_SIZE_INIT	12
5.3.2.7 CSS_SIZE	13
5.3.3 Function Documentation	13
5.3.3.1 counter_absdiff()	13
5.3.3.2 counter_css_completely_false_recip_comiss()	13
5.3.3.3 counter_css_completely_false_recip_omiss()	13
5.3.3.4 counter_css_partially_false_recip_commi()	14
5.3.3.5 counter_css_partially_false_recip_omiss()	14
5.3.3.6 counter_ctriads()	14
5.3.3.7 counter_degree()	14
5.3.3.8 counter_density()	14
5.3.3.9 counter_diff()	15
5.3.3.10 counter_edges()	15
5.3.3.11 counter_idegree()	15
5.3.3.12 counter_idegree15()	15
5.3.3.13 counter_isolates()	15
5.3.3.14 counter_istar2()	16
5.3.3.15 counter_mutual()	16
5.3.3.16 counter_nodecov()	16
5.3.3.17 counter_nodeicov()	16

5.3.3.18 counter_nodematch()	16
5.3.3.19 counter_nodecov()	17
5.3.3.20 counter_odegree()	17
5.3.3.21 counter_odegree15()	17
5.3.3.22 counter_ostar2()	17
5.3.3.23 counter_ttriads()	17
5.3.3.24 NETWORK_COUNTER()	18
5.4 Phylo counters	18
5.4.1 Detailed Description	18
5.4.2 Function Documentation	19
5.4.2.1 counter_co_opt()	19
5.4.2.2 counter_cogain()	19
5.4.2.3 counter_gains()	20
5.4.2.4 counter_gains_k_offspring()	20
5.4.2.5 counter_genes_changing()	20
5.4.2.6 counter_longest()	20
5.4.2.7 counter_loss()	21
5.4.2.8 counter_maxfuns()	21
5.4.2.9 counter_neofun()	21
5.4.2.10 counter_neofun_a2b()	21
5.4.2.11 counter_overall_changes()	22
5.4.2.12 counter_overall_gains()	22
5.4.2.13 counter_overall_loss()	22
5.4.2.14 counter_prop_genes_changing()	22
5.4.2.15 counter_subfun()	23
5.5 Phylo rules	23
5.5.1 Detailed Description	23
5.5.2 Function Documentation	23
5.5.2.1 rule_dyn_limit_changes()	23
6 Namespace Documentation	25
6.1 barry Namespace Reference	25
6.1.1 Detailed Description	25
6.2 barry::counters Namespace Reference	25
6.2.1 Detailed Description	25
6.3 barry::counters::network Namespace Reference	26
6.4 barry::counters::phylo Namespace Reference	26
6.5 CHECK Namespace Reference	26
6.5.1 Detailed Description	26
6.5.2 Variable Documentation	26
6.5.2.1 BOTH	26
6.5.2.2 NONE	26

6.5.2.3 ONE	26
6.5.2.4 TWO	27
6.6 EXISTS Namespace Reference	27
6.6.1 Detailed Description	27
6.6.2 Variable Documentation	27
6.6.2.1 AS_ONE	27
6.6.2.2 AS_ZERO	27
6.6.2.3 BOTH	28
6.6.2.4 NONE	28
6.6.2.5 ONE	28
6.6.2.6 TWO	28
6.6.2.7 UNKNOWN	28
7 Class Documentation	29
7.1 BArray< Cell_Type, Data_Type > Class Template Reference	29
7.1.1 Detailed Description	31
7.1.2 Constructor & Destructor Documentation	32
7.1.2.1 BArray() [1/6]	32
7.1.2.2 BArray() [2/6]	32
7.1.2.3 BArray() [3/6]	32
7.1.2.4 BArray() [4/6]	33
7.1.2.5 BArray() [5/6]	33
7.1.2.6 BArray() [6/6]	33
7.1.2.7 ~BArray()	33
7.1.3 Member Function Documentation	33
7.1.3.1 clear()	33
7.1.3.2 col()	34
7.1.3.3 D() [1/2]	34
7.1.3.4 D() [2/2]	34
7.1.3.5 default_val()	34
7.1.3.6 flush_data()	34
7.1.3.7 get_cell()	34
7.1.3.8 get_col_vec() [1/2]	35
7.1.3.9 get_col_vec() [2/2]	35
7.1.3.10 get_entries()	35
7.1.3.11 get_row_vec() [1/2]	35
7.1.3.12 get_row_vec() [2/2]	35
7.1.3.13 insert_cell() [1/3]	36
7.1.3.14 insert_cell() [2/3]	36
7.1.3.15 insert_cell() [3/3]	36
7.1.3.16 is_empty()	36
7.1.3.17 ncol()	36

7.1.3.18 nnozero()	37
7.1.3.19 nrow()	37
7.1.3.20 operator>() [1/2]	37
7.1.3.21 operator>() [2/2]	37
7.1.3.22 operator*=()	37
7.1.3.23 operator+=() [1/3]	37
7.1.3.24 operator+=() [2/3]	38
7.1.3.25 operator+=() [3/3]	38
7.1.3.26 operator-=() [1/3]	38
7.1.3.27 operator-=() [2/3]	38
7.1.3.28 operator-=() [3/3]	38
7.1.3.29 operator/=()	38
7.1.3.30 operator=() [1/2]	39
7.1.3.31 operator=() [2/2]	39
7.1.3.32 operator==()	39
7.1.3.33 out_of_range()	39
7.1.3.34 print()	39
7.1.3.35 reserve()	39
7.1.3.36 resize()	40
7.1.3.37 rm_cell()	40
7.1.3.38 row()	40
7.1.3.39 set_data()	40
7.1.3.40 swap_cells()	40
7.1.3.41 swap_cols()	41
7.1.3.42 swap_rows()	41
7.1.3.43 toggle_cell()	41
7.1.3.44 toggle_lock()	41
7.1.3.45 transpose()	42
7.1.3.46 zero_col()	42
7.1.3.47 zero_row()	42
7.1.4 Friends And Related Function Documentation	42
7.1.4.1 BArrayCell< Cell_Type, Data_Type >	42
7.1.4.2 BArrayCell_const< Cell_Type, Data_Type >	42
7.1.5 Member Data Documentation	42
7.1.5.1 visited	43
7.2 BArrayCell< Cell_Type, Data_Type > Class Template Reference	43
7.2.1 Detailed Description	43
7.2.2 Constructor & Destructor Documentation	43
7.2.2.1 BArrayCell()	44
7.2.2.2 ~BArrayCell()	44
7.2.3 Member Function Documentation	44
7.2.3.1 operator Cell_Type()	44

7.2.3.2 operator*=()	44
7.2.3.3 operator+=()	44
7.2.3.4 operator-=()	45
7.2.3.5 operator/=()	45
7.2.3.6 operator=()	45
7.2.3.7 operator==()	45
7.3 BArrayCell_const< Cell_Type, Data_Type > Class Template Reference	45
7.3.1 Detailed Description	46
7.3.2 Constructor & Destructor Documentation	46
7.3.2.1 BArrayCell_const()	46
7.3.2.2 ~BArrayCell_const()	46
7.3.3 Member Function Documentation	46
7.3.3.1 operator Cell_Type()	47
7.3.3.2 operator!=()	47
7.3.3.3 operator<()	47
7.3.3.4 operator<=()	47
7.3.3.5 operator==()	47
7.3.3.6 operator>()	48
7.3.3.7 operator>=()	48
7.4 BArrayDense< Cell_Type, Data_Type > Class Template Reference	48
7.4.1 Detailed Description	50
7.4.2 Constructor & Destructor Documentation	51
7.4.2.1 BArrayDense() [1/6]	51
7.4.2.2 BArrayDense() [2/6]	51
7.4.2.3 BArrayDense() [3/6]	51
7.4.2.4 BArrayDense() [4/6]	52
7.4.2.5 BArrayDense() [5/6]	52
7.4.2.6 BArrayDense() [6/6]	52
7.4.2.7 ~BArrayDense()	52
7.4.3 Member Function Documentation	52
7.4.3.1 clear()	53
7.4.3.2 col()	53
7.4.3.3 D() [1/2]	53
7.4.3.4 D() [2/2]	53
7.4.3.5 default_val()	53
7.4.3.6 get_cell()	54
7.4.3.7 get_col_vec() [1/2]	54
7.4.3.8 get_col_vec() [2/2]	54
7.4.3.9 get_entries()	54
7.4.3.10 get_row_vec() [1/2]	55
7.4.3.11 get_row_vec() [2/2]	55
7.4.3.12 insert_cell() [1/3]	55

7.4.3.13 insert_cell() [2/3]	55
7.4.3.14 insert_cell() [3/3]	56
7.4.3.15 is_empty()	56
7.4.3.16 ncol()	56
7.4.3.17 nnozero()	56
7.4.3.18 nrow()	56
7.4.3.19 operator>() [1/2]	57
7.4.3.20 operator>() [2/2]	57
7.4.3.21 operator*=()	57
7.4.3.22 operator+=() [1/3]	57
7.4.3.23 operator+=() [2/3]	57
7.4.3.24 operator+=() [3/3]	58
7.4.3.25 operator-=() [1/3]	58
7.4.3.26 operator-=() [2/3]	58
7.4.3.27 operator-=() [3/3]	58
7.4.3.28 operator/=()	58
7.4.3.29 operator=() [1/2]	59
7.4.3.30 operator=() [2/2]	59
7.4.3.31 operator==()	59
7.4.3.32 out_of_range()	59
7.4.3.33 print()	59
7.4.3.34 reserve()	60
7.4.3.35 resize()	60
7.4.3.36 rm_cell()	60
7.4.3.37 row()	60
7.4.3.38 set_data()	60
7.4.3.39 swap_cells()	61
7.4.3.40 swap_cols()	61
7.4.3.41 swap_rows()	61
7.4.3.42 toggle_cell()	62
7.4.3.43 toggle_lock()	62
7.4.3.44 transpose()	62
7.4.3.45 zero_col()	62
7.4.3.46 zero_row()	62
7.4.4 Friends And Related Function Documentation	63
7.4.4.1 BArrayCell< Cell_Type, Data_Type >	63
7.4.4.2 BArrayCell_const< Cell_Type, Data_Type >	63
7.4.5 Member Data Documentation	63
7.4.5.1 visited	63
7.5 BArrayDenseCell< Cell_Type, Data_Type > Class Template Reference	63
7.5.1 Detailed Description	64
7.5.2 Constructor & Destructor Documentation	64

7.5.2.1 BArrayDenseCell()	64
7.5.2.2 ~BArrayDenseCell()	64
7.5.3 Member Function Documentation	64
7.5.3.1 operator Cell_Type()	65
7.5.3.2 operator*=()	65
7.5.3.3 operator+=()	65
7.5.3.4 operator-=()	65
7.5.3.5 operator/=()	65
7.5.3.6 operator=()	66
7.5.3.7 operator==()	66
7.6 BArrayDenseCell_const< Cell_Type, Data_Type > Class Template Reference	66
7.6.1 Detailed Description	66
7.6.2 Constructor & Destructor Documentation	67
7.6.2.1 BArrayDenseCell_const()	67
7.6.2.2 ~BArrayDenseCell_const()	67
7.6.3 Member Function Documentation	67
7.6.3.1 operator Cell_Type()	67
7.6.3.2 operator!=()	67
7.6.3.3 operator<()	68
7.6.3.4 operator<=()	68
7.6.3.5 operator==()	68
7.6.3.6 operator>()	68
7.6.3.7 operator>=()	68
7.7 BArrayVector< Cell_Type, Data_Type > Class Template Reference	69
7.7.1 Detailed Description	69
7.7.2 Constructor & Destructor Documentation	69
7.7.2.1 BArrayVector()	69
7.7.2.2 ~BArrayVector()	70
7.7.3 Member Function Documentation	70
7.7.3.1 begin()	70
7.7.3.2 end()	70
7.7.3.3 is_col()	71
7.7.3.4 is_row()	71
7.7.3.5 operator std::vector< Cell_Type >()	71
7.7.3.6 operator*=()	71
7.7.3.7 operator+=()	71
7.7.3.8 operator-=()	72
7.7.3.9 operator/=()	72
7.7.3.10 operator=()	72
7.7.3.11 operator==()	72
7.7.3.12 size()	72
7.8 BArrayVector_const< Cell_Type, Data_Type > Class Template Reference	73

7.8.1 Detailed Description	73
7.8.2 Constructor & Destructor Documentation	73
7.8.2.1 BArrayVector_const()	73
7.8.2.2 ~BArrayVector_const()	74
7.8.3 Member Function Documentation	74
7.8.3.1 begin()	74
7.8.3.2 end()	74
7.8.3.3 is_col()	74
7.8.3.4 is_row()	74
7.8.3.5 operator std::vector< Cell_Type >()	74
7.8.3.6 operator!=(())	75
7.8.3.7 operator<()	75
7.8.3.8 operator<=()	75
7.8.3.9 operator==(())	75
7.8.3.10 operator>()	75
7.8.3.11 operator>=()	76
7.8.3.12 size()	76
7.9 Cell< Cell_Type > Class Template Reference	76
7.9.1 Detailed Description	77
7.9.2 Constructor & Destructor Documentation	77
7.9.2.1 Cell() [1/7]	77
7.9.2.2 Cell() [2/7]	77
7.9.2.3 ~Cell()	77
7.9.2.4 Cell() [3/7]	78
7.9.2.5 Cell() [4/7]	78
7.9.2.6 Cell() [5/7]	78
7.9.2.7 Cell() [6/7]	78
7.9.2.8 Cell() [7/7]	78
7.9.3 Member Function Documentation	78
7.9.3.1 add() [1/4]	79
7.9.3.2 add() [2/4]	79
7.9.3.3 add() [3/4]	79
7.9.3.4 add() [4/4]	79
7.9.3.5 operator Cell_Type()	79
7.9.3.6 operator!=(())	79
7.9.3.7 operator=() [1/2]	80
7.9.3.8 operator=() [2/2]	80
7.9.3.9 operator==(())	80
7.9.4 Member Data Documentation	80
7.9.4.1 value	80
7.9.4.2 visited	80
7.10 ConstBArrayRowIter< Cell_Type, Data_Type > Class Template Reference	81

7.10.1 Detailed Description	81
7.10.2 Constructor & Destructor Documentation	81
7.10.2.1 ConstBArrayRowIter()	82
7.10.2.2 ~ConstBArrayRowIter()	82
7.10.3 Member Data Documentation	82
7.10.3.1 Array	82
7.10.3.2 current_col	82
7.10.3.3 current_row	82
7.10.3.4 iter	83
7.11 Counter< Array_Type, Data_Type > Class Template Reference	83
7.11.1 Detailed Description	84
7.11.2 Constructor & Destructor Documentation	84
7.11.2.1 Counter() [1/4]	84
7.11.2.2 Counter() [2/4]	84
7.11.2.3 Counter() [3/4]	85
7.11.2.4 Counter() [4/4]	85
7.11.2.5 ~Counter()	85
7.11.3 Member Function Documentation	85
7.11.3.1 count()	85
7.11.3.2 get_description()	85
7.11.3.3 get_name()	86
7.11.3.4 init()	86
7.11.3.5 operator=() [1/2]	86
7.11.3.6 operator=() [2/2]	86
7.11.4 Member Data Documentation	86
7.11.4.1 count_fun	86
7.11.4.2 data	87
7.11.4.3 delete_data	87
7.11.4.4 desc	87
7.11.4.5 init_fun	87
7.11.4.6 name	87
7.12 Counters< Array_Type, Data_Type > Class Template Reference	88
7.12.1 Detailed Description	88
7.12.2 Constructor & Destructor Documentation	88
7.12.2.1 Counters() [1/3]	89
7.12.2.2 ~Counters()	89
7.12.2.3 Counters() [2/3]	89
7.12.2.4 Counters() [3/3]	89
7.12.3 Member Function Documentation	89
7.12.3.1 add_counter() [1/3]	90
7.12.3.2 add_counter() [2/3]	90
7.12.3.3 add_counter() [3/3]	90

7.12.3.4 clear()	90
7.12.3.5 get_descriptions()	90
7.12.3.6 get_names()	90
7.12.3.7 operator=() [1/2]	90
7.12.3.8 operator=() [2/2]	91
7.12.3.9 operator[]()	91
7.12.3.10 size()	92
7.13 Entries< Cell_Type > Class Template Reference	92
7.13.1 Detailed Description	92
7.13.2 Constructor & Destructor Documentation	93
7.13.2.1 Entries() [1/2]	93
7.13.2.2 Entries() [2/2]	93
7.13.2.3 ~Entries()	93
7.13.3 Member Function Documentation	93
7.13.3.1 resize()	93
7.13.4 Member Data Documentation	93
7.13.4.1 source	94
7.13.4.2 target	94
7.13.4.3 val	94
7.14 Flock Class Reference	94
7.14.1 Detailed Description	95
7.14.2 Constructor & Destructor Documentation	95
7.14.2.1 Flock()	96
7.14.2.2 ~Flock()	96
7.14.3 Member Function Documentation	96
7.14.3.1 add_data()	96
7.14.3.2 colnames()	96
7.14.3.3 get_counters()	97
7.14.3.4 get_model()	97
7.14.3.5 get_support()	97
7.14.3.6 init()	97
7.14.3.7 likelihood_joint()	97
7.14.3.8 nfuncs()	98
7.14.3.9 nleafs()	98
7.14.3.10 nnodes()	98
7.14.3.11 nterms()	98
7.14.3.12 ntrees()	98
7.14.3.13 operator()()	98
7.14.3.14 parse_polytomies()	99
7.14.3.15 print()	99
7.14.3.16 set_seed()	99
7.14.3.17 support_size()	99

7.14.4 Member Data Documentation	100
7.14.4.1 dat	100
7.14.4.2 initialized	100
7.14.4.3 model	100
7.14.4.4 nfunctions	100
7.14.4.5 rengine	100
7.15 FreqTable< T > Class Template Reference	101
7.15.1 Detailed Description	101
7.15.2 Constructor & Destructor Documentation	101
7.15.2.1 FreqTable()	101
7.15.2.2 ~FreqTable()	101
7.15.3 Member Function Documentation	102
7.15.3.1 add()	102
7.15.3.2 as_vector()	102
7.15.3.3 clear()	102
7.15.3.4 get_data()	102
7.15.3.5 get_data_ptr()	102
7.15.3.6 print()	103
7.15.3.7 reserve()	103
7.15.3.8 size()	103
7.16 Geese Class Reference	103
7.16.1 Detailed Description	106
7.16.2 Constructor & Destructor Documentation	106
7.16.2.1 Geese() [1/4]	106
7.16.2.2 Geese() [2/4]	106
7.16.2.3 Geese() [3/4]	106
7.16.2.4 Geese() [4/4]	107
7.16.2.5 ~Geese()	107
7.16.3 Member Function Documentation	107
7.16.3.1 calc_reduced_sequence()	107
7.16.3.2 calc_sequence()	107
7.16.3.3 colnames()	107
7.16.3.4 get_annotated_nodes()	108
7.16.3.5 get_counters()	108
7.16.3.6 get_model()	108
7.16.3.7 get_probabilities()	108
7.16.3.8 get_rengine()	108
7.16.3.9 get_states()	109
7.16.3.10 get_support()	109
7.16.3.11 inherit_support()	109
7.16.3.12 init()	109
7.16.3.13 init_node()	109

7.16.3.14 likelihood()	110
7.16.3.15 likelihood_exhaust()	110
7.16.3.16 nnotations()	110
7.16.3.17 nfuncs()	110
7.16.3.18 nleafs()	110
7.16.3.19 nnodes()	111
7.16.3.20 nterms()	111
7.16.3.21 observed_counts()	111
7.16.3.22 operator=() [1/2]	111
7.16.3.23 operator=() [2/2]	111
7.16.3.24 parse_polytomies()	111
7.16.3.25 predict()	112
7.16.3.26 predict_backend()	112
7.16.3.27 predict_exhaust()	112
7.16.3.28 predict_exhaust_backend()	112
7.16.3.29 predict_sim()	112
7.16.3.30 print()	113
7.16.3.31 print_observed_counts()	113
7.16.3.32 set_seed()	113
7.16.3.33 simulate()	113
7.16.3.34 support_size()	113
7.16.3.35 update_annotations()	114
7.16.4 Member Data Documentation	114
7.16.4.1 delete_engine	114
7.16.4.2 delete_support	114
7.16.4.3 initialized	114
7.16.4.4 map_to_nodes	114
7.16.4.5 nfunctions	115
7.16.4.6 nodes	115
7.16.4.7 reduced_sequence	115
7.16.4.8 sequence	115
7.17 Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference	115
7.17.1 Detailed Description	117
7.17.2 Constructor & Destructor Documentation	119
7.17.2.1 Model() [1/3]	119
7.17.2.2 Model() [2/3]	119
7.17.2.3 Model() [3/3]	119
7.17.2.4 ~Model()	119
7.17.3 Member Function Documentation	120
7.17.3.1 add_array()	120
7.17.3.2 add_counter() [1/3]	120

7.17.3.3 add_counter() [2/3]	120
7.17.3.4 add_counter() [3/3]	121
7.17.3.5 add_rule() [1/3]	121
7.17.3.6 add_rule() [2/3]	121
7.17.3.7 add_rule() [3/3]	121
7.17.3.8 add_rule_dyn() [1/3]	121
7.17.3.9 add_rule_dyn() [2/3]	122
7.17.3.10 add_rule_dyn() [3/3]	122
7.17.3.11 colnames()	122
7.17.3.12 conditional_prob()	122
7.17.3.13 gen_key()	123
7.17.3.14 get_counters()	123
7.17.3.15 get_norm_const()	123
7.17.3.16 get_pset()	123
7.17.3.17 get_pset_stats()	124
7.17.3.18 get_engine()	124
7.17.3.19 get_rules()	124
7.17.3.20 get_rules_dyn()	124
7.17.3.21 get_support()	124
7.17.3.22 likelihood() [1/3]	125
7.17.3.23 likelihood() [2/3]	125
7.17.3.24 likelihood() [3/3]	125
7.17.3.25 likelihood_total()	125
7.17.3.26 nterms()	126
7.17.3.27 operator=()	126
7.17.3.28 print()	126
7.17.3.29 print_stats()	126
7.17.3.30 sample() [1/2]	126
7.17.3.31 sample() [2/2]	127
7.17.3.32 set_counters()	127
7.17.3.33 set_keygen()	127
7.17.3.34 set_engine()	127
7.17.3.35 set_rules()	127
7.17.3.36 set_rules_dyn()	128
7.17.3.37 set_seed()	128
7.17.3.38 size()	128
7.17.3.39 size_unique()	128
7.17.3.40 store_psets()	128
7.17.3.41 support_size()	129
7.18 NetCounterData Class Reference	129
7.18.1 Detailed Description	129
7.18.2 Constructor & Destructor Documentation	129

7.18.2.1 NetCounterData() [1/2]	129
7.18.2.2 NetCounterData() [2/2]	130
7.18.2.3 ~NetCounterData()	130
7.18.3 Member Data Documentation	130
7.18.3.1 indices	130
7.18.3.2 numbers	130
7.19 NetworkData Class Reference	130
7.19.1 Detailed Description	131
7.19.2 Constructor & Destructor Documentation	131
7.19.2.1 NetworkData() [1/3]	131
7.19.2.2 NetworkData() [2/3]	131
7.19.2.3 NetworkData() [3/3]	132
7.19.2.4 ~NetworkData()	132
7.19.3 Member Data Documentation	132
7.19.3.1 directed	132
7.19.3.2 vertex_attr	133
7.20 Node Class Reference	133
7.20.1 Detailed Description	134
7.20.2 Constructor & Destructor Documentation	134
7.20.2.1 Node() [1/5]	134
7.20.2.2 Node() [2/5]	135
7.20.2.3 Node() [3/5]	135
7.20.2.4 Node() [4/5]	135
7.20.2.5 Node() [5/5]	135
7.20.2.6 ~Node()	135
7.20.3 Member Function Documentation	135
7.20.3.1 get_parent()	136
7.20.3.2 is_leaf()	136
7.20.3.3 noffspring()	136
7.20.4 Member Data Documentation	136
7.20.4.1 annotations	136
7.20.4.2 array	136
7.20.4.3 arrays	137
7.20.4.4 duplication	137
7.20.4.5 id	137
7.20.4.6 narray	137
7.20.4.7 offspring	137
7.20.4.8 ord	138
7.20.4.9 parent	138
7.20.4.10 probability	138
7.20.4.11 subtree_prob	138
7.20.4.12 visited	138

7.21 NodeData Class Reference	139
7.21.1 Detailed Description	139
7.21.2 Constructor & Destructor Documentation	139
7.21.2.1 NodeData()	139
7.21.3 Member Data Documentation	139
7.21.3.1 blengths	140
7.21.3.2 duplication	140
7.21.3.3 states	140
7.22 PhyloCounterData Class Reference	140
7.22.1 Detailed Description	141
7.22.2 Constructor & Destructor Documentation	141
7.22.2.1 PhyloCounterData()	141
7.22.3 Member Function Documentation	141
7.22.3.1 at()	141
7.22.3.2 begin()	141
7.22.3.3 empty()	141
7.22.3.4 end()	142
7.22.3.5 get_counters()	142
7.22.3.6 operator()()	142
7.22.3.7 push_back()	142
7.22.3.8 reserve()	142
7.22.3.9 shrink_to_fit()	142
7.22.3.10 size()	143
7.23 PhyloRuleDynData Class Reference	143
7.23.1 Detailed Description	143
7.23.2 Constructor & Destructor Documentation	143
7.23.2.1 PhyloRuleDynData()	143
7.23.2.2 ~PhyloRuleDynData()	144
7.23.3 Member Data Documentation	144
7.23.3.1 counts	144
7.23.3.2 duplication	144
7.23.3.3 lb	144
7.23.3.4 pos	144
7.23.3.5 ub	144
7.24 PowerSet< Array_Type, Data_Rule_Type > Class Template Reference	145
7.24.1 Detailed Description	146
7.24.2 Constructor & Destructor Documentation	146
7.24.2.1 PowerSet() [1/3]	146
7.24.2.2 PowerSet() [2/3]	146
7.24.2.3 PowerSet() [3/3]	147
7.24.2.4 ~PowerSet()	147
7.24.3 Member Function Documentation	147

7.24.3.1 add_rule() [1/3]	147
7.24.3.2 add_rule() [2/3]	147
7.24.3.3 add_rule() [3/3]	147
7.24.3.4 begin()	148
7.24.3.5 calc()	148
7.24.3.6 end()	148
7.24.3.7 get_data()	148
7.24.3.8 get_data_ptr()	148
7.24.3.9 init_support()	149
7.24.3.10 operator[]()	149
7.24.3.11 reset()	149
7.24.3.12 size()	149
7.24.4 Member Data Documentation	149
7.24.4.1 coordinates_free	149
7.24.4.2 coordinates_locked	150
7.24.4.3 data	150
7.24.4.4 EmptyArray	150
7.24.4.5 M	150
7.24.4.6 N	150
7.24.4.7 rules	151
7.24.4.8 rules_deleted	151
7.25 Rule< Array_Type, Data_Type > Class Template Reference	151
7.25.1 Detailed Description	152
7.25.2 Constructor & Destructor Documentation	152
7.25.2.1 Rule() [1/2]	152
7.25.2.2 Rule() [2/2]	152
7.25.2.3 ~Rule()	152
7.25.3 Member Function Documentation	153
7.25.3.1 D()	153
7.25.3.2 operator()()	153
7.26 Rules< Array_Type, Data_Type > Class Template Reference	153
7.26.1 Detailed Description	154
7.26.2 Constructor & Destructor Documentation	154
7.26.2.1 Rules() [1/2]	154
7.26.2.2 Rules() [2/2]	154
7.26.2.3 ~Rules()	155
7.26.3 Member Function Documentation	155
7.26.3.1 add_rule() [1/3]	155
7.26.3.2 add_rule() [2/3]	155
7.26.3.3 add_rule() [3/3]	155
7.26.3.4 clear()	155
7.26.3.5 get_seq()	155

7.26.3.6 operator>()	156
7.26.3.7 operator=()	156
7.26.3.8 size()	157
7.27 StatsCounter< Array_Type, Data_Type > Class Template Reference	157
7.27.1 Detailed Description	157
7.27.2 Constructor & Destructor Documentation	158
7.27.2.1 StatsCounter() [1/2]	158
7.27.2.2 StatsCounter() [2/2]	158
7.27.2.3 ~StatsCounter()	158
7.27.3 Member Function Documentation	158
7.27.3.1 add_counter() [1/2]	158
7.27.3.2 add_counter() [2/2]	159
7.27.3.3 count_all()	159
7.27.3.4 count_current()	159
7.27.3.5 count_init()	159
7.27.3.6 get_counters()	159
7.27.3.7 get_descriptions()	159
7.27.3.8 get_names()	160
7.27.3.9 reset_array()	160
7.27.3.10 set_counters()	160
7.28 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference	160
7.28.1 Detailed Description	162
7.28.2 Constructor & Destructor Documentation	162
7.28.2.1 Support() [1/3]	162
7.28.2.2 Support() [2/3]	163
7.28.2.3 Support() [3/3]	163
7.28.2.4 ~Support()	163
7.28.3 Member Function Documentation	163
7.28.3.1 add_counter() [1/2]	163
7.28.3.2 add_counter() [2/2]	164
7.28.3.3 add_rule() [1/2]	164
7.28.3.4 add_rule() [2/2]	164
7.28.3.5 add_rule_dyn() [1/2]	164
7.28.3.6 add_rule_dyn() [2/2]	164
7.28.3.7 calc()	165
7.28.3.8 eval_rules_dyn()	166
7.28.3.9 get_counters()	166
7.28.3.10 get_counts()	166
7.28.3.11 get_counts_ptr()	166
7.28.3.12 get_current_stats()	167
7.28.3.13 get_data()	167

7.28.3.14	get_rules()	167
7.28.3.15	get_rules_dyn()	167
7.28.3.16	init_support()	167
7.28.3.17	print()	168
7.28.3.18	reset_array() [1/2]	168
7.28.3.19	reset_array() [2/2]	168
7.28.3.20	set_counters()	168
7.28.3.21	set_rules()	168
7.28.3.22	set_rules_dyn()	169
7.28.4	Member Data Documentation	169
7.28.4.1	change_stats	169
7.28.4.2	coordinates_free	169
7.28.4.3	coordinates_locked	169
7.28.4.4	current_stats	170
7.28.4.5	delete_counters	170
7.28.4.6	delete_rules	170
7.28.4.7	delete_rules_dyn	170
7.28.4.8	M	170
7.28.4.9	max_num_elements	171
7.28.4.10	N	171
7.29	vecHasher< T > Struct Template Reference	171
7.29.1	Detailed Description	171
7.29.2	Member Function Documentation	171
7.29.2.1	operator()()	171
8	File Documentation	173
8.1	include/barry/barray-bones.hpp File Reference	173
8.1.1	Macro Definition Documentation	174
8.1.1.1	BARRAY_BONES_HPP	174
8.2	include/barry/barray-iterator.hpp File Reference	174
8.3	include/barry/barray-meat-operators.hpp File Reference	175
8.3.1	Macro Definition Documentation	176
8.3.1.1	BARRY_BARRAY_MEAT_OPERATORS_HPP	176
8.3.1.2	COL	176
8.3.1.3	ROW	176
8.3.2	Function Documentation	176
8.3.2.1	checkdim_()	176
8.4	include/barry/barray-meat.hpp File Reference	177
8.4.1	Macro Definition Documentation	178
8.4.1.1	COL	178
8.4.1.2	ROW	178
8.5	include/barry/barraycell-bones.hpp File Reference	178

8.6 include/barry/barraycell-meat.hpp File Reference	179
8.7 include/barry/barraydense-bones.hpp File Reference	180
8.8 include/barry/barraydense-meat.hpp File Reference	181
8.8.1 Macro Definition Documentation	182
8.8.1.1 BARRY_BARRAYDENSE_MEAT_HPP	182
8.8.1.2 COL	183
8.8.1.3 POS	183
8.8.1.4 ROW	183
8.8.1.5 ZERO_CELL	183
8.9 include/barry/barraydensecell-bones.hpp File Reference	183
8.10 include/barry/barraydensecell-meat.hpp File Reference	184
8.10.1 Macro Definition Documentation	185
8.10.1.1 BARRY_BARRAYDENSECELL_MEAT_HPP	185
8.10.1.2 POS	185
8.11 include/barry/barrayvector-bones.hpp File Reference	185
8.12 include/barry/barrayvector-meat.hpp File Reference	186
8.12.1 Macro Definition Documentation	187
8.12.1.1 BARRY_BARRAYVECTOR_MEAT_HPP	187
8.13 include/barry/barry-configuration.hpp File Reference	187
8.13.1 Macro Definition Documentation	188
8.13.1.1 BARRY_CHECK_SUPPORT	188
8.13.1.2 BARRY_ISFINITE	188
8.13.1.3 BARRY_MAX_NUM_ELEMENTS	188
8.13.1.4 BARRY_SAFE_EXP	188
8.13.1.5 printf_barry	188
8.13.2 Typedef Documentation	188
8.13.2.1 Map	189
8.14 include/barry/barry.hpp File Reference	189
8.14.1 Macro Definition Documentation	190
8.14.1.1 BARRY_HPP	190
8.14.1.2 BARRY_VERSION	190
8.14.1.3 COUNTER_FUNCTION	190
8.14.1.4 COUNTER_LAMBDA	191
8.14.1.5 RULE_FUNCTION	191
8.14.1.6 RULE_LAMBDA	191
8.15 include/barry/cell-bones.hpp File Reference	191
8.16 include/barry/cell-meat.hpp File Reference	192
8.17 include/barry/col-bones.hpp File Reference	193
8.18 include/barry/counters-bones.hpp File Reference	193
8.19 include/barry/counters-meat.hpp File Reference	194
8.19.1 Macro Definition Documentation	197
8.19.1.1 COUNTER_TEMPLATE	197

8.19.1.2 COUNTER_TEMPLATE_ARGS	197
8.19.1.3 COUNTER_TYPE	197
8.19.1.4 COUNTERS_TEMPLATE	198
8.19.1.5 COUNTERS_TEMPLATE_ARGS	198
8.19.1.6 COUNTERS_TYPE	198
8.19.2 Function Documentation	198
8.19.2.1 count_fun()	198
8.19.2.2 COUNTER_TEMPLATE() [1/7]	198
8.19.2.3 COUNTER_TEMPLATE() [2/7]	199
8.19.2.4 COUNTER_TEMPLATE() [3/7]	199
8.19.2.5 COUNTER_TEMPLATE() [4/7]	199
8.19.2.6 COUNTER_TEMPLATE() [5/7]	199
8.19.2.7 COUNTER_TEMPLATE() [6/7]	199
8.19.2.8 COUNTER_TEMPLATE() [7/7]	199
8.19.2.9 COUNTERS_TEMPLATE() [1/8]	200
8.19.2.10 COUNTERS_TEMPLATE() [2/8]	200
8.19.2.11 COUNTERS_TEMPLATE() [3/8]	200
8.19.2.12 COUNTERS_TEMPLATE() [4/8]	200
8.19.2.13 COUNTERS_TEMPLATE() [5/8]	200
8.19.2.14 COUNTERS_TEMPLATE() [6/8]	201
8.19.2.15 COUNTERS_TEMPLATE() [7/8]	201
8.19.2.16 COUNTERS_TEMPLATE() [8/8]	201
8.19.2.17 data()	201
8.19.2.18 delete_data() [1/3]	201
8.19.2.19 delete_data() [2/3]	201
8.19.2.20 delete_data() [3/3]	202
8.19.2.21 delete_to_be_deleted() [1/2]	202
8.19.2.22 delete_to_be_deleted() [2/2]	202
8.19.2.23 desc()	202
8.19.2.24 init_fun() [1/3]	202
8.19.2.25 init_fun() [2/3]	203
8.19.2.26 init_fun() [3/3]	203
8.19.2.27 name()	203
8.19.2.28 push_back() [1/2]	203
8.19.2.29 push_back() [2/2]	203
8.19.2.30 to_be_deleted() [1/2]	203
8.19.2.31 to_be_deleted() [2/2]	203
8.19.3 Variable Documentation	204
8.19.3.1 count_fun_	204
8.19.3.2 counter	204
8.19.3.3 counter_	204
8.19.3.4 data_	204

8.19.3.5 delete_data_	205
8.19.3.6 desc_	205
8.19.3.7 i	205
8.19.3.8 init_fun_	205
8.19.3.9 j	205
8.19.3.10 name_	206
8.19.3.11 noexcept	206
8.19.3.12 return	206
8.20 include/barry/counters/network.hpp File Reference	206
8.20.1 Macro Definition Documentation	209
8.20.1.1 NET_C_DATA_IDX	209
8.20.1.2 NET_C_DATA_NUM	210
8.20.1.3 NETWORK_COUNTER	210
8.20.1.4 NETWORK_COUNTER_LAMBDA	210
8.20.1.5 NETWORK_RULE	210
8.20.1.6 NETWORK_RULE_LAMBDA	211
8.20.2 Typedef Documentation	211
8.20.2.1 NetCounter	211
8.20.2.2 NetCounters	211
8.20.2.3 NetModel	211
8.20.2.4 NetRule	211
8.20.2.5 NetRules	212
8.20.2.6 NetStatsCounter	212
8.20.2.7 NetSupport	212
8.20.2.8 Network	212
8.20.3 Function Documentation	212
8.20.3.1 rules_zerodiag()	212
8.21 include/barry/counters/phylo.hpp File Reference	213
8.21.1 Macro Definition Documentation	215
8.21.1.1 PHYLO_CHECK_MISSING	215
8.21.1.2 PHYLO_COUNTER_LAMBDA	215
8.21.1.3 PHYLO_RULE_DYN_LAMBDA	216
8.21.2 Typedef Documentation	216
8.21.2.1 PhyloArray	216
8.21.2.2 PhyloCounter	216
8.21.2.3 PhyloCounters	216
8.21.2.4 PhyloModel	216
8.21.2.5 PhyloPowerSet	217
8.21.2.6 PhyloRule	217
8.21.2.7 PhyloRuleData	217
8.21.2.8 PhyloRuleDyn	217
8.21.2.9 PhyloRules	217

8.21.2.10 PhyloRulesDyn	217
8.21.2.11 PhyloStatsCounter	218
8.21.2.12 PhyloSupport	218
8.21.3 Function Documentation	218
8.21.3.1 get_last_name()	218
8.22 include/barry/model-bones.hpp File Reference	218
8.22.1 Function Documentation	219
8.22.1.1 keygen_default()	219
8.23 include/barry/model-meat.hpp File Reference	220
8.23.1 Macro Definition Documentation	220
8.23.1.1 MODEL_TEMPLATE	220
8.23.1.2 MODEL_TEMPLATE_ARGS	221
8.23.1.3 MODEL_TYPE	221
8.23.2 Function Documentation	221
8.23.2.1 likelihood_()	221
8.23.2.2 MODEL_TEMPLATE() [1/2]	221
8.23.2.3 MODEL_TEMPLATE() [2/2]	222
8.23.2.4 update_normalizing_constant()	222
8.24 include/barry/models/geese.hpp File Reference	222
8.25 include/barry/models/geese/flock-bones.hpp File Reference	223
8.26 include/barry/models/geese/flock-meet.hpp File Reference	223
8.27 include/barry/models/geese/geese-bones.hpp File Reference	224
8.27.1 Macro Definition Documentation	224
8.27.1.1 INITIALIZED	225
8.27.2 Function Documentation	225
8.27.2.1 keygen_full()	225
8.27.2.2 RULE_FUNCTION()	225
8.27.2.3 vec_diff()	225
8.27.2.4 vector_caster()	225
8.28 include/barry/models/geese/geese-meat-constructors.hpp File Reference	226
8.29 include/barry/models/geese/geese-meat-likelihood.hpp File Reference	226
8.30 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference	227
8.31 include/barry/models/geese/geese-meat-predict.hpp File Reference	228
8.32 include/barry/models/geese/geese-meat-predict_exhaust.hpp File Reference	228
8.33 include/barry/models/geese/geese-meat-predict_sim.hpp File Reference	229
8.34 include/barry/models/geese/geese-meat-simulate.hpp File Reference	229
8.35 include/barry/models/geese/geese-meat.hpp File Reference	230
8.36 include/barry/models/geese/geese-node-bones.hpp File Reference	230
8.37 include/barry/powerset-bones.hpp File Reference	231
8.38 include/barry/powerset-meat.hpp File Reference	232
8.39 include/barry/rules-bones.hpp File Reference	234
8.39.1 Function Documentation	235

