

barry: Your go-to motif accountant

0.0-1

Generated by Doxygen 1.9.1



<b>1 Main Page</b>	<b>1</b>
<b>2 Module Index</b>	<b>5</b>
2.1 Modules	5
<b>3 Class Index</b>	<b>7</b>
3.1 Class List	7
<b>4 File Index</b>	<b>9</b>
4.1 File List	9
<b>5 Module Documentation</b>	<b>11</b>
5.1 Counting	11
5.1.1 Detailed Description	11
5.2 Statistical Models	11
5.2.1 Detailed Description	12
5.3 Network counters	12
5.3.1 Detailed Description	13
5.3.2 Function Documentation	13
5.3.2.1 counter_absdiff()	13
5.3.2.2 counter_ctriads()	13
5.3.2.3 counter_degree()	14
5.3.2.4 counter_density()	14
5.3.2.5 counter_diff()	14
5.3.2.6 counter_edges()	14
5.3.2.7 counter_idegree()	15
5.3.2.8 counter_idegree15()	15
5.3.2.9 counter_isolates()	15
5.3.2.10 counter_istar2()	15
5.3.2.11 counter_mutual()	15
5.3.2.12 counter_nodecov()	16
5.3.2.13 counter_nodeicov()	16
5.3.2.14 counter_nodematch()	16
5.3.2.15 counter_nodeocov()	16
5.3.2.16 counter_odegree()	16
5.3.2.17 counter_odegree15()	17
5.3.2.18 counter_ostar2()	17
5.3.2.19 counter_ttriads()	17
5.3.2.20 NETWORK_COUNTER()	17
5.4 Phylo counters	17
5.4.1 Detailed Description	18
5.4.2 Function Documentation	18
5.4.2.1 counter_co_opt()	19
5.4.2.2 counter_cogain()	19

5.4.2.3 counter_gains()	19
5.4.2.4 counter_gains_k_offspring()	20
5.4.2.5 counter_genes_changing()	20
5.4.2.6 counter_k_genes_changing()	20
5.4.2.7 counter_longest()	20
5.4.2.8 counter_loss()	21
5.4.2.9 counter_maxfun()	21
5.4.2.10 counter_neofun()	21
5.4.2.11 counter_neofun_a2b()	21
5.4.2.12 counter_overall_changes()	22
5.4.2.13 counter_overall_gains()	22
5.4.2.14 counter_overall_loss()	22
5.4.2.15 counter_prop_genes_changing()	22
5.4.2.16 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
<b>6 Namespace Documentation</b>	<b>25</b>
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
<b>7 Class Documentation</b>	<b>29</b>
7.1 BArray< Cell_Type, Data_Type > Class Template Reference	29
7.1.1 Detailed Description	31
7.1.2 Constructor & Destructor Documentation	32
7.1.2.1 BArray() [1/6]	32
7.1.2.2 BArray() [2/6]	32
7.1.2.3 BArray() [3/6]	32
7.1.2.4 BArray() [4/6]	33
7.1.2.5 BArray() [5/6]	33
7.1.2.6 BArray() [6/6]	33
7.1.2.7 ~BArray()	33
7.1.3 Member Function Documentation	33
7.1.3.1 clear()	33
7.1.3.2 col()	34
7.1.3.3 D() [1/2]	34
7.1.3.4 D() [2/2]	34
7.1.3.5 default_val()	34
7.1.3.6 flush_data()	34
7.1.3.7 get_cell()	34
7.1.3.8 get_col_vec() [1/2]	35
7.1.3.9 get_col_vec() [2/2]	35
7.1.3.10 get_entries()	35
7.1.3.11 get_row_vec() [1/2]	35
7.1.3.12 get_row_vec() [2/2]	35
7.1.3.13 insert_cell() [1/3]	36
7.1.3.14 insert_cell() [2/3]	36
7.1.3.15 insert_cell() [3/3]	36
7.1.3.16 is_empty()	36
7.1.3.17 ncol()	36
7.1.3.18 nnozero()	37
7.1.3.19 nrow()	37
7.1.3.20 operator>() [1/2]	37
7.1.3.21 operator>() [2/2]	37
7.1.3.22 operator*=( )	37
7.1.3.23 operator+=( ) [1/3]	37
7.1.3.24 operator+=( ) [2/3]	38
7.1.3.25 operator+=( ) [3/3]	38
7.1.3.26 operator-=( ) [1/3]	38
7.1.3.27 operator-=( ) [2/3]	38
7.1.3.28 operator-=( ) [3/3]	38

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

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

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



7.6.1 Detailed Description . . . . .	65
7.6.2 Constructor & Destructor Documentation . . . . .	65
7.6.2.1 BArrayDenseCell_const() . . . . .	65
7.6.2.2 ~BArrayDenseCell_const() . . . . .	65
7.6.3 Member Function Documentation . . . . .	65
7.6.3.1 operator Cell_Type() . . . . .	66
7.6.3.2 operator!=(()) . . . . .	66
7.6.3.3 operator<() . . . . .	66
7.6.3.4 operator<=() . . . . .	66
7.6.3.5 operator==(()) . . . . .	66
7.6.3.6 operator>() . . . . .	67
7.6.3.7 operator>=() . . . . .	67
7.7 BArrayRow< Cell_Type, Data_Type > Class Template Reference . . . . .	67
7.7.1 Detailed Description . . . . .	67
7.7.2 Constructor & Destructor Documentation . . . . .	68
7.7.2.1 BArrayRow() . . . . .	68
7.7.2.2 ~BArrayRow() . . . . .	68
7.7.3 Member Function Documentation . . . . .	68
7.7.3.1 operator BArrayRow< Cell_Type, Data_Type >() . . . . .	68
7.7.3.2 operator*=(()) . . . . .	68
7.7.3.3 operator+=(()) . . . . .	68
7.7.3.4 operator-=(()) . . . . .	69
7.7.3.5 operator/=(()) . . . . .	69
7.7.3.6 operator=() . . . . .	69
7.7.3.7 operator==(()) . . . . .	69
7.8 BArrayRow_const< Cell_Type, Data_Type > Class Template Reference . . . . .	69
7.8.1 Detailed Description . . . . .	70
7.8.2 Constructor & Destructor Documentation . . . . .	70
7.8.2.1 BArrayRow_const() . . . . .	70
7.8.2.2 ~BArrayRow_const() . . . . .	70
7.8.3 Member Function Documentation . . . . .	70
7.8.3.1 operator BArrayRow_const< Cell_Type, Data_Type >() . . . . .	70
7.8.3.2 operator!=(()) . . . . .	70
7.8.3.3 operator<() . . . . .	71
7.8.3.4 operator<=() . . . . .	71
7.8.3.5 operator==(()) . . . . .	71
7.8.3.6 operator>() . . . . .	71
7.8.3.7 operator>=() . . . . .	71
7.9 BArrayVector< Cell_Type, Data_Type > Class Template Reference . . . . .	71
7.9.1 Detailed Description . . . . .	72
7.9.2 Constructor & Destructor Documentation . . . . .	72
7.9.2.1 BArrayVector() . . . . .	72