8.39.1.1 rule_fun_default()	235
8.40 include/barry/rules-meat.hpp File Reference	235
8.41 include/barry/statscounter-bones.hpp File Reference	236
8.42 include/barry/statscounter-meat.hpp File Reference	237
8.42.1 Macro Definition Documentation	239
8.42.1.1 STATSCOUNTER_TEMPLATE	239
8.42.1.2 STATSCOUNTER_TEMPLATE_ARGS	239
8.42.1.3 STATSCOUNTER_TYPE	239
8.42.2 Function Documentation	240
8.42.2.1 for()	240
8.42.2.2 resize()	240
8.42.2.3 STATSCOUNTER_TEMPLATE() [1/9]	240
8.42.2.4 STATSCOUNTER_TEMPLATE() [2/9]	240
8.42.2.5 STATSCOUNTER_TEMPLATE() [3/9]	240
8.42.2.6 STATSCOUNTER_TEMPLATE() [4/9]	241
8.42.2.7 STATSCOUNTER_TEMPLATE() [5/9]	241
8.42.2.8 STATSCOUNTER_TEMPLATE() [6/9]	241
8.42.2.9 STATSCOUNTER_TEMPLATE() [7/9]	241
8.42.2.10 STATSCOUNTER_TEMPLATE() [8/9]	241
8.42.2.11 STATSCOUNTER_TEMPLATE() [9/9]	241
8.42.3 Variable Documentation	242
8.42.3.1 counter_deleted	242
8.42.3.2 counters	242
8.42.3.3 counters_	242
8.42.3.4 f_	242
8.42.3.5 j	243
8.42.3.6 return	243
8.43 include/barry/statsdb.hpp File Reference	243
8.44 include/barry/support-bones.hpp File Reference	244
8.45 include/barry/support-meat.hpp File Reference	245
8.45.1 Macro Definition Documentation	247
8.45.1.1 BARRY_SUPPORT_MEAT_HPP	248
8.45.1.2 SUPPORT_TEMPLATE	248
8.45.1.3 SUPPORT_TEMPLATE_ARGS	248
8.45.1.4 SUPPORT_TYPE	248
8.45.2 Function Documentation	248
8.45.2.1 calc_backend()	249
8.45.2.2 for()	249
8.45.2.3 if() [1/3]	249
8.45.2.4 if() [2/3]	249
8.45.2.5 if() [3/3]	249
8.45.2.6 insert_cell()	249

8.45.2.7 rm_cell()	250
8.45.2.8 SUPPORT_TEMPLATE() [1/17]	250
8.45.2.9 SUPPORT_TEMPLATE() [2/17]	250
8.45.2.10 SUPPORT_TEMPLATE() [3/17]	250
8.45.2.11 SUPPORT_TEMPLATE() [4/17]	250
8.45.2.12 SUPPORT_TEMPLATE() [5/17]	251
8.45.2.13 SUPPORT_TEMPLATE() [6/17]	251
8.45.2.14 SUPPORT_TEMPLATE() [7/17]	251
8.45.2.15 SUPPORT_TEMPLATE() [8/17]	251
8.45.2.16 SUPPORT_TEMPLATE() [9/17]	251
8.45.2.17 SUPPORT_TEMPLATE() [10/17]	251
8.45.2.18 SUPPORT_TEMPLATE() [11/17]	252
8.45.2.19 SUPPORT_TEMPLATE() [12/17]	252
8.45.2.20 SUPPORT_TEMPLATE() [13/17]	252
8.45.2.21 SUPPORT_TEMPLATE() [14/17]	252
8.45.2.22 SUPPORT_TEMPLATE() [15/17]	252
8.45.2.23 SUPPORT_TEMPLATE() [16/17]	253
8.45.2.24 SUPPORT_TEMPLATE() [17/17]	253
8.45.3 Variable Documentation	253
8.45.3.1 array_bank	253
8.45.3.2 cfree	253
8.45.3.3 counters	253
8.45.3.4 counters_	254
8.45.3.5 delete_counters	254
8.45.3.6 delete_rules	254
8.45.3.7 delete_rules_dyn	254
8.45.3.8 else	254
8.45.3.9 f_	255
8.45.3.10 return	255
8.45.3.11 rules	255
8.45.3.12 rules_	255
8.45.3.13 rules_dyn	255
8.45.3.14 stats_bank	256
8.46 include/barry/typedefs.hpp File Reference	256
8.46.1 Typedef Documentation	258
8.46.1.1 Col_type	258
8.46.1.2 Counter_fun_type	258
8.46.1.3 Counts_type	258
8.46.1.4 MapVec_type	258
8.46.1.5 Row_type	259
8.46.1.6 Rule_fun_type	259
8.46.1.7 uint	259

8.46.2 Function Documentation	259
8.46.2.1 <code>vec_equal()</code>	259
8.46.2.2 <code>vec_equal_approx()</code>	260
8.46.2.3 <code>vec_inner_prod()</code>	260
8.47 README.md File Reference	260
Index	261

Chapter 1

Main Page

Barry: your to-go motif accountant

This repository contains a C++ template library that essentially counts sufficient statistics on binary arrays. The idea of the library is that this can be used together to build exponential family models as those in Exponential Random Graph Models (ERGMs), but as a generalization that also deals with non square arrays.

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?
    std::cout << "Current view" << std::endl;
    net.print();

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

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

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

```

netcounters::counter_ttriads(counter.counters);
netcounters::counter_isolates(counter.counters);
netcounters::counter_ctriads(counter.counters);
netcounters::counter_mutual(counter.counters);

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

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

return 0;
}

```

Compiling this program using g++

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

Yields the following output:

```

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

```

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 statistical models modules avoid double support when possible by keeping track of what datasets (networks, for instance) share the same support.

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	9
Statistical Models	9
Network counters	10
Phylo counters	18
Phylo rules	23

Chapter 3

Class Index

3.1 Class List

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

BArray< Cell_Type, Data_Type >	
Baseline class for binary arrays	29
BArrayCell< Cell_Type, Data_Type >	43
BArrayCell_const< Cell_Type, Data_Type >	45
BArrayDense< Cell_Type, Data_Type >	
Baseline class for binary arrays	48
BArrayDenseCell< Cell_Type, Data_Type >	63
BArrayDenseCell_const< Cell_Type, Data_Type >	66
BArrayVector< Cell_Type, Data_Type >	
Row or column of a BArray	69
BArrayVector_const< Cell_Type, Data_Type >	73
Cell< Cell_Type >	
Entries in BArray . For now, it only has two members:	76
ConstBArrayRowIter< Cell_Type, Data_Type >	81
Counter< Array_Type, Data_Type >	
A counter function based on change statistics	83
Counters< Array_Type, Data_Type >	
Vector of counters	88
Entries< Cell_Type >	
A wrapper class to store <code>source</code> , <code>target</code> , <code>val</code> from a BArray object	92
Flock	
A Flock is a group of Geese	94
FreqTable< T >	
Database of statistics	101
Geese	
Annotated Phylo Model	103
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:	115
NetCounterData	
Data class used to store arbitrary uint or double vectors	129
NetworkData	
Data class for Networks	130
Node	
A single node for the model	133

NodeData	
Data definition for the <code>PhyloArray</code> class	139
PhyloCounterData	140
PhyloRuleDynData	143
PowerSet< Array_Type, Data_Rule_Type >	
Powerset of a binary array	145
Rule< Array_Type, Data_Type >	
Rule for determining if a cell should be included in a sequence	151
Rules< Array_Type, Data_Type >	
Vector of objects of class <code>Rule</code>	153
StatsCounter< Array_Type, Data_Type >	
Count stats for a single <code>Array</code>	157
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >	
Compute the support of sufficient statistics	160
vecHasher< T >	171

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

include/barry/barray-bones.hpp	173
include/barry/barray-iterator.hpp	174
include/barry/barray-meat-operators.hpp	175
include/barry/barray-meat.hpp	177
include/barry/barraycell-bones.hpp	178
include/barry/barraycell-meat.hpp	179
include/barry/barraydense-bones.hpp	180
include/barry/barraydense-meat.hpp	181
include/barry/barraydensecell-bones.hpp	183
include/barry/barraydensecell-meat.hpp	184
include/barry/barrayvector-bones.hpp	185
include/barry/barrayvector-meat.hpp	186
include/barry/barray-configuration.hpp	187
include/barry/barray.hpp	189
include/barry/cell-bones.hpp	191
include/barry/cell-meat.hpp	192
include/barry/col-bones.hpp	193
include/barry/counters-bones.hpp	193
include/barry/counters-meat.hpp	194
include/barry/model-bones.hpp	218
include/barry/model-meat.hpp	220
include/barry/powerset-bones.hpp	231
include/barry/powerset-meat.hpp	232
include/barry/rules-bones.hpp	234
include/barry/rules-meat.hpp	235
include/barry/statscounter-bones.hpp	236
include/barry/statscounter-meat.hpp	237
include/barry/statsdb.hpp	243
include/barry/support-bones.hpp	244
include/barry/support-meat.hpp	245
include/barry/typedefs.hpp	256
include/barry/counters/network.hpp	206
include/barry/counters/phylo.hpp	213
include/barry/models/geese.hpp	222
include/barry/models/geese/flock-bones.hpp	223

include/barry/models/geese/flock-meet.hpp	223
include/barry/models/geese/geese-bones.hpp	224
include/barry/models/geese/geese-meat-constructors.hpp	226
include/barry/models/geese/geese-meat-likelihood.hpp	226
include/barry/models/geese/geese-meat-likelihood_exhaust.hpp	227
include/barry/models/geese/geese-meat-predict.hpp	228
include/barry/models/geese/geese-meat-predict_exhaust.hpp	228
include/barry/models/geese/geese-meat-predict_sim.hpp	229
include/barry/models/geese/geese-meat-simulate.hpp	229
include/barry/models/geese/geese-meat.hpp	230
include/barry/models/geese/geese-node-bones.hpp	230

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 empty to have a one, is defined as

$$\delta(y_{ij}) = s_{ij}^+(y) - s_{ij}^-(y),$$

where $s_{ij}^+(y)$ and $s_{ij}^-(y)$ represent the motif statistic with and without the ij -cell. For example, in the case of networks, the change statistic for the number of edges is always 1.

To count statistics in an array, the [\[Counter\]](#) class will empty the array, initialize the counters, and then start counting while adding at each step a single cell, until matching the original array.

5.2 Statistical Models

Statistical models available in `barry`.

Classes

- class [Model](#)< [Array_Type](#), [Data_Counter_Type](#), [Data_Rule_Type](#), [Data_Rule_Dyn_Type](#) >
General framework for discrete exponential models. This class allows generating discrete exponential models in the form of a linear exponential model:
- class [Flock](#)
A [Flock](#) is a group of [Geese](#).
- class [Geese](#)
Annotated Phylo [Model](#).

5.2.1 Detailed Description

Statistical models available in `barry`.

5.3 Network counters

[Counters](#) for network models.

Macros

- `#define CSS_SIZE()`
- `#define CSS_CASE_TRUTH() if ((i < n) && (j < n))`
- `#define CSS_CASE_PERCEIVED() else if (((i >= s) && (i < e)) & ((j >= s) && (j < e)))`
- `#define CSS_CASE_ELSE()`
- `#define CSS_CHECK_SIZE_INIT()`
- `#define CSS_CHECK_SIZE()`
- `#define CSS_APPEND(name)`

Functions

- void [counter_edges](#) ([NetCounters](#) *counters)
Number of edges.
- void [counter_isolates](#) ([NetCounters](#) *counters)
Number of isolated vertices.
- void [counter_mutual](#) ([NetCounters](#) *counters)
Number of mutual ties.
- void [counter_istar2](#) ([NetCounters](#) *counters)
- void [counter_ostar2](#) ([NetCounters](#) *counters)
- void [counter_ttriads](#) ([NetCounters](#) *counters)
- void [counter_ctriads](#) ([NetCounters](#) *counters)
- void [counter_density](#) ([NetCounters](#) *counters)
- void [counter_idegree15](#) ([NetCounters](#) *counters)
- void [counter_odegree15](#) ([NetCounters](#) *counters)
- void [counter_absdiff](#) ([NetCounters](#) *counters, uint attr_id, double alpha=1.0)
Sum of absolute attribute difference between ego and alter.
- void [counter_diff](#) ([NetCounters](#) *counters, uint attr_id, double alpha=1.0, double tail_head=true)
Sum of attribute difference between ego and alter to pow(alpha)
- [NETWORK_COUNTER](#) (init_single_attr)

- void `counter_nodeicov` (`NetCounters *counters`, `uint attr_id`)
- void `counter_nodeocov` (`NetCounters *counters`, `uint attr_id`)
- void `counter_nodcov` (`NetCounters *counters`, `uint attr_id`)
- void `counter_nodematch` (`NetCounters *counters`, `uint attr_id`)
- void `counter_iddegree` (`NetCounters *counters`, `std::vector< uint > d`)
Counts number of vertices with a given in-degree.
- void `counter_odegree` (`NetCounters *counters`, `std::vector< uint > d`)
Counts number of vertices with a given out-degree.
- void `counter_degree` (`NetCounters *counters`, `std::vector< uint > d`)
Counts number of vertices with a given out-degree.
- void `counter_css_partially_false_recip_commi` (`NetCounters *counters`, `uint netsize`, `const std::vector< uint > &end_`)
Counts errors of commission.
- void `counter_css_partially_false_recip_omiss` (`NetCounters *counters`, `uint netsize`, `const std::vector< uint > &end_`)
Counts errors of omission.
- void `counter_css_completely_false_recip_comiss` (`NetCounters *counters`, `uint netsize`, `const std::vector< uint > &end_`)
Counts completely false reciprocity (comission)
- void `counter_css_completely_false_recip_omiss` (`NetCounters *counters`, `uint netsize`, `const std::vector< uint > &end_`)
Counts completely false reciprocity (omission)

5.3.1 Detailed Description

`Counters` for network models.

Parameters

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

5.3.2 Macro Definition Documentation

5.3.2.1 CSS_APPEND

```
#define CSS_APPEND(  
    name )
```

Value:

```
std::string name_ = (name);\nfor (uint i = 0u; i < end_.size(); ++i) { \nstd::string tmpname = name_ + " (" + std::to_string(i) + ")";\ncounters->add_counter(tmp_count, tmp_init,\n    new NetCounterData({netsize, i == 0u ? netsize : end_[i-1], end_[i]}, {}),\n    true, tmpname);}
```

Definition at line 841 of file `network.hpp`.

5.3.2.2 CSS_CASE_ELSE

```
#define CSS_CASE_ELSE( )
```

Definition at line 831 of file network.hpp.

5.3.2.3 CSS_CASE_PERCEIVED

```
#define CSS_CASE_PERCEIVED( ) else if (((i >= s) && (i < e)) & ((j >= s) && (j < e)))
```

Definition at line 830 of file network.hpp.

5.3.2.4 CSS_CASE_TRUTH

```
#define CSS_CASE_TRUTH( ) if ((i < n) && (j < n))
```

Definition at line 829 of file network.hpp.

5.3.2.5 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 836 of file network.hpp.

5.3.2.6 CSS_CHECK_SIZE_INIT

```
#define CSS_CHECK_SIZE_INIT( )
```

Value:

```
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 832 of file network.hpp.

5.3.2.7 CSS_SIZE

```
#define CSS_SIZE( )
```

Value:

```
uint n = data->indices[0u]; \  
uint s = data->indices[1u]; \  
uint e = data->indices[2u];
```

Definition at line 824 of file network.hpp.

5.3.3 Function Documentation

5.3.3.1 counter_absdiff()

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

Sum of absolute attribute difference between ego and alter.

Definition at line 480 of file network.hpp.

5.3.3.2 counter_css_completely_false_recip_comiss()

```
void counter_css_completely_false_recip_comiss (  
    NetCounters * counters,  
    uint netsize,  
    const std::vector< uint > & end_ ) [inline]
```

Counts completely false reciprocity (comission)

Definition at line 960 of file network.hpp.

5.3.3.3 counter_css_completely_false_recip_omiss()

```
void counter_css_completely_false_recip_omiss (  
    NetCounters * counters,  
    uint netsize,  
    const std::vector< uint > & end_ ) [inline]
```

Counts completely false reciprocity (omission)

Definition at line 1016 of file network.hpp.

5.3.3.4 counter_css_partially_false_recip_commi()

```
void counter_css_partially_false_recip_commi (
    NetCounters * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Counts errors of commission.

Definition at line 850 of file network.hpp.

5.3.3.5 counter_css_partially_false_recip_omiss()

```
void counter_css_partially_false_recip_omiss (
    NetCounters * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Counts errors of omission.

Definition at line 904 of file network.hpp.

5.3.3.6 counter_ctriads()

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

Definition at line 365 of file network.hpp.

5.3.3.7 counter_degree()

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

Counts number of vertices with a given out-degree.

Definition at line 777 of file network.hpp.

5.3.3.8 counter_density()

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

Definition at line 413 of file network.hpp.

5.3.3.9 counter_diff()

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

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

Definition at line 524 of file network.hpp.

5.3.3.10 counter_edges()

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

Number of edges.

Definition at line 128 of file network.hpp.

5.3.3.11 counter_idegree()

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

Counts number of vertices with a given in-degree.

Definition at line 680 of file network.hpp.

5.3.3.12 counter_idegree15()

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

Definition at line 434 of file network.hpp.

5.3.3.13 counter_isolates()

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

Number of isolated vertices.

Definition at line 149 of file network.hpp.

5.3.3.14 counter_istar2()

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

Definition at line 234 of file network.hpp.

5.3.3.15 counter_mutual()

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

Number of mutual ties.

Definition at line 187 of file network.hpp.

5.3.3.16 counter_nodecov()

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

Definition at line 630 of file network.hpp.

5.3.3.17 counter_nodeicov()

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

Definition at line 586 of file network.hpp.

5.3.3.18 counter_nodematch()

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

Definition at line 652 of file network.hpp.

5.3.3.19 counter_nodecov()

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

Definition at line 608 of file network.hpp.

5.3.3.20 counter_odegree()

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

Counts number of vertices with a given out-degree.

Definition at line 728 of file network.hpp.

5.3.3.21 counter_odegree15()

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

Definition at line 456 of file network.hpp.

5.3.3.22 counter_ostar2()

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

Definition at line 255 of file network.hpp.

5.3.3.23 counter_ttriads()

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

Definition at line 279 of file network.hpp.

5.3.3.24 NETWORK_COUNTER()

```
NETWORK_COUNTER (
    init_single_attr )
```

Definition at line 568 of file network.hpp.

5.4 Phylo counters

[Counters](#) for phylogenetic modeling.

Functions

- void [counter_overall_gains](#) ([PhyloCounters](#) *counters, bool duplication=true)
Overall functional gains.
- void [counter_gains](#) ([PhyloCounters](#) *counters, std::vector< [uint](#) > nfun, bool duplication=true)
Functional gains for a specific function (nfun).
- void [counter_gains_k_offspring](#) ([PhyloCounters](#) *counters, std::vector< [uint](#) > nfun, [uint](#) k=1u, bool duplication=true)
k genes gain function nfun
- void [counter_genes_changing](#) ([PhyloCounters](#) *counters, bool duplication=true)
Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)
- void [counter_prop_genes_changing](#) ([PhyloCounters](#) *counters, bool duplication=true)
Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)
- void [counter_overall_loss](#) ([PhyloCounters](#) *counters, bool duplication=true)
Overall functional loss.
- void [counter_maxfuns](#) ([PhyloCounters](#) *counters, [uint](#) lb, [uint](#) ub, bool duplication=true)
Cap the number of functions per gene.
- void [counter_loss](#) ([PhyloCounters](#) *counters, std::vector< [uint](#) > nfun, bool duplication=true)
Total count of losses for an specific function.
- void [counter_overall_changes](#) ([PhyloCounters](#) *counters, bool duplication=true)
Total number of changes. Use this statistic to account for "preservation".
- void [counter_subfun](#) ([PhyloCounters](#) *counters, [uint](#) nfunA, [uint](#) nfunB, bool duplication=true)
Total count of Sub-functionalization events.
- void [counter_cogain](#) ([PhyloCounters](#) *counters, [uint](#) nfunA, [uint](#) nfunB, bool duplication=true)
Co-evolution (joint gain or loss)
- void [counter_longest](#) ([PhyloCounters](#) *counters)
Longest branch mutates (either by gain or by loss)
- void [counter_neofun](#) ([PhyloCounters](#) *counters, [uint](#) nfunA, [uint](#) nfunB, bool duplication=true)
Total number of neofunctionalization events.
- void [counter_neofun_a2b](#) ([PhyloCounters](#) *counters, [uint](#) nfunA, [uint](#) nfunB, bool duplication=true)
Total number of neofunctionalization events.
- void [counter_co_opt](#) ([PhyloCounters](#) *counters, [uint](#) nfunA, [uint](#) nfunB, bool duplication=true)
Function co-opting.

5.4.1 Detailed Description

[Counters](#) for phylogenetic modeling.

Parameters

<i>counters</i>	A pointer to a <code>PhyloCounters</code> object (<code>Counters<PhyloArray, PhyloCounterData></code>).
-----------------	---

5.4.2 Function Documentation

5.4.2.1 counter_co_opt()

```
void counter_co_opt (
    PhyloCounters * counters,
    uint nfunA,
    uint nfunB,
    bool duplication = true ) [inline]
```

Function co-opting.

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

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

This algorithm implements the change statistic.

Definition at line 1184 of file phylo.hpp.

5.4.2.2 counter_cogain()

```
void counter_cogain (
    PhyloCounters * counters,
    uint nfunA,
    uint nfunB,
    bool duplication = true ) [inline]
```

Co-evolution (joint gain or loss)

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

Definition at line 814 of file phylo.hpp.

5.4.2.3 counter_gains()

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

Functional gains for a specific function (nfun).

Definition at line 179 of file phylo.hpp.

5.4.2.4 counter_gains_k_offspring()

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

k genes gain function nfun

Definition at line 222 of file phylo.hpp.

5.4.2.5 counter_genes_changing()

```
void counter_genes_changing (
    PhyloCounters * counters,
    bool duplication = true ) [inline]
```

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

Definition at line 296 of file phylo.hpp.

5.4.2.6 counter_longest()

```
void counter_longest (
    PhyloCounters * counters ) [inline]
```

Longest branch mutates (either by gain or by loss)

Definition at line 873 of file phylo.hpp.

5.4.2.7 counter_loss()

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

Total count of losses for an specific function.

Definition at line 612 of file phylo.hpp.

5.4.2.8 counter_maxfuns()

```
void counter_maxfuns (
    PhyloCounters * counters,
    uint lb,
    uint ub,
    bool duplication = true ) [inline]
```

Cap the number of functions per gene.

Definition at line 528 of file phylo.hpp.

5.4.2.9 counter_neofun()

```
void counter_neofun (
    PhyloCounters * counters,
    uint nfunA,
    uint nfunB,
    bool duplication = true ) [inline]
```

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 978 of file phylo.hpp.

5.4.2.10 counter_neofun_a2b()

```
void counter_neofun_a2b (
    PhyloCounters * counters,
    uint nfunA,
    uint nfunB,
    bool duplication = true ) [inline]
```

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 1063 of file phylo.hpp.

5.4.2.11 counter_overall_changes()

```
void counter_overall_changes (
    PhyloCounters * counters,
    bool duplication = true ) [inline]
```

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

Definition at line 661 of file phylo.hpp.

5.4.2.12 counter_overall_gains()

```
void counter_overall_gains (
    PhyloCounters * counters,
    bool duplication = true ) [inline]
```

Overall functional gains.

Total number of gains (irrespective of the function).

Definition at line 139 of file phylo.hpp.

5.4.2.13 counter_overall_loss()

```
void counter_overall_loss (
    PhyloCounters * counters,
    bool duplication = true ) [inline]
```

Overall functional loss.

Definition at line 482 of file phylo.hpp.

5.4.2.14 counter_prop_genes_changing()

```
void counter_prop_genes_changing (
    PhyloCounters * counters,
    bool duplication = true ) [inline]
```

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

Definition at line 367 of file phylo.hpp.

5.4.2.15 counter_subfun()

```
void counter_subfun (
    PhyloCounters * counters,
    uint nfunA,
    uint nfunB,
    bool duplication = true ) [inline]
```

Total count of Sub-functionalization events.

It requires to specify data = {funA, funB}

Definition at line 728 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, bool duplication=true)
Overall functional gains.

5.5.1 Detailed Description

[Rules](#) for phylogenetic modeling.

Parameters

<i>rules</i>	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,  
bool duplication = true ) [inline]
```

Overall functional gains.

Parameters

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

Returns

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

Definition at line 1317 of file phylo.hpp.

Chapter 6

Namespace Documentation

6.1 `barry` Namespace Reference

`barry`: Your go-to motif accountant

Namespaces

- [counters](#)

Tree class and Treeliterator class.

6.1.1 Detailed Description

`barry`: Your go-to motif accountant

6.2 `barry::counters` Namespace Reference

Tree class and Treeliterator class.

Namespaces

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

6.2.1 Detailed Description

Tree class and Treeliterator class.

6.3 barry::counters::network Namespace Reference

6.4 barry::counters::phylo Namespace Reference

6.5 CHECK Namespace Reference

Integer constants used to specify which cell should be check.

Variables

- const int [BOTH](#) = -1
- const int [NONE](#) = 0
- const int [ONE](#) = 1
- const int [TWO](#) = 2

6.5.1 Detailed Description

Integer constants used to specify which cell should be check.

6.5.2 Variable Documentation

6.5.2.1 BOTH

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

Definition at line 20 of file typedefs.hpp.

6.5.2.2 NONE

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

Definition at line 21 of file typedefs.hpp.

6.5.2.3 ONE

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

Definition at line 22 of file typedefs.hpp.

6.5.2.4 TWO

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

Definition at line 23 of file typedefs.hpp.

6.6 EXISTS Namespace Reference

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

Variables

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

6.6.1 Detailed Description

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

6.6.2 Variable Documentation

6.6.2.1 AS_ONE

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

Definition at line 38 of file typedefs.hpp.

6.6.2.2 AS_ZERO

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

Definition at line 37 of file typedefs.hpp.

6.6.2.3 BOTH

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

Definition at line 31 of file typedefs.hpp.

6.6.2.4 NONE

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

Definition at line 32 of file typedefs.hpp.

6.6.2.5 ONE

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

Definition at line 33 of file typedefs.hpp.

6.6.2.6 TWO

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

Definition at line 34 of file typedefs.hpp.

6.6.2.7 UNKNOWN

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

Definition at line 36 of file typedefs.hpp.

Chapter 7

Class Documentation

7.1 BArray< Cell_Type, Data_Type > Class Template Reference

Baseline class for binary arrays.

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

Public Member Functions

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

Get the edgelist.

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

Constructors

Parameters

N_	Number of rows
M_	Number of columns
source	An unsigned vector ranging from 0 to N_
target	An unsigned int vector ranging from 0 to M_
target	When <code>true</code> tries to add repeated observations.

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

Queries

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

Parameters

i,j	<i>Coordinates</i>
check_bounds	<i>If false avoids checking bounds.</i>

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

- **BArray**< Cell_Type, Data_Type > & **operator+=** (const std::pair< uint, uint > &coords)

- `BArray< Cell_Type, Data_Type > & operator-=` (const std::pair< uint, uint > &coords)
- `BArrayCell< Cell_Type, Data_Type > operator()` (uint i, uint j, bool check_bounds=true)
- const `BArrayCell_const< Cell_Type, Data_Type > operator()` (uint i, uint j, bool check_bounds=true) const
- void `rm_cell` (uint i, uint j, bool check_bounds=true, bool check_exists=true)
- void `insert_cell` (uint i, uint j, const `Cell< Cell_Type > &v`, bool check_bounds, bool check_exists)
- void `insert_cell` (uint i, uint j, `Cell< Cell_Type > &&v`, bool check_bounds, bool check_exists)
- void `insert_cell` (uint i, uint j, `Cell_Type v`, bool check_bounds, bool check_exists)
- void `swap_cells` (uint i0, uint j0, uint i1, uint j1, bool check_bounds=true, int check_exists=`CHECK::BOTH`, int *report=nullptr)
- void `toggle_cell` (uint i, uint j, bool check_bounds=true, int check_exists=`EXISTS::UNKNOWN`)
- 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_maps`, i.e. `std::vector< std::unordered_map<unsigned int, Cell_Type> >`.

Template Parameters

<i>Cell_Type</i>	Type of cell (any type).
<i>Data_Type</i>	Data type of the array (bool default).

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

7.1.2 Constructor & Destructor Documentation

7.1.2.1 BArray() [1/6]

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

Zero-size array.

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

7.1.2.2 BArray() [2/6]

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

Empty array.

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

7.1.2.3 BArray() [3/6]

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

Edgelist with data.

7.1.2.4 BArray() [4/6]

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

Edgelist with no data (simpler)

7.1.2.5 BArray() [5/6]

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

Copy constructor.

7.1.2.6 BArray() [6/6]

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

Move operator.

7.1.2.7 ~BArray()

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

7.1.3 Member Function Documentation

7.1.3.1 clear()

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

7.1.3.2 col()

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

7.1.3.3 D() [1/2]

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

7.1.3.4 D() [2/2]

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

7.1.3.5 default_val()

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

7.1.3.6 flush_data()

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

7.1.3.7 get_cell()

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

7.1.3.8 get_col_vec() [1/2]

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

7.1.3.9 get_col_vec() [2/2]

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

7.1.3.10 get_entries()

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

Get the edgelist.

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

Returns

`Entries<Cell_Type>`

7.1.3.11 get_row_vec() [1/2]

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

7.1.3.12 get_row_vec() [2/2]

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

7.1.3.13 insert_cell() [1/3]

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

7.1.3.14 insert_cell() [2/3]

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

7.1.3.15 insert_cell() [3/3]

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

7.1.3.16 is_empty()

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

7.1.3.17 ncol()

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


7.1.3.18 nnozero()

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

7.1.3.19 nrow()

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

7.1.3.20 operator()() [1/2]

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

7.1.3.21 operator()() [2/2]

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

7.1.3.22 operator*=()

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

7.1.3.23 operator+=() [1/3]

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

7.1.3.24 operator+=() [2/3]

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

7.1.3.25 operator+=() [3/3]

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

7.1.3.26 operator-=() [1/3]

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

7.1.3.27 operator-=() [2/3]

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

7.1.3.28 operator-=() [3/3]

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

7.1.3.29 operator/=()

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

7.1.3.30 operator=() [1/2]

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

Move assignment.

7.1.3.31 operator=() [2/2]

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

Assignment constructor.

7.1.3.32 operator==()

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

7.1.3.33 out_of_range()

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

7.1.3.34 print()

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

7.1.3.35 reserve()

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

7.1.3.36 resize()

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

7.1.3.37 rm_cell()

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

7.1.3.38 row()

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

7.1.3.39 set_data()

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

Set the data object.

Parameters

<i>data_</i>	
<i>delete_ ↔ data_</i>	

7.1.3.40 swap_cells()

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

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

7.1.3.41 swap_cols()

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

7.1.3.42 swap_rows()

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

7.1.3.43 toggle_cell()

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

7.1.3.44 toggle_lock()

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

7.1.3.45 transpose()

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

7.1.3.46 zero_col()

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

7.1.3.47 zero_row()

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

7.1.4 Friends And Related Function Documentation

7.1.4.1 BArrayCell< Cell_Type, Data_Type >

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

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

7.1.4.2 BArrayCell_const< Cell_Type, Data_Type >

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

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

7.1.5 Member Data Documentation

7.1.5.1 visited

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

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

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

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

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

7.2 BArrayCell< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

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

7.2.1 Detailed Description

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

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

7.2.2 Constructor & Destructor Documentation

7.2.2.1 BArrayCell()

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

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

7.2.2.2 ~BArrayCell()

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

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

7.2.3 Member Function Documentation

7.2.3.1 operator Cell_Type()

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

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

7.2.3.2 operator*=()

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

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

7.2.3.3 operator+=()

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

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

7.2.3.4 operator-=()

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

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

7.2.3.5 operator/=()

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

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

7.2.3.6 operator=()

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

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

7.2.3.7 operator==()

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

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

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

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

7.3 BArrayCell_const< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

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

7.3.1 Detailed Description

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

Definition at line 41 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 50 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 62 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 bararraycell-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 bararraycell-meat.hpp.

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

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

7.4 BArrayDense< Cell_Type, Data_Type > Class Template Reference

Baseline class for binary arrays.

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

Public Member Functions

- bool [operator==](#) (const [BArrayDense](#)< Cell_Type, Data_Type > &Array_)
- [~BArrayDense](#) ()
- void [out_of_range](#) (uint i, uint j) const
- Cell_Type [get_cell](#) (uint i, uint j, bool check_bounds=true) const
- std::vector< Cell_Type > [get_col_vec](#) (uint i, bool check_bounds=true) const
- std::vector< Cell_Type > [get_row_vec](#) (uint i, bool check_bounds=true) const
- void [get_col_vec](#) (std::vector< Cell_Type > *x, uint i, bool check_bounds=true) const
- void [get_row_vec](#) (std::vector< Cell_Type > *x, uint i, bool check_bounds=true) const
- const [Row_type](#)< Cell_Type > & [row](#) (uint i, bool check_bounds=true) const
- const [Col_type](#)< Cell_Type > & [col](#) (uint i, bool check_bounds=true) const
- [Entries](#)< Cell_Type > [get_entries](#) () const
- *Get the edgelist.*
- void [transpose](#) ()
- void [clear](#) (bool hard=true)
- void [resize](#) (uint N_, uint M_)
- void [reserve](#) ()
- void [print](#) () const

Constructors

Parameters

N_	Number of rows
M_	Number of columns
source	An unsigned vector ranging from 0 to N_
target	An unsigned int vector ranging from 0 to M_
target	When <code>true</code> tries to add repeated observations.

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

Queries

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

Parameters

i,j	Coordinates
check_bounds	If <code>false</code> 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 <code>true</code> 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 <code>swap_cells</code> , check if either of both cells exists/don't exist.

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

Column/row wise interchange

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

Arithmetic operators

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

Public Attributes

- bool `visited` = false

Friends

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

7.4.1 Detailed Description

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

Baseline class for binary arrays.

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

Template Parameters

<i>Cell_Type</i>	Type of cell (any type).
<i>Data_Type</i>	Data type of the array (bool default).

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

7.4.2 Constructor & Destructor Documentation

7.4.2.1 BArrayDense() [1/6]

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

Zero-size array.

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

7.4.2.2 BArrayDense() [2/6]

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

Empty array.

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

7.4.2.3 BArrayDense() [3/6]

```
template<typename Cell_Type , typename Data_Type >
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 ) [inline]
```

Edgelist with data.

Definition at line 18 of file barraydense-meet.hpp.

7.4.2.4 BArrayDense() [4/6]

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

Edgelist with no data (simpler)

Definition at line 65 of file barraydense-meet.hpp.

7.4.2.5 BArrayDense() [5/6]

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

Copy constructor.

Definition at line 120 of file barraydense-meet.hpp.

7.4.2.6 BArrayDense() [6/6]

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

Move operator.

Definition at line 196 of file barraydense-meet.hpp.

7.4.2.7 ~BArrayDense()

```
template<typename Cell_Type , typename Data_Type >
BArrayDense< Cell_Type, Data_Type >::~~BArrayDense [inline]
```

Definition at line 273 of file barraydense-meet.hpp.

7.4.3 Member Function Documentation

7.4.3.1 clear()

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

Definition at line 1004 of file barraydense-meet.hpp.

7.4.3.2 col()

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

Definition at line 401 of file barraydense-meet.hpp.

7.4.3.3 D() [1/2]

```
template<typename Cell_Type , typename Data_Type >
Data_Type * BArrayDense< Cell_Type, Data_Type >::D [inline]
```

Definition at line 297 of file barraydense-meet.hpp.

7.4.3.4 D() [2/2]

```
template<typename Cell_Type , typename Data_Type >
const Data_Type * BArrayDense< Cell_Type, Data_Type >::D [inline]
```

Definition at line 302 of file barraydense-meet.hpp.

7.4.3.5 default_val()

```
template<typename Cell_Type , typename Data_Type >
Cell< Cell_Type > BArrayDense< Cell_Type, Data_Type >::default_val [inline]
```

Definition at line 467 of file barraydense-meet.hpp.

7.4.3.6 get_cell()

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

Definition at line 317 of file bararraydense-meet.hpp.

7.4.3.7 get_col_vec() [1/2]

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

Definition at line 377 of file bararraydense-meet.hpp.

7.4.3.8 get_col_vec() [2/2]

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

Definition at line 362 of file bararraydense-meet.hpp.

7.4.3.9 get_entries()

```
template<typename Cell_Type , typename Data_Type >
Entries< Cell_Type > BArrayDense< Cell_Type, Data_Type >::get_entries [inline]
```

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>`

Definition at line 412 of file bararraydense-meet.hpp.

7.4.3.10 get_row_vec() [1/2]

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

Definition at line 346 of file `barraydense-meet.hpp`.

7.4.3.11 get_row_vec() [2/2]

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

Definition at line 331 of file `barraydense-meet.hpp`.

7.4.3.12 insert_cell() [1/3]

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

Definition at line 601 of file `barraydense-meet.hpp`.

7.4.3.13 insert_cell() [2/3]

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

Definition at line 649 of file `barraydense-meet.hpp`.

7.4.3.14 insert_cell() [3/3]

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

Definition at line 553 of file barraydense-meet.hpp.

7.4.3.15 is_empty()

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

Definition at line 432 of file barraydense-meet.hpp.

7.4.3.16 ncol()

```
template<typename Cell_Type , typename Data_Type >
uint BArrayDense< Cell_Type, Data_Type >::ncol [inline], [noexcept]
```

Definition at line 457 of file barraydense-meet.hpp.

7.4.3.17 nnozero()

```
template<typename Cell_Type , typename Data_Type >
uint BArrayDense< Cell_Type, Data_Type >::nnozero [inline], [noexcept]
```

Definition at line 462 of file barraydense-meet.hpp.

7.4.3.18 nrow()

```
template<typename Cell_Type , typename Data_Type >
uint BArrayDense< Cell_Type, Data_Type >::nrow [inline], [noexcept]
```

Definition at line 452 of file barraydense-meet.hpp.

7.4.3.19 operator>() [1/2]

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

Definition at line 503 of file barraydense-meet.hpp.