7.9.2.2 ~BArrayVector()	73
7.9.3 Member Function Documentation	73
7.9.3.1 begin()	73
7.9.3.2 end()	73
7.9.3.3 is_col()	73
7.9.3.4 is_row()	74
7.9.3.5 operator std::vector< Cell_Type >()	74
7.9.3.6 operator*=(())	74
7.9.3.7 operator+=(())	74
7.9.3.8 operator-=(())	74
7.9.3.9 operator/=(())	75
7.9.3.10 operator=()	75
7.9.3.11 operator==(())	75
7.9.3.12 size()	75
7.10 BArrayVector_const< Cell_Type, Data_Type > Class Template Reference	75
7.10.1 Detailed Description	76
7.10.2 Constructor & Destructor Documentation	76
7.10.2.1 BArrayVector_const()	76
7.10.2.2 ~BArrayVector_const()	76
7.10.3 Member Function Documentation	77
7.10.3.1 begin()	77
7.10.3.2 end()	77
7.10.3.3 is_col()	77
7.10.3.4 is_row()	77
7.10.3.5 operator std::vector< Cell_Type >()	77
7.10.3.6 operator!=(())	78
7.10.3.7 operator<()	78
7.10.3.8 operator<=()	78
7.10.3.9 operator==(())	78
7.10.3.10 operator>()	78
7.10.3.11 operator>=()	79
7.10.3.12 size()	79
7.11 Cell< Cell_Type > Class Template Reference	79
7.11.1 Detailed Description	80
7.11.2 Constructor & Destructor Documentation	80
7.11.2.1 Cell() [1/7]	80
7.11.2.2 Cell() [2/7]	80
7.11.2.3 ~Cell()	80
7.11.2.4 Cell() [3/7]	81
7.11.2.5 Cell() [4/7]	81
7.11.2.6 Cell() [5/7]	81
7.11.2.7 Cell() [6/7]	81

7.11.2.8 Cell() [7/7]	81
7.11.3 Member Function Documentation	81
7.11.3.1 add() [1/4]	82
7.11.3.2 add() [2/4]	82
7.11.3.3 add() [3/4]	82
7.11.3.4 add() [4/4]	82
7.11.3.5 operator Cell_Type()	82
7.11.3.6 operator"!=()	82
7.11.3.7 operator=() [1/2]	83
7.11.3.8 operator=() [2/2]	83
7.11.3.9 operator==()	83
7.11.4 Member Data Documentation	83
7.11.4.1 active	83
7.11.4.2 value	83
7.11.4.3 visited	84
7.12 ConstBArrayRowIter< Cell_Type, Data_Type > Class Template Reference	84
7.12.1 Detailed Description	85
7.12.2 Constructor & Destructor Documentation	85
7.12.2.1 ConstBArrayRowIter()	85
7.12.2.2 ~ConstBArrayRowIter()	85
7.12.3 Member Data Documentation	85
7.12.3.1 Array	85
7.12.3.2 current_col	85
7.12.3.3 current_row	86
7.12.3.4 iter	86
7.13 Counter< Array_Type, Data_Type > Class Template Reference	86
7.13.1 Detailed Description	87
7.13.2 Constructor & Destructor Documentation	87
7.13.2.1 Counter() [1/4]	87
7.13.2.2 Counter() [2/4]	88
7.13.2.3 Counter() [3/4]	88
7.13.2.4 Counter() [4/4]	88
7.13.2.5 ~Counter()	88
7.13.3 Member Function Documentation	88
7.13.3.1 count()	89
7.13.3.2 get_description()	89
7.13.3.3 get_name()	89
7.13.3.4 init()	89
7.13.3.5 operator=() [1/2]	89
7.13.3.6 operator=() [2/2]	89
7.13.4 Member Data Documentation	90
7.13.4.1 count_fun	90

7.13.4.2 data	90
7.13.4.3 delete_data	90
7.13.4.4 desc	90
7.13.4.5 init_fun	90
7.13.4.6 name	91
7.14 Counters< Array_Type, Data_Type > Class Template Reference	91
7.14.1 Detailed Description	91
7.14.2 Constructor & Destructor Documentation	92
7.14.2.1 Counters() [1/3]	92
7.14.2.2 ~Counters()	92
7.14.2.3 Counters() [2/3]	92
7.14.2.4 Counters() [3/3]	92
7.14.3 Member Function Documentation	93
7.14.3.1 add_counter() [1/3]	93
7.14.3.2 add_counter() [2/3]	93
7.14.3.3 add_counter() [3/3]	93
7.14.3.4 clear()	93
7.14.3.5 get_descriptions()	93
7.14.3.6 get_names()	94
7.14.3.7 operator=() [1/2]	94
7.14.3.8 operator=() [2/2]	94
7.14.3.9 operator[]()	94
7.14.3.10 size()	96
7.15 Entries< Cell_Type > Class Template Reference	96
7.15.1 Detailed Description	96
7.15.2 Constructor & Destructor Documentation	97
7.15.2.1 Entries() [1/2]	97
7.15.2.2 Entries() [2/2]	97
7.15.2.3 ~Entries()	97
7.15.3 Member Function Documentation	97
7.15.3.1 resize()	97
7.15.4 Member Data Documentation	98
7.15.4.1 source	98
7.15.4.2 target	98
7.15.4.3 val	98
7.16 Flock Class Reference	98
7.16.1 Detailed Description	99
7.16.2 Constructor & Destructor Documentation	99
7.16.2.1 Flock()	100
7.16.2.2 ~Flock()	100
7.16.3 Member Function Documentation	100
7.16.3.1 add_data()	100

7.16.3.2 colnames()	100
7.16.3.3 get_counters()	101
7.16.3.4 get_model()	101
7.16.3.5 get_support()	101
7.16.3.6 init()	101
7.16.3.7 likelihood_joint()	101
7.16.3.8 nfuncs()	102
7.16.3.9 nleafs()	102
7.16.3.10 nnodes()	102
7.16.3.11 nterms()	102
7.16.3.12 ntrees()	102
7.16.3.13 operator()	102
7.16.3.14 parse_polytomies()	103
7.16.3.15 print()	103
7.16.3.16 set_seed()	103
7.16.3.17 support_size()	103
7.16.4 Member Data Documentation	104
7.16.4.1 dat	104
7.16.4.2 initialized	104
7.16.4.3 model	104
7.16.4.4 nfunctions	104
7.16.4.5 rengine	104
7.17 FreqTable< T > Class Template Reference	105
7.17.1 Detailed Description	105
7.17.2 Constructor & Destructor Documentation	105
7.17.2.1 FreqTable()	105
7.17.2.2 ~FreqTable()	105
7.17.3 Member Function Documentation	106
7.17.3.1 add()	106
7.17.3.2 as_vector()	106
7.17.3.3 clear()	106
7.17.3.4 get_data()	106
7.17.3.5 get_data_ptr()	106
7.17.3.6 print()	107
7.17.3.7 reserve()	107
7.17.3.8 size()	107
7.18 Geese Class Reference	107
7.18.1 Detailed Description	110
7.18.2 Constructor & Destructor Documentation	110
7.18.2.1 Geese() [1/4]	110
7.18.2.2 Geese() [2/4]	110
7.18.2.3 Geese() [3/4]	111

7.18.2.4 Geese() [ 4 / 4 ]	111
7.18.2.5 ~Geese()	111
7.18.3 Member Function Documentation	111
7.18.3.1 calc_reduced_sequence()	111
7.18.3.2 calc_sequence()	111
7.18.3.3 colnames()	112
7.18.3.4 get_annotated_nodes()	112
7.18.3.5 get_counters()	112
7.18.3.6 get_model()	112
7.18.3.7 get_probabilities()	112
7.18.3.8 get_engine()	112
7.18.3.9 get_states()	113
7.18.3.10 get_support()	113
7.18.3.11 inherit_support()	113
7.18.3.12 init()	113
7.18.3.13 init_node()	113
7.18.3.14 likelihood()	114
7.18.3.15 likelihood_exhaust()	114
7.18.3.16 nannotations()	114
7.18.3.17 nfuncs()	114
7.18.3.18 nleafs()	114
7.18.3.19 nnodes()	115
7.18.3.20 nterms()	115
7.18.3.21 observed_counts()	115
7.18.3.22 operator=() [ 1 / 2 ]	115
7.18.3.23 operator=() [ 2 / 2 ]	115
7.18.3.24 parse_polytomies()	115
7.18.3.25 predict()	116
7.18.3.26 predict_backend()	116
7.18.3.27 predict_exhaust()	116
7.18.3.28 predict_exhaust_backend()	116
7.18.3.29 predict_sim()	116
7.18.3.30 print()	117
7.18.3.31 print_observed_counts()	117
7.18.3.32 set_seed()	117
7.18.3.33 simulate()	117
7.18.3.34 support_size()	117
7.18.3.35 update_annotations()	118
7.18.4 Member Data Documentation	118
7.18.4.1 delete_engine	118
7.18.4.2 delete_support	118
7.18.4.3 initialized	118

7.18.4.4 map_to_nodes . . . . .	118
7.18.4.5 nfunctions . . . . .	119
7.18.4.6 nodes . . . . .	119
7.18.4.7 reduced_sequence . . . . .	119
7.18.4.8 sequence . . . . .	119
7.19 Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference . . . . .	119
7.19.1 Detailed Description . . . . .	121
7.19.2 Constructor & Destructor Documentation . . . . .	123
7.19.2.1 Model() [1/3] . . . . .	123
7.19.2.2 Model() [2/3] . . . . .	123
7.19.2.3 Model() [3/3] . . . . .	123
7.19.2.4 ~Model() . . . . .	123
7.19.3 Member Function Documentation . . . . .	124
7.19.3.1 add_array() . . . . .	124
7.19.3.2 add_counter() [1/3] . . . . .	124
7.19.3.3 add_counter() [2/3] . . . . .	124
7.19.3.4 add_counter() [3/3] . . . . .	125
7.19.3.5 add_rule() [1/3] . . . . .	125
7.19.3.6 add_rule() [2/3] . . . . .	125
7.19.3.7 add_rule() [3/3] . . . . .	125
7.19.3.8 add_rule_dyn() [1/3] . . . . .	125
7.19.3.9 add_rule_dyn() [2/3] . . . . .	126
7.19.3.10 add_rule_dyn() [3/3] . . . . .	126
7.19.3.11 colnames() . . . . .	126
7.19.3.12 conditional_prob() . . . . .	126
7.19.3.13 gen_key() . . . . .	127
7.19.3.14 get_counters() . . . . .	127
7.19.3.15 get_norm_const() . . . . .	127
7.19.3.16 get_pset() . . . . .	127
7.19.3.17 get_pset_stats() . . . . .	128
7.19.3.18 get_rengine() . . . . .	128
7.19.3.19 get_rules() . . . . .	128
7.19.3.20 get_rules_dyn() . . . . .	128
7.19.3.21 get_support() . . . . .	128
7.19.3.22 likelihood() [1/3] . . . . .	129
7.19.3.23 likelihood() [2/3] . . . . .	129
7.19.3.24 likelihood() [3/3] . . . . .	129
7.19.3.25 likelihood_total() . . . . .	129
7.19.3.26 nterms() . . . . .	130
7.19.3.27 operator=() . . . . .	130
7.19.3.28 print() . . . . .	130

7.19.3.29 print_stats()	130
7.19.3.30 sample() [1/2]	130
7.19.3.31 sample() [2/2]	131
7.19.3.32 set_counters()	131
7.19.3.33 set_keygen()	131
7.19.3.34 set_rengine()	131
7.19.3.35 set_rules()	131
7.19.3.36 set_rules_dyn()	132
7.19.3.37 set_seed()	132
7.19.3.38 size()	132
7.19.3.39 size_unique()	132
7.19.3.40 store_psets()	132
7.19.3.41 support_size()	133
7.20 NetCounterData Class Reference	133
7.20.1 Detailed Description	133
7.20.2 Constructor & Destructor Documentation	133
7.20.2.1 NetCounterData() [1/2]	133
7.20.2.2 NetCounterData() [2/2]	134
7.20.2.3 ~NetCounterData()	134
7.20.3 Member Data Documentation	134
7.20.3.1 indices	134
7.20.3.2 numbers	134
7.21 NetworkData Class Reference	134
7.21.1 Detailed Description	135
7.21.2 Constructor & Destructor Documentation	135
7.21.2.1 NetworkData() [1/3]	135
7.21.2.2 NetworkData() [2/3]	135
7.21.2.3 NetworkData() [3/3]	136
7.21.2.4 ~NetworkData()	136
7.21.3 Member Data Documentation	136
7.21.3.1 directed	136
7.21.3.2 vertex_attr	137
7.22 Node Class Reference	137
7.22.1 Detailed Description	138
7.22.2 Constructor & Destructor Documentation	138
7.22.2.1 Node() [1/5]	138
7.22.2.2 Node() [2/5]	139
7.22.2.3 Node() [3/5]	139
7.22.2.4 Node() [4/5]	139
7.22.2.5 Node() [5/5]	139
7.22.2.6 ~Node()	139
7.22.3 Member Function Documentation	139