7.4.3.20 operator>() [2/2]

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

Definition at line 512 of file barraydense-meet.hpp.

7.4.3.21 operator*=()

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

7.4.3.22 operator+=() [1/3]

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

7.4.3.23 operator+=() [2/3]

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

7.4.3.24 operator+=() [3/3]

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

Definition at line 472 of file barraydense-meet.hpp.

7.4.3.25 operator-=() [1/3]

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

7.4.3.26 operator-=() [2/3]

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

7.4.3.27 operator-=() [3/3]

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

Definition at line 488 of file barraydense-meet.hpp.

7.4.3.28 operator/=()

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

7.4.3.29 operator=() [1/2]

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

Move assignment.

Definition at line 212 of file barraydense-meet.hpp.

7.4.3.30 operator=() [2/2]

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

Assignment constructor.

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

7.4.3.31 operator==()

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

Definition at line 254 of file barraydense-meet.hpp.

7.4.3.32 out_of_range()

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::out_of_range (
    uint i,
    uint j ) const [inline]
```

Definition at line 307 of file barraydense-meet.hpp.

7.4.3.33 print()

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::print [inline]
```

Definition at line 1072 of file barraydense-meet.hpp.

7.4.3.34 reserve()

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::reserve [inline]
```

Definition at line 1059 of file barraydense-meet.hpp.

7.4.3.35 resize()

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

Definition at line 1027 of file barraydense-meet.hpp.

7.4.3.36 rm_cell()

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

Definition at line 521 of file barraydense-meet.hpp.

7.4.3.37 row()

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

Definition at line 391 of file barraydense-meet.hpp.

7.4.3.38 set_data()

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

Set the data object.

Parameters

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

Definition at line 282 of file `barraydense-meet.hpp`.

7.4.3.39 swap_cells()

```
template<typename Cell_Type , typename Data_Type >
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 ) [inline]
```

Definition at line 657 of file `barraydense-meet.hpp`.

7.4.3.40 swap_cols()

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

Definition at line 838 of file `barraydense-meet.hpp`.

7.4.3.41 swap_rows()

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

Definition at line 792 of file `barraydense-meet.hpp`.

7.4.3.42 toggle_cell()

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

Definition at line 758 of file bararraydense-meet.hpp.

7.4.3.43 toggle_lock()

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

7.4.3.44 transpose()

```
template<typename Cell_Type , typename Data_Type >
void BArrayDense< Cell_Type, Data_Type >::transpose [inline]
```

Definition at line 947 of file bararraydense-meet.hpp.

7.4.3.45 zero_col()

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

Definition at line 925 of file bararraydense-meet.hpp.

7.4.3.46 zero_row()

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

Definition at line 906 of file bararraydense-meet.hpp.

7.4.4 Friends And Related Function Documentation

7.4.4.1 BArrayCell< Cell_Type, Data_Type >

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

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

7.4.4.2 BArrayCell_const< Cell_Type, Data_Type >

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

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

7.4.5 Member Data Documentation

7.4.5.1 visited

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

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

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

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

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

7.5 BArrayDenseCell< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

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

7.5.1 Detailed Description

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

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

7.5.2 Constructor & Destructor Documentation

7.5.2.1 BArrayDenseCell()

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

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

7.5.2.2 ~BArrayDenseCell()

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

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

7.5.3 Member Function Documentation

7.5.3.1 operator Cell_Type()

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

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

7.5.3.2 operator*=()

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

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

7.5.3.3 operator+=()

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

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

7.5.3.4 operator-=()

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

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

7.5.3.5 operator/=()

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

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

7.5.3.6 operator=()

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

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

7.5.3.7 operator==()

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

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

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

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

7.6 BArrayDenseCell_const< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

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

7.6.1 Detailed Description

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

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

7.6.2 Constructor & Destructor Documentation

7.6.2.1 BArrayDenseCell_const()

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

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

7.6.2.2 ~BArrayDenseCell_const()

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

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

7.6.3 Member Function Documentation

7.6.3.1 operator Cell_Type()

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

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

7.6.3.2 operator"!=(

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

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

7.6.3.3 operator<()

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

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

7.6.3.4 operator<=()

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

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

7.6.3.5 operator==()

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

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

7.6.3.6 operator>()

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

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

7.6.3.7 operator>=()

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

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

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

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

7.7 BArrayVector< Cell_Type, Data_Type > Class Template Reference

Row or column of a [BArray](#)

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

Public Member Functions

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

7.7.1 Detailed Description

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

Row or column of a [BArray](#)

Template Parameters

<i>Cell_Type</i>	
<i>Data_Type</i>	

Definition at line 13 of file [barrayvector-bones.hpp](#).

7.7.2 Constructor & Destructor Documentation

7.7.2.1 BArrayVector()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArrayVector< Cell_Type, Data_Type >::BArrayVector (
```

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

Construct a new [BArrayVector](#) object.

Parameters

<i>Array_</i>	Pointer to a BArray object
<i>dim_</i>	Dimension. 0 means row and 1 means column.
<i>i_</i>	Element to point.
<i>check_bounds</i>	When <code>true</code> , check boundaries.

Definition at line 34 of file `barrayvector-bones.hpp`.

7.7.2.2 ~BArrayVector()

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

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

7.7.3 Member Function Documentation

7.7.3.1 begin()

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

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

7.7.3.2 end()

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

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

7.7.3.3 is_col()

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

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

7.7.3.4 is_row()

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

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

7.7.3.5 operator std::vector< Cell_Type >()

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

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

7.7.3.6 operator*=()

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

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

7.7.3.7 operator+=()

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

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

7.7.3.8 operator-=()

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

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

7.7.3.9 operator/=()

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

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

7.7.3.10 operator=()

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

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

7.7.3.11 operator==()

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

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

7.7.3.12 size()

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

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

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

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

7.8 BArrayVector_const< Cell_Type, Data_Type > Class Template Reference

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

Public Member Functions

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

7.8.1 Detailed Description

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

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

7.8.2 Constructor & Destructor Documentation

7.8.2.1 BArrayVector_const()

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

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

7.8.2.2 ~BArrayVector_const()

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

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

7.8.3 Member Function Documentation

7.8.3.1 begin()

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

7.8.3.2 end()

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

7.8.3.3 is_col()

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

7.8.3.4 is_row()

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

7.8.3.5 operator std::vector< Cell_Type >()

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

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

7.8.3.6 operator!=(())

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

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

7.8.3.7 operator<()

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

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

7.8.3.8 operator<=()

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

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

7.8.3.9 operator==(())

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

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

7.8.3.10 operator>()

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

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

7.8.3.11 operator>=()

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

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

7.8.3.12 size()

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

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

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

7.9 Cell< Cell_Type > Class Template Reference

Entries in [BArray](#). For now, it only has two members:

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

Public Member Functions

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

Public Attributes

- Cell_Type [value](#)
- bool [visited](#)

7.9.1 Detailed Description

```
template<class Cell_Type>
class Cell< Cell_Type >
```

[Entries](#) in [BArray](#). For now, it only has two members:

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

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

7.9.2 Constructor & Destructor Documentation

7.9.2.1 Cell() [1/7]

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

7.9.2.2 Cell() [2/7]

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

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

7.9.2.3 ~Cell()

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

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

7.9.2.4 Cell() [3/7]

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

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

7.9.2.5 Cell() [4/7]

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

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

7.9.2.6 Cell() [5/7]

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

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

7.9.2.7 Cell() [6/7]

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

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

7.9.2.8 Cell() [7/7]

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

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

7.9.3 Member Function Documentation

7.9.3.1 add() [1/4]

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

7.9.3.2 add() [2/4]

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

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

7.9.3.3 add() [3/4]

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

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

7.9.3.4 add() [4/4]

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

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

7.9.3.5 operator Cell_Type()

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

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

7.9.3.6 operator"!="()

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

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

7.9.3.7 operator=() [1/2]

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

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

7.9.3.8 operator=() [2/2]

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

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

7.9.3.9 operator==()

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

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

7.9.4 Member Data Documentation

7.9.4.1 value

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

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

7.9.4.2 visited

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

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

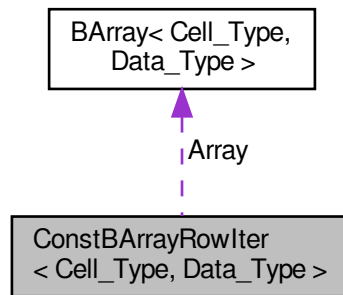
The documentation for this class was generated from the following files:

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

7.10 ConstBArrayRowIter< Cell_Type, Data_Type > Class Template Reference

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

Collaboration diagram for ConstBArrayRowIter< Cell_Type, Data_Type >:



Public Member Functions

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

Public Attributes

- [uint](#) `current_row`
- [uint](#) `current_col`
- [Row_type](#)< Cell_Type >::const_iterator `iter`
- const [BArray](#)< Cell_Type, Data_Type > * `Array`

7.10.1 Detailed Description

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

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

7.10.2 Constructor & Destructor Documentation

7.10.2.1 ConstBArrayRowIter()

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

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

7.10.2.2 ~ConstBArrayRowIter()

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

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

7.10.3 Member Data Documentation

7.10.3.1 Array

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

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

7.10.3.2 current_col

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

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

7.10.3.3 current_row

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

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

7.10.3.4 iter

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

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

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

- include/barry/barray-iterator.hpp

7.11 Counter< Array_Type, Data_Type > Class Template Reference

A counter function based on change statistics.

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

Public Member Functions

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

Creator passing a counter and an initializer

Parameters

count_fun_↔ _	The main counter function.
init_fun_	The initializer function can also be used to check if the BArray as the needed variables (see BArray::data).
data_	Data to be used with the counter.
delete_↔ data_	When <code>true</code> , the destructor will delete the pointer in the main data.

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

Public Attributes

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

7.11.1 Detailed Description

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

A counter function based on change statistics.

This class is used by `CountStats` and [StatsCounter](#) as a way to count statistics using change statistics.

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

7.11.2 Constructor & Destructor Documentation

7.11.2.1 Counter() [1/4]

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

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

7.11.2.2 Counter() [2/4]

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

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

7.11.2.3 Counter() [3/4]

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

Copy constructor.

7.11.2.4 Counter() [4/4]

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

Move constructor.

7.11.2.5 ~Counter()

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

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

7.11.3 Member Function Documentation**7.11.3.1 count()**

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

7.11.3.2 get_description()

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

7.11.3.3 get_name()

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

7.11.3.4 init()

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

7.11.3.5 operator=() [1/2]

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

Copy assignment.

7.11.3.6 operator=() [2/2]

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

Move assignment.

7.11.4 Member Data Documentation

7.11.4.1 count_fun

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

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

7.11.4.2 data

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

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

7.11.4.3 delete_data

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

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

7.11.4.4 desc

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

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

7.11.4.5 init_fun

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

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

7.11.4.6 name

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

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

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

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

7.12 Counters< Array_Type, Data_Type > Class Template Reference

Vector of counters.

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

Public Member Functions

- [Counters](#) ()
- [~Counters](#) ()
- [Counters](#) (const [Counters](#)< Array_Type, Data_Type > &[counter_](#))
Copy constructor.
- [Counters](#) ([Counters](#)< Array_Type, Data_Type > &&[counters_](#)) **noexcept**
Move constructor.
- [Counters](#)< Array_Type, Data_Type > **operator=** (const [Counters](#)< Array_Type, Data_Type > &[counter_](#))
Copy assignment constructor.
- [Counters](#)< Array_Type, Data_Type > & **operator=** ([Counters](#)< Array_Type, Data_Type > &&[counter_](#)) **noexcept**
Move assignment constructor.
- [Counter](#)< Array_Type, Data_Type > & **operator[]** (uint idx)
Returns a pointer to a particular counter.
- std::size_t [size](#) () const **noexcept**
Number of counters in the set.
- void [add_counter](#) ([Counter](#)< Array_Type, Data_Type > &[counter](#))
- void [add_counter](#) ([Counter](#)< Array_Type, Data_Type > *[counter](#))
- void [add_counter](#) ([Counter_fun_type](#)< Array_Type, Data_Type > [count_fun_](#), [Counter_fun_type](#)< Array_Type, Data_Type > [init_fun_](#)=nullptr, Data_Type *[data_](#)=nullptr, bool [delete_data_](#)=false, std::string [name_](#)="", std::string [desc_](#)="")
- void [clear](#) ()
- std::vector< std::string > [get_names](#) () const
- std::vector< std::string > [get_descriptions](#) () const

7.12.1 Detailed Description

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
```

```
class Counters< Array_Type, Data_Type >
```

Vector of counters.

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

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

7.12.2 Constructor & Destructor Documentation

7.12.2.1 Counters() [1/3]

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

7.12.2.2 ~Counters()

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

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

7.12.2.3 Counters() [2/3]

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

Copy constructor.

Parameters

<i>counter_↔</i>	
—	

7.12.2.4 Counters() [3/3]

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

Move constructor.

Parameters

<i>counters_↔</i>	
—	

7.12.3 Member Function Documentation

7.12.3.1 add_counter() [1/3]

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

7.12.3.2 add_counter() [2/3]

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

7.12.3.3 add_counter() [3/3]

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

7.12.3.4 clear()

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

7.12.3.5 get_descriptions()

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

7.12.3.6 get_names()

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

7.12.3.7 operator=() [1/2]

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

Copy assignment constructor.

Parameters

<i>counter</i> ↔	
—	

Returns

Counters<Array_Type,Data_Type>

7.12.3.8 operator=() [2/2]

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

Move assignment constructor.

Parameters

<i>counter</i> ↔	
—	

Returns

Counters<Array_Type,Data_Type>&

7.12.3.9 operator[]()

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

Returns a pointer to a particular counter.

Parameters

<i>idx</i>	Id of the counter
------------	-------------------

Returns

Counter<Array_Type,Data_Type>*

7.12.3.10 size()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::size_t Counters< Array_Type, Data_Type >::size ( ) const [inline], [noexcept]
```

Number of counters in the set.

Returns

uint

Definition at line 161 of file counters-bones.hpp.

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

- include/barry/counters-bones.hpp

7.13 Entries< Cell_Type > Class Template Reference

A wrapper class to store `source`, `target`, `val` from a [BArray](#) object.

```
#include <typedefs.hpp>
```

Public Member Functions

- [Entries](#) ()
- [Entries](#) (uint n)
- [~Entries](#) ()
- void [resize](#) (uint n)

Public Attributes

- std::vector< [uint](#) > [source](#)
- std::vector< [uint](#) > [target](#)
- std::vector< [Cell_Type](#) > [val](#)

7.13.1 Detailed Description

```
template<typename Cell_Type>
class Entries< Cell_Type >
```

A wrapper class to store `source`, `target`, `val` from a [BArray](#) object.

Template Parameters

<i>Cell_Type</i>	Any type
------------------	----------

Definition at line 59 of file typedefs.hpp.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 Entries() [1/2]

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

Definition at line 65 of file typedefs.hpp.

7.13.2.2 Entries() [2/2]

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

Definition at line 66 of file typedefs.hpp.

7.13.2.3 ~Entries()

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

Definition at line 73 of file typedefs.hpp.

7.13.3 Member Function Documentation

7.13.3.1 resize()

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

Definition at line 75 of file typedefs.hpp.

7.13.4 Member Data Documentation

7.13.4.1 source

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

Definition at line 61 of file typedefs.hpp.

7.13.4.2 target

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

Definition at line 62 of file typedefs.hpp.

7.13.4.3 val

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

Definition at line 63 of file typedefs.hpp.

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

- [include/barry/typedefs.hpp](#)

7.14 Flock Class Reference

A [Flock](#) is a group of [Geese](#).

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

Public Member Functions

- [Flock](#) ()
- [~Flock](#) ()
- unsigned int [add_data](#) (std::vector< std::vector< unsigned int > > &annotations, std::vector< unsigned int > &geneid, std::vector< int > &parent, std::vector< bool > &duplication)
Add a tree to the flock.
- void [set_seed](#) (const unsigned int &s)
Set the seed of the model.
- void [init](#) (bool verb=true)
- [phylocounters::PhyloCounters](#) * [get_counters](#) ()
- [phylocounters::PhyloSupport](#) * [get_support](#) ()
- [phylocounters::PhyloModel](#) * [get_model](#) ()
- double [likelihood_joint](#) (const std::vector< double > &par, bool as_log=false, bool use_reduced_↔ sequence=true)
Returns the joint likelihood of the model.
- [Geese](#) * [operator\(\)](#) (unsigned int i, bool check_bounds=true)
Access the i-th geese element.

Information about the model

- unsigned int [nfuncs](#) () const [noexcept](#)
- unsigned int [ntrees](#) () const [noexcept](#)
- std::vector< unsigned int > [nnodes](#) () const [noexcept](#)
- std::vector< unsigned int > [nleafs](#) () const [noexcept](#)
- unsigned int [nterms](#) () const
- unsigned int [support_size](#) () const [noexcept](#)
- std::vector< std::string > [colnames](#) () const
- unsigned int [parse_polytomies](#) (bool verb=true) const [noexcept](#)
- void [print](#) () const

Public Attributes

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

7.14.1 Detailed Description

A [Flock](#) is a group of [Geese](#).

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

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

7.14.2 Constructor & Destructor Documentation

7.14.2.1 Flock()

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

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

7.14.2.2 ~Flock()

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

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

7.14.3 Member Function Documentation

7.14.3.1 add_data()

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

Add a tree to the flock.

Parameters

<i>annotations</i>	see Geese::Geese .
<i>geneid</i>	see Geese .
<i>parent</i>	see Geese .
<i>duplication</i>	see Geese .

Returns

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

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

7.14.3.2 colnames()

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

Definition at line 159 of file flock-meet.hpp.

7.14.3.3 get_counters()

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

Definition at line 69 of file flock-meet.hpp.

7.14.3.4 get_model()

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

Definition at line 82 of file flock-meet.hpp.

7.14.3.5 get_support()

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

Definition at line 78 of file flock-meet.hpp.

7.14.3.6 init()

```
void Flock::init (
    bool verb = true ) [inline]
```

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

7.14.3.7 likelihood_joint()

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

Returns the joint likelihood of the model.

Parameters

<i>par</i>	Vector of model parameters.
<i>as_log</i>	When <code>true</code> it will return the value as log.
<i>use_reduced_sequence</i>	When <code>true</code> (default) will compute the likelihood using the reduced sequence, which is faster.

Returns

double

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

7.14.3.8 nfuncs()

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

Definition at line 111 of file flock-meet.hpp.

7.14.3.9 nleafs()

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

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

7.14.3.10 nnodes()

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

Definition at line 123 of file flock-meet.hpp.

7.14.3.11 nterms()

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

Definition at line 146 of file flock-meet.hpp.

7.14.3.12 ntrees()

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

Definition at line 117 of file flock-meet.hpp.

7.14.3.13 operator>()()

```
Geese * Flock::operator() (
    unsigned int i,
    bool check_bounds = true ) [inline]
```

Access the i-th geese element.

Parameters

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

Returns

Geese*

Definition at line 217 of file flock-meet.hpp.

7.14.3.14 parse_polytomies()

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

Definition at line 165 of file flock-meet.hpp.

7.14.3.15 print()

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

Definition at line 184 of file flock-meet.hpp.

7.14.3.16 set_seed()

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

Set the seed of the model.

Parameters

<i>s</i>	Passed to the <code>engine.seed()</code> member object.
----------	---

Definition at line 37 of file flock-meet.hpp.

7.14.3.17 support_size()

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

Definition at line 153 of file flock-meet.hpp.

7.14.4 Member Data Documentation

7.14.4.1 dat

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

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

7.14.4.2 initialized

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

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

7.14.4.3 model

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

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

7.14.4.4 nfunctions

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

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

7.14.4.5 engine

```
std::mt19937 Flock::engine
```

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

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

- [include/barry/models/geese/flock-bones.hpp](#)
- [include/barry/models/geese/flock-meet.hpp](#)

7.15 FreqTable< T > Class Template Reference

Database of statistics.

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

Public Member Functions

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

7.15.1 Detailed Description

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

Database of statistics.

This is mostly used in [Support](#).

Definition at line 16 of file statsdb.hpp.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 FreqTable()

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

Definition at line 28 of file statsdb.hpp.

7.15.2.2 ~FreqTable()

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

Definition at line 29 of file statsdb.hpp.

7.15.3 Member Function Documentation

7.15.3.1 add()

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

Definition at line 47 of file statsdb.hpp.

7.15.3.2 as_vector()

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

Definition at line 61 of file statsdb.hpp.

7.15.3.3 clear()

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

Definition at line 83 of file statsdb.hpp.

7.15.3.4 get_data()

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

Definition at line 73 of file statsdb.hpp.

7.15.3.5 get_data_ptr()

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

Definition at line 78 of file statsdb.hpp.

7.15.3.6 print()

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

Definition at line 102 of file statsdb.hpp.

7.15.3.7 reserve()

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

Definition at line 89 of file statsdb.hpp.

7.15.3.8 size()

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

Definition at line 126 of file statsdb.hpp.

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

- include/barry/statsdb.hpp

7.16 Geese Class Reference

Annotated Phyl^o [Model](#).

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

Public Member Functions

- `~Geese ()`
- `void init (bool verb=true)`
- `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

- `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 <code>true</code> it will print out information about the encountered polytomies.
------	--

- `unsigned int nfuncs () const noexcept`
Number of functions analyzed.
- `unsigned int nnodes () const noexcept`
Number of nodes (interior + leaf)

- unsigned int [nleafs](#) () const [noexcept](#)
Number of leaf.
- unsigned int [nterms](#) () const
Number of terms included.
- unsigned int [support_size](#) () const [noexcept](#)
Number of unique sets of sufficient stats.
- std::vector< unsigned int > [nannotations](#) () const [noexcept](#)
Number of annotations.
- std::vector< std::string > [colnames](#) () const
Names of the terms in the model.
- unsigned int [parse_polytomies](#) (bool verb=true) const [noexcept](#)
Check polytomies and return the largest.

Geese prediction

Calculate the conditional probability

Parameters

par	<i>Vector of parameters (terms + root).</i>
res_prob	<i>Vector indicating each nodes' state probability.</i>
leave_one_out	<i>When <code>true</code>, it will compute the predictions using leave-one-out, thus the prediction will be repeated <code>nleafs</code> times.</i>
only_annotated	<i>When <code>true</code>, it will make the predictions only on the induced sub-tree with annotated leaves.</i>
use_reduced_sequence	<i>Passed to the <code>likelihood</code> method.</i>
preorder	<i>For the tree traversal.</i>

When `res_prob` is specified, the function will attach the member vector probabilities from the [Nodes](#) objects. This contains the probability that the *i*th node has either of the possible states.

Returns

`std::vector< double >` Returns the posterior probability

- `std::vector< std::vector< double > >` [predict](#) (const `std::vector< double >` &par, `std::vector< std::vector< double > >` *res_prob=nullptr, bool leave_one_out=false, bool only_annotated=false, bool use_reduced_sequence=true)
- `std::vector< std::vector< double > >` [predict_backend](#) (const `std::vector< double >` &par, bool use_reduced_sequence, const `std::vector< 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_engine](#) () returns the Pseudo-RNG engine used.
[get_counters](#) () returns the vector of counters used.
[get_model](#) () returns the [Model](#) object used.
[get_support](#) () returns the computed support of the model.

- `std::mt19937` * [get_engine](#) ()
- `phylocounters::PhyloCounters` * [get_counters](#) ()
- `phylocounters::PhyloModel` * [get_model](#) ()
- `phylocounters::PhyloSupport` * [get_support](#) ()

Public Attributes

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

7.16.1 Detailed Description

Annotated Phylo [Model](#).

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

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

7.16.2 Constructor & Destructor Documentation

7.16.2.1 Geese() [1/4]

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

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

7.16.2.2 Geese() [2/4]

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

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

7.16.2.3 Geese() [3/4]

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

Definition at line 207 of file `geese-meat-constructors.hpp`.

7.16.2.4 Geese() [4/4]

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

Definition at line 280 of file geese-meat-constructors.hpp.

7.16.2.5 ~Geese()

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

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

7.16.3 Member Function Documentation

7.16.3.1 calc_reduced_sequence()

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

Definition at line 274 of file geese-meat.hpp.

7.16.3.2 calc_sequence()

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

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

7.16.3.3 colnames()

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

Names of the terms in the model.

Definition at line 375 of file geese-meat.hpp.

7.16.3.4 `get_annotated_nodes()`

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

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

Definition at line 556 of file geese-meat.hpp.

7.16.3.5 `get_counters()`

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

Definition at line 539 of file geese-meat.hpp.

7.16.3.6 `get_model()`

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

Definition at line 544 of file geese-meat.hpp.

7.16.3.7 `get_probabilities()`

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

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

7.16.3.8 `get_engine()`

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

Definition at line 534 of file geese-meat.hpp.

7.16.3.9 get_states()

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

Powerset of a gene's possible states.

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

Returns

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

Definition at line 552 of file `geese-meat.hpp`.

7.16.3.10 get_support()

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

Definition at line 548 of file `geese-meat.hpp`.

7.16.3.11 inherit_support()

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

Definition at line 183 of file `geese-meat.hpp`.

7.16.3.12 init()

```
void Geese::init (
    bool verb = true ) [inline]
```

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

7.16.3.13 init_node()

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

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

7.16.3.14 likelihood()

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

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

7.16.3.15 likelihood_exhaust()

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

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

7.16.3.16 nannotations()

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

Number of annotations.

Definition at line 369 of file geese-meat.hpp.

7.16.3.17 nfuncs()

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

Number of functions analyzed.

Definition at line 333 of file geese-meat.hpp.

7.16.3.18 nleafs()

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

Number of leaf.

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

7.16.3.19 nnodes()

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

Number of nodes (interior + leaf)

Definition at line 337 of file geese-meat.hpp.

7.16.3.20 nterms()

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

Number of terms included.

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

7.16.3.21 observed_counts()

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

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

7.16.3.22 operator=() [1/2]

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

7.16.3.23 operator=() [2/2]

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

7.16.3.24 parse_polytomies()

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

Check polytomies and return the largest.

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

7.16.3.25 predict()

```
std::vector< std::vector< double > > Geese::predict (
    const std::vector< double > & par,
    std::vector< std::vector< double > > * res_prob = nullptr,
    bool leave_one_out = false,
    bool only_annotated = false,
    bool use_reduced_sequence = true ) [inline]
```

Definition at line 240 of file geese-meat-predict.hpp.

7.16.3.26 predict_backend()

```
std::vector< std::vector< double > > Geese::predict_backend (
    const std::vector< double > & par,
    bool use_reduced_sequence,
    const std::vector< uint > & preorder ) [inline]
```

< True if the array belongs to the set

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

7.16.3.27 predict_exhaust()

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

Definition at line 5 of file geese-meat-predict_exhaust.hpp.

7.16.3.28 predict_exhaust_backend()

```
std::vector< std::vector< double > > Geese::predict_exhaust_backend (
    const std::vector< double > & par,
    const std::vector< uint > & preorder ) [inline]
```

Definition at line 47 of file geese-meat-predict_exhaust.hpp.

7.16.3.29 predict_sim()

```
std::vector< std::vector< double > > Geese::predict_sim (
    const std::vector< double > & par,
    bool only_annotated = false,
    unsigned int nsims = 10000u ) [inline]
```

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

7.16.3.30 print()

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

Prints information about the GESE.

Definition at line 516 of file geese-meat.hpp.

7.16.3.31 print_observed_counts()

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

Definition at line 454 of file geese-meat.hpp.

7.16.3.32 set_seed()

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

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

7.16.3.33 simulate()

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

Definition at line 8 of file geese-meat-simulate.hpp.

7.16.3.34 support_size()

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

Number of unique sets of sufficient stats.

Definition at line 359 of file geese-meat.hpp.

7.16.3.35 update_annotations()

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

Definition at line 206 of file geese-meat.hpp.

7.16.4 Member Data Documentation

7.16.4.1 delete_engine

```
bool Geese::delete_engine = false
```

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

7.16.4.2 delete_support

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

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

7.16.4.3 initialized

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

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

7.16.4.4 map_to_nodes

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

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

7.16.4.5 nfunctions

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

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

7.16.4.6 nodes

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

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

7.16.4.7 reduced_sequence

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

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

7.16.4.8 sequence

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

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

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

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

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

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

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

Public Member Functions

- void [set_engine](#) (std::mt19937 *engine_, 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 > ¶ms={})
- Array_Type [sample](#) (const [uint](#) &i, const std::vector< double > ¶ms)
- double [conditional_prob](#) (const Array_Type &Array_, const std::vector< double > ¶ms, unsigned int i, unsigned int j)
Conditional probability ("Gibbs sampler")
- const std::mt19937 * [get_engine](#) () const
- [Counters](#)< Array_Type, Data_Counter_Type > * [get_counters](#) ()
- [Rules](#)< Array_Type, Data_Rule_Type > * [get_rules](#) ()
- [Rules](#)< Array_Type, Data_Rule_Dyn_Type > * [get_rules_dyn](#) ()
- [Support](#)< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > * [get_support](#) ()

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

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

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

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

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

- void [add_rule](#) ([Rule](#)< Array_Type, Data_Rule_Type > &rule)
- void [add_rule](#) ([Rule](#)< Array_Type, Data_Rule_Type > *rule)
- void [add_rule](#) ([Rule_fun_type](#)< Array_Type, Data_Rule_Type > count_fun_, Data_Rule_Type *data_=nullptr, bool delete_data=false)
- void [set_rules](#) ([Rules](#)< Array_Type, Data_Rule_Type > *rules_)
- void [add_rule_dyn](#) ([Rule](#)< Array_Type, Data_Rule_Dyn_Type > &rule)
- void [add_rule_dyn](#) ([Rule](#)< Array_Type, Data_Rule_Dyn_Type > *rule)
- void [add_rule_dyn](#) ([Rule_fun_type](#)< Array_Type, Data_Rule_Dyn_Type > count_fun_, Data_Rule_Dyn_Type *data_=nullptr, bool delete_data=false)
- void [set_rules_dyn](#) ([Rules](#)< Array_Type, Data_Rule_Dyn_Type > *rules_)

Likelihood functions.

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

Parameters

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

- double [likelihood](#) (const std::vector< double > ¶ms, const [uint](#) &i, bool as_log=false)
- double [likelihood](#) (const std::vector< double > ¶ms, const Array_Type &Array_, int i=-1, bool as_log=false)
- double [likelihood](#) (const std::vector< double > ¶ms, const std::vector< double > &target_, const [uint](#) &i, bool as_log=false)
- double [likelihood_total](#) (const std::vector< double > ¶ms, 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 <i>true</i> returns the logged version of the normalizing constant.

- double [get_norm_const](#) (const std::vector< double > ¶ms, const [uint](#) &i, bool as_log=false)
- const std::vector< Array_Type > * [get_pset](#) (const [uint](#) &i)
- const std::vector< std::vector< double > > * [get_pset_stats](#) (const [uint](#) &i)

Size of the model

Number of different supports included in the model

This will return the size of *stats*.

Returns

[size\(\)](#) returns the number of arrays in the model.
[size_unique\(\)](#) returns the number of unique arrays (according to the hasher) in the model.
[nterms\(\)](#) returns the number of terms in the model.

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

7.17.1 Detailed Description

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

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

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

This implementation aims to reduce the number of times that the support needs to be computed. Models included here use more than a single array, and thus allow the function to recycle support sets as needed. For example, if we are looking at directed graphs all of the same size and without vertex level features, i.e. a model that only counts edges, triangles, etc. then the support needs to be fully computed only once.

Template Parameters

<i>Array_Type</i>	Class of BArray object.
<i>Data_Counter_Type</i>	Any type.
<i>Data_Rule_Type</i>	Any type.

Definition at line 47 of file model-bones.hpp.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 Model() [1/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Model ( )
```

7.17.2.2 Model() [2/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Model (
    uint size_ )
```

7.17.2.3 Model() [3/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Model (
    const Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >
    & Model_ )
```

7.17.2.4 ~Model()

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

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

7.17.3 Member Function Documentation

7.17.3.1 add_array()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_array (
    const Array_Type & Array_,
    bool force_new = false )
```

Adds an array to the support of not already included.

Parameters

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

Returns

The number of the array.

7.17.3.2 add_counter() [1/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_counter (
    Counter< Array_Type, Data_Counter_Type > & counter )
```

7.17.3.3 add_counter() [2/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_counter (
    Counter< Array_Type, Data_Counter_Type > * counter )
```

7.17.3.4 add_counter() [3/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_counter (
    Counter_fun_type< Array_Type, Data_Counter_Type > count_fun_,
    Counter_fun_type< Array_Type, Data_Counter_Type > init_fun_ = nullptr,
    Data_Counter_Type * data_ = nullptr,
    bool delete_data_ = false )
```

7.17.3.5 add_rule() [1/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
    Rule< Array_Type, Data_Rule_Type > & rule )
```

7.17.3.6 add_rule() [2/3]

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

7.17.3.7 add_rule() [3/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule (
    Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_,
    Data_Rule_Type * data_ = nullptr,
    bool delete_data_ = false )
```

7.17.3.8 add_rule_dyn() [1/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_dyn
(
    Rule< Array_Type, Data_Rule_Dyn_Type > & rule )
```

7.17.3.9 add_rule_dyn() [2/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_dyn
(
    Rule< Array_Type, Data_Rule_Dyn_Type > * rule )
```

7.17.3.10 add_rule_dyn() [3/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_dyn
(
    Rule_fun_type< Array_Type, Data_Rule_Dyn_Type > count_fun_,
    Data_Rule_Dyn_Type * data_ = nullptr,
    bool delete_data_ = false )
```

7.17.3.11 colnames()

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

7.17.3.12 conditional_prob()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::conditional↵
_prob (
    const Array_Type & Array_,
    const std::vector< double > & params,
    unsigned int i,
    unsigned int j )
```

Conditional probability ("Gibbs sampler")

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

Parameters

<i>Array_↵</i>	Array to check
<i>params</i>	Vector of parameters
<i>i</i>	Row entry
<i>j</i>	Column entry

Returns

double The conditional probability

7.17.3.13 gen_key()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< double > Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_↵
Type >::gen_key (
    const Array_Type & Array_ )
```

7.17.3.14 get_counters()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Counters<Array_Type,Data_Counter_Type>* Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_counters ( )
```

7.17.3.15 get_norm_const()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_norm_↵
const (
    const std::vector< double > & params,
    const uint & i,
    bool as_log = false )
```

7.17.3.16 get_pset()

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

7.17.3.17 get_pset_stats()

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

7.17.3.18 get_rengine()

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

7.17.3.19 get_rules()

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

7.17.3.20 get_rules_dyn()

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

7.17.3.21 get_support()

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


7.17.3.22 likelihood() [1/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::likelihood
(
    const std::vector< double > & params,
    const Array_Type & Array_,
    int i = -1,
    bool as_log = false )
```

7.17.3.23 likelihood() [2/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::likelihood
(
    const std::vector< double > & params,
    const std::vector< double > & target_,
    const uint & i,
    bool as_log = false )
```

7.17.3.24 likelihood() [3/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::likelihood
(
    const std::vector< double > & params,
    const uint & i,
    bool as_log = false )
```

7.17.3.25 likelihood_total()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::likelihood_total
(
    const std::vector< double > & params,
    bool as_log = false )
```

7.17.3.26 nterms()

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

7.17.3.27 operator=()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Model<Array_Type,Data_Counter_Type,Data_Rule_Type,Data_Rule_Dyn_Type>& Model< Array_Type,
Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::operator= (
    const Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >
    & Model_ )
```

7.17.3.28 print()

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

Prints information about the model.

7.17.3.29 print_stats()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::print_stats (
    uint i ) const
```

7.17.3.30 sample() [1/2]

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

7.17.3.31 sample() [2/2]

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

7.17.3.32 set_counters()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_counters
(
    Counters< Array_Type, Data_Counter_Type > * counters_ )
```

7.17.3.33 set_keygen()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_keygen (
    std::function< std::vector< double >(const Array_Type &)> keygen_ )
```

7.17.3.34 set_rengine()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_rengine (
    std::mt19937 * rengine_,
    bool delete_ = false ) [inline]
```

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

7.17.3.35 set_rules()

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

7.17.3.36 set_rules_dyn()

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

7.17.3.37 set_seed()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_seed (
    unsigned int s ) [inline]
```

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

7.17.3.38 size()

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

7.17.3.39 size_unique()

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

7.17.3.40 store_psets()

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

7.17.3.41 support_size()

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

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

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

7.18 NetCounterData Class Reference

Data class used to store arbitrary uint or double vectors.

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

Public Member Functions

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

Public Attributes

- std::vector< [uint](#) > [indices](#)
- std::vector< double > [numbers](#)

7.18.1 Detailed Description

Data class used to store arbitrary uint or double vectors.

Definition at line 61 of file network.hpp.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 NetCounterData() [1/2]

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

Definition at line 67 of file network.hpp.

7.18.2.2 NetCounterData() [2/2]

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

Definition at line 68 of file network.hpp.

7.18.2.3 ~NetCounterData()

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

Definition at line 73 of file network.hpp.

7.18.3 Member Data Documentation

7.18.3.1 indices

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

Definition at line 64 of file network.hpp.

7.18.3.2 numbers

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

Definition at line 65 of file network.hpp.

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

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

7.19 NetworkData Class Reference

Data class for Networks.

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

Public Member Functions

- [NetworkData](#) ()
- [NetworkData](#) (std::vector< double > vertex_attr_, bool directed_=true)
Constructor using a single attribute.
- [NetworkData](#) (std::vector< std::vector< double > > vertex_attr_, bool directed_=true)
Constructor using multiple attributes.
- [~NetworkData](#) ()

Public Attributes

- bool [directed](#) = true
- std::vector< std::vector< double > > [vertex_attr](#)

7.19.1 Detailed Description

Data class for Networks.

Details on the available counters for [NetworkData](#) can be found in the [Network counters](#) section.

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

Definition at line 24 of file network.hpp.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 NetworkData() [1/3]

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

Definition at line 30 of file network.hpp.

7.19.2.2 NetworkData() [2/3]

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

Constructor using a single attribute.

Parameters

<i>vertex_↔ attr_</i>	Double vector of length equal to the number of vertices in the data.
<i>directed_</i>	When <code>true</code> the graph as treated as directed.

Definition at line 38 of file network.hpp.

7.19.2.3 NetworkData() [3/3]

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

Constructor using multiple attributes.

Parameters

<i>vertex_↔ attr_</i>	Vector of double vectors. The size equals to the number of attributes to be created. Each individual vector should be of length equal to the number of vertices.
<i>directed_</i>	When <code>true</code> the graph as treated as directed.

Definition at line 50 of file network.hpp.

7.19.2.4 ~NetworkData()

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

Definition at line 56 of file network.hpp.

7.19.3 Member Data Documentation**7.19.3.1 directed**

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

Definition at line 27 of file network.hpp.

7.19.3.2 vertex_attr

```
std::vector< std::vector< double > > NetworkData::vertex_attr
```

Definition at line 28 of file network.hpp.

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

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

7.20 Node Class Reference

A single node for the model.

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

Collaboration diagram for Node:



Public Member Functions

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

Construct a new Node object

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

Public Attributes

- unsigned int `id`
Id of the node (as specified in the input)
- unsigned int `ord`
Order in which the node was created.
- `phylocounters::PhyloArray` `array`
- `std::vector< unsigned int >` `annotations`
Observed annotations (only defined for [Geese](#))
- bool `duplication`
- `std::vector< phylocounters::PhyloArray >` `arrays` = {}
Arrays given all possible states.
- `Node *` `parent` = nullptr
Parent node.
- `std::vector< Node * >` `offspring` = {}
Offspring nodes.
- `std::vector< unsigned int >` `narray` = {}
ID of the array in the model.
- bool `visited` = false
- `std::vector< double >` `subtree_prob`
Induced subtree probabilities.
- `std::vector< double >` `probability`
The probability of observing each state.