7.22.3.1 <code>get_parent()</code>	140
7.22.3.2 <code>is_leaf()</code>	140
7.22.3.3 <code>noffspring()</code>	140
7.22.4 Member Data Documentation	140
7.22.4.1 annotations	140
7.22.4.2 array	140
7.22.4.3 arrays	141
7.22.4.4 duplication	141
7.22.4.5 id	141
7.22.4.6 narray	141
7.22.4.7 offspring	141
7.22.4.8 ord	142
7.22.4.9 parent	142
7.22.4.10 probability	142
7.22.4.11 subtree_prob	142
7.22.4.12 visited	142
7.23 NodeData Class Reference	143
7.23.1 Detailed Description	143
7.23.2 Constructor & Destructor Documentation	143
7.23.2.1 <code>NodeData()</code>	143
7.23.3 Member Data Documentation	143
7.23.3.1 blengths	144
7.23.3.2 duplication	144
7.23.3.3 states	144
7.24 PhyloCounterData Class Reference	144
7.24.1 Detailed Description	145
7.24.2 Constructor & Destructor Documentation	145
7.24.2.1 <code>PhyloCounterData()</code>	145
7.24.3 Member Function Documentation	145
7.24.3.1 <code>at()</code>	145
7.24.3.2 <code>begin()</code>	145
7.24.3.3 <code>empty()</code>	145
7.24.3.4 <code>end()</code>	146
7.24.3.5 <code>get_counters()</code>	146
7.24.3.6 <code>operator&gt;()</code>	146
7.24.3.7 <code>push_back()</code>	146
7.24.3.8 <code>reserve()</code>	146
7.24.3.9 <code>shrink_to_fit()</code>	146
7.24.3.10 <code>size()</code>	147
7.25 PhyloRuleDynData Class Reference	147
7.25.1 Detailed Description	147
7.25.2 Constructor & Destructor Documentation	147

7.25.2.1 PhyloRuleDynData()	147
7.25.2.2 ~PhyloRuleDynData()	148
7.25.3 Member Data Documentation	148
7.25.3.1 counts	148
7.25.3.2 duplication	148
7.25.3.3 lb	148
7.25.3.4 pos	148
7.25.3.5 ub	148
7.26 PowerSet< Array_Type, Data_Rule_Type > Class Template Reference	149
7.26.1 Detailed Description	150
7.26.2 Constructor & Destructor Documentation	150
7.26.2.1 PowerSet() [1/3]	150
7.26.2.2 PowerSet() [2/3]	150
7.26.2.3 PowerSet() [3/3]	151
7.26.2.4 ~PowerSet()	151
7.26.3 Member Function Documentation	151
7.26.3.1 add_rule() [1/3]	151
7.26.3.2 add_rule() [2/3]	151
7.26.3.3 add_rule() [3/3]	151
7.26.3.4 begin()	152
7.26.3.5 calc()	152
7.26.3.6 end()	152
7.26.3.7 get_data()	152
7.26.3.8 get_data_ptr()	152
7.26.3.9 init_support()	153
7.26.3.10 operator[]()	153
7.26.3.11 reset()	153
7.26.3.12 size()	153
7.26.4 Member Data Documentation	153
7.26.4.1 coordinates_free	153
7.26.4.2 coordinates_locked	154
7.26.4.3 data	154
7.26.4.4 EmptyArray	154
7.26.4.5 M	154
7.26.4.6 N	154
7.26.4.7 rules	155
7.26.4.8 rules_deleted	155
7.27 Progress Class Reference	155
7.27.1 Detailed Description	155
7.27.2 Constructor & Destructor Documentation	155
7.27.2.1 Progress()	156
7.27.2.2 ~Progress()	156

7.27.3 Member Function Documentation	156
7.27.3.1 end()	156
7.27.3.2 next()	156
7.28 Rule< Array_Type, Data_Type > Class Template Reference	156
7.28.1 Detailed Description	157
7.28.2 Constructor & Destructor Documentation	157
7.28.2.1 Rule() [1/2]	157
7.28.2.2 Rule() [2/2]	157
7.28.2.3 ~Rule()	158
7.28.3 Member Function Documentation	158
7.28.3.1 D()	158
7.28.3.2 operator()()	158
7.29 Rules< Array_Type, Data_Type > Class Template Reference	158
7.29.1 Detailed Description	159
7.29.2 Constructor & Destructor Documentation	159
7.29.2.1 Rules() [1/2]	159
7.29.2.2 Rules() [2/2]	159
7.29.2.3 ~Rules()	160
7.29.3 Member Function Documentation	160
7.29.3.1 add_rule() [1/3]	160
7.29.3.2 add_rule() [2/3]	160
7.29.3.3 add_rule() [3/3]	160
7.29.3.4 clear()	160
7.29.3.5 get_seq()	160
7.29.3.6 operator()()	161
7.29.3.7 operator=()	161
7.29.3.8 size()	162
7.30 StatsCounter< Array_Type, Data_Type > Class Template Reference	162
7.30.1 Detailed Description	162
7.30.2 Constructor & Destructor Documentation	163
7.30.2.1 StatsCounter() [1/2]	163
7.30.2.2 StatsCounter() [2/2]	163
7.30.2.3 ~StatsCounter()	163
7.30.3 Member Function Documentation	163
7.30.3.1 add_counter() [1/2]	163
7.30.3.2 add_counter() [2/2]	164
7.30.3.3 count_all()	164
7.30.3.4 count_current()	164
7.30.3.5 count_init()	164
7.30.3.6 get_counters()	164
7.30.3.7 get_descriptions()	164
7.30.3.8 get_names()	165

7.30.3.9 reset_array()	165
7.30.3.10 set_counters()	165
7.31 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference	165
7.31.1 Detailed Description	167
7.31.2 Constructor & Destructor Documentation	167
7.31.2.1 Support() [1/3]	167
7.31.2.2 Support() [2/3]	168
7.31.2.3 Support() [3/3]	168
7.31.2.4 ~Support()	168
7.31.3 Member Function Documentation	168
7.31.3.1 add_counter() [1/2]	168
7.31.3.2 add_counter() [2/2]	169
7.31.3.3 add_rule() [1/2]	169
7.31.3.4 add_rule() [2/2]	169
7.31.3.5 add_rule_dyn() [1/2]	169
7.31.3.6 add_rule_dyn() [2/2]	169
7.31.3.7 calc()	170
7.31.3.8 eval_rules_dyn()	171
7.31.3.9 get_counters()	171
7.31.3.10 get_counts()	171
7.31.3.11 get_counts_ptr()	171
7.31.3.12 get_current_stats()	172
7.31.3.13 get_data()	172
7.31.3.14 get_rules()	172
7.31.3.15 get_rules_dyn()	172
7.31.3.16 init_support()	172
7.31.3.17 print()	173
7.31.3.18 reset_array() [1/2]	173
7.31.3.19 reset_array() [2/2]	173
7.31.3.20 set_counters()	173
7.31.3.21 set_rules()	173
7.31.3.22 set_rules_dyn()	174
7.31.4 Member Data Documentation	174
7.31.4.1 change_stats	174
7.31.4.2 coordinates_free	174
7.31.4.3 coordinates_locked	174
7.31.4.4 current_stats	175
7.31.4.5 delete_counters	175
7.31.4.6 delete_rules	175
7.31.4.7 delete_rules_dyn	175
7.31.4.8 M	175

7.31.4.9 max_num_elements . . . . .	176
7.31.4.10 N . . . . .	176
7.32 vecHasher< T > Struct Template Reference . . . . .	176
7.32.1 Detailed Description . . . . .	176
7.32.2 Member Function Documentation . . . . .	176
7.32.2.1 operator>() . . . . .	176
<b>8 File Documentation</b>	<b>177</b>
8.1 include/barry/barray-bones.hpp File Reference . . . . .	177
8.1.1 Macro Definition Documentation . . . . .	178
8.1.1.1 BARRAY_BONES_HPP . . . . .	178
8.2 include/barry/barray-iterator.hpp File Reference . . . . .	178
8.3 include/barry/barray-meat-operators.hpp File Reference . . . . .	179
8.3.1 Macro Definition Documentation . . . . .	180
8.3.1.1 BARRAY_TEMPLATE . . . . .	180
8.3.1.2 BARRAY_TEMPLATE_ARGS . . . . .	180
8.3.1.3 BARRAY_TYPE . . . . .	180
8.3.1.4 BARRY_BARRAY_MEAT_OPERATORS_HPP . . . . .	181
8.3.1.5 COL . . . . .	181
8.3.1.6 ROW . . . . .	181
8.3.2 Function Documentation . . . . .	181
8.3.2.1 BARRAY_TEMPLATE() [1/6] . . . . .	181
8.3.2.2 BARRAY_TEMPLATE() [2/6] . . . . .	181
8.3.2.3 BARRAY_TEMPLATE() [3/6] . . . . .	182
8.3.2.4 BARRAY_TEMPLATE() [4/6] . . . . .	182
8.3.2.5 BARRAY_TEMPLATE() [5/6] . . . . .	182
8.3.2.6 BARRAY_TEMPLATE() [6/6] . . . . .	182
8.3.2.7 BARRAY_TEMPLATE_ARGS() . . . . .	182
8.3.2.8 BARRAY_TYPE() . . . . .	182
8.3.2.9 for() . . . . .	183
8.3.2.10 operator>() . . . . .	183
8.3.3 Variable Documentation . . . . .	183
8.3.3.1 rhs . . . . .	183
8.3.3.2 this . . . . .	183
8.4 include/barry/barray-meat.hpp File Reference . . . . .	184
8.4.1 Macro Definition Documentation . . . . .	186
8.4.1.1 BARRAY_TEMPLATE . . . . .	186
8.4.1.2 BARRAY_TEMPLATE_ARGS . . . . .	186
8.4.1.3 BARRAY_TYPE . . . . .	187
8.4.1.4 COL . . . . .	187
8.4.1.5 ROW . . . . .	187
8.4.2 Function Documentation . . . . .	187

8.4.2.1 ans()	187
8.4.2.2 BARRAY_TEMPLATE() [1/23]	187
8.4.2.3 BARRAY_TEMPLATE() [2/23]	188
8.4.2.4 BARRAY_TEMPLATE() [3/23]	188
8.4.2.5 BARRAY_TEMPLATE() [4/23]	188
8.4.2.6 BARRAY_TEMPLATE() [5/23]	188
8.4.2.7 BARRAY_TEMPLATE() [6/23]	188
8.4.2.8 BARRAY_TEMPLATE() [7/23]	188
8.4.2.9 BARRAY_TEMPLATE() [8/23]	189
8.4.2.10 BARRAY_TEMPLATE() [9/23]	189
8.4.2.11 BARRAY_TEMPLATE() [10/23]	189
8.4.2.12 BARRAY_TEMPLATE() [11/23]	189
8.4.2.13 BARRAY_TEMPLATE() [12/23]	189
8.4.2.14 BARRAY_TEMPLATE() [13/23]	190
8.4.2.15 BARRAY_TEMPLATE() [14/23]	190
8.4.2.16 BARRAY_TEMPLATE() [15/23]	190
8.4.2.17 BARRAY_TEMPLATE() [16/23]	190
8.4.2.18 BARRAY_TEMPLATE() [17/23]	190
8.4.2.19 BARRAY_TEMPLATE() [18/23]	190
8.4.2.20 BARRAY_TEMPLATE() [19/23]	191
8.4.2.21 BARRAY_TEMPLATE() [20/23]	191
8.4.2.22 BARRAY_TEMPLATE() [21/23]	191
8.4.2.23 BARRAY_TEMPLATE() [22/23]	191
8.4.2.24 BARRAY_TEMPLATE() [23/23]	191
8.4.2.25 COL()	191
8.4.2.26 for() [1/3]	192
8.4.2.27 for() [2/3]	192
8.4.2.28 for() [3/3]	192
8.4.2.29 if() [1/17]	192
8.4.2.30 if() [2/17]	192
8.4.2.31 if() [3/17]	192
8.4.2.32 if() [4/17]	193
8.4.2.33 if() [5/17]	193
8.4.2.34 if() [6/17]	193
8.4.2.35 if() [7/17]	193
8.4.2.36 if() [8/17]	193
8.4.2.37 if() [9/17]	193
8.4.2.38 if() [10/17]	194
8.4.2.39 if() [11/17]	194
8.4.2.40 if() [12/17]	194
8.4.2.41 if() [13/17]	194
8.4.2.42 if() [14/17]	194