7.20.1 Detailed Description

A single node for the model.

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

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

7.20.2 Constructor & Destructor Documentation

7.20.2.1 `Node()` [1/5]

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

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

7.20.2.2 Node() [2/5]

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

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

7.20.2.3 Node() [3/5]

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

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

7.20.2.4 Node() [4/5]

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

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

7.20.2.5 Node() [5/5]

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

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

7.20.2.6 ~Node()

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

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

7.20.3 Member Function Documentation

7.20.3.1 `get_parent()`

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

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

7.20.3.2 `is_leaf()`

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

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

7.20.3.3 `noffspring()`

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

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

7.20.4 Member Data Documentation

7.20.4.1 `annotations`

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

Observed annotations (only defined for [Geese](#))

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

7.20.4.2 `array`

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

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

7.20.4.3 arrays

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

Arrays given all possible states.

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

7.20.4.4 duplication

```
bool Node::duplication
```

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

7.20.4.5 id

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

Id of the node (as specified in the input)

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

7.20.4.6 narray

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

ID of the array in the model.

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

7.20.4.7 offspring

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

Offspring nodes.

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

7.20.4.8 ord

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

Order in which the node was created.

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

7.20.4.9 parent

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

Parent node.

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

7.20.4.10 probability

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

The probability of observing each state.

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

7.20.4.11 subtree_prob

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

Induced subtree probabilities.

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

7.20.4.12 visited

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

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

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

- [include/barry/models/geese/geese-node-bones.hpp](#)

7.21 NodeData Class Reference

Data definition for the `PhyloArray` class.

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

Public Member Functions

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

Public Attributes

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

7.21.1 Detailed Description

Data definition for the `PhyloArray` class.

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

This holds basic information about a given node.

Definition at line 23 of file `phylo.hpp`.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 NodeData()

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

Definition at line 43 of file `phylo.hpp`.

7.21.3 Member Data Documentation

7.21.3.1 blengths

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

Branch length.

Definition at line 29 of file phylo.hpp.

7.21.3.2 duplication

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

Definition at line 39 of file phylo.hpp.

7.21.3.3 states

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

State of the parent node.

Definition at line 34 of file phylo.hpp.

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

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

7.22 PhyloCounterData Class Reference

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

Public Member Functions

- [PhyloCounterData](#) (std::vector< [uint](#) > [data_](#), std::vector< double > *[counters_](#)=nullptr)
- [uint](#) at ([uint](#) d)
- [uint](#) operator() ([uint](#) d)
- void [reserve](#) ([uint](#) x)
- void [push_back](#) ([uint](#) x)
- void [shrink_to_fit](#) ()
- [uint](#) size ()
- std::vector< [uint](#) >::iterator [begin](#) ()
- std::vector< [uint](#) >::iterator [end](#) ()
- bool [empty](#) ()
- std::vector< double > * [get_counters](#) ()

7.22.1 Detailed Description

Definition at line 54 of file phylo.hpp.

7.22.2 Constructor & Destructor Documentation

7.22.2.1 PhyloCounterData()

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

Definition at line 60 of file phylo.hpp.

7.22.3 Member Function Documentation

7.22.3.1 at()

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

Definition at line 65 of file phylo.hpp.

7.22.3.2 begin()

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

Definition at line 72 of file phylo.hpp.

7.22.3.3 empty()

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

Definition at line 75 of file phylo.hpp.

7.22.3.4 end()

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

Definition at line 73 of file phylo.hpp.

7.22.3.5 get_counters()

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

Definition at line 76 of file phylo.hpp.

7.22.3.6 operator()()

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

Definition at line 66 of file phylo.hpp.

7.22.3.7 push_back()

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

Definition at line 68 of file phylo.hpp.

7.22.3.8 reserve()

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

Definition at line 67 of file phylo.hpp.

7.22.3.9 shrink_to_fit()

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

Definition at line 69 of file phylo.hpp.

7.22.3.10 size()

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

Definition at line 70 of file phylo.hpp.

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

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

7.23 PhyloRuleDynData Class Reference

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

Public Member Functions

- [PhyloRuleDynData](#) (const std::vector< double > *counts_, [uint](#) pos_, [uint](#) lb_, [uint](#) ub_, bool duplication_)
- [~PhyloRuleDynData](#) ()

Public Attributes

- const std::vector< double > * [counts](#)
- [uint](#) [pos](#)
- [uint](#) [lb](#)
- [uint](#) [ub](#)
- bool [duplication](#)

7.23.1 Detailed Description

Definition at line 1289 of file phylo.hpp.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 PhyloRuleDynData()

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

Definition at line 1296 of file phylo.hpp.

7.23.2.2 ~PhyloRuleDynData()

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

Definition at line 1305 of file phylo.hpp.

7.23.3 Member Data Documentation

7.23.3.1 counts

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

Definition at line 1291 of file phylo.hpp.

7.23.3.2 duplication

```
bool PhyloRuleDynData::duplication
```

Definition at line 1295 of file phylo.hpp.

7.23.3.3 lb

```
uint PhyloRuleDynData::lb
```

Definition at line 1293 of file phylo.hpp.

7.23.3.4 pos

```
uint PhyloRuleDynData::pos
```

Definition at line 1292 of file phylo.hpp.

7.23.3.5 ub

```
uint PhyloRuleDynData::ub
```

Definition at line 1294 of file phylo.hpp.

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

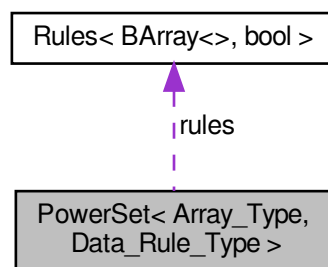
- [include/barry/counters/phylo.hpp](#)

7.24 PowerSet< Array_Type, Data_Rule_Type > Class Template Reference

Powerset of a binary array.

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

Collaboration diagram for PowerSet< Array_Type, Data_Rule_Type >:



Public Member Functions

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

Construct and destroy a PowerSet object

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

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

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

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

Getter functions

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

Public Attributes

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

7.24.1 Detailed Description

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
class PowerSet< Array_Type, Data_Rule_Type >
```

Powerset of a binary array.

Template Parameters

<i>Array_Type</i>	
<i>Data_Rule_Type</i>	

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

7.24.2 Constructor & Destructor Documentation

7.24.2.1 PowerSet() [1/3]

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

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

7.24.2.2 PowerSet() [2/3]

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

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

7.24.2.3 PowerSet() [3/3]

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

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

7.24.2.4 ~PowerSet()

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

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

7.24.3 Member Function Documentation**7.24.3.1 add_rule()** [1/3]

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::add_rule (
    Rule< Array_Type, Data_Rule_Type > & rule ) [inline]
```

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

7.24.3.2 add_rule() [2/3]

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::add_rule (
    Rule< Array_Type, Data_Rule_Type > * rule ) [inline]
```

Definition at line 122 of file powerset-meat.hpp.

7.24.3.3 add_rule() [3/3]

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::add_rule (
    Rule_fun_type< Array_Type, Data_Rule_Type > count_fun_,
    Data_Rule_Type * data_ = nullptr,
    bool delete_data_ = false ) [inline]
```

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

7.24.3.4 begin()

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

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

7.24.3.5 calc()

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

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

7.24.3.6 end()

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

Definition at line 74 of file powerset-bones.hpp.

7.24.3.7 get_data()

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

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

7.24.3.8 get_data_ptr()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
const std::vector< Array_Type >* PowerSet< Array_Type, Data_Rule_Type >::get_data_ptr ( )
const [inline]
```

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

7.24.3.9 init_support()

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

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

7.24.3.10 operator[]()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
const Array_Type& PowerSet< Array_Type, Data_Rule_Type >::operator[] (
    const unsigned int & i ) const [inline]
```

Definition at line 76 of file powerset-bones.hpp.

7.24.3.11 reset()

```
template<typename Array_Type , typename Data_Rule_Type >
void PowerSet< Array_Type, Data_Rule_Type >::reset (
    uint N_,
    uint M_ ) [inline]
```

Definition at line 101 of file powerset-meat.hpp.

7.24.3.12 size()

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::size_t PowerSet< Array_Type, Data_Rule_Type >::size ( ) const [inline], [noexcept]
```

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

7.24.4 Member Data Documentation

7.24.4.1 coordinates_free

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< std::pair<uint,uint> > PowerSet< Array_Type, Data_Rule_Type >::coordinates_free
```

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

7.24.4.2 coordinates_locked

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< std::pair<uint, uint> > PowerSet< Array_Type, Data_Rule_Type >::coordinates_←
locked
```

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

7.24.4.3 data

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< Array_Type > PowerSet< Array_Type, Data_Rule_Type >::data
```

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

7.24.4.4 EmptyArray

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
Array_Type PowerSet< Array_Type, Data_Rule_Type >::EmptyArray
```

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

7.24.4.5 M

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
uint PowerSet< Array_Type, Data_Rule_Type >::M
```

Definition at line 27 of file powerset-bones.hpp.

7.24.4.6 N

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
uint PowerSet< Array_Type, Data_Rule_Type >::N
```

Definition at line 27 of file powerset-bones.hpp.

7.24.4.7 rules

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
Rules<Array_Type, Data_Rule_Type>* PowerSet< Array_Type, Data_Rule_Type >::rules
```

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

7.24.4.8 rules_deleted

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
bool PowerSet< Array_Type, Data_Rule_Type >::rules_deleted = false
```

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

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

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

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

- [Rule](#) ()
- [Rule](#) ([Rule_fun_type](#)< Array_Type, Data_Type > fun_, Data_Type *dat_=nullptr, bool delete_dat_=false)

7.25.1 Detailed Description

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
class Rule< Array_Type, Data_Type >
```

[Rule](#) for determining if a cell should be included in a sequence.

[Rules](#) can be used together with [Support](#) and [PowerSet](#) to determine which cells should be included when enumerating all possible realizations of a binary array.

Template Parameters

<i>Array_Type</i>	An object of class BArray .
<i>Data_Type</i>	Any type.

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

7.25.2 Constructor & Destructor Documentation

7.25.2.1 Rule() [1/2]

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

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

7.25.2.2 Rule() [2/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Rule< Array_Type, Data_Type >::Rule (
    Rule_fun_type< Array_Type, Data_Type > fun_,
    Data_Type * dat_ = nullptr,
    bool delete_dat_ = false ) [inline]
```

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

7.25.2.3 ~Rule()

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

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

7.25.3 Member Function Documentation

7.25.3.1 D()

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

Read/Write access to the data.

7.25.3.2 operator()()

```
template<typename Array_Type , typename Data_Type >
bool Rule< Array_Type, Data_Type >::operator() (
    const Array_Type & a,
    uint i,
    uint j ) [inline]
```

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

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

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

7.26 Rules< Array_Type, Data_Type > Class Template Reference

Vector of objects of class [Rule](#).

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

Public Member Functions

- [Rules](#) ()
- [Rules](#) (const [Rules](#)< Array_Type, Data_Type > &rules_)
- [Rules](#)< Array_Type, Data_Type > [operator=](#) (const [Rules](#)< Array_Type, Data_Type > &rules_)
- [~Rules](#) ()
- [uint size](#) () const [noexcept](#)
- bool [operator\(\)](#) (const Array_Type &a, [uint](#) i, [uint](#) j)
Check whether a given cell is free or locked.
- void [clear](#) ()
- void [get_seq](#) (const Array_Type &a, std::vector< std::pair< [uint](#), [uint](#) > > *free, std::vector< std::pair< [uint](#), [uint](#) > > *locked=nullptr)
Computes the sequence of free and locked cells in an [BArray](#).

Rule adding

Parameters

rule	
------	--

- void [add_rule](#) ([Rule](#)< Array_Type, Data_Type > &rule)
- void [add_rule](#) ([Rule](#)< Array_Type, Data_Type > *rule)
- void [add_rule](#) ([Rule_fun_type](#)< Array_Type, Data_Type > rule_, Data_Type *[data_](#)=nullptr, bool [delete_data_](#)=false)

7.26.1 Detailed Description

```
template<typename Array_Type, typename Data_Type>
class Rules< Array_Type, Data_Type >
```

Vector of objects of class [Rule](#).

Template Parameters

<i>Array_Type</i>	An object of class BArray
<i>Data_Type</i>	Any type.

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

7.26.2 Constructor & Destructor Documentation**7.26.2.1 Rules() [1/2]**

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::Rules ( ) [inline]
```

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

7.26.2.2 Rules() [2/2]

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::Rules (
    const Rules< Array_Type, Data_Type > & rules_ ) [inline]
```

Definition at line 10 of file rules-meat.hpp.

7.26.2.3 ~Rules()

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::~Rules ( ) [inline]
```

Definition at line 81 of file rules-bones.hpp.

7.26.3 Member Function Documentation**7.26.3.1 add_rule() [1/3]**

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::add_rule (
    Rule< Array_Type, Data_Type > & rule ) [inline]
```

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

7.26.3.2 add_rule() [2/3]

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::add_rule (
    Rule< Array_Type, Data_Type > * rule ) [inline]
```

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

7.26.3.3 add_rule() [3/3]

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::add_rule (
    Rule_fun_type< Array_Type, Data_Type > rule_,
    Data_Type * data_ = nullptr,
    bool delete_data_ = false ) [inline]
```

Definition at line 89 of file rules-meat.hpp.

7.26.3.4 clear()

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::clear [inline]
```

Definition at line 127 of file rules-meat.hpp.

7.26.3.5 get_seq()

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::get_seq (
    const Array_Type & a,
    std::vector< std::pair< uint, uint > > * free,
    std::vector< std::pair< uint, uint > > * locked = nullptr ) [inline]
```

Computes the sequence of free and locked cells in an [BArray](#).

Parameters

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

Returns

Nothing.

Definition at line 139 of file rules-meat.hpp.

7.26.3.6 operator()

```
template<typename Array_Type , typename Data_Type >
bool Rules< Array_Type, Data_Type >::operator() (
    const Array_Type & a,
    uint i,
    uint j ) [inline]
```

Check whether a given cell is free or locked.

Parameters

<i>a</i>	A BArray object
<i>i</i>	row position
<i>j</i>	col position

Returns

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

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

7.26.3.7 operator=()

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type > Rules< Array_Type, Data_Type >::operator= (
    const Rules< Array_Type, Data_Type > & rules_ )
```

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

7.26.3.8 size()

```
template<typename Array_Type , typename Data_Type >
uint Rules< Array_Type, Data_Type >::size ( ) const [inline], [noexcept]
```

Definition at line 86 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

7.27 StatsCounter< Array_Type, Data_Type > Class Template Reference

Count stats for a single Array.

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

Public Member Functions

- [StatsCounter](#) (const Array_Type *Array_)
Creator of a StatsCounter
- [StatsCounter](#) ()
Can be created without setting the array.
- [~StatsCounter](#) ()
- void [reset_array](#) (const Array_Type *Array_)
Changes the reference array for the counting.
- void [add_counter](#) (Counter< Array_Type, Data_Type > *f_)
- void [add_counter](#) (Counter< Array_Type, Data_Type > f_)
- void [set_counters](#) (Counters< Array_Type, Data_Type > *counters_)
- void [count_init](#) (uint i, uint j)
Counter functions This function recurses through the entries of Array and at each step of adding a new cell it uses the functions to list the statistics.
- void [count_current](#) (uint i, uint j)
- std::vector< double > [count_all](#) ()
- Counters< Array_Type, Data_Type > * [get_counters](#) ()
- std::vector< std::string > [get_names](#) () const
- std::vector< std::string > [get_descriptions](#) () const

7.27.1 Detailed Description

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
class StatsCounter< Array_Type, Data_Type >
```

Count stats for a single Array.

Users can a list of functions that can be used with this. The baseline set of arguments is a pointer to a binary array and a dataset to add the counts to.

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

7.27.2 Constructor & Destructor Documentation

7.27.2.1 StatsCounter() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
StatsCounter< Array_Type, Data_Type >::StatsCounter (
    const Array_Type * Array_ ) [inline]
```

Creator of a [StatsCounter](#)

Parameters

Array ↔	A const pointer to a BArray .
—	

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

7.27.2.2 StatsCounter() [2/2]

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

Can be created without setting the array.

Definition at line 51 of file statscounter-bones.hpp.

7.27.2.3 ~StatsCounter()

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

7.27.3 Member Function Documentation

7.27.3.1 add_counter() [1/2]

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

7.27.3.2 add_counter() [2/2]

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

7.27.3.3 count_all()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::vector< double > StatsCounter< Array_Type, Data_Type >::count_all ( )
```

7.27.3.4 count_current()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
void StatsCounter< Array_Type, Data_Type >::count_current (
    uint i,
    uint j )
```

7.27.3.5 count_init()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
void StatsCounter< Array_Type, Data_Type >::count_init (
    uint i,
    uint j )
```

Counter functions This function recurses through the entries of `Array` and at each step of adding a new cell it uses the functions to list the statistics.

7.27.3.6 get_counters()

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

7.27.3.7 get_descriptions()

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

7.27.3.8 get_names()

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

7.27.3.9 reset_array()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
void StatsCounter< Array_Type, Data_Type >::reset_array (
    const Array_Type * Array_ )
```

Changes the reference array for the counting.

Parameters

<i>Array_</i>	A pointer to an array of class Array_Type.
—	

7.27.3.10 set_counters()

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
void StatsCounter< Array_Type, Data_Type >::set_counters (
    Counters< Array_Type, Data_Type > * counters_ )
```

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

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

7.28 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference

Compute the support of sufficient statistics.

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

Public Member Functions

- [Support](#) (const Array_Type &Array_)
Constructor passing a reference Array.
- [Support](#) (uint N_, uint M_)
Constructor specifying the dimensions of the array (empty).
- [Support](#) ()
- [~Support](#) ()

- void [init_support](#) (std::vector< Array_Type > *[array_bank](#)=nullptr, std::vector< std::vector< double > > *[stats_bank](#)=nullptr)
- void [calc](#) (std::vector< Array_Type > *[array_bank](#)=nullptr, std::vector< std::vector< double > > *[stats_bank](#)=nullptr, unsigned int max_num_elements_=0u)
Computes the entire support.
- [Counts_type](#) [get_counts](#) () const
- const [MapVec_type](#) * [get_counts_ptr](#) () const
- std::vector< double > * [get_current_stats](#) ()
List current statistics.
- void [print](#) () const
- const [FreqTable](#) & [get_data](#) () const
- [Counters](#)< Array_Type, Data_Counter_Type > * [get_counters](#) ()
Vector of counter functions.
- [Rules](#)< Array_Type, Data_Rule_Type > * [get_rules](#) ()
Vector of static rules (cells to iterate).
- [Rules](#)< Array_Type, Data_Rule_Dyn_Type > * [get_rules_dyn](#) ()
Vector of dynamic rules (to include/exclude a realization).

Resets the support calculator

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

Parameters

Array↔	New array over which the support will be computed.
—	

- void [reset_array](#) ()
- void [reset_array](#) (const Array_Type &Array_)

Manage counters

Parameters

f_	A counter to be added.
counters↔	A vector of counters to be added.
—	

- void [add_counter](#) ([Counter](#)< Array_Type, Data_Counter_Type > *[f_](#))
- void [add_counter](#) ([Counter](#)< Array_Type, Data_Counter_Type > [f_](#))
- void [set_counters](#) ([Counters](#)< Array_Type, Data_Counter_Type > *[counters_](#))

Manage rules

Parameters

f_	A rule to be added.
counters↔	A vector of rules to be added.
—	

- void [add_rule](#) ([Rule](#)< Array_Type, Data_Rule_Type > *[f_](#))

- void `add_rule` (`Rule`< `Array_Type`, `Data_Rule_Type` > `f_`)
- void `set_rules` (`Rules`< `Array_Type`, `Data_Rule_Type` > `*rules_`)
- void `add_rule_dyn` (`Rule`< `Array_Type`, `Data_Rule_Dyn_Type` > `*f_`)
- void `add_rule_dyn` (`Rule`< `Array_Type`, `Data_Rule_Dyn_Type` > `f_`)
- void `set_rules_dyn` (`Rules`< `Array_Type`, `Data_Rule_Dyn_Type` > `*rules_`)
- bool `eval_rules_dyn` (const std::vector< double > &counts, const `uint` &i, const `uint` &j)

Public Attributes

- `uint` `N`
- `uint` `M`
- bool `delete_counters` = true
- bool `delete_rules` = true
- bool `delete_rules_dyn` = true
- `uint` `max_num_elements` = `BARRY_MAX_NUM_ELEMENTS`
- std::vector< double > `current_stats`
- std::vector< std::pair< `uint`, `uint` > > `coordinates_free`
- std::vector< std::pair< `uint`, `uint` > > `coordinates_locked`
- std::vector< std::vector< double > > `change_stats`

7.28.1 Detailed Description

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

```
class Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >
```

Compute the support of sufficient statistics.

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

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

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

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

7.28.2 Constructor & Destructor Documentation

7.28.2.1 Support() [1/3]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↔
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::Support (
    const Array_Type & Array_ ) [inline]
```

Constructor passing a reference Array.

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

7.28.2.2 Support() [2/3]

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

Constructor specifying the dimensions of the array (empty).

Definition at line 78 of file support-bones.hpp.

7.28.2.3 Support() [3/3]

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

Definition at line 85 of file support-bones.hpp.

7.28.2.4 ~Support()

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

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

7.28.3 Member Function Documentation

7.28.3.1 add_counter() [1/2]

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

7.28.3.2 add_counter() [2/2]

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

7.28.3.3 add_rule() [1/2]

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

7.28.3.4 add_rule() [2/2]

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

7.28.3.5 add_rule_dyn() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_↵
dyn (
    Rule< Array_Type, Data_Rule_Dyn_Type > * f_ )
```

7.28.3.6 add_rule_dyn() [2/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::add_rule_↵
dyn (
    Rule< Array_Type, Data_Rule_Dyn_Type > f_ )
```


7.28.3.7 calc()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::calc (
    std::vector< Array_Type > * array_bank = nullptr,
    std::vector< std::vector< double > > * stats_bank = nullptr,
    unsigned int max_num_elements_ = 0u )
```

Computes the entire support.

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

Parameters

<i>array_bank</i>	If specified, the counter will add to the vector each possible state of the array, as it counts.
<i>stats_bank</i>	If specified, the counter will add to the vector each possible set of statistics, as it counts.

7.28.3.8 eval_rules_dyn()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::eval_rules_dyn (
    const std::vector< double > & counts,
    const uint & i,
    const uint & j )
```

7.28.3.9 get_counters()

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

Vector of counter functions.

7.28.3.10 get_counts()

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

7.28.3.11 get_counts_ptr()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
const MapVec_type* Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_counts_ptr ( ) const
```

7.28.3.12 get_current_stats()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< double >* Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_current_stats ( )
```

List current statistics.

7.28.3.13 get_data()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
const FreqTable& Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_data ( ) const
```

7.28.3.14 get_rules()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Rules<Array_Type,Data_Rule_Type>* Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_rules ( )
```

Vector of static rules (cells to iterate).

7.28.3.15 get_rules_dyn()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Rules<Array_Type,Data_Rule_Dyn_Type>* Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_rules_dyn ( )
```

Vector of dynamic rules (to include/exclude a realization).

7.28.3.16 init_support()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::init_support (
    std::vector< Array_Type > * array_bank = nullptr,
    std::vector< std::vector< double > > * stats_bank = nullptr )
```

7.28.3.17 print()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::print ( )
const
```

7.28.3.18 reset_array() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::reset_array
( )
```

7.28.3.19 reset_array() [2/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::reset_array
(
    const Array_Type & Array_ )
```

7.28.3.20 set_counters()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_↵
counters (
    Counters< Array_Type, Data_Counter_Type > * counters_ )
```

7.28.3.21 set_rules()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_rules (
    Rules< Array_Type, Data_Rule_Type > * rules_ )
```

7.28.3.22 set_rules_dyn()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_rules_dyn (
    Rules< Array_Type, Data_Rule_Dyn_Type > * rules_ )
```

7.28.4 Member Data Documentation

7.28.4.1 change_stats

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector< double > > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::change_stats
```

Definition at line 65 of file support-bones.hpp.

7.28.4.2 coordinates_free

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::pair<uint, uint> > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::coordinates_free
```

Definition at line 63 of file support-bones.hpp.

7.28.4.3 coordinates_locked

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::pair<uint, uint> > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::coordinates_locked
```

Definition at line 64 of file support-bones.hpp.

7.28.4.4 current_stats

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵  
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>  
std::vector< double > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_↵  
_Type >::current_stats
```

Definition at line 62 of file support-bones.hpp.

7.28.4.5 delete_counters

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵  
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>  
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↵  
counters = true
```

Definition at line 56 of file support-bones.hpp.

7.28.4.6 delete_rules

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵  
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>  
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↵  
rules = true
```

Definition at line 57 of file support-bones.hpp.

7.28.4.7 delete_rules_dyn

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵  
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>  
bool Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::delete_↵  
rules_dyn = true
```

Definition at line 58 of file support-bones.hpp.

7.28.4.8 M

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵  
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>  
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::M
```

Definition at line 55 of file support-bones.hpp.

7.28.4.9 max_num_elements

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::max_num_elements = BARRY_MAX_NUM_ELEMENTS
```

Definition at line 59 of file support-bones.hpp.

7.28.4.10 N

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
uint Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::N
```

Definition at line 55 of file support-bones.hpp.

The documentation for this class was generated from the following file:

- [include/barry/support-bones.hpp](#)

7.29 vecHasher< T > Struct Template Reference

```
#include <typedefs.hpp>
```

Public Member Functions

- `std::size_t operator() (std::vector< T > const &dat) const` [noexcept](#)

7.29.1 Detailed Description

```
template<typename T>
struct vecHasher< T >
```

Definition at line 86 of file typedefs.hpp.

7.29.2 Member Function Documentation

7.29.2.1 operator()()

```
template<typename T >
std::size_t vecHasher< T >::operator() (
    std::vector< T > const & dat ) const [inline], [noexcept]
```

Definition at line 87 of file typedefs.hpp.

The documentation for this struct was generated from the following file:

- [include/barry/typedefs.hpp](#)

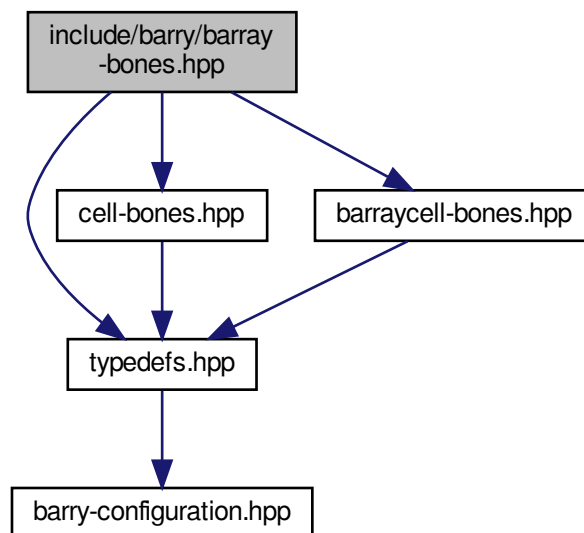
Chapter 8

File Documentation

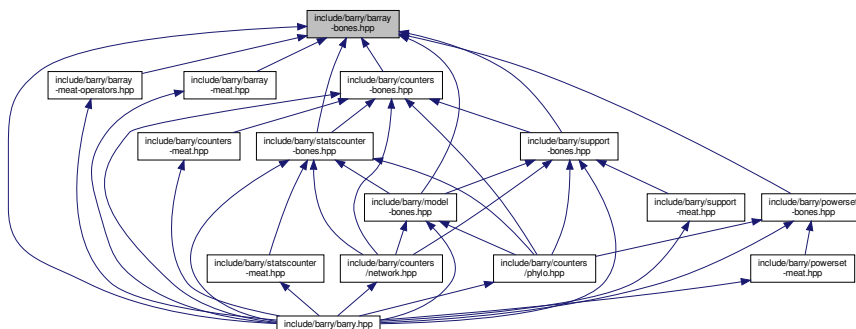
8.1 include/barry/barray-bones.hpp File Reference

```
#include "typedefs.hpp"  
#include "cell-bones.hpp"  
#include "barraycell-bones.hpp"
```

Include dependency graph for barray-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [BArray< Cell_Type, Data_Type >](#)
Baseline class for binary arrays.

Macros

- `#define BARRAY_BONES_HPP 1`

8.1.1 Macro Definition Documentation

8.1.1.1 BARRAY_BONES_HPP

```
#define BARRAY_BONES_HPP 1
```

Definition at line 8 of file `barray-bones.hpp`.

8.2 include/barry/barray-iterator.hpp File Reference

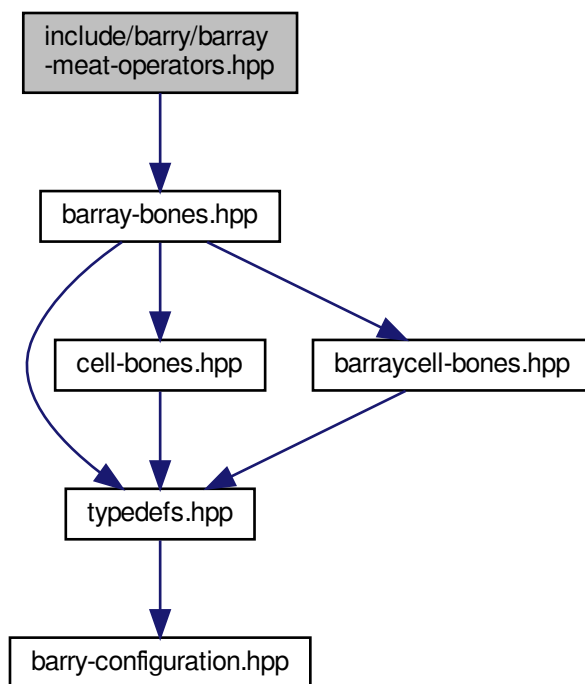
Classes

- class [ConstBArrayRowIter< Cell_Type, Data_Type >](#)

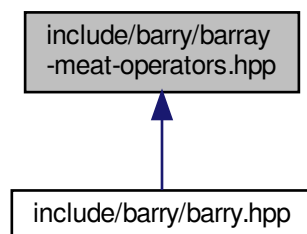
8.3 include/barry/barray-meat-operators.hpp File Reference

```
#include "barray-bones.hpp"
```

Include dependency graph for barray-meat-operators.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- `#define BARRY_BARRAY_MEAT_OPERATORS_HPP 1`
- `#define ROW(a) this->el_ij[a]`
- `#define COL(a) this->el_ji[a]`

Functions

- `template<typename Cell_Type , typename Data_Type >`
`void checkdim_ (const BArray< Cell_Type, Data_Type > &lhs, const BArray< Cell_Type, Data_Type > &rhs)`

8.3.1 Macro Definition Documentation

8.3.1.1 BARRY_BARRAY_MEAT_OPERATORS_HPP

```
#define BARRY_BARRAY_MEAT_OPERATORS_HPP 1
```

Definition at line 5 of file `barray-meat-operators.hpp`.

8.3.1.2 COL

```
#define COL(  
    a ) this->el_ji[a]
```

Definition at line 8 of file `barray-meat-operators.hpp`.

8.3.1.3 ROW

```
#define ROW(  
    a ) this->el_ij[a]
```

Definition at line 7 of file `barray-meat-operators.hpp`.

8.3.2 Function Documentation

8.3.2.1 checkdim_()

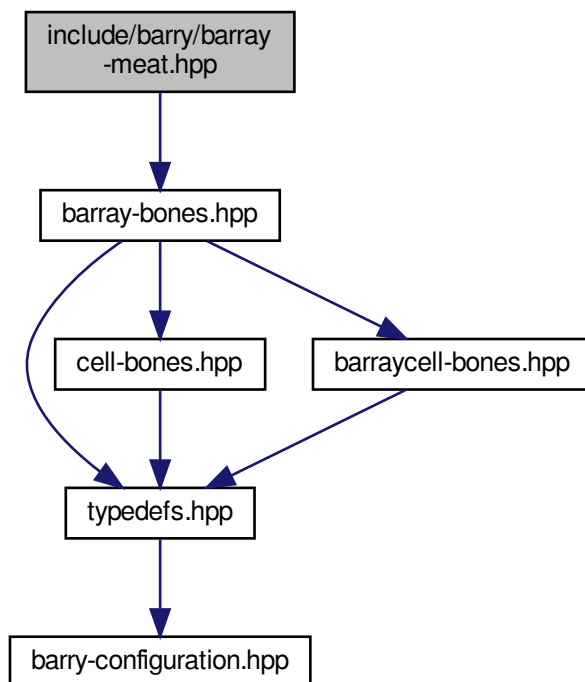
```
template<typename Cell_Type , typename Data_Type >  
void checkdim_ (  
    const BArray< Cell_Type, Data_Type > & lhs,  
    const BArray< Cell_Type, Data_Type > & rhs ) [inline]
```

Definition at line 11 of file `barray-meat-operators.hpp`.

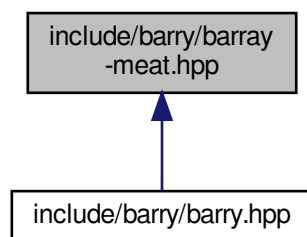
8.4 include/barry/barray-meat.hpp File Reference

```
#include "barray-bones.hpp"
```

Include dependency graph for barray-meat.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- `#define ROW(a) this->el_ij[a]`
- `#define COL(a) this->el_ji[a]`

8.4.1 Macro Definition Documentation

8.4.1.1 COL

```
#define COL(  
    a ) this->el_ji[a]
```

Definition at line 8 of file `barray-meat.hpp`.

8.4.1.2 ROW

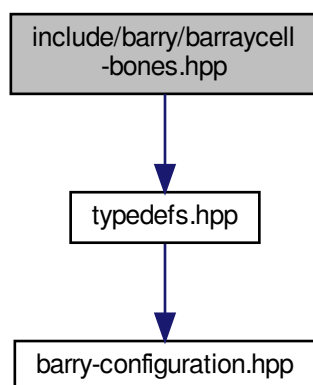
```
#define ROW(  
    a ) this->el_ij[a]
```

Definition at line 7 of file `barray-meat.hpp`.

8.5 `include/barry/barraycell-bones.hpp` File Reference

```
#include "typedefs.hpp"
```

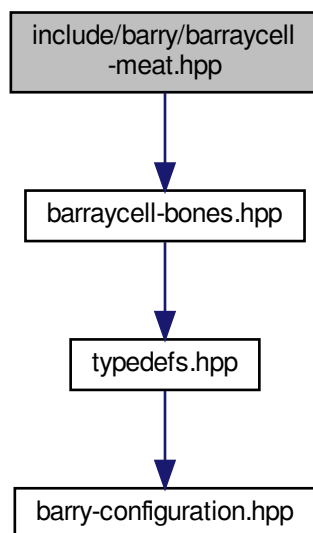
Include dependency graph for `barraycell-bones.hpp`:



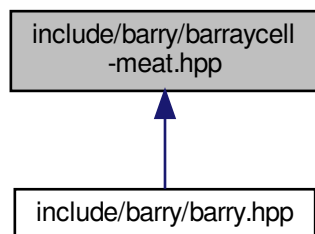
[illegible]

- class `BArrayCell< Cell_Type, Data_Type >`
- class `BArrayCell_const< Cell_Type, Data_Type >`

```
#include "barraycell-bones.hpp"
Include dependency graph for barraycell-meat.hpp:
```



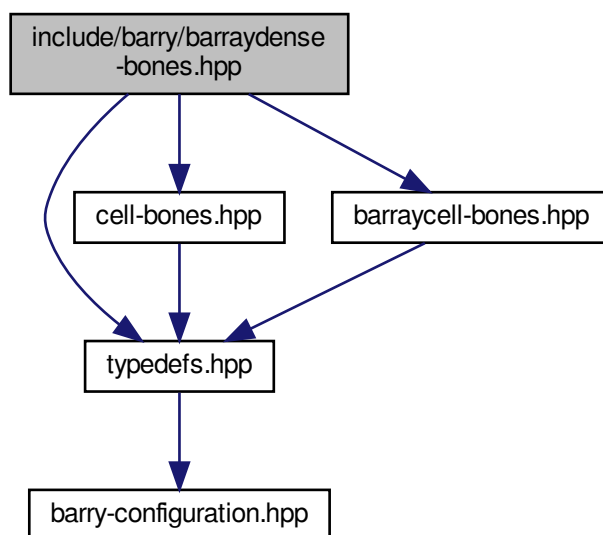
This graph shows which files directly or indirectly include this file:



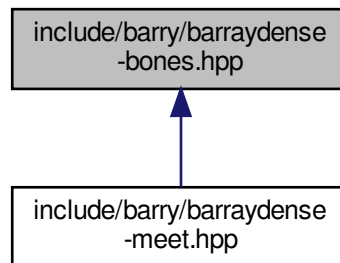
8.7 include/barry/barraydense-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "cell-bones.hpp"
#include "barraycell-bones.hpp"
```

Include dependency graph for barraydense-bones.hpp:



This graph shows which files directly or indirectly include this file:



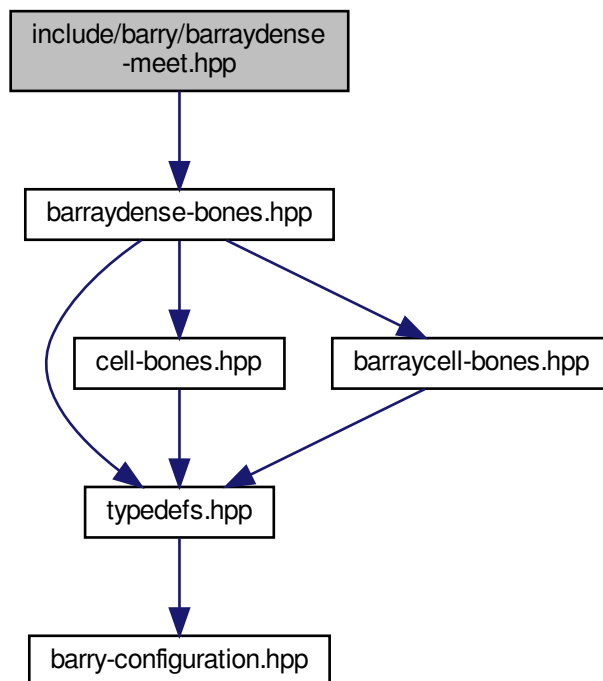
Classes

- class [BArrayDense](#)< [Cell_Type](#), [Data_Type](#) >
Baseline class for binary arrays.

8.8 include/barry/barraydense-meet.hpp File Reference

```
#include "barraydense-bones.hpp"
```

Include dependency graph for `barraydense-meet.hpp`:



Macros

- `#define BARRY_BARRAYDENSE_MEAT_HPP`
- `#define ROW(a) this->el_ij[a]`
- `#define COL(a) this->el_ji[a]`
- `#define POS(a, b) (b)*N + (a)`
- `#define ZERO_CELL Cell< Cell_Type >(static_cast< Cell_Type >(0.0))`

8.8.1 Macro Definition Documentation

8.8.1.1 BARRY_BARRAYDENSE_MEAT_HPP

```
#define BARRY_BARRAYDENSE_MEAT_HPP
```

Definition at line 5 of file `barraydense-meet.hpp`.

8.8.1.2 COL

```
#define COL(  
    a ) this->el_ji[a]
```

Definition at line 8 of file barraydense-meet.hpp.

8.8.1.3 POS

```
#define POS(  
    a,  
    b ) (b)*N + (a)
```

Definition at line 9 of file barraydense-meet.hpp.

8.8.1.4 ROW

```
#define ROW(  
    a ) this->el_ij[a]
```

Definition at line 7 of file barraydense-meet.hpp.

8.8.1.5 ZERO_CELL

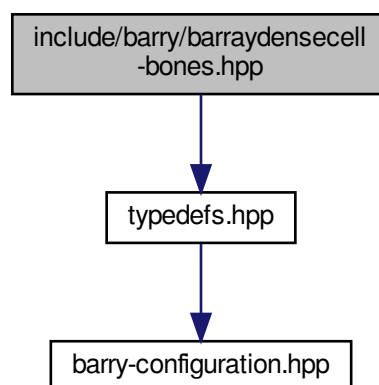
```
#define ZERO_CELL Cell< Cell_Type >(static_cast< Cell_Type >(0.0))
```

Definition at line 14 of file barraydense-meet.hpp.

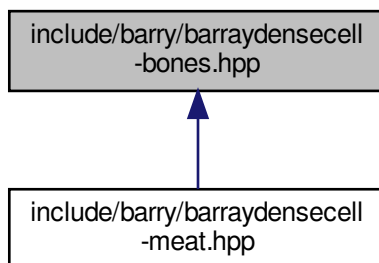
8.9 include/barry/barraydensecell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barraydensecell-bones.hpp:



This graph shows which files directly or indirectly include this file:



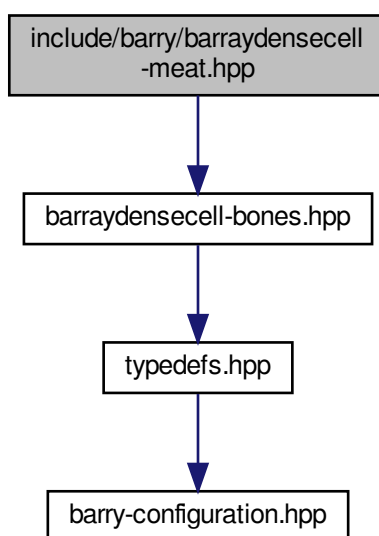
Classes

- class [BArrayDenseCell< Cell_Type, Data_Type >](#)
- class [BArrayDenseCell_const< Cell_Type, Data_Type >](#)

8.10 include/barry/barraydensecell-meat.hpp File Reference

```
#include "barraydensecell-bones.hpp"
```

Include dependency graph for `barraydensecell-meat.hpp`:



Macros

- `#define BARRY_BARRAYDENSECELL_MEAT_HPP 1`
- `#define POS(a, b) (a) + (b) * Array->N`

8.10.1 Macro Definition Documentation

8.10.1.1 BARRY_BARRAYDENSECELL_MEAT_HPP

```
#define BARRY_BARRAYDENSECELL_MEAT_HPP 1
```

Definition at line 4 of file barraydensecell-meat.hpp.

8.10.1.2 POS

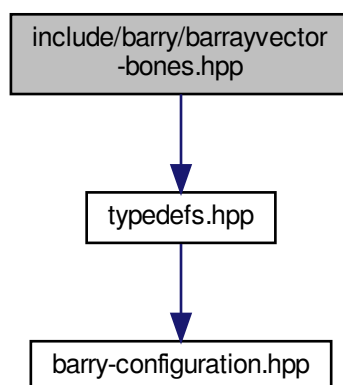
```
#define POS(  
    a,  
    b ) (a) + (b) * Array->N
```

Definition at line 6 of file barraydensecell-meat.hpp.

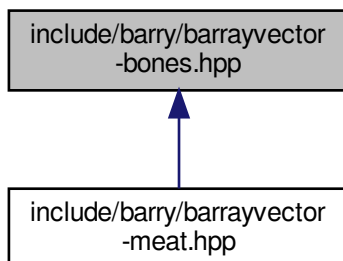
8.11 include/barry/barrayvector-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barrayvector-bones.hpp:



This graph shows which files directly or indirectly include this file:



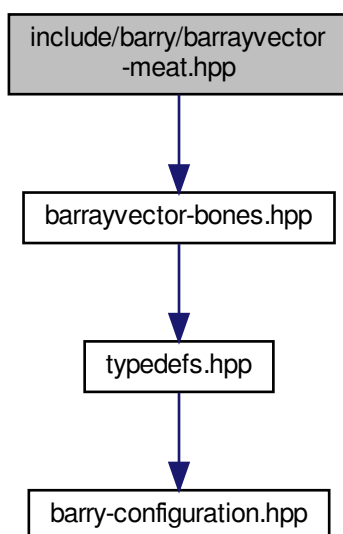
Classes

- class [BArrayVector< Cell_Type, Data_Type >](#)
Row or column of a [BArray](#)
- class [BArrayVector_const< Cell_Type, Data_Type >](#)

8.12 include/barry/barrayvector-meat.hpp File Reference

```
#include "barrayvector-bones.hpp"
```

Include dependency graph for `barrayvector-meat.hpp`:



Macros

- `#define BARRY_BARRAYVECTOR_MEAT_HPP 1`

8.12.1 Macro Definition Documentation

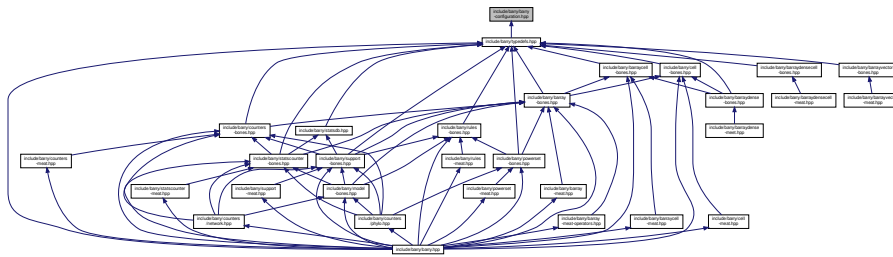
8.12.1.1 BARRY_BARRAYVECTOR_MEAT_HPP

```
#define BARRY_BARRAYVECTOR_MEAT_HPP 1
```

Definition at line 4 of file `barrayvector-meat.hpp`.

8.13 include/barry/barry-configuration.hpp File Reference

This graph shows which files directly or indirectly include this file:



Configuration MACROS

These are mostly related to performance. The definitions follow:

- `BARRY_USE_UNORDERED_MAP` If specified, then barry is compiled using `std::unordered_map`. Otherwise it will use `std::map` for the arrays.
- `BARRY_USE_SAFE_EXP` When specified, it will multiply all likelihoods in `Model` by $(1/100)/(1/100)$ so that numerical overflows are avoided.
- `BARRY_USE_ISFINITE` When specified, it will introduce a macro that checks whether the likelihood is finite or not.
- `printf_barry` If not specified, will be defined as `printf`.
- `#define BARRY_SAFE_EXP -100.0`
- `#define BARRY_ISFINITE(a)`
- `#define BARRY_CHECK_SUPPORT(x, maxs)`
- `#define printf_barry printf`
- `#define BARRY_MAX_NUM_ELEMENTS static_cast< unsigned int >(UINT_MAX/2u)`
- `template<typename Ta, typename Tb >`
`using Map = std::map< Ta, Tb >`

8.13.1 Macro Definition Documentation

8.13.1.1 BARRY_CHECK_SUPPORT

```
#define BARRY_CHECK_SUPPORT(  
    x,  
    maxs )
```

Definition at line 45 of file barry-configuration.hpp.

8.13.1.2 BARRY_ISFINITE

```
#define BARRY_ISFINITE(  
    a )
```

Definition at line 38 of file barry-configuration.hpp.

8.13.1.3 BARRY_MAX_NUM_ELEMENTS

```
#define BARRY_MAX_NUM_ELEMENTS static_cast< unsigned int >(UINT_MAX/2u)
```

Definition at line 53 of file barry-configuration.hpp.

8.13.1.4 BARRY_SAFE_EXP

```
#define BARRY_SAFE_EXP -100.0
```

Definition at line 31 of file barry-configuration.hpp.

8.13.1.5 printf_barry

```
#define printf_barry printf
```

Definition at line 49 of file barry-configuration.hpp.

8.13.2 Typedef Documentation

8.13.2.1 Map

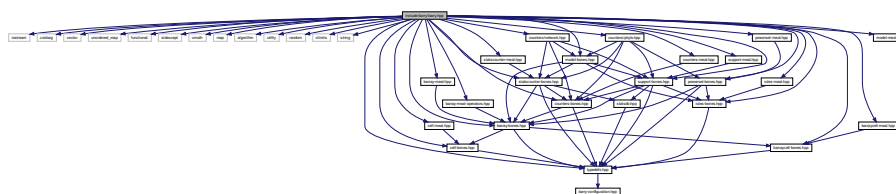
```
template<typename Ta , typename Tb >
using Map = std::map<Ta,Tb>
```

Definition at line 25 of file barry-configuration.hpp.

8.14 include/barry/barry.hpp File Reference

```
#include <iostream>
#include <cstdint>
#include <vector>
#include <unordered_map>
#include <functional>
#include <stdexcept>
#include <cmath>
#include <map>
#include <algorithm>
#include <utility>
#include <random>
#include <climits>
#include <string>
#include "typedefs.hpp"
#include "cell-bones.hpp"
#include "cell-meat.hpp"
#include "barray-bones.hpp"
#include "barraycell-bones.hpp"
#include "barray-meat.hpp"
#include "barraycell-meat.hpp"
#include "barray-meat-operators.hpp"
#include "counters-bones.hpp"
#include "counters-meat.hpp"
#include "statscounter-bones.hpp"
#include "statscounter-meat.hpp"
#include "support-bones.hpp"
#include "support-meat.hpp"
#include "powerset-bones.hpp"
#include "powerset-meat.hpp"
#include "model-bones.hpp"
#include "model-meat.hpp"
#include "rules-bones.hpp"
#include "rules-meat.hpp"
#include "counters/network.hpp"
#include "counters/phylo.hpp"
```

Include dependency graph for barry.hpp:



Namespaces

- [barry](#)
barry: Your go-to motif accountant
- [barry::counters](#)
Tree class and Treeliterator class.
- [barry::counters::network](#)
- [barry::counters::phylo](#)

Macros

- `#define BARRY_HPP`
- `#define BARRY_VERSION 0.1`
- `#define COUNTER_FUNCTION(a)`
- `#define COUNTER_LAMBDA(a)`
- `#define RULE_FUNCTION(a)`
- `#define RULE_LAMBDA(a)`

8.14.1 Macro Definition Documentation

8.14.1.1 BARRY_HPP

```
#define BARRY_HPP
```

Definition at line 20 of file barry.hpp.

8.14.1.2 BARRY_VERSION

```
#define BARRY_VERSION 0.1
```

Definition at line 22 of file barry.hpp.

8.14.1.3 COUNTER_FUNCTION

```
#define COUNTER_FUNCTION(  
    a )
```

Value:

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \  
inline double (a) (const Array_Type & Array, uint i, uint j, Data_Type * data) \  

```

Definition at line 73 of file barry.hpp.

8.14.1.4 COUNTER_LAMBDA

```
#define COUNTER_LAMBDA(  
    a )
```

Value:

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \  
Counter_fun_type<Array_Type, Data_Type> a = \  
[] (const Array_Type & Array, uint i, uint j, Data_Type * data)
```

Definition at line 76 of file barry.hpp.

8.14.1.5 RULE_FUNCTION

```
#define RULE_FUNCTION(  
    a )
```

Value:

```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \  
inline bool (a) (const Array_Type & Array, uint i, uint j, Data_Type * data) \  
{
```

Definition at line 80 of file barry.hpp.

8.14.1.6 RULE_LAMBDA

```
#define RULE_LAMBDA(  
    a )
```

Value:

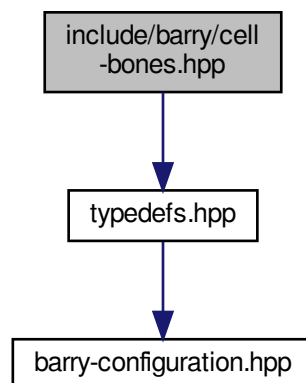
```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \  
Rule_fun_type<Array_Type, Data_Type> a = \  
[] (const Array_Type & Array, uint i, uint j, Data_Type * data)
```

Definition at line 83 of file barry.hpp.

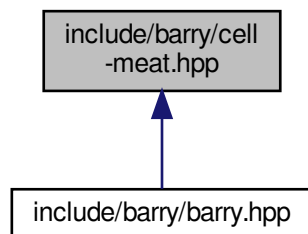
8.15 include/barry/cell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for cell-bones.hpp:



This graph shows which files directly or indirectly include this file:



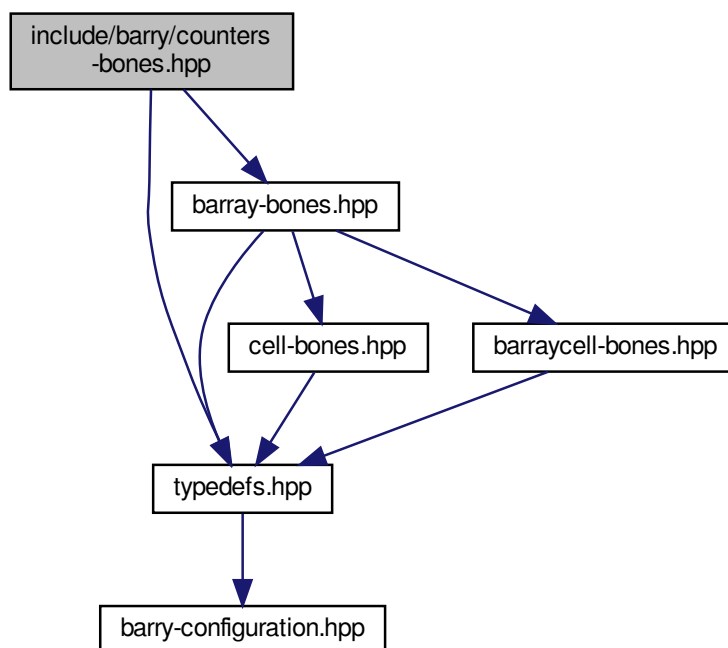
8.17 include/barry/col-bones.hpp File Reference

8.18 include/barry/counters-bones.hpp File Reference

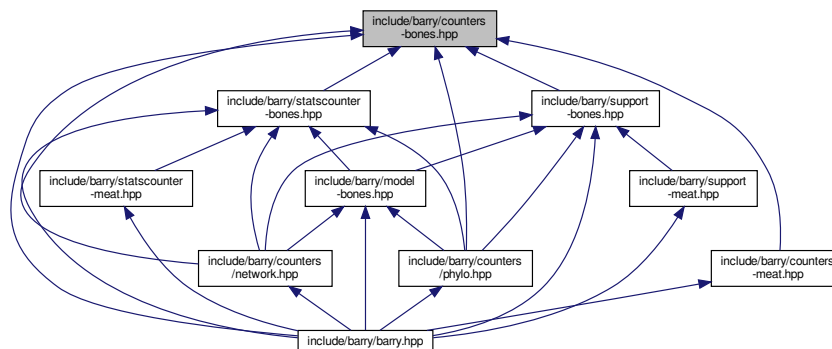
```
#include "typedefs.hpp"
```

```
#include "barray-bones.hpp"
```

Include dependency graph for counters-bones.hpp:



This graph shows which files directly or indirectly include this file:



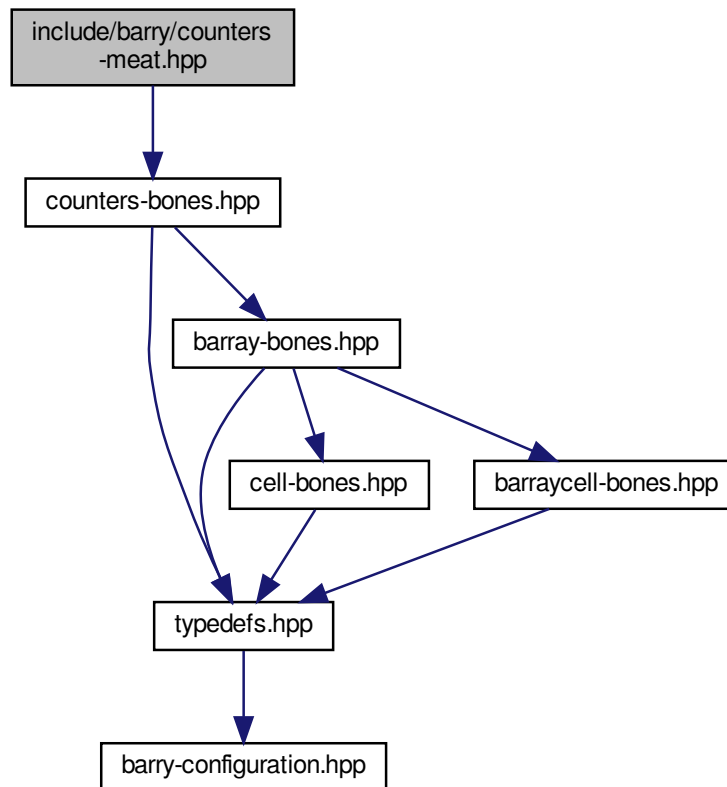
Classes

- class [Counter< Array_Type, Data_Type >](#)
A counter function based on change statistics.
- class [Counters< Array_Type, Data_Type >](#)
Vector of counters.

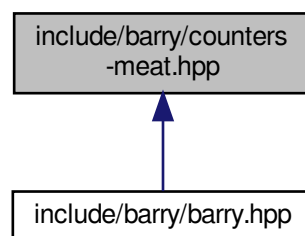
8.19 include/barry/counters-meat.hpp File Reference

```
#include "counters-bones.hpp"
```

Include dependency graph for counters-meat.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- `#define COUNTER_TYPE() Counter<Array_Type,Data_Type>`

- #define COUNTER_TEMPLATE_ARGS() <typename Array_Type, typename Data_Type>
- #define COUNTER_TEMPLATE(a, b) template COUNTER_TEMPLATE_ARGS() inline a COUNTER_TYPE()↔
::b
- #define COUNTERS_TYPE() Counters<Array_Type,Data_Type>
- #define COUNTERS_TEMPLATE_ARGS() <typename Array_Type, typename Data_Type>
- #define COUNTERS_TEMPLATE(a, b) template COUNTERS_TEMPLATE_ARGS() inline a COUNTERS_TYPE()↔
::b

Functions

- COUNTER_TEMPLATE (, Counter)(const Counter< Array_Type
Data_Type init_fun (counter_.init_fun)
Data_Type &&counter_ init_fun (std::move(counter_.init_fun))
Data_Type &&counter_ data (std::move(counter_.data))
Data_Type &&counter_ delete_data (std::move(counter_.delete_data))
Data_Type &&counter_ name (std::move(counter_.name))
Data_Type &&counter_ desc (std::move(counter_.desc))

Move constructor.
- COUNTER_TEMPLATE (COUNTER_TYPE(), operator=)(const Counter< Array_Type
COUNTER_TEMPLATE (COUNTER_TYPE() &, operator=)(Counter< Array_Type
COUNTER_TEMPLATE (double, count)(Array_Type &Array
 < Move assignment
- return count_fun (Array, i, j, data)
- COUNTER_TEMPLATE (double, init)(Array_Type &Array
• return init_fun (Array, i, j, data)
- COUNTER_TEMPLATE (std::string, get_name)() const
- COUNTER_TEMPLATE (std::string, get_description)() const
- COUNTERS_TEMPLATE (, Counters)()
- COUNTERS_TEMPLATE (COUNTER_TYPE() &, operator[])(uint idx)
- Data_Type Data_Type to_be_deleted (new std::vector< uint >(0u))
- Data_Type Data_Type delete_data (true)
- Data_Type Data_Type delete_to_be_deleted (true)
- Data_Type &&counters_ to_be_deleted (std::move(counters_.to_be_deleted))
- Data_Type &&counters_ delete_data (std::move(counters_.delete_data))
- Data_Type &&counters_ delete_to_be_deleted (std::move(counters_.delete_to_be_deleted))
- COUNTERS_TEMPLATE (COUNTERS_TYPE(), operator=)(const Counters< Array_Type
COUNTERS_TEMPLATE (COUNTERS_TYPE() &, operator=)(Counters< Array_Type
COUNTERS_TEMPLATE (void, add_counter)(Counter< Array_Type
data push_back (new Counter< Array_Type, Data_Type >(counter))
data push_back (new Counter< Array_Type, Data_Type >(count_fun_, init_fun_, data_, delete_data_,
name_, desc_))
COUNTERS_TEMPLATE (void, clear)()
- COUNTERS_TEMPLATE (std::vector< std::string >, get_names)() const
- COUNTERS_TEMPLATE (std::vector< std::string >, get_descriptions)() const

Variables

- Data_Type & [counter_](#)
- Data_Type &&[counter_ noexcept](#)
- [uint i](#)
- [uint uint j](#)
- Data_Type & [counter](#)
- [return](#)
- Data_Type [count_fun_](#)
- Data_Type [Counter_fun_type](#)< Array_Type, Data_Type > [init_fun_](#)
- Data_Type [Counter_fun_type](#)< Array_Type, Data_Type > Data_Type * [data_](#)
- Data_Type [Counter_fun_type](#)< Array_Type, Data_Type > Data_Type bool [delete_data_](#)
- Data_Type [Counter_fun_type](#)< Array_Type, Data_Type > Data_Type bool std::string [name_](#)
- Data_Type [Counter_fun_type](#)< Array_Type, Data_Type > Data_Type bool std::string std::string [desc_](#)

8.19.1 Macro Definition Documentation

8.19.1.1 COUNTER_TEMPLATE

```
#define COUNTER_TEMPLATE(
    a,
    b )  template COUNTER_TEMPLATE_ARGS() inline a COUNTER_TYPE()::b
```

Definition at line 10 of file counters-meat.hpp.

8.19.1.2 COUNTER_TEMPLATE_ARGS

```
#define COUNTER_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

Definition at line 8 of file counters-meat.hpp.

8.19.1.3 COUNTER_TYPE

```
#define COUNTER_TYPE( ) Counter<Array_Type,Data_Type>
```

Definition at line 6 of file counters-meat.hpp.

8.19.1.4 COUNTERS_TEMPLATE

```
#define COUNTERS_TEMPLATE(  
    a,  
    b )  template COUNTERS_TEMPLATE_ARGS() inline a COUNTERS_TYPE()::b
```

Definition at line 153 of file counters-meat.hpp.

8.19.1.5 COUNTERS_TEMPLATE_ARGS

```
#define COUNTERS_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

Definition at line 151 of file counters-meat.hpp.

8.19.1.6 COUNTERS_TYPE

```
#define COUNTERS_TYPE( ) Counters<Array_Type,Data_Type>
```

Definition at line 149 of file counters-meat.hpp.

8.19.2 Function Documentation

8.19.2.1 count_fun()

```
return count_fun (  
    Array ,  
    i ,  
    j ,  
    data )
```

8.19.2.2 COUNTER_TEMPLATE() [1/7]

```
COUNTER_TEMPLATE (  
    Counter ) const
```

8.19.2.3 COUNTER_TEMPLATE() [2/7]

```
COUNTER_TEMPLATE (
    COUNTER_TYPE() & ,
    operator )
```

8.19.2.4 COUNTER_TEMPLATE() [3/7]

```
COUNTER_TEMPLATE (
    COUNTER_TYPE() ,
    operator ) const
```

8.19.2.5 COUNTER_TEMPLATE() [4/7]

```
COUNTER_TEMPLATE (
    double ,
    count ) &
```

< Move assignment

8.19.2.6 COUNTER_TEMPLATE() [5/7]

```
COUNTER_TEMPLATE (
    double ,
    init ) &
```

8.19.2.7 COUNTER_TEMPLATE() [6/7]

```
COUNTER_TEMPLATE (
    std::string ,
    get_description ) const
```

Definition at line 141 of file counters-meat.hpp.

8.19.2.8 COUNTER_TEMPLATE() [7/7]

```
COUNTER_TEMPLATE (
    std::string ,
    get_name ) const
```

Definition at line 137 of file counters-meat.hpp.

8.19.2.9 COUNTERS_TEMPLATE() [1/8]

```
COUNTERS_TEMPLATE (
    Counters )
```

Definition at line 156 of file counters-meat.hpp.

8.19.2.10 COUNTERS_TEMPLATE() [2/8]

```
COUNTERS_TEMPLATE (
    COUNTER_TYPE() & ,
    operator [ ] )
```

Definition at line 163 of file counters-meat.hpp.

8.19.2.11 COUNTERS_TEMPLATE() [3/8]

```
COUNTERS_TEMPLATE (
    COUNTERS_TYPE() & ,
    operator )
```

8.19.2.12 COUNTERS_TEMPLATE() [4/8]

```
COUNTERS_TEMPLATE (
    COUNTERS_TYPE() ,
    operator ) const
```

8.19.2.13 COUNTERS_TEMPLATE() [5/8]

```
COUNTERS_TEMPLATE (
    std::vector< std::string > ,
    get_descriptions ) const
```

Definition at line 348 of file counters-meat.hpp.

8.19.2.14 COUNTERS_TEMPLATE() [6/8]

```
COUNTERS_TEMPLATE (
    std::vector< std::string > ,
    get_names ) const
```

Definition at line 337 of file counters-meat.hpp.

8.19.2.15 COUNTERS_TEMPLATE() [7/8]

```
COUNTERS_TEMPLATE (
    void ,
    add_counter )
```

8.19.2.16 COUNTERS_TEMPLATE() [8/8]

```
COUNTERS_TEMPLATE (
    void ,
    clear )
```

Definition at line 318 of file counters-meat.hpp.

8.19.2.17 data()

```
Data_Type&& counter_ data (
    std::move(counter_.data) )
```

8.19.2.18 delete_data() [1/3]

```
Data_Type&& counter_ delete_data (
    std::move(counter_.delete_data) )
```

8.19.2.19 delete_data() [2/3]

```
Data_Type&& counters_ delete_data (
    std::move(counters_.delete_data) )
```

8.19.2.20 delete_data() [3/3]

```
Data_Type Data_Type delete_data (  
    true )
```

8.19.2.21 delete_to_be_deleted() [1/2]

```
Data_Type&& counters_ delete_to_be_deleted (  
    std::move(counters_.delete_to_be_deleted) )
```

Definition at line 201 of file counters-meat.hpp.

8.19.2.22 delete_to_be_deleted() [2/2]

```
Data_Type Data_Type delete_to_be_deleted (  
    true )
```

Definition at line 173 of file counters-meat.hpp.

8.19.2.23 desc()

```
Data_Type&& counter_ desc (  
    std::move(counter_.desc) )
```

Move constructor.

Definition at line 46 of file counters-meat.hpp.

8.19.2.24 init_fun() [1/3]

```
return init_fun (  
    Array ,  
    i ,  
    j ,  
    data )
```

8.19.2.25 init_fun() [2/3]

```
Data_Type init_fun (  
    counter_.  init_fun )
```

Definition at line 15 of file counters-meat.hpp.

8.19.2.26 init_fun() [3/3]

```
Data_Type&& counter_ init_fun (  
    std::move(counter_.init_fun) )
```

8.19.2.27 name()

```
Data_Type&& counter_ name (  
    std::move(counter_.name) )
```

8.19.2.28 push_back() [1/2]

```
data push_back (  
    new Counter< Array_Type, Data_Type > count_fun_, init_fun_, data_, delete_data_,  
    name_, desc_ )
```

8.19.2.29 push_back() [2/2]

```
data push_back (  
    new Counter< Array_Type, Data_Type > counter )
```

8.19.2.30 to_be_deleted() [1/2]

```
Data_Type Data_Type to_be_deleted (  
    new std::vector< uint > 0u )
```

8.19.2.31 to_be_deleted() [2/2]

```
Data_Type&& counters_ to_be_deleted (  
    std::move(counters_.to_be_deleted) )
```

8.19.3 Variable Documentation

8.19.3.1 count_fun_

Data_Type count_fun_

Definition at line 291 of file counters-meat.hpp.

8.19.3.2 counter

Data_Type * counter

Initial value:

```
{  
    to_be_deleted->push_back(data->size())
```

Definition at line 273 of file counters-meat.hpp.

8.19.3.3 counter_

Data_Type & counter_

Initial value:

```
{  
    if (this != &counter_) {  
        this->count_fun = counter_.count_fun;  
        this->init_fun = counter_.init_fun;  
        if (counter_.delete_data)  
        {  
            this->data = new Data_Type(*counter_.data);  
            this->delete_data = true;  
        }  
        else {  
            this->data = counter_.data;  
            this->delete_data = false;  
        }  
        this->name = counter_.name;  
        this->desc = counter_.desc;  
    }  
    return *this
```

Definition at line 14 of file counters-meat.hpp.

8.19.3.4 data_

Data_Type Counter_fun_type<Array_Type,Data_Type> Data_Type* data_

Definition at line 293 of file counters-meat.hpp.

8.19.3.5 delete_data_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> Data_Type bool delete_data_
```

Definition at line 294 of file counters-meat.hpp.

8.19.3.6 desc_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> Data_Type bool std::string std::string desc_↵  
-
```

Initial value:

```
{  
  
    to_be_deleted->push_back(data->size())
```

Definition at line 296 of file counters-meat.hpp.

8.19.3.7 i

```
uint i
```

Definition at line 117 of file counters-meat.hpp.

8.19.3.8 init_fun_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> init_fun_
```

Definition at line 292 of file counters-meat.hpp.

8.19.3.9 j

```
uint j
```

Initial value:

```
{  
    if (count_fun == nullptr)  
        return 0.0
```

Definition at line 117 of file counters-meat.hpp.

8.19.3.10 name_

Data_Type Counter_fun_type<Array_Type,Data_Type> Data_Type bool std::string name_

Definition at line 295 of file counters-meat.hpp.

8.19.3.11 noexcept

Data_Type &&counters_ noexcept

Initial value:

```
{
    if (this != &counter_)
    {
        if (delete_data)
            delete data;

        this->data = std::move(counter_.data);
        this->delete_data = std::move(counter_.delete_data);
        counter_.data = nullptr;
        counter_.delete_data = false;

        this->count_fun = std::move(counter_.count_fun);
        this->init_fun = std::move(counter_.init_fun);

        this->name = std::move(counter_.name);
        this->desc = std::move(counter_.desc);
    }
    return *this
}
```

Definition at line 40 of file counters-meat.hpp.

8.19.3.12 return

return

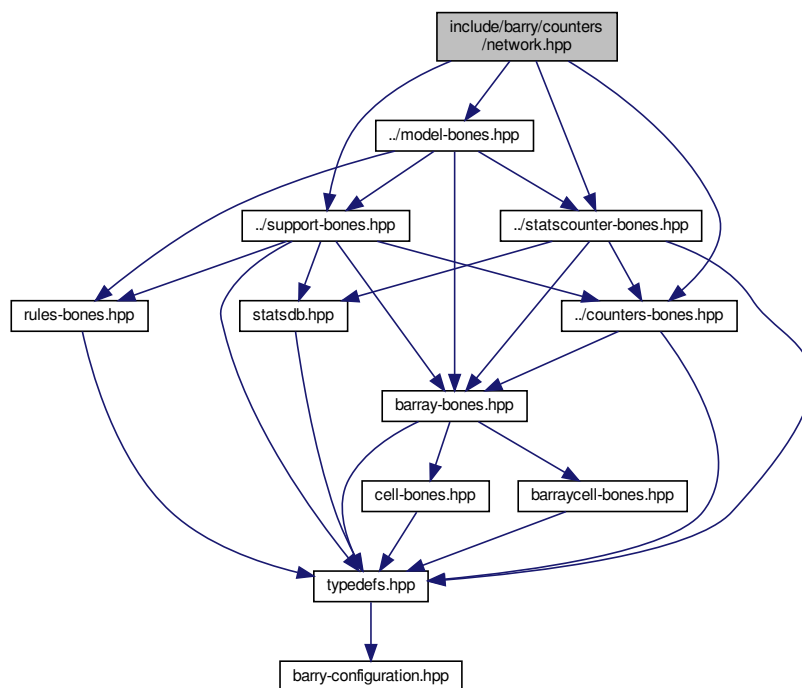
Definition at line 279 of file counters-meat.hpp.

8.20 include/barry/counters/network.hpp File Reference

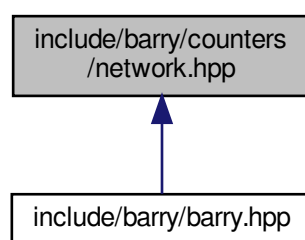
```
#include "../counters-bones.hpp"
#include "../support-bones.hpp"
#include "../statscounter-bones.hpp"
```

```
#include "../model-bones.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 CSS_SIZE()`
 - `#define CSS_CASE_TRUTH() if ((i < n) && (j < n))`
 - `#define CSS_CASE_PERCEIVED() else if (((i >= s) && (i < e)) & ((j >= s) && (j < e)))`
 - `#define CSS_CASE_ELSE()`
 - `#define CSS_CHECK_SIZE_INIT()`
 - `#define CSS_CHECK_SIZE()`
 - `#define CSS_APPEND(name)`
-
- `#define NET_C_DATA_IDX(i) (data->indices[i])`
 - `#define NET_C_DATA_NUM(i) (data->numbers[i])`

Macros for defining counters

- `#define NETWORK_COUNTER(a)`
- `#define NETWORK_COUNTER_LAMBDA(a)`

Macros for defining rules

- `#define NETWORK_RULE(a)`
- `#define NETWORK_RULE_LAMBDA(a)`

Typedefs

Convenient typedefs for network objects.

- `typedef BArray< double, NetworkData > Network`
- `typedef Counter< Network, NetCounterData > NetCounter`
- `typedef Counters< Network, NetCounterData > NetCounters`
- `typedef Support< Network, NetCounterData > NetSupport`
- `typedef StatsCounter< Network, NetCounterData > NetStatsCounter`
- `typedef Model< Network, NetCounterData > NetModel`
- `typedef Rule< Network, bool > NetRule`
- `typedef Rules< Network, bool > NetRules`

Functions

- `void counter_edges (NetCounters *counters)`
Number of edges.
- `void counter_isolates (NetCounters *counters)`
Number of isolated vertices.
- `void counter_mutual (NetCounters *counters)`
Number of mutual ties.
- `void counter_istar2 (NetCounters *counters)`
- `void counter_ostar2 (NetCounters *counters)`
- `void counter_ttriads (NetCounters *counters)`
- `void counter_ctriads (NetCounters *counters)`
- `void counter_density (NetCounters *counters)`
- `void counter_idegree15 (NetCounters *counters)`

- void `counter_odegree15` (`NetCounters *counters`)
- void `counter_absdiff` (`NetCounters *counters`, `uint attr_id`, `double alpha=1.0`)
Sum of absolute attribute difference between ego and alter.
- void `counter_diff` (`NetCounters *counters`, `uint attr_id`, `double alpha=1.0`, `double tail_head=true`)
Sum of attribute difference between ego and alter to pow(alpha)
- `NETWORK_COUNTER` (`init_single_attr`)
- void `counter_nodeicov` (`NetCounters *counters`, `uint attr_id`)
- void `counter_nodeocov` (`NetCounters *counters`, `uint attr_id`)
- void `counter_nodecov` (`NetCounters *counters`, `uint attr_id`)
- void `counter_nodematch` (`NetCounters *counters`, `uint attr_id`)
- void `counter_iddegree` (`NetCounters *counters`, `std::vector< uint > d`)
Counts number of vertices with a given in-degree.
- void `counter_odegree` (`NetCounters *counters`, `std::vector< uint > d`)
Counts number of vertices with a given out-degree.
- void `counter_degree` (`NetCounters *counters`, `std::vector< uint > d`)
Counts number of vertices with a given out-degree.
- void `counter_css_partially_false_recip_commi` (`NetCounters *counters`, `uint netsize`, `const std::vector< uint > &end_`)
Counts errors of commission.
- void `counter_css_partially_false_recip_omiss` (`NetCounters *counters`, `uint netsize`, `const std::vector< uint > &end_`)
Counts errors of omission.
- void `counter_css_completely_false_recip_comiss` (`NetCounters *counters`, `uint netsize`, `const std::vector< uint > &end_`)
Counts completely false reciprocity (comission)
- void `counter_css_completely_false_recip_omiss` (`NetCounters *counters`, `uint netsize`, `const std::vector< uint > &end_`)
Counts completely false reciprocity (omission)

Rules for network models

Parameters

rules	A pointer to a <code>NetRules</code> object (<code>Rules<Network, bool></code>).
-------	--

- void `rules_zerodiag` (`NetRules *rules`)
Number of edges.

8.20.1 Macro Definition Documentation

8.20.1.1 NET_C_DATA_IDX

```
#define NET_C_DATA_IDX(  
    i ) (data->indices[i])
```

Definition at line 79 of file network.hpp.

8.20.1.2 NET_C_DATA_NUM

```
#define NET_C_DATA_NUM(  
    i ) (data->numbers[i])
```

Definition at line 80 of file network.hpp.

8.20.1.3 NETWORK_COUNTER

```
#define NETWORK_COUNTER(  
    a )
```

Value:

```
inline double (a) \  
(const Network & Array, uint i, uint j, NetCounterData * data)
```

Function for definition of a network counter function

Definition at line 101 of file network.hpp.

8.20.1.4 NETWORK_COUNTER_LAMBDA

```
#define NETWORK_COUNTER_LAMBDA(  
    a )
```

Value:

```
Counter_fun_type<Network, NetCounterData> a = \  
[] (const Network & Array, uint i, uint j, NetCounterData * data)
```

Lambda function for definition of a network counter function

Definition at line 104 of file network.hpp.

8.20.1.5 NETWORK_RULE

```
#define NETWORK_RULE(  
    a )
```

Value:

```
inline bool (a) \  
(const Network & Array, uint i, uint j, bool * data)
```

Function for definition of a network counter function

Definition at line 113 of file network.hpp.

8.20.1.6 NETWORK_RULE_LAMBDA

```
#define NETWORK_RULE_LAMBDA(  
    a )
```

Value:

```
Rule_fun_type<Network, bool> a = \  
[] (const Network & Array, uint i, uint j, bool * data)
```

Lambda function for definition of a network counter function

Definition at line 116 of file network.hpp.

8.20.2 Typedef Documentation

8.20.2.1 NetCounter

```
typedef Counter<Network, NetCounterData > NetCounter
```

Definition at line 88 of file network.hpp.

8.20.2.2 NetCounters

```
typedef Counters< Network, NetCounterData> NetCounters
```

Definition at line 89 of file network.hpp.

8.20.2.3 NetModel

```
typedef Model<Network, NetCounterData> NetModel
```

Definition at line 92 of file network.hpp.

8.20.2.4 NetRule

```
typedef Rule<Network, bool> NetRule
```

Definition at line 93 of file network.hpp.

8.20.2.5 NetRules

```
typedef Rules<Network, bool> NetRules
```

Definition at line 94 of file network.hpp.

8.20.2.6 NetStatsCounter

```
typedef StatsCounter<Network, NetCounterData> NetStatsCounter
```

Definition at line 91 of file network.hpp.

8.20.2.7 NetSupport