8.4.2.43 if() [15/17]	194
8.4.2.44 if() [16/17]	194
8.4.2.45 if() [17/17]	195
8.4.2.46 M()	195
8.4.2.47 resize() [1/2]	195
8.4.2.48 resize() [2/2]	195
8.4.2.49 return()	195
8.4.2.50 ROW() [1/2]	195
8.4.2.51 ROW() [2/2]	195
8.4.3 Variable Documentation	196
8.4.3.1 add	196
8.4.3.2 ans	196
8.4.3.3 Array_	196
8.4.3.4 check_bounds	196
8.4.3.5 check_exists	197
8.4.3.6 col0	197
8.4.3.7 const	197
8.4.3.8 copy_data	197
8.4.3.9 data	197
8.4.3.10 delete_data	198
8.4.3.11 delete_data_	198
8.4.3.12 else	198
8.4.3.13 false	198
8.4.3.14 first	198
8.4.3.15 i1	199
8.4.3.16 j	199
8.4.3.17 j0	199
8.4.3.18 j1	199
8.4.3.19 M	199
8.4.3.20 M_	200
8.4.3.21 N	200
8.4.3.22 NCells	200
8.4.3.23 report	200
8.4.3.24 return	200
8.4.3.25 row0	201
8.4.3.26 search	201
8.4.3.27 source	201
8.4.3.28 target	201
8.4.3.29 v	201
8.4.3.30 value	201
8.5 include/barry/barraycell-bones.hpp File Reference	202
8.6 include/barry/barraycell-meat.hpp File Reference	202

8.7 include/barry/barraydense-bones.hpp File Reference . . . . .	203
8.7.1 Macro Definition Documentation . . . . .	205
8.7.1.1 BARRY_BARRAYDENSE_BONES_HPP . . . . .	205
8.8 include/barry/barraydense-meat-operators.hpp File Reference . . . . .	205
8.8.1 Macro Definition Documentation . . . . .	206
8.8.1.1 BARRY_BARRAY_MEAT_OPERATORS_HPP . . . . .	206
8.8.1.2 BDENSE_TEMPLATE . . . . .	206
8.8.1.3 BDENSE_TEMPLATE_ARGS . . . . .	206
8.8.1.4 BDENSE_TYPE . . . . .	207
8.8.1.5 COL . . . . .	207
8.8.1.6 POS . . . . .	207
8.8.1.7 POS_N . . . . .	207
8.8.1.8 ROW . . . . .	207
8.8.2 Function Documentation . . . . .	207
8.8.2.1 BDENSE_TEMPLATE() [1/4] . . . . .	208
8.8.2.2 BDENSE_TEMPLATE() [2/4] . . . . .	208
8.8.2.3 BDENSE_TEMPLATE() [3/4] . . . . .	208
8.8.2.4 BDENSE_TEMPLATE() [4/4] . . . . .	208
8.8.2.5 BDENSE_TEMPLATE_ARGS() . . . . .	208
8.8.2.6 BDENSE_TYPE() . . . . .	208
8.9 include/barry/barraydense-meat.hpp File Reference . . . . .	209
8.9.1 Macro Definition Documentation . . . . .	211
8.9.1.1 BDENSE_TEMPLATE . . . . .	211
8.9.1.2 BDENSE_TEMPLATE_ARGS . . . . .	211
8.9.1.3 BDENSE_TYPE . . . . .	211
8.9.1.4 COL . . . . .	212
8.9.1.5 POS . . . . .	212
8.9.1.6 POS_N . . . . .	212
8.9.1.7 ROW . . . . .	212
8.9.1.8 ZERO_CELL . . . . .	212
8.9.2 Function Documentation . . . . .	212
8.9.2.1 ans() . . . . .	213
8.9.2.2 BDENSE_TEMPLATE() [1/27] . . . . .	213
8.9.2.3 BDENSE_TEMPLATE() [2/27] . . . . .	213
8.9.2.4 BDENSE_TEMPLATE() [3/27] . . . . .	213
8.9.2.5 BDENSE_TEMPLATE() [4/27] . . . . .	213
8.9.2.6 BDENSE_TEMPLATE() [5/27] . . . . .	213
8.9.2.7 BDENSE_TEMPLATE() [6/27] . . . . .	214
8.9.2.8 BDENSE_TEMPLATE() [7/27] . . . . .	214
8.9.2.9 BDENSE_TEMPLATE() [8/27] . . . . .	214
8.9.2.10 BDENSE_TEMPLATE() [9/27] . . . . .	214
8.9.2.11 BDENSE_TEMPLATE() [10/27] . . . . .	214



8.9.2.12	BDENSE_TEMPLATE() [11/27]	215
8.9.2.13	BDENSE_TEMPLATE() [12/27]	215
8.9.2.14	BDENSE_TEMPLATE() [13/27]	215
8.9.2.15	BDENSE_TEMPLATE() [14/27]	215
8.9.2.16	BDENSE_TEMPLATE() [15/27]	215
8.9.2.17	BDENSE_TEMPLATE() [16/27]	216
8.9.2.18	BDENSE_TEMPLATE() [17/27]	216
8.9.2.19	BDENSE_TEMPLATE() [18/27]	216
8.9.2.20	BDENSE_TEMPLATE() [19/27]	216
8.9.2.21	BDENSE_TEMPLATE() [20/27]	216
8.9.2.22	BDENSE_TEMPLATE() [21/27]	216
8.9.2.23	BDENSE_TEMPLATE() [22/27]	217
8.9.2.24	BDENSE_TEMPLATE() [23/27]	217
8.9.2.25	BDENSE_TEMPLATE() [24/27]	217
8.9.2.26	BDENSE_TEMPLATE() [25/27]	217
8.9.2.27	BDENSE_TEMPLATE() [26/27]	217
8.9.2.28	BDENSE_TEMPLATE() [27/27]	217
8.9.2.29	for()	218
8.9.2.30	if() [1/6]	218
8.9.2.31	if() [2/6]	218
8.9.2.32	if() [3/6]	218
8.9.2.33	if() [4/6]	218
8.9.2.34	if() [5/6]	218
8.9.2.35	if() [6/6]	219
8.9.2.36	M()	219
8.9.2.37	resize() [1/3]	219
8.9.2.38	resize() [2/3]	219
8.9.2.39	resize() [3/3]	219
8.9.3	Variable Documentation	219
8.9.3.1	add	219
8.9.3.2	ans	220
8.9.3.3	check_bounds	220
8.9.3.4	check_exists	220
8.9.3.5	col	220
8.9.3.6	const	220
8.9.3.7	copy_data	221
8.9.3.8	data	221
8.9.3.9	delete_data	221
8.9.3.10	delete_data_	221
8.9.3.11	el	221
8.9.3.12	else	222
8.9.3.13	false	222

8.9.3.14 i1	222
8.9.3.15 j	222
8.9.3.16 j0	222
8.9.3.17 j1	222
8.9.3.18 M	223
8.9.3.19 M_	223
8.9.3.20 N	223
8.9.3.21 NCells	223
8.9.3.22 report	223
8.9.3.23 return	224
8.9.3.24 source	224
8.9.3.25 target	224
8.9.3.26 v	224
8.9.3.27 value	224
8.10 include/barry/barraydensecell-bones.hpp File Reference	225
8.11 include/barry/barraydensecell-meat.hpp File Reference	225
8.11.1 Macro Definition Documentation	226
8.11.1.1 POS	227
8.12 include/barry/barrayrow-bones.hpp File Reference	227
8.13 include/barry/barrayrow-meat.hpp File Reference	228
8.13.1 Macro Definition Documentation	228
8.13.1.1 BARRY_BARRAYROW_MEAT_HPP	228
8.13.1.2 BROW_TEMPLATE	229
8.13.1.3 BROW_TEMPLATE_ARGS	229
8.13.1.4 BROW_TYPE	229
8.13.2 Function Documentation	229
8.13.2.1 BROW_TEMPLATE() [1/5]	229
8.13.2.2 BROW_TEMPLATE() [2/5]	229
8.13.2.3 BROW_TEMPLATE() [3/5]	230
8.13.2.4 BROW_TEMPLATE() [4/5]	230
8.13.2.5 BROW_TEMPLATE() [5/5]	230
8.14 include/barry/barrayvector-bones.hpp File Reference	230
8.15 include/barry/barrayvector-meat.hpp File Reference	231
8.15.1 Macro Definition Documentation	232
8.15.1.1 BARRY_BARRAYVECTOR_MEAT_HPP	232
8.16 include/barry/barry-configuration.hpp File Reference	232
8.16.1 Macro Definition Documentation	233
8.16.1.1 BARRY_CHECK_SUPPORT	233
8.16.1.2 BARRY_ISFINITE	233
8.16.1.3 BARRY_MAX_NUM_ELEMENTS	233
8.16.1.4 BARRY_SAFE_EXP	233
8.16.1.5 printf_barry	233

8.16.2 Typedef Documentation	233
8.16.2.1 Map	234
8.17 include/barry/barry-debug.hpp File Reference	234
8.17.1 Macro Definition Documentation	234
8.17.1.1 BARRY_DEBUG_LEVEL	234
8.18 include/barry/barry.hpp File Reference	234
8.18.1 Macro Definition Documentation	236
8.18.1.1 BARRY_HPP	236
8.18.1.2 BARRY_VERSION	236
8.18.1.3 COUNTER_FUNCTION	236
8.18.1.4 COUNTER_LAMBDA	236
8.18.1.5 RULE_FUNCTION	237
8.18.1.6 RULE_LAMBDA	237
8.19 include/barry/cell-bones.hpp File Reference	237
8.20 include/barry/cell-meat.hpp File Reference	238
8.21 include/barry/col-bones.hpp File Reference	239
8.22 include/barry/counters-bones.hpp File Reference	239
8.23 include/barry/counters-meat.hpp File Reference	240
8.23.1 Macro Definition Documentation	242
8.23.1.1 COUNTER_TEMPLATE	242
8.23.1.2 COUNTER_TEMPLATE_ARGS	242
8.23.1.3 COUNTER_TYPE	242
8.23.1.4 COUNTERS_TEMPLATE	242
8.23.1.5 COUNTERS_TEMPLATE_ARGS	242
8.23.1.6 COUNTERS_TYPE	243
8.23.2 Function Documentation	243
8.23.2.1 count_fun()	243
8.23.2.2 COUNTER_TEMPLATE() [1/7]	243
8.23.2.3 COUNTER_TEMPLATE() [2/7]	243
8.23.2.4 COUNTER_TEMPLATE() [3/7]	243
8.23.2.5 COUNTER_TEMPLATE() [4/7]	244
8.23.2.6 COUNTER_TEMPLATE() [5/7]	244
8.23.2.7 COUNTER_TEMPLATE() [6/7]	244
8.23.2.8 COUNTER_TEMPLATE() [7/7]	244
8.23.2.9 COUNTERS_TEMPLATE() [1/8]	244
8.23.2.10 COUNTERS_TEMPLATE() [2/8]	245
8.23.2.11 COUNTERS_TEMPLATE() [3/8]	245
8.23.2.12 COUNTERS_TEMPLATE() [4/8]	245
8.23.2.13 COUNTERS_TEMPLATE() [5/8]	245
8.23.2.14 COUNTERS_TEMPLATE() [6/8]	245
8.23.2.15 COUNTERS_TEMPLATE() [7/8]	246
8.23.2.16 COUNTERS_TEMPLATE() [8/8]	246

8.23.2.17 data()	246
8.23.2.18 delete_data() [1/3]	246
8.23.2.19 delete_data() [2/3]	246
8.23.2.20 delete_data() [3/3]	246
8.23.2.21 delete_to_be_deleted() [1/2]	247
8.23.2.22 delete_to_be_deleted() [2/2]	247
8.23.2.23 desc()	247
8.23.2.24 init_fun() [1/3]	247
8.23.2.25 init_fun() [2/3]	247
8.23.2.26 init_fun() [3/3]	248
8.23.2.27 name()	248
8.23.2.28 push_back() [1/2]	248
8.23.2.29 push_back() [2/2]	248
8.23.2.30 to_be_deleted() [1/2]	248
8.23.2.31 to_be_deleted() [2/2]	248
8.23.3 Variable Documentation	248
8.23.3.1 count_fun_	249
8.23.3.2 counter	249
8.23.3.3 counter_	249
8.23.3.4 data_	249
8.23.3.5 delete_data_	250
8.23.3.6 desc_	250
8.23.3.7 i	250
8.23.3.8 init_fun_	250
8.23.3.9 j	250
8.23.3.10 name_	251
8.23.3.11 noexcept	251
8.23.3.12 return	251
8.24 include/barry/counters/network-css.hpp File Reference	252
8.24.1 Macro Definition Documentation	253
8.24.1.1 CSS_APPEND	253
8.24.1.2 CSS_CASE_ELSE	253
8.24.1.3 CSS_CASE_PERCEIVED	253
8.24.1.4 CSS_CASE_TRUTH	254
8.24.1.5 CSS_CHECK_SIZE	254
8.24.1.6 CSS_CHECK_SIZE_INIT	254
8.24.1.7 CSS_NET_COUNTER_LAMBDA_INIT	254
8.24.1.8 CSS_PERCEIVED_CELLS	255
8.24.1.9 CSS_SIZE	255
8.24.1.10 CSS_TRUE_CELLS	255
8.24.2 Function Documentation	255
8.24.2.1 counter_css_census01()	255