```
typedef Support<Network, NetCounterData > NetSupport
```

Definition at line 90 of file network.hpp.

8.20.2.8 Network

```
typedef BArray<double, NetworkData> Network
```

Definition at line 87 of file network.hpp.

8.20.3 Function Documentation

8.20.3.1 rules_zerodiag()

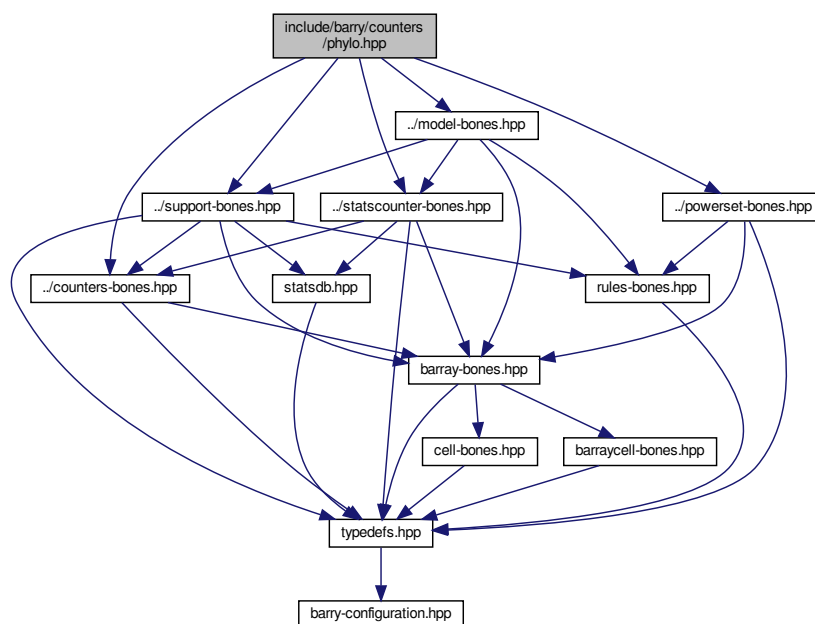
```
void rules_zerodiag (  
    NetRules * rules ) [inline]
```

Number of edges.

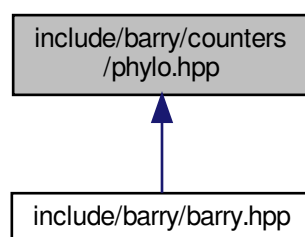
Definition at line 1085 of file network.hpp.

8.21 include/barry/counters/phylo.hpp File Reference

```
#include "../counters-bones.hpp"
#include "../support-bones.hpp"
#include "../statscounter-bones.hpp"
#include "../model-bones.hpp"
#include "../powerset-bones.hpp"
Include dependency graph for phylo.hpp:
```



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 PHYLO_COUNTER_LAMBDA(a)`
Extension of a simple counter.
- `#define PHYLO_RULE_DYN_LAMBDA(a)`
- `#define PHYLO_CHECK_MISSING()`

Typedefs

- `typedef std::vector< std::pair< uint, uint > > PhyloRuleData`

Convenient typedefs for Node objects.

- `typedef BArray< uint, NodeData > PhyloArray`
- `typedef Counter< PhyloArray, PhyloCounterData > PhyloCounter`
- `typedef Counters< PhyloArray, PhyloCounterData > PhyloCounters`
- `typedef Rule< PhyloArray, PhyloRuleData > PhyloRule`
- `typedef Rules< PhyloArray, PhyloRuleData > PhyloRules`
- `typedef Rule< PhyloArray, PhyloRuleDynData > PhyloRuleDyn`
- `typedef Rules< PhyloArray, PhyloRuleDynData > PhyloRulesDyn`
- `typedef Support< PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport`
- `typedef StatsCounter< PhyloArray, PhyloCounterData > PhyloStatsCounter`
- `typedef Model< PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel`
- `typedef PowerSet< PhyloArray, PhyloRuleData > PhyloPowerSet`

Functions

- `std::string get_last_name (bool d)`
- `void counter_overall_gains (PhyloCounters *counters, bool duplication=true)`
Overall functional gains.
- `void counter_gains (PhyloCounters *counters, std::vector< uint > nfun, bool duplication=true)`
Functional gains for a specific function (nfun).
- `void counter_gains_k_offspring (PhyloCounters *counters, std::vector< uint > nfun, uint k=1u, bool duplication=true)`
k genes gain function nfun
- `void counter_genes_changing (PhyloCounters *counters, bool duplication=true)`
Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)
- `void counter_prop_genes_changing (PhyloCounters *counters, bool duplication=true)`
Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)
- `void counter_overall_loss (PhyloCounters *counters, bool duplication=true)`
Overall functional loss.
- `void counter_maxfuns (PhyloCounters *counters, uint lb, uint ub, bool duplication=true)`
Cap the number of functions per gene.
- `void counter_loss (PhyloCounters *counters, std::vector< uint > nfun, bool duplication=true)`
Total count of losses for an specific function.
- `void counter_overall_changes (PhyloCounters *counters, bool duplication=true)`
Total number of changes. Use this statistic to account for "preservation".
- `void counter_subfun (PhyloCounters *counters, uint nfunA, uint nfunB, bool duplication=true)`
Total count of Sub-functionalization events.

- void `counter_cogain` (`PhyloCounters` *counters, `uint` nfunA, `uint` nfunB, `bool` duplication=true)
Co-evolution (joint gain or loss)
- void `counter_longest` (`PhyloCounters` *counters)
Longest branch mutates (either by gain or by loss)
- void `counter_neofun` (`PhyloCounters` *counters, `uint` nfunA, `uint` nfunB, `bool` duplication=true)
Total number of neofunctionalization events.
- void `counter_neofun_a2b` (`PhyloCounters` *counters, `uint` nfunA, `uint` nfunB, `bool` duplication=true)
Total number of neofunctionalization events.
- void `counter_co_opt` (`PhyloCounters` *counters, `uint` nfunA, `uint` nfunB, `bool` duplication=true)
Function co-opting.
- void `rule_dyn_limit_changes` (`PhyloSupport` *support, `uint` pos, `uint` lb, `uint` ub, `bool` duplication=true)
Overall functional gains.

8.21.1 Macro Definition Documentation

8.21.1.1 PHYLO_CHECK_MISSING

```
#define PHYLO_CHECK_MISSING( )
```

Value:

```
if (Array.D() == nullptr) \
    throw std::logic_error("The array data is nullptr."); \
if (data == nullptr) \
    throw std::logic_error("The counter/rule data is nullptr.")
```

Definition at line 121 of file phylo.hpp.

8.21.1.2 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 115 of file phylo.hpp.

8.21.1.3 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 118 of file phylo.hpp.

8.21.2 Typedef Documentation

8.21.2.1 PhyloArray

```
typedef BArray<uint, NodeData> PhyloArray
```

Definition at line 88 of file phylo.hpp.

8.21.2.2 PhyloCounter

```
typedef Counter<PhyloArray, PhyloCounterData > PhyloCounter
```

Definition at line 89 of file phylo.hpp.

8.21.2.3 PhyloCounters

```
typedef Counters< PhyloArray, PhyloCounterData> PhyloCounters
```

Definition at line 90 of file phylo.hpp.

8.21.2.4 PhyloModel

```
typedef Model<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel
```

Definition at line 100 of file phylo.hpp.

8.21.2.5 PhyloPowerSet

```
typedef PowerSet<PhyloArray, PhyloRuleData> PhyloPowerSet
```

Definition at line 101 of file phylo.hpp.

8.21.2.6 PhyloRule

```
typedef Rule<PhyloArray,PhyloRuleData> PhyloRule
```

Definition at line 92 of file phylo.hpp.

8.21.2.7 PhyloRuleData

```
typedef std::vector< std::pair< uint, uint > > PhyloRuleData
```

Definition at line 81 of file phylo.hpp.

8.21.2.8 PhyloRuleDyn

```
typedef Rule<PhyloArray,PhyloRuleDynData> PhyloRuleDyn
```

Definition at line 95 of file phylo.hpp.

8.21.2.9 PhyloRules

```
typedef Rules<PhyloArray,PhyloRuleData> PhyloRules
```

Definition at line 93 of file phylo.hpp.

8.21.2.10 PhyloRulesDyn

```
typedef Rules<PhyloArray,PhyloRuleDynData> PhyloRulesDyn
```

Definition at line 96 of file phylo.hpp.

8.21.2.11 PhyloStatsCounter

```
typedef StatsCounter<PhyloArray, PhyloCounterData> PhyloStatsCounter
```

Definition at line 99 of file phylo.hpp.

8.21.2.12 PhyloSupport

```
typedef Support<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport
```

Definition at line 98 of file phylo.hpp.

8.21.3 Function Documentation

8.21.3.1 get_last_name()

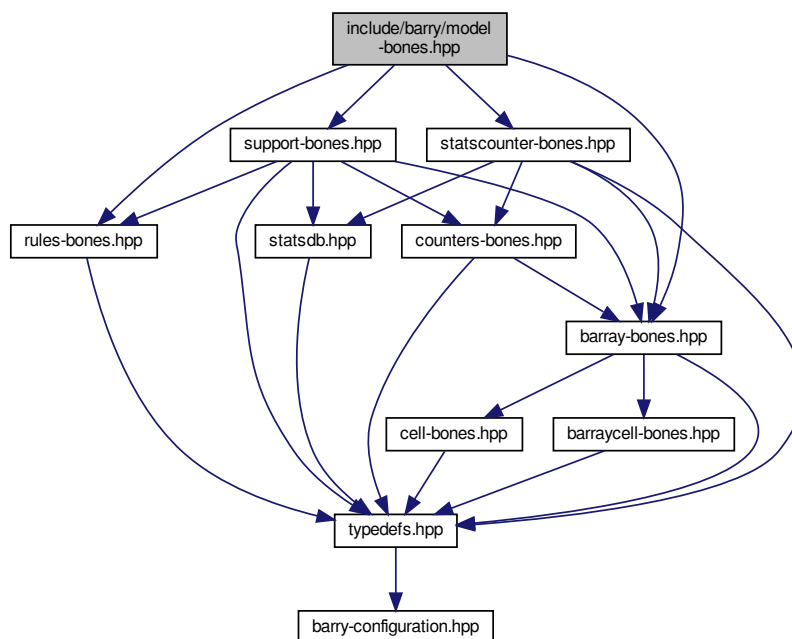
```
std::string get_last_name (
    bool d ) [inline]
```

Definition at line 126 of file phylo.hpp.

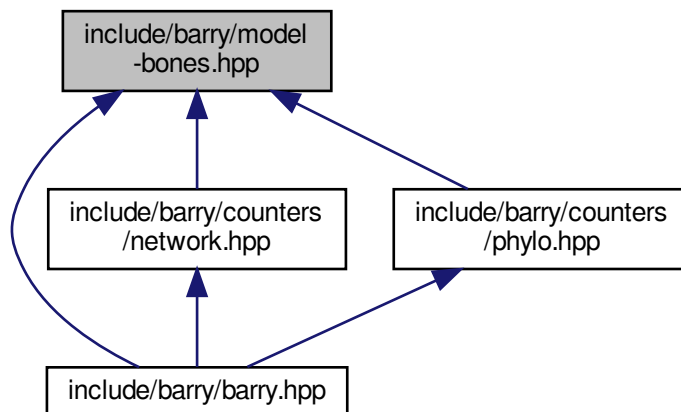
8.22 include/barry/model-bones.hpp File Reference

```
#include "barray-bones.hpp"
#include "support-bones.hpp"
#include "statscounter-bones.hpp"
#include "rules-bones.hpp"
```

Include dependency graph for model-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >](#)

General framework for discrete exponential models. This class allows generating discrete exponential models in the form of a linear exponential model:

Functions

- template<typename Array_Type >
std::vector< double > [keygen_default](#) (const Array_Type &Array_)
- Array Hasher class (used for computing support)*

8.22.1 Function Documentation

8.22.1.1 keygen_default()

```

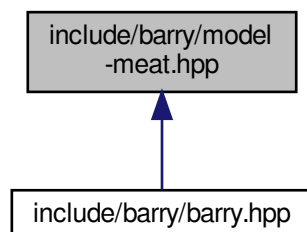
template<typename Array_Type >
std::vector< double > keygen_default (
    const Array_Type & Array_ ) [inline]
  
```

Array Hasher class (used for computing support)

Definition at line 17 of file model-bones.hpp.

8.23 include/barry/model-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



Macros

- `#define MODEL_TYPE()`
- `#define MODEL_TEMPLATE_ARGS()`
- `#define MODEL_TEMPLATE(a, b) template MODEL_TEMPLATE_ARGS() inline a MODEL_TYPE()::b`

Functions

- `double update_normalizing_constant` (const std::vector< double > ¶ms, const Counts_type &support)
- `double likelihood_` (const std::vector< double > &target_stats, const std::vector< double > ¶ms, const double normalizing_constant, bool log_=false)
- `MODEL_TEMPLATE` (, Model)()
- `MODEL_TEMPLATE` (, Model)(const MODEL_TYPE() &Model_)

8.23.1 Macro Definition Documentation

8.23.1.1 MODEL_TEMPLATE

```

#define MODEL_TEMPLATE(
    a,
    b ) template MODEL_TEMPLATE_ARGS() inline a MODEL_TYPE()::b
  
```

Definition at line 75 of file model-meat.hpp.

8.23.1.2 MODEL_TEMPLATE_ARGS

```
#define MODEL_TEMPLATE_ARGS( )
```

Value:

```
<typename Array_Type, typename Data_Counter_Type,\  
  typename Data_Rule_Type, typename Data_Rule_Dyn_Type>
```

Definition at line 72 of file model-meat.hpp.

8.23.1.3 MODEL_TYPE

```
#define MODEL_TYPE( )
```

Value:

```
Model<Array_Type, Data_Counter_Type, Data_Rule_Type,\  
  Data_Rule_Dyn_Type>
```

Definition at line 69 of file model-meat.hpp.

8.23.2 Function Documentation

8.23.2.1 likelihood_()

```
double likelihood_ (  
    const std::vector< double > & target_stats,  
    const std::vector< double > & params,  
    const double normalizing_constant,  
    bool log_ = false ) [inline]
```

Definition at line 37 of file model-meat.hpp.

8.23.2.2 MODEL_TEMPLATE() [1/2]

```
MODEL_TEMPLATE (  
    Model )
```

Definition at line 79 of file model-meat.hpp.

8.23.2.3 MODEL_TEMPLATE() [2/2]

```
MODEL_TEMPLATE (
    Model ) const &
```

Definition at line 134 of file model-meat.hpp.

8.23.2.4 update_normalizing_constant()

```
double update_normalizing_constant (
    const std::vector< double > & params,
    const Counts_type & support ) [inline]
```

Definition at line 11 of file model-meat.hpp.

8.24 include/barry/models/geese.hpp File Reference

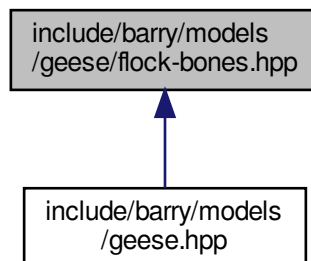
```
#include "geese/geese-node-bones.hpp"
#include "geese/geese-bones.hpp"
#include "geese/geese-meat.hpp"
#include "geese/geese-meat-constructors.hpp"
#include "geese/geese-meat-likelihood.hpp"
#include "geese/geese-meat-likelihood_exhaust.hpp"
#include "geese/geese-meat-simulate.hpp"
#include "geese/geese-meat-predict.hpp"
#include "geese/geese-meat-predict_exhaust.hpp"
#include "geese/geese-meat-predict_sim.hpp"
#include "geese/flock-bones.hpp"
#include "geese/flock-meet.hpp"
```

Include dependency graph for geese.hpp:



8.25 include/barry/models/geese/flock-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:

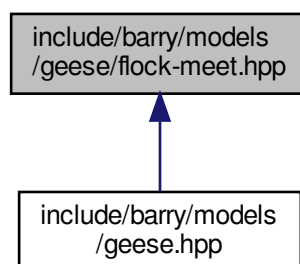


Classes

- class [Flock](#)
A [Flock](#) is a group of [Geese](#).

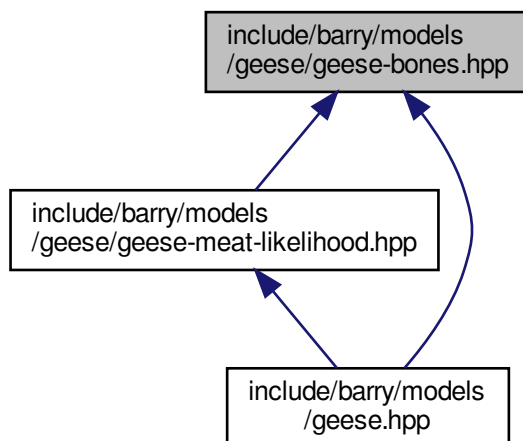
8.26 include/barry/models/geese/flock-meet.hpp File Reference

This graph shows which files directly or indirectly include this file:



8.27 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)`

8.27.1 Macro Definition Documentation

8.27.1.1 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.

8.27.2 Function Documentation

8.27.2.1 keygen_full()

```
std::vector< double > keygen_full ( \  
    const phylocounters::PhyloArray & array ) [inline]
```

Definition at line 35 of file geese-bones.hpp.

8.27.2.2 RULE_FUNCTION()

```
RULE_FUNCTION ( \  
    rule_empty_free )
```

Definition at line 26 of file geese-bones.hpp.

8.27.2.3 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.

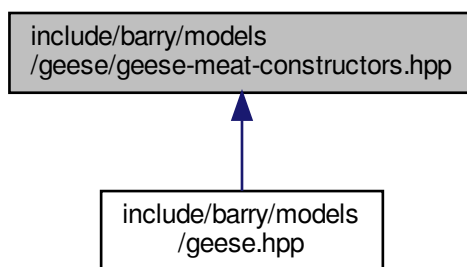
8.27.2.4 vector_caster()

```
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.

8.28 include/barry/models/geese/geese-meat-constructors.hpp File Reference

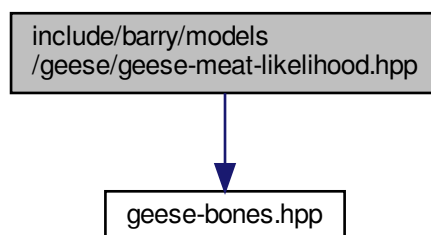
This graph shows which files directly or indirectly include this file:



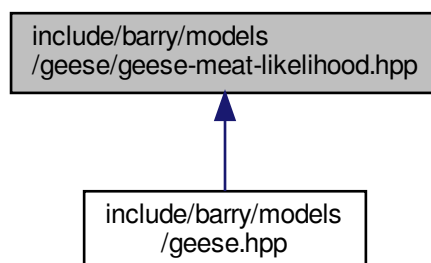
8.29 include/barry/models/geese/geese-meat-likelihood.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for geese-meat-likelihood.hpp:

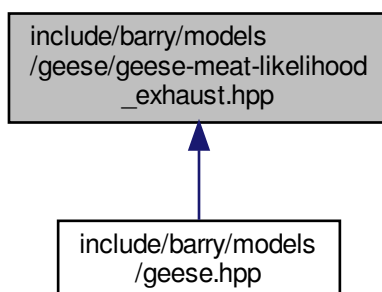


This graph shows which files directly or indirectly include this file:



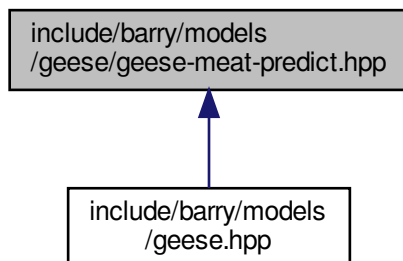
8.30 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference

This graph shows which files directly or indirectly include this file:



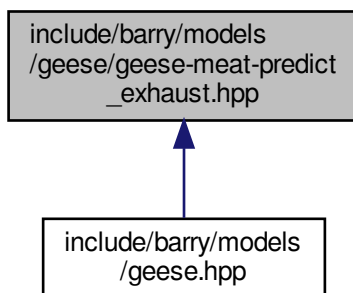
8.31 `include/barry/models/geese/geese-meat-predict.hpp` File Reference

This graph shows which files directly or indirectly include this file:



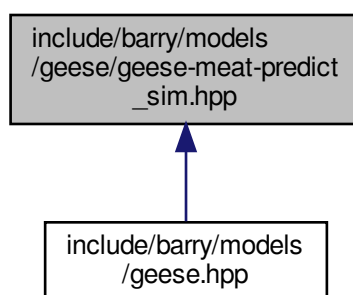
8.32 `include/barry/models/geese/geese-meat-predict_exhaust.hpp` File Reference

This graph shows which files directly or indirectly include this file:



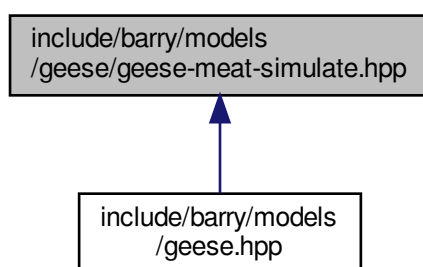
8.33 include/barry/models/geese/geese-meat-predict_sim.hpp File Reference

This graph shows which files directly or indirectly include this file:



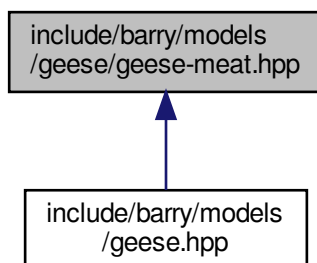
8.34 include/barry/models/geese/geese-meat-simulate.hpp File Reference

This graph shows which files directly or indirectly include this file:



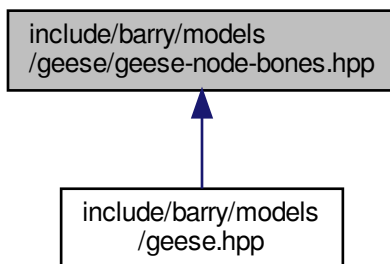
8.35 include/barry/models/geese/geese-meat.hpp File Reference

This graph shows which files directly or indirectly include this file:



8.36 include/barry/models/geese/geese-node-bones.hpp File Reference

This graph shows which files directly or indirectly include this file:



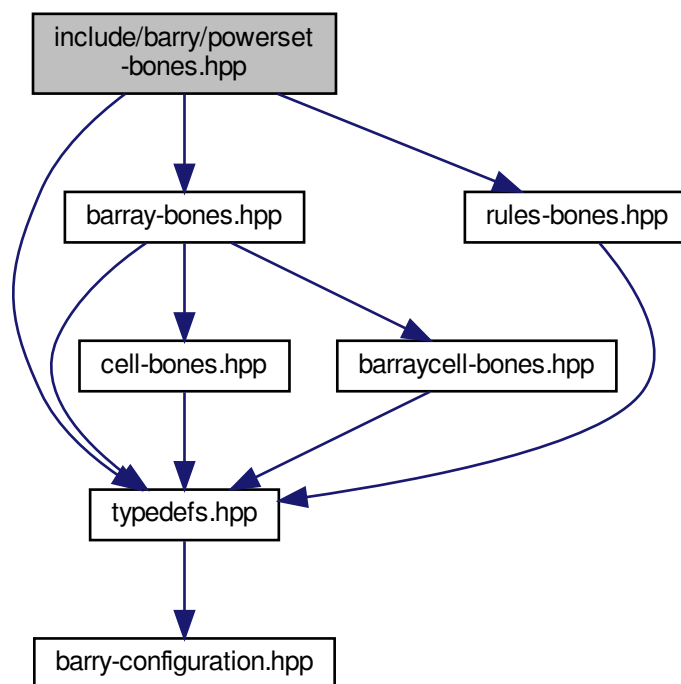
Classes

- class [Node](#)
A single node for the model.

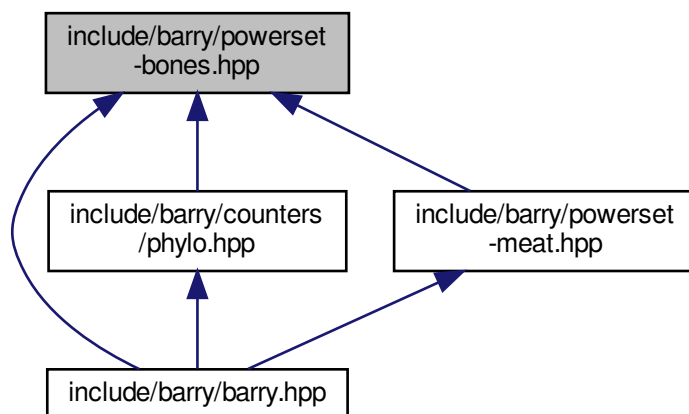
8.37 include/barry/powerset-bones.hpp File Reference

```
#include "typedefs.hpp"  
#include "barray-bones.hpp"  
#include "rules-bones.hpp"
```

Include dependency graph for powerset-bones.hpp:



This graph shows which files directly or indirectly include this file:



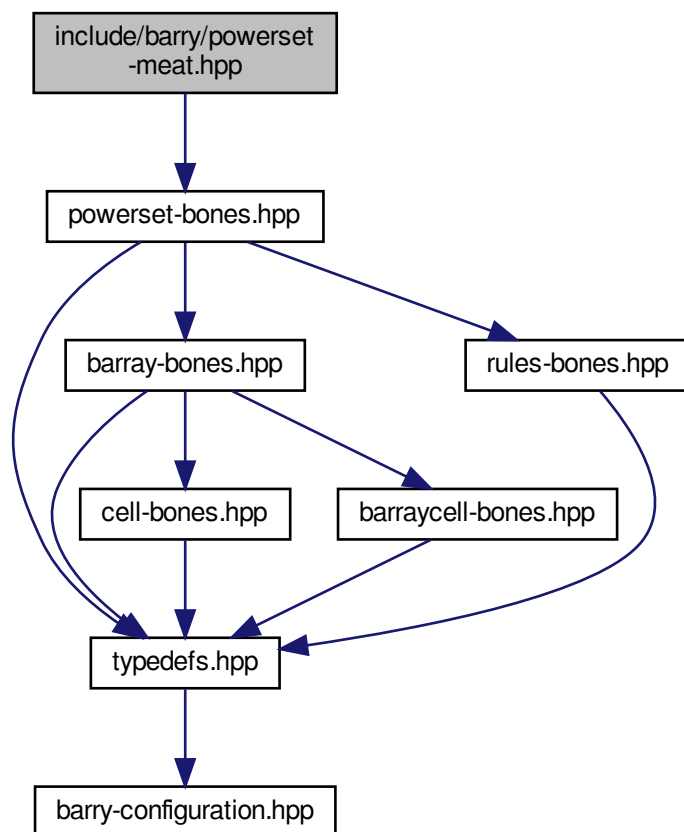
Classes

- class [PowerSet< Array_Type, Data_Rule_Type >](#)
PowerSet of a binary array.

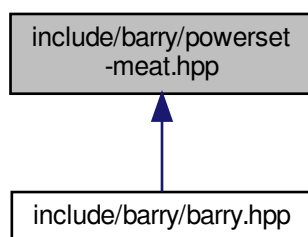
8.38 include/barry/powerset-meat.hpp File Reference

```
#include "powerset-bones.hpp"
```

Include dependency graph for powerset-meat.hpp:



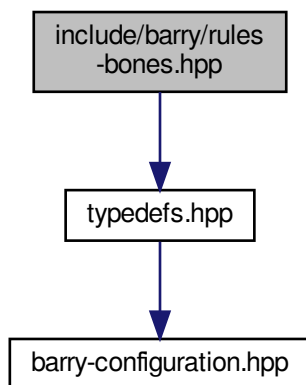
This graph shows which files directly or indirectly include this file:



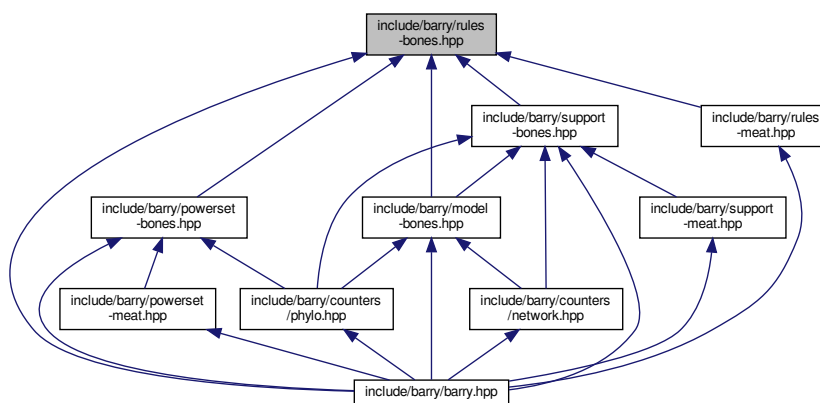
8.39 include/barry/rules-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for rules-bones.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Rule< Array_Type, Data_Type >](#)
Rule for determining if a cell should be included in a sequence.
- class [Rules< Array_Type, Data_Type >](#)
Vector of objects of class *Rule*.

Functions

- `template<typename Array_Type , typename Data_Type >`
`bool rule_fun_default (const Array_Type *array, uint i, uint j, Data_Type *dat)`

8.39.1 Function Documentation

8.39.1.1 rule_fun_default()

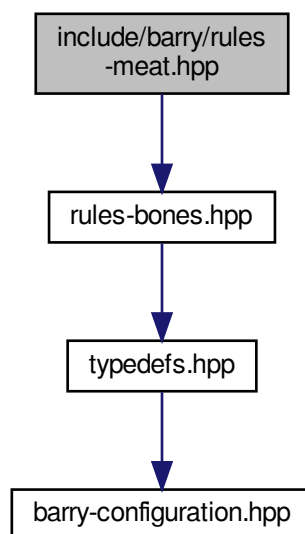
```
template<typename Array_Type , typename Data_Type >
bool rule_fun_default (
    const Array_Type * array,
    uint i,
    uint j,
    Data_Type * dat )
```

Definition at line 10 of file rules-bones.hpp.

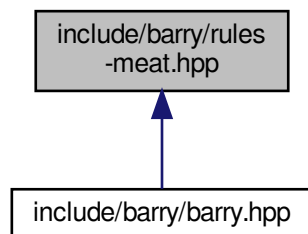
8.40 include/barry/rules-meat.hpp File Reference

```
#include "rules-bones.hpp"
```

Include dependency graph for rules-meat.hpp:

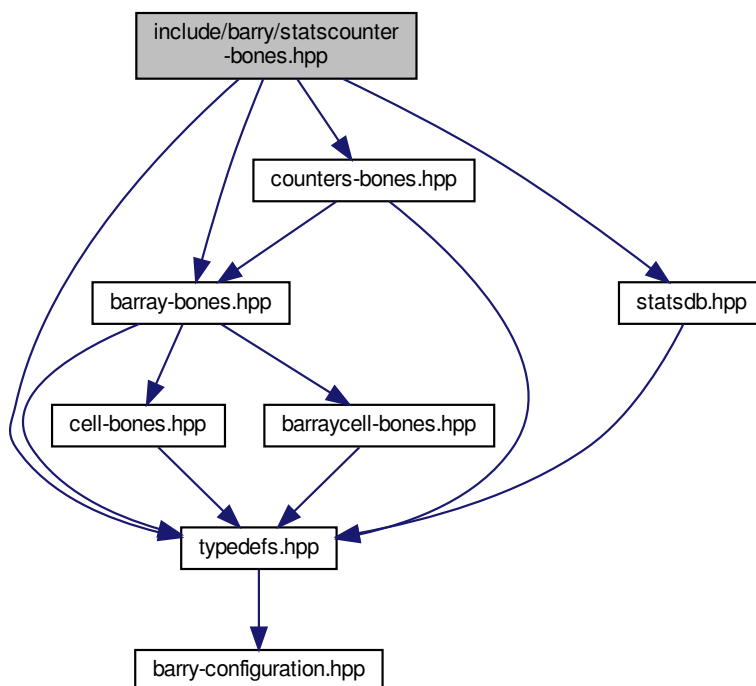


This graph shows which files directly or indirectly include this file:

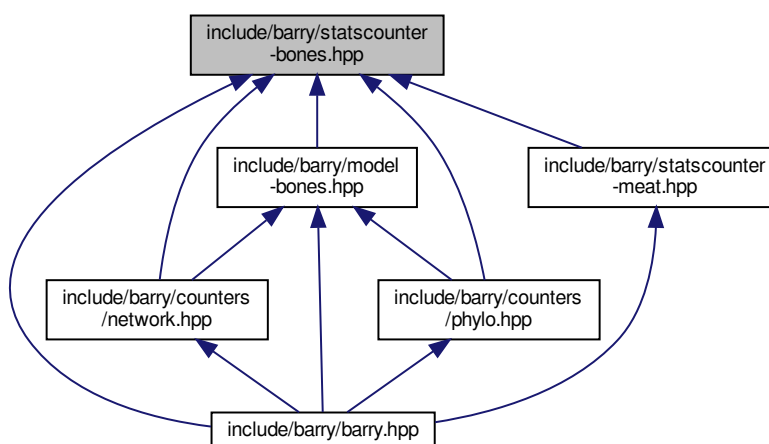


8.41 include/barry/statscounter-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "barray-bones.hpp"
#include "statsdb.hpp"
#include "counters-bones.hpp"
Include dependency graph for statscounter-bones.hpp:
```



This graph shows which files directly or indirectly include this file:



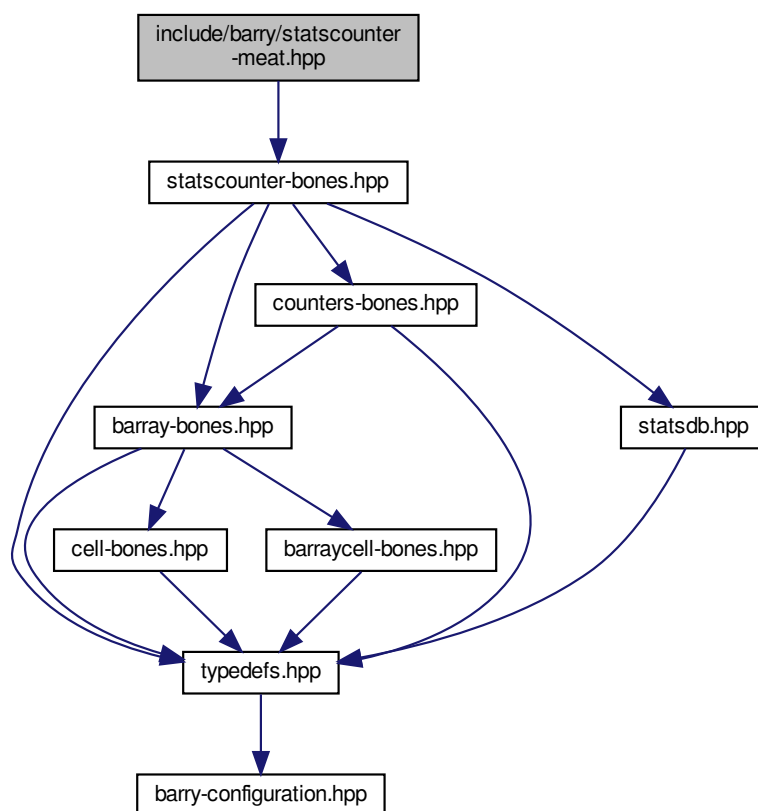
Classes

- class [StatsCounter](#)< [Array_Type](#), [Data_Type](#) >
Count stats for a single Array.

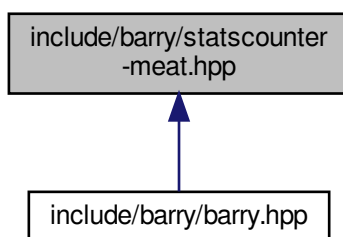
8.42 include/barry/statscounter-meat.hpp File Reference

```
#include "statscounter-bones.hpp"
```

Include dependency graph for statscounter-meat.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- `#define STATSCOUNTER_TYPE() StatsCounter<Array_Type,Data_Type>`
- `#define STATSCOUNTER_TEMPLATE_ARGS() <typename Array_Type, typename Data_Type>`
- `#define STATSCOUNTER_TEMPLATE(a, b) template STATSCOUNTER_TEMPLATE_ARGS() inline a STATSCOUNTER_TYPE()::b`

Functions

- [STATSCOUNTER_TEMPLATE](#) ([~,StatsCounter](#)())
- [STATSCOUNTER_TEMPLATE](#) (void, reset_array)(const Array_Type *Array_)
- [STATSCOUNTER_TEMPLATE](#) (void, add_counter)(Counter< Array_Type
- [STATSCOUNTER_TEMPLATE](#) (void, set_counters)(Counters< Array_Type
- [STATSCOUNTER_TEMPLATE](#) (void, count_init)(uint i
- current_stats [resize](#) (counters->size(), 0.0)
- [for](#) (uint n=0u;n< counters->size();++n) current_stats[n] = change_stats[pos][n]
- [STATSCOUNTER_TEMPLATE](#) (void, count_current)(uint i
- [STATSCOUNTER_TEMPLATE](#) (std::vector< double >, count_all)()
- [STATSCOUNTER_TEMPLATE](#) (std::vector< std::string >, get_names)() const
- [STATSCOUNTER_TEMPLATE](#) (std::vector< std::string >, get_descriptions)() const

Variables

- Data_Type * [f_](#)
- [return](#)
- Data_Type * [counters_](#)
- [counter_deleted](#) = true
- [counters](#) = [counters_](#)
- [uint j](#)

8.42.1 Macro Definition Documentation

8.42.1.1 STATSCOUNTER_TEMPLATE

```
#define STATSCOUNTER_TEMPLATE(
    a,
    b )  template STATSCOUNTER\_TEMPLATE\_ARGS() inline a STATSCOUNTER\_TYPE()::b
```

Definition at line 10 of file statscounter-meat.hpp.

8.42.1.2 STATSCOUNTER_TEMPLATE_ARGS

```
template STATSCOUNTER\_TEMPLATE\_ARGS( ) <typename Array_Type, typename Data_Type>
```

Definition at line 8 of file statscounter-meat.hpp.

8.42.1.3 STATSCOUNTER_TYPE

```
template Data_Type * STATSCOUNTER\_TYPE( ) StatsCounter<Array_Type,Data_Type>
```

Definition at line 6 of file statscounter-meat.hpp.

8.42.2 Function Documentation

8.42.2.1 for()

```
for (
    uint n = 0u; n < counters->size(); ++n ) = change_stats[pos][n]
```

Definition at line 134 of file support-meat.hpp.

8.42.2.2 resize()

```
current_stats resize (
    counters-> size(),
    0. 0 )
```

8.42.2.3 STATSCOUNTER_TEMPLATE() [1/9]

```
STATSCOUNTER_TEMPLATE (
    ~ StatsCounter )
```

Definition at line 13 of file statscounter-meat.hpp.

8.42.2.4 STATSCOUNTER_TEMPLATE() [2/9]

```
STATSCOUNTER_TEMPLATE (
    std::vector< double > ,
    count_all )
```

Definition at line 91 of file statscounter-meat.hpp.

8.42.2.5 STATSCOUNTER_TEMPLATE() [3/9]

```
STATSCOUNTER_TEMPLATE (
    std::vector< std::string > ,
    get_descriptions ) const
```

Definition at line 143 of file statscounter-meat.hpp.

8.42.2.6 STATS_COUNTER_TEMPLATE() [4/9]

```
STATSCOUNTER_TEMPLATE (
    std::vector< std::string > ,
    get_names ) const
```

Definition at line 138 of file statscounter-meat.hpp.

8.42.2.7 STATS_COUNTER_TEMPLATE() [5/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    add_counter )
```

8.42.2.8 STATS_COUNTER_TEMPLATE() [6/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    count_current )
```

8.42.2.9 STATS_COUNTER_TEMPLATE() [7/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    count_init )
```

8.42.2.10 STATS_COUNTER_TEMPLATE() [8/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    reset_array ) const
```

Definition at line 20 of file statscounter-meat.hpp.

8.42.2.11 STATS_COUNTER_TEMPLATE() [9/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    set_counters )
```

8.42.3 Variable Documentation

8.42.3.1 counter_deleted

```
counter_deleted = true
```

Definition at line 52 of file statscounter-meat.hpp.

8.42.3.2 counters

```
counters = counters_
```

Definition at line 53 of file statscounter-meat.hpp.

8.42.3.3 counters_

```
Data_Type* counters_
```

Initial value:

```
{  
  
    if (!counter_deleted)  
        delete counters
```

Definition at line 46 of file statscounter-meat.hpp.

8.42.3.4 f_

```
Data_Rule_Dyn_Type f_
```

Initial value:

```
{  
  
    counters->add_counter(f_)
```

Definition at line 29 of file statscounter-meat.hpp.

8.42.3.5 j

```
uint j
```

Initial value:

```
{  
  
    if (counters->size() == 0u)  
        throw std::logic_error("No counters added: Cannot count without knowing what to count!")  
}
```

Definition at line 59 of file statscounter-meat.hpp.

8.42.3.6 return

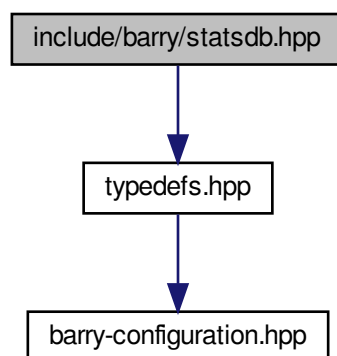
```
return
```

Definition at line 33 of file statscounter-meat.hpp.

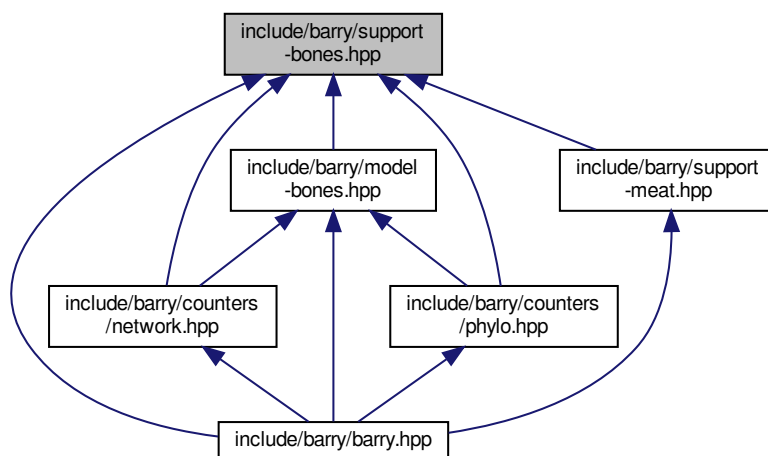
8.43 include/barry/statsdb.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for statsdb.hpp:



This graph shows which files directly or indirectly include this file:



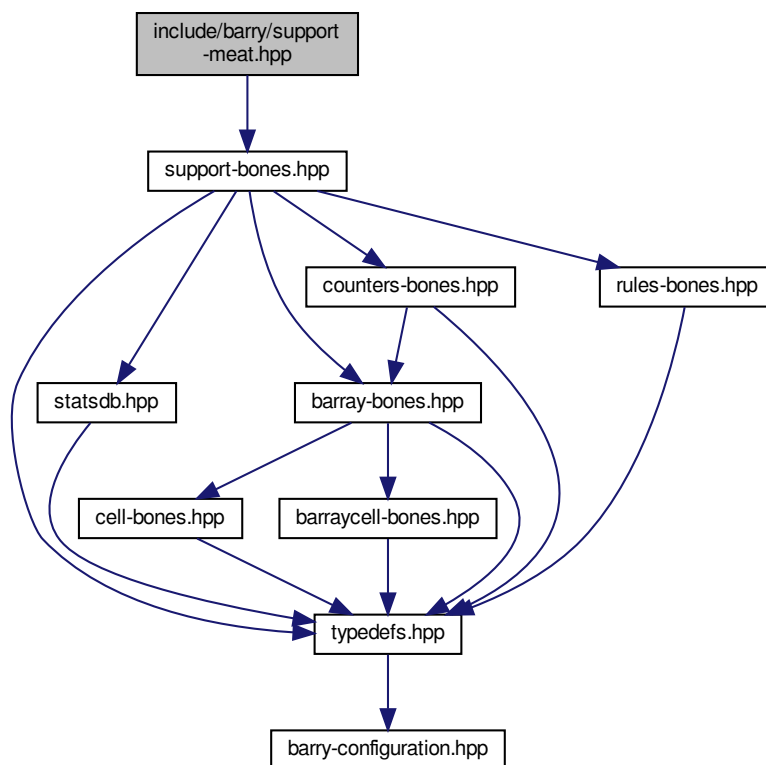
Classes

- class [Support](#)< [Array_Type](#), [Data_Counter_Type](#), [Data_Rule_Type](#), [Data_Rule_Dyn_Type](#) >
Compute the support of sufficient statistics.

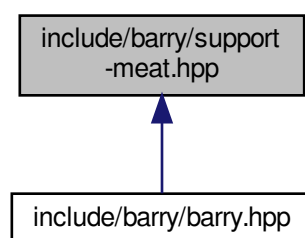
8.45 include/barry/support-meat.hpp File Reference

```
#include "support-bones.hpp"
```

Include dependency graph for support-meat.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define `BARRY_SUPPORT_MEAT_HPP` 1
- #define `SUPPORT_TEMPLATE_ARGS()`
- #define `SUPPORT_TYPE()`
- #define `SUPPORT_TEMPLATE(a, b)`

Functions

- [SUPPORT_TEMPLATE](#) (void, init_support)(std
- [SUPPORT_TEMPLATE](#) (void, reset_array)()
- [SUPPORT_TEMPLATE](#) (void, reset_array)(const Array_Type &Array_)
- [SUPPORT_TEMPLATE](#) (void, [calc_backend](#))(uint pos
- [calc_backend](#) (pos+1u, [array_bank](#), [stats_bank](#))
- EmptyArray [insert_cell](#) (cfree.first, cfree.second, EmptyArray.default_val().value, false, false)
- [for](#) (uint n=0u;n< [counters](#)->size();++n)
- [if](#) ([rules_dyn](#)->size() > 0u)
- [if](#) ([array_bank](#) !=nullptr) [array_bank](#) -> [push_back](#)(EmptyArray)
- [if](#) ([stats_bank](#) !=nullptr) [stats_bank](#) -> [push_back](#)(current_stats)
- EmptyArray [rm_cell](#) (cfree.first, cfree.second, false, false)
- [SUPPORT_TEMPLATE](#) (void, calc)(std
- [SUPPORT_TEMPLATE](#) (void, add_counter)(Counter< Array_Type
- [SUPPORT_TEMPLATE](#) (void, set_counters)(Counters< Array_Type
- [SUPPORT_TEMPLATE](#) (void, add_rule)(Rule< Array_Type
- [SUPPORT_TEMPLATE](#) (void, set_rules)(Rules< Array_Type
- [SUPPORT_TEMPLATE](#) (void, add_rule_dyn)(Rule< Array_Type
- [SUPPORT_TEMPLATE](#) (void, set_rules_dyn)(Rules< Array_Type
- [SUPPORT_TEMPLATE](#) (bool, eval_rules_dyn)(const std
- [SUPPORT_TEMPLATE](#) ([Counts_type](#), get_counts)() const
- [SUPPORT_TEMPLATE](#) (const [MapVec_type](#)<> *, get_counts_ptr)() const
- [SUPPORT_TEMPLATE](#) (std::vector< double > *, get_current_stats)()
- [SUPPORT_TEMPLATE](#) (void, print)() const
- [SUPPORT_TEMPLATE](#) (const [FreqTable](#)<> &, get_data)() const

Variables

- std::vector< Array_Type > * [array_bank](#)
- std::vector< Array_Type > std::vector< std::vector< double > > * [stats_bank](#)
- const std::pair< uint, uint > & [cfree](#) = coordinates_free[pos]
- [else](#)
- [return](#)
- Data_Counter_Type * [f_](#)
- Data_Counter_Type * [counters_](#)
- [delete_counters](#) = false
- [counters](#) = [counters_](#)
- Data_Rule_Type * [rules_](#)
- [delete_rules](#) = false
- [rules](#) = [rules_](#)
- [delete_rules_dyn](#) = false
- [rules_dyn](#) = [rules_](#)

8.45.1 Macro Definition Documentation

8.45.1.1 BARRY_SUPPORT_MEAT_HPP

```
#define BARRY_SUPPORT_MEAT_HPP 1
```

Definition at line 4 of file support-meat.hpp.

8.45.1.2 SUPPORT_TEMPLATE

```
#define SUPPORT_TEMPLATE(  
    a,  
    b )
```

Value:

```
template SUPPORT_TEMPLATE_ARGS() \  
inline a SUPPORT_TYPE()::b
```

Definition at line 12 of file support-meat.hpp.

8.45.1.3 SUPPORT_TEMPLATE_ARGS

```
template SUPPORT_TEMPLATE_ARGS( )
```

Value:

```
<typename Array_Type, typename \  
Data_Counter_Type, typename Data_Rule_Type, typename Data_Rule_Dyn_Type>
```

Definition at line 6 of file support-meat.hpp.

8.45.1.4 SUPPORT_TYPE

```
template Data_Rule_Dyn_Type * SUPPORT_TYPE( )
```

Value:

```
Support<Array_Type, Data_Counter_Type, Data_Rule_Type, \  
Data_Rule_Dyn_Type>
```

Definition at line 9 of file support-meat.hpp.

8.45.2 Function Documentation

8.45.2.1 calc_backend()

```
calc_backend (
    pos+ 1u,
    array_bank ,
    stats_bank )
```

8.45.2.2 for()

```
for (
    uint n = 0u; n < counters->size(); ++n )
```

Definition at line 134 of file support-meat.hpp.

8.45.2.3 if() [1/3]

```
if (
    array_bank !    = nullptr ) -> push_back(EmptyArray)
```

8.45.2.4 if() [2/3]

```
if (
    rules_dyn-> size(),
    0u )
```

Definition at line 148 of file support-meat.hpp.

8.45.2.5 if() [3/3]

```
if (
    stats_bank !    = nullptr ) -> push_back(current_stats)
```

8.45.2.6 insert_cell()

```
EmptyArray insert_cell (
    cfree. first,
    cfree. second,
    EmptyArray.default_val(). value,
    false ,
    false )
```

8.45.2.7 `rm_cell()`

```
EmptyArray rm_cell (
    cfree. first,
    cfree. second,
    false ,
    false )
```

8.45.2.8 `SUPPORT_TEMPLATE()` [1/17]

```
SUPPORT_TEMPLATE (
    bool ,
    eval_rules_dyn ) const
```

Definition at line 320 of file support-meat.hpp.

8.45.2.9 `SUPPORT_TEMPLATE()` [2/17]

```
SUPPORT_TEMPLATE (
    const FreqTable<> & ,
    get_data ) const
```

Definition at line 369 of file support-meat.hpp.

8.45.2.10 `SUPPORT_TEMPLATE()` [3/17]

```
SUPPORT_TEMPLATE (
    const MapVec_type<> * ,
    get_counts_ptr ) const
```

Definition at line 348 of file support-meat.hpp.

8.45.2.11 `SUPPORT_TEMPLATE()` [4/17]

```
SUPPORT_TEMPLATE (
    Counts_type ,
    get_counts ) const
```

Definition at line 342 of file support-meat.hpp.

8.45.2.12 SUPPORT_TEMPLATE() [5/17]

```
SUPPORT_TEMPLATE (
    std::vector< double > * ,
    get_current_stats )
```

Definition at line 354 of file support-meat.hpp.

8.45.2.13 SUPPORT_TEMPLATE() [6/17]

```
SUPPORT_TEMPLATE (
    void ,
    add_counter )
```

8.45.2.14 SUPPORT_TEMPLATE() [7/17]

```
SUPPORT_TEMPLATE (
    void ,
    add_rule )
```

8.45.2.15 SUPPORT_TEMPLATE() [8/17]

```
SUPPORT_TEMPLATE (
    void ,
    add_rule_dyn )
```

8.45.2.16 SUPPORT_TEMPLATE() [9/17]

```
SUPPORT_TEMPLATE (
    void ,
    calc )
```

Definition at line 197 of file support-meat.hpp.

8.45.2.17 SUPPORT_TEMPLATE() [10/17]

```
SUPPORT_TEMPLATE (
    void ,
    calc_backend )
```

8.45.2.18 SUPPORT_TEMPLATE() [11/17]

```
SUPPORT_TEMPLATE (
    void ,
    init_support )
```

Definition at line 15 of file support-meat.hpp.

8.45.2.19 SUPPORT_TEMPLATE() [12/17]

```
SUPPORT_TEMPLATE (
    void ,
    print ) const
```

Definition at line 358 of file support-meat.hpp.

8.45.2.20 SUPPORT_TEMPLATE() [13/17]

```
SUPPORT_TEMPLATE (
    void ,
    reset_array )
```

Definition at line 91 of file support-meat.hpp.

8.45.2.21 SUPPORT_TEMPLATE() [14/17]

```
SUPPORT_TEMPLATE (
    void ,
    reset_array ) const &
```

Definition at line 97 of file support-meat.hpp.

8.45.2.22 SUPPORT_TEMPLATE() [15/17]

```
SUPPORT_TEMPLATE (
    void ,
    set_counters )
```


8.45.2.23 SUPPORT_TEMPLATE() [16/17]

```
SUPPORT_TEMPLATE (
    void ,
    set_rules )
```

8.45.2.24 SUPPORT_TEMPLATE() [17/17]

```
SUPPORT_TEMPLATE (
    void ,
    set_rules_dyn )
```

8.45.3 Variable Documentation

8.45.3.1 array_bank

```
std::vector< Array_Type >* array_bank
```

Definition at line 109 of file support-meat.hpp.

8.45.3.2 cfree

```
const std::pair<uint,uint>& cfree = coordinates_free[pos]
```

Definition at line 123 of file support-meat.hpp.

8.45.3.3 counters

```
counters = counters_
```

Definition at line 248 of file support-meat.hpp.

8.45.3.4 counters_

```
Data_Counter_Type* counters_
```

Initial value:

```
{  
  
    if (delete_counters)  
        delete counters
```

Definition at line 241 of file support-meat.hpp.

8.45.3.5 delete_counters

```
delete_counters = false
```

Definition at line 247 of file support-meat.hpp.

8.45.3.6 delete_rules

```
delete_rules = false
```

Definition at line 281 of file support-meat.hpp.

8.45.3.7 delete_rules_dyn

```
delete_rules_dyn = false
```

Definition at line 313 of file support-meat.hpp.

8.45.3.8 else

```
else
```

Initial value:

```
{  
    data.add(current_stats)
```

Definition at line 166 of file support-meat.hpp.

8.45.3.9 f_

Data_Rule_Dyn_Type f_

Initial value:

```
{  
    counters->add_counter(f_)
```

Definition at line 223 of file support-meat.hpp.

8.45.3.10 return

return

Definition at line 193 of file support-meat.hpp.

8.45.3.11 rules

rules = rules_

Definition at line 282 of file support-meat.hpp.

8.45.3.12 rules_

Data_Rule_Dyn_Type * rules_

Initial value:

```
{  
  
    if (delete_rules)  
        delete rules
```

Definition at line 275 of file support-meat.hpp.

8.45.3.13 rules_dyn

rules_dyn = rules_

Definition at line 314 of file support-meat.hpp.

8.45.3.14 stats_bank

```
std::vector< Array_Type > std::vector< std::vector< double > >* stats_bank
```

Initial value:

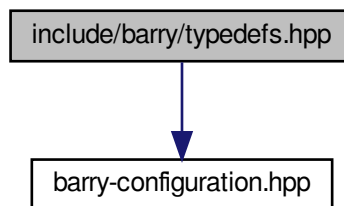
```
{
    if (pos >= coordinates_free.size())
        return
```

Definition at line 110 of file support-meat.hpp.

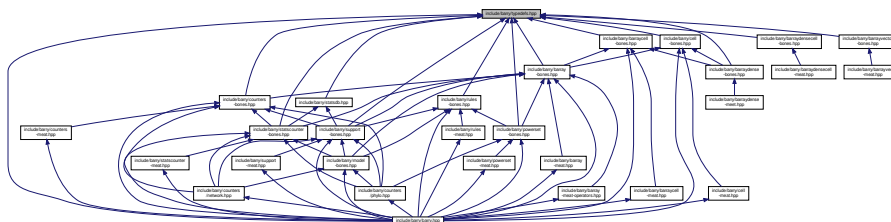
8.46 include/barry/typedefs.hpp File Reference

```
#include "barry-configuration.hpp"
```

Include dependency graph for typedefs.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Entries< Cell_Type >](#)
A wrapper class to store *source*, *target*, *val* from a [BArray](#) object.
- struct [vecHasher< T >](#)

Namespaces

- [CHECK](#)
Integer constants used to specify which cell should be check.
- [EXISTS](#)
Integer constants used to specify which cell should be check to exist or not.

Typedefs

- typedef unsigned int [uint](#)
- typedef std::vector< std::pair< std::vector< double >, [uint](#) > > [Counts_type](#)
- template<typename Cell_Type >
using [Row_type](#) = Map< [uint](#), Cell< Cell_Type > >
- template<typename Cell_Type >
using [Col_type](#) = Map< [uint](#), Cell< Cell_Type > * >
- template<typename Ta = double, typename Tb = uint>
using [MapVec_type](#) = std::unordered_map< std::vector< Ta >, Tb, [vecHasher](#)< Ta > >
- template<typename Array_Type , typename Data_Type >
using [Counter_fun_type](#) = std::function< double(const Array_Type &, [uint](#), [uint](#), Data_Type *)>
[Counter](#) and rule functions.
- template<typename Array_Type , typename Data_Type >
using [Rule_fun_type](#) = std::function< bool(const Array_Type &, [uint](#), [uint](#), Data_Type *)>

Functions

- template<typename T >
T [vec_inner_prod](#) (const std::vector< T > &a, const std::vector< T > &b)
- template<typename T >
bool [vec_equal](#) (const std::vector< T > &a, const std::vector< T > &b)
Compares if -a- and -b- are equal.
- template<typename T >
bool [vec_equal_approx](#) (const std::vector< T > &a, const std::vector< T > &b, double eps=1e-10)

Variables

- const int [CHECK::BOTH](#) = -1
- const int [CHECK::NONE](#) = 0
- const int [CHECK::ONE](#) = 1
- const int [CHECK::TWO](#) = 2
- const int [EXISTS::BOTH](#) = -1
- const int [EXISTS::NONE](#) = 0
- const int [EXISTS::ONE](#) = 1
- const int [EXISTS::TWO](#) = 1
- const int [EXISTS::UNKNOWN](#) = -1
- const int [EXISTS::AS_ZERO](#) = 0
- const int [EXISTS::AS_ONE](#) = 1

8.46.1 Typedef Documentation

8.46.1.1 Col_type

```
template<typename Cell_Type >
using Col_type = Map< uint, Cell<Cell_Type>* >
```

Definition at line 51 of file typedefs.hpp.

8.46.1.2 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

<i>Array_Type</i>	a BArray
<i>unit,uint</i>	Focal cell
<i>Data_Type</i>	Data associated with the function, for example, id of the attribute in the Array.

Returns

`Counter_fun_type` a double (the change statistic)

`Rule_fun_type` a bool. True if the cell is blocked.

Definition at line 123 of file typedefs.hpp.

8.46.1.3 Counts_type

```
typedef std::vector< std::pair< std::vector<double>, uint > > Counts_type
```

Definition at line 44 of file typedefs.hpp.

8.46.1.4 MapVec_type

```
template<typename Ta = double, typename Tb = uint>
using MapVec_type = std::unordered_map< std::vector< Ta >, Tb, vecHasher<Ta> >
```

Definition at line 105 of file typedefs.hpp.

8.46.1.5 Row_type

```
template<typename Cell_Type >
using Row_type = Map< uint, Cell<Cell_Type> >
```

Definition at line 48 of file typedefs.hpp.

8.46.1.6 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 126 of file typedefs.hpp.

8.46.1.7 uint

```
typedef unsigned int uint
```

Definition at line 10 of file typedefs.hpp.

8.46.2 Function Documentation

8.46.2.1 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

<i>a,b</i>	Two vectors of the same length
------------	--------------------------------

Returns

`true` if all elements are equal.

Definition at line 137 of file typedefs.hpp.

8.46.2.2 `vec_equal_approx()`

```
template<typename T >
bool vec_equal_approx (
    const std::vector< T > & a,
    const std::vector< T > & b,
    double eps = 1e-10 ) [inline]
```

Definition at line 155 of file typedefs.hpp.

8.46.2.3 `vec_inner_prod()`

```
template<typename T >
T vec_inner_prod (
    const std::vector< T > & a,
    const std::vector< T > & b ) [inline]
```

Definition at line 175 of file typedefs.hpp.

8.47 README.md File Reference

Index

- ~BArray
 - BArray< Cell_Type, Data_Type >, [33](#)
- ~BArrayCell
 - BArrayCell< Cell_Type, Data_Type >, [44](#)
- ~BArrayCell_const
 - BArrayCell_const< Cell_Type, Data_Type >, [46](#)
- ~BArrayDense
 - BArrayDense< Cell_Type, Data_Type >, [52](#)
- ~BArrayDenseCell
 - BArrayDenseCell< Cell_Type, Data_Type >, [64](#)
- ~BArrayDenseCell_const
 - BArrayDenseCell_const< Cell_Type, Data_Type >, [67](#)
- ~BArrayVector
 - BArrayVector< Cell_Type, Data_Type >, [70](#)
- ~BArrayVector_const
 - BArrayVector_const< Cell_Type, Data_Type >, [73](#)
- ~Cell
 - Cell< Cell_Type >, [77](#)
- ~ConstBArrayRowIter
 - ConstBArrayRowIter< Cell_Type, Data_Type >, [82](#)
- ~Counter
 - Counter< Array_Type, Data_Type >, [85](#)
- ~Counters
 - Counters< Array_Type, Data_Type >, [89](#)
- ~Entries
 - Entries< Cell_Type >, [93](#)
- ~Flock
 - Flock, [96](#)
- ~FreqTable
 - FreqTable< T >, [101](#)
- ~Geese
 - Geese, [107](#)
- ~Model
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [119](#)
- ~NetCounterData
 - NetCounterData, [130](#)
- ~NetworkData
 - NetworkData, [132](#)
- ~Node
 - Node, [135](#)
- ~PhyloRuleDynData
 - PhyloRuleDynData, [143](#)
- ~PowerSet
 - PowerSet< Array_Type, Data_Rule_Type >, [147](#)
- ~Rule
 - Rule< Array_Type, Data_Type >, [152](#)
- ~Rules
 - Rules< Array_Type, Data_Type >, [154](#)
- ~StatsCounter
 - StatsCounter< Array_Type, Data_Type >, [158](#)
- ~Support
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [163](#)
- add
 - Cell< Cell_Type >, [78](#), [79](#)
 - FreqTable< T >, [102](#)
- add_array
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [120](#)
- add_counter
 - Counters< Array_Type, Data_Type >, [89](#), [90](#)
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [120](#)
 - StatsCounter< Array_Type, Data_Type >, [158](#)
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [163](#)
- add_data
 - Flock, [96](#)
- add_rule
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [121](#)
 - PowerSet< Array_Type, Data_Rule_Type >, [147](#)
 - Rules< Array_Type, Data_Type >, [155](#)
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [164](#)
- add_rule_dyn
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [121](#), [122](#)
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [164](#)
- annotations
 - Node, [136](#)
- Array
 - ConstBArrayRowIter< Cell_Type, Data_Type >, [82](#)
- array
 - Node, [136](#)
- array_bank

- support-meat.hpp, 253
- arrays
 - Node, 136
- AS_ONE
 - EXISTS, 27
- as_vector
 - FreqTable< T >, 102
- AS_ZERO
 - EXISTS, 27
- at
 - PhyloCounterData, 141
- BArray
 - BArray< Cell_Type, Data_Type >, 32, 33
- BArray< Cell_Type, Data_Type >, 29
 - ~BArray, 33
 - BArray, 32, 33
 - BArrayCell< Cell_Type, Data_Type >, 42
 - BArrayCell_const< Cell_Type, Data_Type >, 42
 - clear, 33
 - col, 33
 - D, 34
 - default_val, 34
 - flush_data, 34
 - get_cell, 34
 - get_col_vec, 34, 35
 - get_entries, 35
 - get_row_vec, 35
 - insert_cell, 35, 36
 - is_empty, 36
 - ncol, 36
 - nnozero, 36
 - nrow, 37
 - operator*=: 37
 - operator(), 37
 - operator+=, 37, 38
 - operator-=, 38
 - operator/=, 38
 - operator=, 38, 39
 - operator==, 39
 - out_of_range, 39
 - print, 39
 - reserve, 39
 - resize, 39
 - rm_cell, 40
 - row, 40
 - set_data, 40
 - swap_cells, 40
 - swap_cols, 41
 - swap_rows, 41
 - toggle_cell, 41
 - toggle_lock, 41
 - transpose, 41
 - visited, 42
 - zero_col, 42
 - zero_row, 42
- barray-bones.hpp
 - BARRAY_BONES_HPP, 174
- barray-meat-operators.hpp
 - BARRY_BARRAY_MEAT_OPERATORS_HPP, 176
 - checkdim_, 176
 - COL, 176
 - ROW, 176
- barray-meat.hpp
 - COL, 178
 - ROW, 178
- BARRAY_BONES_HPP
 - barray-bones.hpp, 174
- BArrayCell
 - BArrayCell< Cell_Type, Data_Type >, 43
- BArrayCell< Cell_Type, Data_Type >, 43
 - ~BArrayCell, 44
 - BArray< Cell_Type, Data_Type >, 42
 - BArrayCell, 43
 - BArrayDense< Cell_Type, Data_Type >, 63
 - operator Cell_Type, 44
 - operator*=: 44
 - operator+=, 44
 - operator-=, 44
 - operator/=, 45
 - operator=, 45
 - operator==, 45
- BArrayCell_const
 - BArrayCell_const< Cell_Type, Data_Type >, 46
- BArrayCell_const< Cell_Type, Data_Type >, 45
 - ~BArrayCell_const, 46
 - BArray< Cell_Type, Data_Type >, 42
 - BArrayCell_const, 46
 - BArrayDense< Cell_Type, Data_Type >, 63
 - operator Cell_Type, 46
 - operator!=, 47
 - operator<, 47
 - operator<=, 47
 - operator>, 47
 - operator>=, 48
 - operator==, 47
- BArrayDense
 - BArrayDense< Cell_Type, Data_Type >, 51, 52
- BArrayDense< Cell_Type, Data_Type >, 48
 - ~BArrayDense, 52
 - BArrayCell< Cell_Type, Data_Type >, 63
 - BArrayCell_const< Cell_Type, Data_Type >, 63
 - BArrayDense, 51, 52
 - clear, 52
 - col, 53
 - D, 53
 - default_val, 53
 - get_cell, 53
 - get_col_vec, 54
 - get_entries, 54
 - get_row_vec, 54, 55
 - insert_cell, 55
 - is_empty, 56
 - ncol, 56
 - nnozero, 56
 - nrow, 56

- operator*=[, 57](#)
- operator()[, 56, 57](#)
- operator+=[, 57](#)
- operator-=[, 58](#)
- operator/= [, 58](#)
- operator=[, 58, 59](#)
- operator==[, 59](#)
- out_of_range[, 59](#)
- print[, 59](#)
- reserve[, 59](#)
- resize[, 60](#)
- rm_cell[, 60](#)
- row[, 60](#)
- set_data[, 60](#)
- swap_cells[, 61](#)
- swap_cols[, 61](#)
- swap_rows[, 61](#)
- toggle_cell[, 61](#)
- toggle_lock[, 62](#)
- transpose[, 62](#)
- visited[, 63](#)
- zero_col[, 62](#)
- zero_row[, 62](#)
- barrydense-meet.hpp
 - BARRY_BARRAYDENSE_MEAT_HPP [, 182](#)
 - COL [, 182](#)
 - POS [, 183](#)
 - ROW [, 183](#)
 - ZERO_CELL [, 183](#)
- BArrayDenseCell
 - BArrayDenseCell< Cell_Type, Data_Type > [, 64](#)
- BArrayDenseCell< Cell_Type, Data_Type > [, 63](#)
 - ~BArrayDenseCell [, 64](#)
 - BArrayDenseCell [, 64](#)
 - operator Cell_Type [, 64](#)
 - operator*=[, 65](#)
 - operator+=[, 65](#)
 - operator-=[, 65](#)
 - operator/= [, 65](#)
 - operator=[, 65](#)
 - operator==[, 66](#)
- barrydensecell-meet.hpp
 - BARRY_BARRAYDENSECELL_MEAT_HPP [, 185](#)
 - POS [, 185](#)
- BArrayDenseCell_const
 - BArrayDenseCell_const< Cell_Type, Data_Type > [, 67](#)
- BArrayDenseCell_const< Cell_Type, Data_Type > [, 66](#)
 - ~BArrayDenseCell_const [, 67](#)
 - BArrayDenseCell_const [, 67](#)
 - operator Cell_Type [, 67](#)
 - operator!=[, 67](#)
 - operator<[, 67](#)
 - operator<=[, 68](#)
 - operator>[, 68](#)
 - operator>=[, 68](#)
 - operator==[, 68](#)
- BArrayVector
 - BArrayVector< Cell_Type, Data_Type > [, 69](#)
- BArrayVector< Cell_Type, Data_Type > [, 69](#)
 - ~BArrayVector [, 70](#)
 - BArrayVector [, 69](#)
 - begin [, 70](#)
 - end [, 70](#)
 - is_col [, 70](#)
 - is_row [, 71](#)
 - operator std::vector< Cell_Type > [, 71](#)
 - operator*=[, 71](#)
 - operator+=[, 71](#)
 - operator-=[, 71](#)
 - operator/= [, 72](#)
 - operator=[, 72](#)
 - operator==[, 72](#)
 - size [, 72](#)
- barryvector-meat.hpp
 - BARRY_BARRAYVECTOR_MEAT_HPP [, 187](#)
- BArrayVector_const
 - BArrayVector_const< Cell_Type, Data_Type > [, 73](#)
- BArrayVector_const< Cell_Type, Data_Type > [, 73](#)
 - ~BArrayVector_const [, 73](#)
 - BArrayVector_const [, 73](#)
 - begin [, 74](#)
 - end [, 74](#)
 - is_col [, 74](#)
 - is_row [, 74](#)
 - operator std::vector< Cell_Type > [, 74](#)
 - operator!=[, 74](#)
 - operator<[, 75](#)
 - operator<=[, 75](#)
 - operator>[, 75](#)
 - operator>=[, 75](#)
 - operator==[, 75](#)
 - size [, 76](#)
- barry [, 25](#)
- barry-configuration.hpp
 - BARRY_CHECK_SUPPORT [, 188](#)
 - BARRY_ISFINITE [, 188](#)
 - BARRY_MAX_NUM_ELEMENTS [, 188](#)
 - BARRY_SAFE_EXP [, 188](#)
 - Map [, 188](#)
 - printf_barry [, 188](#)
- barry.hpp
 - BARRY_HPP [, 190](#)
 - BARRY_VERSION [, 190](#)
 - COUNTER_FUNCTION [, 190](#)
 - COUNTER_LAMBDA [, 190](#)
 - RULE_FUNCTION [, 191](#)
 - RULE_LAMBDA [, 191](#)
- barry::counters [, 25](#)
- barry::counters::network [, 26](#)
- barry::counters::phylo [, 26](#)
- BARRY_BARRAY_MEAT_OPERATORS_HPP
 - barry-meat-operators.hpp [, 176](#)
- BARRY_BARRAYDENSE_MEAT_HPP
 - barrydense-meet.hpp [, 182](#)
- BARRY_BARRAYDENSECELL_MEAT_HPP

- barraydensecell-meat.hpp, 185
- BARRY_BARRAYVECTOR_MEAT_HPP
 - barrayvector-meat.hpp, 187
- BARRY_CHECK_SUPPORT
 - barry-configuration.hpp, 188
- BARRY_HPP
 - barry.hpp, 190
- BARRY_ISFINITE
 - barry-configuration.hpp, 188
- BARRY_MAX_NUM_ELEMENTS
 - barry-configuration.hpp, 188
- BARRY_SAFE_EXP
 - barry-configuration.hpp, 188
- BARRY_SUPPORT_MEAT_HPP
 - support-meat.hpp, 247
- BARRY_VERSION
 - barry.hpp, 190
- begin
 - BArrayVector< Cell_Type, Data_Type >, 70
 - BArrayVector_const< Cell_Type, Data_Type >, 74
 - PhyloCounterData, 141
 - PowerSet< Array_Type, Data_Rule_Type >, 147
- blengths
 - NodeData, 139
- BOTH
 - CHECK, 26
 - EXISTS, 27
- calc
 - PowerSet< Array_Type, Data_Rule_Type >, 148
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 164
- calc_backend
 - support-meat.hpp, 248
- calc_reduced_sequence
 - Geese, 107
- calc_sequence
 - Geese, 107
- Cell
 - Cell< Cell_Type >, 77, 78
- Cell< Cell_Type >, 76
 - ~Cell, 77
 - add, 78, 79
 - Cell, 77, 78
 - operator Cell_Type, 79
 - operator!=, 79
 - operator=, 79, 80
 - operator==, 80
 - value, 80
 - visited, 80
- cfree
 - support-meat.hpp, 253
- change_stats
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 169
- CHECK, 26
 - BOTH, 26
- NONE, 26
- ONE, 26
- TWO, 26
- checkdim_
 - barray-meat-operators.hpp, 176
- clear
 - BArray< Cell_Type, Data_Type >, 33
 - BArrayDense< Cell_Type, Data_Type >, 52
 - Counters< Array_Type, Data_Type >, 90
 - FreqTable< T >, 102
 - Rules< Array_Type, Data_Type >, 155
- COL
 - barray-meat-operators.hpp, 176
 - barray-meat.hpp, 178
 - barraydense-meet.hpp, 182
- col
 - BArray< Cell_Type, Data_Type >, 33
 - BArrayDense< Cell_Type, Data_Type >, 53
- Col_type
 - typedefs.hpp, 258
- colnames
 - Flock, 96
 - Geese, 107
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 122
- conditional_prob
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 122
- ConstBArrayRowIter
 - ConstBArrayRowIter< Cell_Type, Data_Type >, 81
- ConstBArrayRowIter< Cell_Type, Data_Type >, 81
 - ~ConstBArrayRowIter, 82
 - Array, 82
 - ConstBArrayRowIter, 81
 - current_col, 82
 - current_row, 82
 - iter, 82
- coordinates_free
 - PowerSet< Array_Type, Data_Rule_Type >, 149
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 169
- coordinates_locked
 - PowerSet< Array_Type, Data_Rule_Type >, 149
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 169
- count
 - Counter< Array_Type, Data_Type >, 85
- count_all
 - StatsCounter< Array_Type, Data_Type >, 159
- count_current
 - StatsCounter< Array_Type, Data_Type >, 159
- count_fun
 - Counter< Array_Type, Data_Type >, 86
 - counters-meat.hpp, 198

- count_fun_
 - counters-meat.hpp, 204
- count_init
 - StatsCounter< Array_Type, Data_Type >, 159
- Counter
 - Counter< Array_Type, Data_Type >, 84, 85
- counter
 - counters-meat.hpp, 204
- Counter< Array_Type, Data_Type >, 83
 - ~Counter, 85
 - count, 85
 - count_fun, 86
 - Counter, 84, 85
 - data, 86
 - delete_data, 87
 - desc, 87
 - get_description, 85
 - get_name, 85
 - init, 86
 - init_fun, 87
 - name, 87
 - operator=, 86
- counter_
 - counters-meat.hpp, 204
- counter_absdiff
 - Network counters, 13
- counter_co_opt
 - Phylo counters, 19
- counter_cogain
 - Phylo counters, 19
- counter_css_completely_false_recip_comiss
 - Network counters, 13
- counter_css_completely_false_recip_omiss
 - Network counters, 13
- counter_css_partially_false_recip_commi
 - Network counters, 13
- counter_css_partially_false_recip_omiss
 - Network counters, 14
- counter_ctriads
 - Network counters, 14
- counter_degree
 - Network counters, 14
- counter_deleted
 - statscounter-meat.hpp, 242
- counter_density
 - Network counters, 14
- counter_diff
 - Network counters, 14
- counter_edges
 - Network counters, 15
- Counter_fun_type
 - typedefs.hpp, 258
- COUNTER_FUNCTION
 - barry.hpp, 190
- counter_gains
 - Phylo counters, 19
- counter_gains_k_offspring
 - Phylo counters, 20
- counter_genes_changing
 - Phylo counters, 20
- counter_idegree
 - Network counters, 15
- counter_idegree15
 - Network counters, 15
- counter_isolates
 - Network counters, 15
- counter_istar2
 - Network counters, 15
- COUNTER_LAMBDA
 - barry.hpp, 190
- counter_longest
 - Phylo counters, 20
- counter_loss
 - Phylo counters, 20
- counter_maxfun
 - Phylo counters, 21
- counter_mutual
 - Network counters, 16
- counter_neofun
 - Phylo counters, 21
- counter_neofun_a2b
 - Phylo counters, 21
- counter_nodecov
 - Network counters, 16
- counter_nodeicov
 - Network counters, 16
- counter_nodematch
 - Network counters, 16
- counter_nodecov
 - Network counters, 16
- counter_odegree
 - Network counters, 17
- counter_odegree15
 - Network counters, 17
- counter_ostar2
 - Network counters, 17
- counter_overall_changes
 - Phylo counters, 21
- counter_overall_gains
 - Phylo counters, 22
- counter_overall_loss
 - Phylo counters, 22
- counter_prop_genes_changing
 - Phylo counters, 22
- counter_subfun
 - Phylo counters, 22
- COUNTER_TEMPLATE
 - counters-meat.hpp, 197–199
- COUNTER_TEMPLATE_ARGS
 - counters-meat.hpp, 197
- counter_ttriads
 - Network counters, 17
- COUNTER_TYPE
 - counters-meat.hpp, 197
- Counters
 - Counters< Array_Type, Data_Type >, 88, 89

- counters
 - statscounter-meat.hpp, 242
 - support-meat.hpp, 253
- Counters< Array_Type, Data_Type >, 88
 - ~Counters, 89
 - add_counter, 89, 90
 - clear, 90
 - Counters, 88, 89
 - get_descriptions, 90
 - get_names, 90
 - operator=, 90, 91
 - operator[], 91
 - size, 91
- counters-meat.hpp
 - count_fun, 198
 - count_fun_, 204
 - counter, 204
 - counter_, 204
 - COUNTER_TEMPLATE, 197–199
 - COUNTER_TEMPLATE_ARGS, 197
 - COUNTER_TYPE, 197
 - COUNTERS_TEMPLATE, 197, 199–201
 - COUNTERS_TEMPLATE_ARGS, 198
 - COUNTERS_TYPE, 198
 - data, 201
 - data_, 204
 - delete_data, 201
 - delete_data_, 204
 - delete_to_be_deleted, 202
 - desc, 202
 - desc_, 205
 - i, 205
 - init_fun, 202, 203
 - init_fun_, 205
 - j, 205
 - name, 203
 - name_, 205
 - noexcept, 206
 - push_back, 203
 - return, 206
 - to_be_deleted, 203
- counters_
 - statscounter-meat.hpp, 242
 - support-meat.hpp, 253
- COUNTERS_TEMPLATE
 - counters-meat.hpp, 197, 199–201
- COUNTERS_TEMPLATE_ARGS
 - counters-meat.hpp, 198
- COUNTERS_TYPE
 - counters-meat.hpp, 198
- Counting, 9
- counts
 - PhyloRuleDynData, 144
- Counts_type
 - typedefs.hpp, 258
- CSS_APPEND
 - Network counters, 11
- CSS_CASE_ELSE
 - Network counters, 11
- CSS_CASE_PERCEIVED
 - Network counters, 12
- CSS_CASE_TRUTH
 - Network counters, 12
- CSS_CHECK_SIZE
 - Network counters, 12
- CSS_CHECK_SIZE_INIT
 - Network counters, 12
- CSS_SIZE
 - Network counters, 12
- current_col
 - ConstBArrayRowIter< Cell_Type, Data_Type >, 82
- current_row
 - ConstBArrayRowIter< Cell_Type, Data_Type >, 82
- current_stats
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 169
- D
 - BArray< Cell_Type, Data_Type >, 34
 - BArrayDense< Cell_Type, Data_Type >, 53
 - Rule< Array_Type, Data_Type >, 153
- dat
 - Flock, 100
- data
 - Counter< Array_Type, Data_Type >, 86
 - counters-meat.hpp, 201
 - PowerSet< Array_Type, Data_Rule_Type >, 150
- data_
 - counters-meat.hpp, 204
- default_val
 - BArray< Cell_Type, Data_Type >, 34
 - BArrayDense< Cell_Type, Data_Type >, 53
- delete_counters
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 170
 - support-meat.hpp, 254
- delete_data
 - Counter< Array_Type, Data_Type >, 87
 - counters-meat.hpp, 201
- delete_data_
 - counters-meat.hpp, 204
- delete_rengine
 - Geese, 114
- delete_rules
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 170
 - support-meat.hpp, 254
- delete_rules_dyn
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 170
 - support-meat.hpp, 254
- delete_support
 - Geese, 114

- delete_to_be_deleted
 - counters-meat.hpp, 202
- desc
 - Counter< Array_Type, Data_Type >, 87
 - counters-meat.hpp, 202
- desc_
 - counters-meat.hpp, 205
- directed
 - NetworkData, 132
- duplication
 - Node, 137
 - NodeData, 140
 - PhyloRuleDynData, 144
- else
 - support-meat.hpp, 254
- empty
 - PhyloCounterData, 141
- EmptyArray
 - PowerSet< Array_Type, Data_Rule_Type >, 150
- end
 - BArrayVector< Cell_Type, Data_Type >, 70
 - BArrayVector_const< Cell_Type, Data_Type >, 74
 - PhyloCounterData, 141
 - PowerSet< Array_Type, Data_Rule_Type >, 148
- Entries
 - Entries< Cell_Type >, 93
- Entries< Cell_Type >, 92
 - ~Entries, 93
 - Entries, 93
 - resize, 93
 - source, 93
 - target, 94
 - val, 94
- eval_rules_dyn
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 166
- EXISTS, 27
 - AS_ONE, 27
 - AS_ZERO, 27
 - BOTH, 27
 - NONE, 28
 - ONE, 28
 - TWO, 28
 - UNKNOWN, 28
- f_
 - statscounter-meat.hpp, 242
 - support-meat.hpp, 254
- Flock, 94
 - ~Flock, 96
 - add_data, 96
 - colnames, 96
 - dat, 100
 - Flock, 95
 - get_counters, 96
 - get_model, 97
 - get_support, 97
 - init, 97
 - initialized, 100
 - likelihood_joint, 97
 - model, 100
 - nfunctions, 100
 - nfuncs, 98
 - nleaves, 98
 - nnodes, 98
 - nterms, 98
 - ntrees, 98
 - operator(), 98
 - parse_polytomies, 99
 - print, 99
 - rengine, 100
 - set_seed, 99
 - support_size, 99
- flush_data
 - BArray< Cell_Type, Data_Type >, 34
- for
 - statscounter-meat.hpp, 240
 - support-meat.hpp, 249
- FreqTable
 - FreqTable< T >, 101
- FreqTable< T >, 101
 - ~FreqTable, 101
 - add, 102
 - as_vector, 102
 - clear, 102
 - FreqTable, 101
 - get_data, 102
 - get_data_ptr, 102
 - print, 102
 - reserve, 103
 - size, 103
- Geese, 103
 - ~Geese, 107
 - calc_reduced_sequence, 107
 - calc_sequence, 107
 - colnames, 107
 - delete_engine, 114
 - delete_support, 114
 - Geese, 106
 - get_annotated_nodes, 107
 - get_counters, 108
 - get_model, 108
 - get_probabilities, 108
 - get_rengine, 108
 - get_states, 108
 - get_support, 109
 - inherit_support, 109
 - init, 109
 - init_node, 109
 - initialized, 114
 - likelihood, 109
 - likelihood_exhaust, 110
 - map_to_nodes, 114
 - nannotations, 110
 - nfunctions, 114

- nfuns, [110](#)
 - nleafs, [110](#)
 - nnodes, [110](#)
 - nodes, [115](#)
 - nterms, [111](#)
 - observed_counts, [111](#)
 - operator=, [111](#)
 - parse_polytomies, [111](#)
 - predict, [111](#)
 - predict_backend, [112](#)
 - predict_exhaust, [112](#)
 - predict_exhaust_backend, [112](#)
 - predict_sim, [112](#)
 - print, [112](#)
 - print_observed_counts, [113](#)
 - reduced_sequence, [115](#)
 - sequence, [115](#)
 - set_seed, [113](#)
 - simulate, [113](#)
 - support_size, [113](#)
 - update_annotations, [113](#)
- geese-bones.hpp
 - INITIALIZED, [224](#)
 - keygen_full, [225](#)
 - RULE_FUNCTION, [225](#)
 - vec_diff, [225](#)
 - vector_caster, [225](#)
- gen_key
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [123](#)
- get_annotated_nodes
 - Geese, [107](#)
- get_cell
 - BArray< Cell_Type, Data_Type >, [34](#)
 - BArrayDense< Cell_Type, Data_Type >, [53](#)
- get_col_vec
 - BArray< Cell_Type, Data_Type >, [34](#), [35](#)
 - BArrayDense< Cell_Type, Data_Type >, [54](#)
- get_counters
 - Flock, [96](#)
 - Geese, [108](#)
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [123](#)
 - PhyloCounterData, [142](#)
 - StatsCounter< Array_Type, Data_Type >, [159](#)
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [166](#)
- get_counts
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [166](#)
- get_counts_ptr
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [166](#)
- get_current_stats
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [166](#)
- get_data
 - FreqTable< T >, [102](#)
 - PowerSet< Array_Type, Data_Rule_Type >, [148](#)
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [167](#)
- get_data_ptr
 - FreqTable< T >, [102](#)
 - PowerSet< Array_Type, Data_Rule_Type >, [148](#)
- get_description
 - Counter< Array_Type, Data_Type >, [85](#)
- get_descriptions
 - Counters< Array_Type, Data_Type >, [90](#)
 - StatsCounter< Array_Type, Data_Type >, [159](#)
- get_entries
 - BArray< Cell_Type, Data_Type >, [35](#)
 - BArrayDense< Cell_Type, Data_Type >, [54](#)
- get_last_name
 - phylo.hpp, [218](#)
- get_model
 - Flock, [97](#)
 - Geese, [108](#)
- get_name
 - Counter< Array_Type, Data_Type >, [85](#)
- get_names
 - Counters< Array_Type, Data_Type >, [90](#)
 - StatsCounter< Array_Type, Data_Type >, [159](#)
- get_norm_const
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [123](#)
- get_parent
 - Node, [135](#)
- get_probabilities
 - Geese, [108](#)
- get_pset
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [123](#)
- get_pset_stats
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [123](#)
- get_rengine
 - Geese, [108](#)
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [124](#)
- get_row_vec
 - BArray< Cell_Type, Data_Type >, [35](#)
 - BArrayDense< Cell_Type, Data_Type >, [54](#), [55](#)
- get_rules
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [166](#)

- 124
- Support< Array_Type, Data_Counter_Type,
Data_Rule_Type, Data_Rule_Dyn_Type >,
167
- get_rules_dyn
 - Model< Array_Type, Data_Counter_Type,
Data_Rule_Type, Data_Rule_Dyn_Type >,
124
 - Support< Array_Type, Data_Counter_Type,
Data_Rule_Type, Data_Rule_Dyn_Type >,
167
- get_seq
 - Rules< Array_Type, Data_Type >, 155
- get_states
 - Geese, 108
- get_support
 - Flock, 97
 - Geese, 109
 - Model< Array_Type, Data_Counter_Type,
Data_Rule_Type, Data_Rule_Dyn_Type >,
124
- i
 - counters-meat.hpp, 205
- id
 - Node, 137
- if
 - support-meat.hpp, 249
- include/barry/barray-bones.hpp, 173
- include/barry/barray-iterator.hpp, 174
- include/barry/barray-meat-operators.hpp, 175
- include/barry/barray-meat.hpp, 177
- include/barry/barraycell-bones.hpp, 178
- include/barry/barraycell-meat.hpp, 179
- include/barry/barraydense-bones.hpp, 180
- include/barry/barraydense-meet.hpp, 181
- include/barry/barraydensecell-bones.hpp, 183
- include/barry/barraydensecell-meat.hpp, 184
- include/barry/barrayvector-bones.hpp, 185
- include/barry/barrayvector-meat.hpp, 186
- include/barry/barray-configuration.hpp, 187
- include/barry/barray.hpp, 189
- include/barry/cell-bones.hpp, 191
- include/barry/cell-meat.hpp, 192
- include/barry/col-bones.hpp, 193
- include/barry/counters-bones.hpp, 193
- include/barry/counters-meat.hpp, 194
- include/barry/counters/network.hpp, 206
- include/barry/counters/phylo.hpp, 213
- include/barry/model-bones.hpp, 218
- include/barry/model-meat.hpp, 220
- include/barry/models/geese.hpp, 222
- include/barry/models/geese/flock-bones.hpp, 223
- include/barry/models/geese/flock-meet.hpp, 223
- include/barry/models/geese/geese-bones.hpp, 224
- include/barry/models/geese/geese-meat-constructors.hpp, 226
- include/barry/models/geese/geese-meat-likelihood.hpp,
226
- include/barry/models/geese/geese-meat-likelihood_exhaust.hpp,
227
- include/barry/models/geese/geese-meat-predict.hpp,
228
- include/barry/models/geese/geese-meat-predict_exhaust.hpp,
228
- include/barry/models/geese/geese-meat-predict_sim.hpp,
229
- include/barry/models/geese/geese-meat-simulate.hpp,
229
- include/barry/models/geese/geese-meat.hpp, 230
- include/barry/models/geese/geese-node-bones.hpp,
230
- include/barry/powerset-bones.hpp, 231
- include/barry/powerset-meat.hpp, 232
- include/barry/rules-bones.hpp, 234
- include/barry/rules-meat.hpp, 235
- include/barry/statscounter-bones.hpp, 236
- include/barry/statscounter-meat.hpp, 237
- include/barry/statsdb.hpp, 243
- include/barry/support-bones.hpp, 244
- include/barry/support-meat.hpp, 245
- include/barry/typedefs.hpp, 256
- indices
 - NetCounterData, 130
- inherit_support
 - Geese, 109
- init
 - Counter< Array_Type, Data_Type >, 86
 - Flock, 97
 - Geese, 109
- init_fun
 - Counter< Array_Type, Data_Type >, 87
 - counters-meat.hpp, 202, 203
- init_fun_
 - counters-meat.hpp, 205
- init_node
 - Geese, 109
- init_support
 - PowerSet< Array_Type, Data_Rule_Type >, 148
 - Support< Array_Type, Data_Counter_Type,
Data_Rule_Type, Data_Rule_Dyn_Type >,
167
- INITIALIZED
 - geese-bones.hpp, 224
- initialized
 - Flock, 100
 - Geese, 114
- insert_cell
 - BArray< Cell_Type, Data_Type >, 35, 36
 - BArrayDense< Cell_Type, Data_Type >, 55
 - support-meat.hpp, 249
- is_col
 - BArrayVector< Cell_Type, Data_Type >, 70
 - BArrayVector_const< Cell_Type, Data_Type >, 74
- is_empty
 - BArray< Cell_Type, Data_Type >, 36
 - BArrayDense< Cell_Type, Data_Type >, 56

- is_leaf
 - Node, [136](#)
- is_row
 - BArrayVector< Cell_Type, Data_Type >, [71](#)
 - BArrayVector_const< Cell_Type, Data_Type >, [74](#)
- iter
 - ConstBArrayRowIter< Cell_Type, Data_Type >, [82](#)
- j
 - counters-meat.hpp, [205](#)
 - statscounter-meat.hpp, [242](#)
- keygen_default
 - model-bones.hpp, [219](#)
- keygen_full
 - geese-bones.hpp, [225](#)
- lb
 - PhyloRuleDynData, [144](#)
- likelihood
 - Geese, [109](#)
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [124](#), [125](#)
- likelihood_
 - model-meat.hpp, [221](#)
- likelihood_exhaust
 - Geese, [110](#)
- likelihood_joint
 - Flock, [97](#)
- likelihood_total
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [125](#)
- M
 - PowerSet< Array_Type, Data_Rule_Type >, [150](#)
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [170](#)
- Map
 - barry-configuration.hpp, [188](#)
- map_to_nodes
 - Geese, [114](#)
- MapVec_type
 - typedefs.hpp, [258](#)
- max_num_elements
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [170](#)
- Model
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [119](#)
- model
 - Flock, [100](#)
- Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [115](#)
- ~Model, [119](#)
- add_array, [120](#)
- add_counter, [120](#)
- add_rule, [121](#)
- add_rule_dyn, [121](#), [122](#)
- colnames, [122](#)
- conditional_prob, [122](#)
- gen_key, [123](#)
- get_counters, [123](#)
- get_norm_const, [123](#)
- get_pset, [123](#)
- get_pset_stats, [123](#)
- get_engine, [124](#)
- get_rules, [124](#)
- get_rules_dyn, [124](#)
- get_support, [124](#)
- likelihood, [124](#), [125](#)
- likelihood_total, [125](#)
- Model, [119](#)
- nterms, [125](#)
- operator=, [126](#)
- print, [126](#)
- print_stats, [126](#)
- sample, [126](#)
- set_counters, [127](#)
- set_keygen, [127](#)
- set_engine, [127](#)
- set_rules, [127](#)
- set_rules_dyn, [127](#)
- set_seed, [128](#)
- size, [128](#)
- size_unique, [128](#)
- store_psets, [128](#)
- support_size, [128](#)
- model-bones.hpp
 - keygen_default, [219](#)
- model-meat.hpp
 - likelihood_, [221](#)
 - MODEL_TEMPLATE, [220](#), [221](#)
 - MODEL_TEMPLATE_ARGS, [220](#)
 - MODEL_TYPE, [221](#)
 - update_normalizing_constant, [222](#)
- MODEL_TEMPLATE
 - model-meat.hpp, [220](#), [221](#)
- MODEL_TEMPLATE_ARGS
 - model-meat.hpp, [220](#)
- MODEL_TYPE
 - model-meat.hpp, [221](#)
- N
 - PowerSet< Array_Type, Data_Rule_Type >, [150](#)
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [171](#)
- name
 - Counter< Array_Type, Data_Type >, [87](#)
 - counters-meat.hpp, [203](#)
 - name_
 - counters-meat.hpp, [205](#)
- nannotations

- Geese, 110
- narray
 - Node, 137
- ncol
 - BArray< Cell_Type, Data_Type >, 36
 - BArrayDense< Cell_Type, Data_Type >, 56
- NET_C_DATA_IDX
 - network.hpp, 209
- NET_C_DATA_NUM
 - network.hpp, 209
- NetCounter
 - network.hpp, 211
- NetCounterData, 129
 - ~NetCounterData, 130
 - indices, 130
 - NetCounterData, 129
 - numbers, 130
- NetCounters
 - network.hpp, 211
- NetModel
 - network.hpp, 211
- NetRule
 - network.hpp, 211
- NetRules
 - network.hpp, 211
- NetStatsCounter
 - network.hpp, 212
- NetSupport
 - network.hpp, 212
- Network
 - network.hpp, 212
- Network counters, 10
 - counter_absdiff, 13
 - counter_css_completely_false_recip_comiss, 13
 - counter_css_completely_false_recip_omiss, 13
 - counter_css_partially_false_recip_commi, 13
 - counter_css_partially_false_recip_omiss, 14
 - counter_ctriads, 14
 - counter_degree, 14
 - counter_density, 14
 - counter_diff, 14
 - counter_edges, 15
 - counter_iddegree, 15
 - counter_iddegree15, 15
 - counter_isolates, 15
 - counter_istar2, 15
 - counter_mutual, 16
 - counter_nodecov, 16
 - counter_nodeicov, 16
 - counter_nodematch, 16
 - counter_nodeocov, 16
 - counter_odegree, 17
 - counter_odegree15, 17
 - counter_ostar2, 17
 - counter_ttriads, 17
 - CSS_APPEND, 11
 - CSS_CASE_ELSE, 11
 - CSS_CASE_PERCEIVED, 12
 - CSS_CASE_TRUTH, 12
 - CSS_CHECK_SIZE, 12
 - CSS_CHECK_SIZE_INIT, 12
 - CSS_SIZE, 12
 - NETWORK_COUNTER, 17
- network.hpp
 - NET_C_DATA_IDX, 209
 - NET_C_DATA_NUM, 209
 - NetCounter, 211
 - NetCounters, 211
 - NetModel, 211
 - NetRule, 211
 - NetRules, 211
 - NetStatsCounter, 212
 - NetSupport, 212
 - Network, 212
 - NETWORK_COUNTER, 210
 - NETWORK_COUNTER_LAMBDA, 210
 - NETWORK_RULE, 210
 - NETWORK_RULE_LAMBDA, 210
 - rules_zerodiag, 212
- NETWORK_COUNTER
 - Network counters, 17
 - network.hpp, 210
- NETWORK_COUNTER_LAMBDA
 - network.hpp, 210
- NETWORK_RULE
 - network.hpp, 210
- NETWORK_RULE_LAMBDA
 - network.hpp, 210
- NetworkData, 130
 - ~NetworkData, 132
 - directed, 132
 - NetworkData, 131, 132
 - vertex_attr, 132
- nfunctions
 - Flock, 100
 - Geese, 114
- nfuncs
 - Flock, 98
 - Geese, 110
- nleafs
 - Flock, 98
 - Geese, 110
- nnodes
 - Flock, 98
 - Geese, 110
- nnozero
 - BArray< Cell_Type, Data_Type >, 36
 - BArrayDense< Cell_Type, Data_Type >, 56
- Node, 133
 - ~Node, 135
 - annotations, 136
 - array, 136
 - arrays, 136
 - duplication, 137
 - get_parent, 135
 - id, 137

- is_leaf, [136](#)
- narray, [137](#)
- Node, [134](#), [135](#)
- noffspring, [136](#)
- offspring, [137](#)
- ord, [137](#)
- parent, [138](#)
- probability, [138](#)
- subtree_prob, [138](#)
- visited, [138](#)
- NodeData, [139](#)
 - blengths, [139](#)
 - duplication, [140](#)
 - NodeData, [139](#)
 - states, [140](#)
- nodes
 - Geese, [115](#)
- noexcept
 - counters-meat.hpp, [206](#)
- noffspring
 - Node, [136](#)
- NONE
 - CHECK, [26](#)
 - EXISTS, [28](#)
- nrow
 - BArray< Cell_Type, Data_Type >, [37](#)
 - BArrayDense< Cell_Type, Data_Type >, [56](#)
- nterms
 - Flock, [98](#)
 - Geese, [111](#)
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [125](#)
- ntrees
 - Flock, [98](#)
- numbers
 - NetCounterData, [130](#)
- observed_counts
 - Geese, [111](#)
- offspring
 - Node, [137](#)
- ONE
 - CHECK, [26](#)
 - EXISTS, [28](#)
- operator Cell_Type
 - BArrayCell< Cell_Type, Data_Type >, [44](#)
 - BArrayCell_const< Cell_Type, Data_Type >, [46](#)
 - BArrayDenseCell< Cell_Type, Data_Type >, [64](#)
 - BArrayDenseCell_const< Cell_Type, Data_Type >, [67](#)
 - Cell< Cell_Type >, [79](#)
- operator std::vector< Cell_Type >
 - BArrayVector< Cell_Type, Data_Type >, [71](#)
 - BArrayVector_const< Cell_Type, Data_Type >, [74](#)
- operator!=
 - BArrayCell_const< Cell_Type, Data_Type >, [47](#)
 - BArrayDenseCell_const< Cell_Type, Data_Type >, [67](#)
 - BArrayVector_const< Cell_Type, Data_Type >, [74](#)
 - Cell< Cell_Type >, [79](#)
 - BArrayCell_const< Cell_Type, Data_Type >, [47](#)
 - BArrayDenseCell_const< Cell_Type, Data_Type >, [67](#)
 - BArrayVector_const< Cell_Type, Data_Type >, [75](#)
- operator<=
 - BArrayCell_const< Cell_Type, Data_Type >, [47](#)
 - BArrayDenseCell_const< Cell_Type, Data_Type >, [68](#)
 - BArrayVector_const< Cell_Type, Data_Type >, [75](#)
- operator<
 - BArrayCell_const< Cell_Type, Data_Type >, [47](#)
 - BArrayDenseCell_const< Cell_Type, Data_Type >, [67](#)
 - BArrayVector_const< Cell_Type, Data_Type >, [75](#)
- operator>=
 - BArrayCell_const< Cell_Type, Data_Type >, [48](#)
 - BArrayDenseCell_const< Cell_Type, Data_Type >, [68](#)
 - BArrayVector_const< Cell_Type, Data_Type >, [75](#)
- operator*=
 - BArray< Cell_Type, Data_Type >, [37](#)
 - BArrayCell< Cell_Type, Data_Type >, [44](#)
 - BArrayDense< Cell_Type, Data_Type >, [57](#)
 - BArrayDenseCell< Cell_Type, Data_Type >, [65](#)
 - BArrayVector< Cell_Type, Data_Type >, [71](#)
- operator()
 - BArray< Cell_Type, Data_Type >, [37](#)
 - BArrayDense< Cell_Type, Data_Type >, [56](#), [57](#)
 - Flock, [98](#)
 - PhyloCounterData, [142](#)
 - Rule< Array_Type, Data_Type >, [153](#)
 - Rules< Array_Type, Data_Type >, [156](#)
 - vecHasher< T >, [171](#)
- operator+=
 - BArray< Cell_Type, Data_Type >, [37](#), [38](#)
 - BArrayCell< Cell_Type, Data_Type >, [44](#)
 - BArrayDense< Cell_Type, Data_Type >, [57](#)
 - BArrayDenseCell< Cell_Type, Data_Type >, [65](#)
 - BArrayVector< Cell_Type, Data_Type >, [71](#)
- operator-=
 - BArray< Cell_Type, Data_Type >, [38](#)
 - BArrayCell< Cell_Type, Data_Type >, [44](#)
 - BArrayDense< Cell_Type, Data_Type >, [58](#)
 - BArrayDenseCell< Cell_Type, Data_Type >, [65](#)
 - BArrayVector< Cell_Type, Data_Type >, [71](#)
- operator/=
 - BArray< Cell_Type, Data_Type >, [38](#)
 - BArrayCell< Cell_Type, Data_Type >, [45](#)
 - BArrayDense< Cell_Type, Data_Type >, [58](#)
 - BArrayDenseCell< Cell_Type, Data_Type >, [65](#)
 - BArrayVector< Cell_Type, Data_Type >, [72](#)
- operator=
 - BArray< Cell_Type, Data_Type >, [38](#), [39](#)
 - BArrayCell< Cell_Type, Data_Type >, [45](#)
 - BArrayDense< Cell_Type, Data_Type >, [58](#), [59](#)

- BArrayDenseCell< Cell_Type, Data_Type >, 65
- BArrayVector< Cell_Type, Data_Type >, 72
- Cell< Cell_Type >, 79, 80
- Counter< Array_Type, Data_Type >, 86
- Counters< Array_Type, Data_Type >, 90, 91
- Geese, 111
- Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 126
- Rules< Array_Type, Data_Type >, 156
- operator==
 - BArray< Cell_Type, Data_Type >, 39
 - BArrayCell< Cell_Type, Data_Type >, 45
 - BArrayCell_const< Cell_Type, Data_Type >, 47
 - BArrayDense< Cell_Type, Data_Type >, 59
 - BArrayDenseCell< Cell_Type, Data_Type >, 66
 - BArrayDenseCell_const< Cell_Type, Data_Type >, 68
 - BArrayVector< Cell_Type, Data_Type >, 72
 - BArrayVector_const< Cell_Type, Data_Type >, 75
 - Cell< Cell_Type >, 80
- operator[]
 - Counters< Array_Type, Data_Type >, 91
 - PowerSet< Array_Type, Data_Rule_Type >, 149
- ord
 - Node, 137
- out_of_range
 - BArray< Cell_Type, Data_Type >, 39
 - BArrayDense< Cell_Type, Data_Type >, 59
- parent
 - Node, 138
- parse_polytomies
 - Flock, 99
 - Geese, 111
- Phylo counters, 18
 - counter_co_opt, 19
 - counter_cogain, 19
 - counter_gains, 19
 - counter_gains_k_offspring, 20
 - counter_genes_changing, 20
 - counter_longest, 20
 - counter_loss, 20
 - counter_maxfuns, 21
 - counter_neofun, 21
 - counter_neofun_a2b, 21
 - counter_overall_changes, 21
 - counter_overall_gains, 22
 - counter_overall_loss, 22
 - counter_prop_genes_changing, 22
 - counter_subfun, 22
- Phylo rules, 23
 - rule_dyn_limit_changes, 23
- phylo.hpp
 - get_last_name, 218
 - PHYLO_CHECK_MISSING, 215
 - PHYLO_COUNTER_LAMBDA, 215
 - PHYLO_RULE_DYN_LAMBDA, 215
 - PhyloArray, 216
 - PhyloCounter, 216
 - PhyloCounters, 216
 - PhyloModel, 216
 - PhyloPowerSet, 216
 - PhyloRule, 217
 - PhyloRuleData, 217
 - PhyloRuleDyn, 217
 - PhyloRules, 217
 - PhyloRulesDyn, 217
 - PhyloStatsCounter, 217
 - PhyloSupport, 218
 - PHYLO_CHECK_MISSING
 - phylo.hpp, 215
 - PHYLO_COUNTER_LAMBDA
 - phylo.hpp, 215
 - PHYLO_RULE_DYN_LAMBDA
 - phylo.hpp, 215
 - PhyloArray
 - phylo.hpp, 216
 - PhyloCounter
 - phylo.hpp, 216
 - PhyloCounterData, 140
 - at, 141
 - begin, 141
 - empty, 141
 - end, 141
 - get_counters, 142
 - operator(), 142
 - PhyloCounterData, 141
 - push_back, 142
 - reserve, 142
 - shrink_to_fit, 142
 - size, 142
 - PhyloCounters
 - phylo.hpp, 216
 - PhyloModel
 - phylo.hpp, 216
 - PhyloPowerSet
 - phylo.hpp, 216
 - PhyloRule
 - phylo.hpp, 217
 - PhyloRuleData
 - phylo.hpp, 217
 - PhyloRuleDyn
 - phylo.hpp, 217
 - PhyloRuleDynData, 143
 - ~PhyloRuleDynData, 143
 - counts, 144
 - duplication, 144
 - lb, 144
 - PhyloRuleDynData, 143
 - pos, 144
 - ub, 144
 - PhyloRules
 - phylo.hpp, 217
 - PhyloRulesDyn
 - phylo.hpp, 217
 - PhyloStatsCounter

- phylo.hpp, 217
- PhyloSupport
 - phylo.hpp, 218
- POS
 - barraydense-meet.hpp, 183
 - barraydensecell-meet.hpp, 185
- pos
 - PhyloRuleDynData, 144
- PowerSet
 - PowerSet< Array_Type, Data_Rule_Type >, 146
- PowerSet< Array_Type, Data_Rule_Type >, 145
 - ~PowerSet, 147
 - add_rule, 147
 - begin, 147
 - calc, 148
 - coordinates_free, 149
 - coordinates_locked, 149
 - data, 150
 - EmptyArray, 150
 - end, 148
 - get_data, 148
 - get_data_ptr, 148
 - init_support, 148
 - M, 150
 - N, 150
 - operator[], 149
 - PowerSet, 146
 - reset, 149
 - rules, 150
 - rules_deleted, 151
 - size, 149
- predict
 - Geese, 111
- predict_backend
 - Geese, 112
- predict_exhaust
 - Geese, 112
- predict_exhaust_backend
 - Geese, 112
- predict_sim
 - Geese, 112
- print
 - BArray< Cell_Type, Data_Type >, 39
 - BArrayDense< Cell_Type, Data_Type >, 59
 - Flock, 99
 - FreqTable< T >, 102
 - Geese, 112
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 126
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 167
- print_observed_counts
 - Geese, 113
- print_stats
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 126
- printf_barry
 - barry-configuration.hpp, 188
- probability
 - Node, 138
- push_back
 - counters-meet.hpp, 203
 - PhyloCounterData, 142
- README.md, 260
- reduced_sequence
 - Geese, 115
- engine
 - Flock, 100
- reserve
 - BArray< Cell_Type, Data_Type >, 39
 - BArrayDense< Cell_Type, Data_Type >, 59
 - FreqTable< T >, 103
 - PhyloCounterData, 142
- reset
 - PowerSet< Array_Type, Data_Rule_Type >, 149
- reset_array
 - StatsCounter< Array_Type, Data_Type >, 160
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 168
- resize
 - BArray< Cell_Type, Data_Type >, 39
 - BArrayDense< Cell_Type, Data_Type >, 60
 - Entries< Cell_Type >, 93
 - statscounter-meet.hpp, 240
- return
 - counters-meet.hpp, 206
 - statscounter-meet.hpp, 243
 - support-meet.hpp, 255
- rm_cell
 - BArray< Cell_Type, Data_Type >, 40
 - BArrayDense< Cell_Type, Data_Type >, 60
 - support-meet.hpp, 249
- ROW
 - barray-meet-operators.hpp, 176
 - barray-meet.hpp, 178
 - barraydense-meet.hpp, 183
- row
 - BArray< Cell_Type, Data_Type >, 40
 - BArrayDense< Cell_Type, Data_Type >, 60
- Row_type
 - typedefs.hpp, 258
- Rule
 - Rule< Array_Type, Data_Type >, 152
- Rule< Array_Type, Data_Type >, 151
 - ~Rule, 152
 - D, 153
 - operator(), 153
 - Rule, 152
- rule_dyn_limit_changes
 - Phylo rules, 23
- rule_fun_default
 - rules-bones.hpp, 235

- Rule_fun_type
 - typedefs.hpp, [259](#)
- RULE_FUNCTION
 - barry.hpp, [191](#)
 - geese-bones.hpp, [225](#)
- RULE_LAMBDA
 - barry.hpp, [191](#)
- Rules
 - Rules< Array_Type, Data_Type >, [154](#)
- rules
 - PowerSet< Array_Type, Data_Rule_Type >, [150](#)
 - support-meat.hpp, [255](#)
- Rules< Array_Type, Data_Type >, [153](#)
 - ~Rules, [154](#)
 - add_rule, [155](#)
 - clear, [155](#)
 - get_seq, [155](#)
 - operator(), [156](#)
 - operator=, [156](#)
 - Rules, [154](#)
 - size, [156](#)
- rules-bones.hpp
 - rule_fun_default, [235](#)
- rules_
 - support-meat.hpp, [255](#)
- rules_deleted
 - PowerSet< Array_Type, Data_Rule_Type >, [151](#)
- rules_dyn
 - support-meat.hpp, [255](#)
- rules_zerodiag
 - network.hpp, [212](#)
- sample
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [126](#)
- sequence
 - Geese, [115](#)
- set_counters
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [127](#)
 - StatsCounter< Array_Type, Data_Type >, [160](#)
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [168](#)
- set_data
 - BArray< Cell_Type, Data_Type >, [40](#)
 - BArrayDense< Cell_Type, Data_Type >, [60](#)
- set_keygen
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [127](#)
- set_engine
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [127](#)
- set_rules
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [127](#)
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [168](#)
- set_rules_dyn
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [127](#)
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [168](#)
- set_seed
 - Flock, [99](#)
 - Geese, [113](#)
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [128](#)
- shrink_to_fit
 - PhyloCounterData, [142](#)
- simulate
 - Geese, [113](#)
- size
 - BArrayVector< Cell_Type, Data_Type >, [72](#)
 - BArrayVector_const< Cell_Type, Data_Type >, [76](#)
 - Counters< Array_Type, Data_Type >, [91](#)
 - FreqTable< T >, [103](#)
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [128](#)
 - PhyloCounterData, [142](#)
 - PowerSet< Array_Type, Data_Rule_Type >, [149](#)
 - Rules< Array_Type, Data_Type >, [156](#)
- size_unique
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, [128](#)
- source
 - Entries< Cell_Type >, [93](#)
- states
 - NodeData, [140](#)
- Statistical Models, [9](#)
- stats_bank
 - support-meat.hpp, [255](#)
- StatsCounter
 - StatsCounter< Array_Type, Data_Type >, [158](#)
- StatsCounter< Array_Type, Data_Type >, [157](#)
 - ~StatsCounter, [158](#)
 - add_counter, [158](#)
 - count_all, [159](#)
 - count_current, [159](#)
 - count_init, [159](#)
 - get_counters, [159](#)
 - get_descriptions, [159](#)
 - get_names, [159](#)
 - reset_array, [160](#)
 - set_counters, [160](#)

- StatsCounter, 158
- statscounter-meat.hpp
 - counter_deleted, 242
 - counters, 242
 - counters_, 242
 - f_, 242
 - for, 240
 - j, 242
 - resize, 240
 - return, 243
 - STATSCOUNTER_TEMPLATE, 239–241
 - STATSCOUNTER_TEMPLATE_ARGS, 239
 - STATSCOUNTER_TYPE, 239
- STATSCOUNTER_TEMPLATE
 - statscounter-meat.hpp, 239–241
- STATSCOUNTER_TEMPLATE_ARGS
 - statscounter-meat.hpp, 239
- STATSCOUNTER_TYPE
 - statscounter-meat.hpp, 239
- store_psets
 - Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 128
- subtree_prob
 - Node, 138
- Support
 - Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 162, 163
- Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 160
 - ~Support, 163
 - add_counter, 163
 - add_rule, 164
 - add_rule_dyn, 164
 - calc, 164
 - change_stats, 169
 - coordinates_free, 169
 - coordinates_locked, 169
 - current_stats, 169
 - delete_counters, 170
 - delete_rules, 170
 - delete_rules_dyn, 170
 - eval_rules_dyn, 166
 - get_counters, 166
 - get_counts, 166
 - get_counts_ptr, 166
 - get_current_stats, 166
 - get_data, 167
 - get_rules, 167
 - get_rules_dyn, 167
 - init_support, 167
 - M, 170
 - max_num_elements, 170
 - N, 171
 - print, 167
 - reset_array, 168
 - set_counters, 168
 - set_rules, 168
 - set_rules_dyn, 168
 - Support, 162, 163
- support-meat.hpp
 - array_bank, 253
 - BARRY_SUPPORT_MEAT_HPP, 247
 - calc_backend, 248
 - cfree, 253
 - counters, 253
 - counters_, 253
 - delete_counters, 254
 - delete_rules, 254
 - delete_rules_dyn, 254
 - else, 254
 - f_, 254
 - for, 249
 - if, 249
 - insert_cell, 249
 - return, 255
 - rm_cell, 249
 - rules, 255
 - rules_, 255
 - rules_dyn, 255
 - stats_bank, 255
 - SUPPORT_TEMPLATE, 248, 250–253
 - SUPPORT_TEMPLATE_ARGS, 248
 - SUPPORT_TYPE, 248
- support_size
 - Flock, 99
 - Geese, 113
- Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >, 128
 - SUPPORT_TEMPLATE
 - support-meat.hpp, 248, 250–253
 - SUPPORT_TEMPLATE_ARGS
 - support-meat.hpp, 248
 - SUPPORT_TYPE
 - support-meat.hpp, 248
- swap_cells
 - BArray< Cell_Type, Data_Type >, 40
 - BArrayDense< Cell_Type, Data_Type >, 61
- swap_cols
 - BArray< Cell_Type, Data_Type >, 41
 - BArrayDense< Cell_Type, Data_Type >, 61
- swap_rows
 - BArray< Cell_Type, Data_Type >, 41
 - BArrayDense< Cell_Type, Data_Type >, 61
- target
 - Entries< Cell_Type >, 94
- to_be_deleted
 - counters-meat.hpp, 203
- toggle_cell
 - BArray< Cell_Type, Data_Type >, 41
 - BArrayDense< Cell_Type, Data_Type >, 61
- toggle_lock
 - BArray< Cell_Type, Data_Type >, 41
 - BArrayDense< Cell_Type, Data_Type >, 62

transpose
 BArray< Cell_Type, Data_Type >, [41](#)
 BArrayDense< Cell_Type, Data_Type >, [62](#)

TWO
 CHECK, [26](#)
 EXISTS, [28](#)

typedefs.hpp
 Col_type, [258](#)
 Counter_fun_type, [258](#)
 Counts_type, [258](#)
 MapVec_type, [258](#)
 Row_type, [258](#)
 Rule_fun_type, [259](#)
 uint, [259](#)
 vec_equal, [259](#)
 vec_equal_approx, [259](#)
 vec_inner_prod, [260](#)

ub
 PhyloRuleDynData, [144](#)

uint
 typedefs.hpp, [259](#)

UNKNOWN
 EXISTS, [28](#)

update_annotations
 Geese, [113](#)

update_normalizing_constant
 model-meat.hpp, [222](#)

val
 Entries< Cell_Type >, [94](#)

value
 Cell< Cell_Type >, [80](#)

vec_diff
 geese-bones.hpp, [225](#)

vec_equal
 typedefs.hpp, [259](#)

vec_equal_approx
 typedefs.hpp, [259](#)

vec_inner_prod
 typedefs.hpp, [260](#)

vecHasher< T >, [171](#)
 operator(), [171](#)

vector_caster
 geese-bones.hpp, [225](#)

vertex_attr
 NetworkData, [132](#)

visited
 BArray< Cell_Type, Data_Type >, [42](#)
 BArrayDense< Cell_Type, Data_Type >, [63](#)
 Cell< Cell_Type >, [80](#)
 Node, [138](#)

ZERO_CELL
 barraydense-meet.hpp, [183](#)

zero_col
 BArray< Cell_Type, Data_Type >, [42](#)
 BArrayDense< Cell_Type, Data_Type >, [62](#)

zero_row