8.24.2.2 counter_css_census02()	256
8.24.2.3 counter_css_census03()	256
8.24.2.4 counter_css_census04()	256
8.24.2.5 counter_css_census05()	256
8.24.2.6 counter_css_census06()	256
8.24.2.7 counter_css_census07()	257
8.24.2.8 counter_css_census08()	257
8.24.2.9 counter_css_census09()	257
8.24.2.10 counter_css_census10()	257
8.24.2.11 counter_css_completely_false_recip_comiss()	257
8.24.2.12 counter_css_completely_false_recip_omiss()	258
8.24.2.13 counter_css_mixed_recip()	258
8.24.2.14 counter_css_partially_false_recip_commi()	258
8.24.2.15 counter_css_partially_false_recip_omiss()	258
8.25 include/barry/counters/network.hpp File Reference	259
8.25.1 Macro Definition Documentation	261
8.25.1.1 NET_C_DATA_IDX	262
8.25.1.2 NET_C_DATA_NUM	262
8.25.1.3 NETWORK_COUNTER	262
8.25.1.4 NETWORK_COUNTER_LAMBDA	262
8.25.1.5 NETWORK_RULE	263
8.25.1.6 NETWORK_RULE_LAMBDA	263
8.25.2 Typedef Documentation	263
8.25.2.1 NetCounter	263
8.25.2.2 NetCounters	263
8.25.2.3 NetModel	264
8.25.2.4 NetRule	264
8.25.2.5 NetRules	264
8.25.2.6 NetStatsCounter	264
8.25.2.7 NetSupport	264
8.25.2.8 Network	264
8.25.2.9 NetworkDense	265
8.25.3 Function Documentation	265
8.25.3.1 rules_zerodiag()	265
8.26 include/barry/counters/phylo.hpp File Reference	265
8.26.1 Macro Definition Documentation	267
8.26.1.1 DEFAULT_DUPLICATION	267
8.26.1.2 DUPL_DUPL	267
8.26.1.3 DUPL_EITH	267
8.26.1.4 DUPL_SPEC	268
8.26.1.5 IF_MATCHES	268
8.26.1.6 IF_NOTMATCHES	268

8.26.1.7 IS_DUPLICATION	268
8.26.1.8 IS_EITHER	268
8.26.1.9 IS_SPECIATION	269
8.26.1.10 MAKE_DUPL_VARS	269
8.26.1.11 PHYLO_CHECK_MISSING	269
8.26.1.12 PHYLO_COUNTER_LAMBDA	269
8.26.1.13 PHYLO_RULE_DYN_LAMBDA	270
8.26.2 Typedef Documentation	270
8.26.2.1 PhyloArray	270
8.26.2.2 PhyloCounter	270
8.26.2.3 PhyloCounters	270
8.26.2.4 PhyloModel	270
8.26.2.5 PhyloPowerSet	271
8.26.2.6 PhyloRule	271
8.26.2.7 PhyloRuleData	271
8.26.2.8 PhyloRuleDyn	271
8.26.2.9 PhyloRules	271
8.26.2.10 PhyloRulesDyn	271
8.26.2.11 PhyloStatsCounter	272
8.26.2.12 PhyloSupport	272
8.26.3 Function Documentation	272
8.26.3.1 get_last_name()	272
8.27 include/barry/model-bones.hpp File Reference	272
8.27.1 Function Documentation	273
8.27.1.1 keygen_default()	273
8.28 include/barry/model-meat.hpp File Reference	274
8.28.1 Macro Definition Documentation	274
8.28.1.1 MODEL_TEMPLATE	274
8.28.1.2 MODEL_TEMPLATE_ARGS	275
8.28.1.3 MODEL_TYPE	275
8.28.2 Function Documentation	275
8.28.2.1 likelihood_()	275
8.28.2.2 MODEL_TEMPLATE() [1/2]	275
8.28.2.3 MODEL_TEMPLATE() [2/2]	276
8.28.2.4 update_normalizing_constant()	276
8.29 include/barry/models/geese.hpp File Reference	276
8.30 include/barry/models/geese/flock-bones.hpp File Reference	277
8.31 include/barry/models/geese/flock-meat.hpp File Reference	277
8.32 include/barry/models/geese/geese-bones.hpp File Reference	278
8.32.1 Macro Definition Documentation	278
8.32.1.1 INITIALIZED	279
8.32.2 Function Documentation	279

8.32.2.1	<a href="#">keygen_full()</a>	279
8.32.2.2	<a href="#">RULE_FUNCTION()</a>	279
8.32.2.3	<a href="#">vec_diff()</a>	279
8.32.2.4	<a href="#">vector_caster()</a>	279
8.33	<a href="#">include/barry/models/geese/geese-meat-constructors.hpp File Reference</a>	280
8.34	<a href="#">include/barry/models/geese/geese-meat-likelihood.hpp File Reference</a>	280
8.35	<a href="#">include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference</a>	281
8.36	<a href="#">include/barry/models/geese/geese-meat-predict.hpp File Reference</a>	282
8.37	<a href="#">include/barry/models/geese/geese-meat-predict_exhaust.hpp File Reference</a>	282
8.38	<a href="#">include/barry/models/geese/geese-meat-predict_sim.hpp File Reference</a>	283
8.39	<a href="#">include/barry/models/geese/geese-meat-simulate.hpp File Reference</a>	283
8.40	<a href="#">include/barry/models/geese/geese-meat.hpp File Reference</a>	284
8.41	<a href="#">include/barry/models/geese/geese-node-bones.hpp File Reference</a>	284
8.42	<a href="#">include/barry/powerset-bones.hpp File Reference</a>	285
8.43	<a href="#">include/barry/powerset-meat.hpp File Reference</a>	286
8.44	<a href="#">include/barry/progress.hpp File Reference</a>	287
8.44.1	Macro Definition Documentation	287
8.44.1.1	<a href="#">BARRY_PROGRESS_BAR_WIDTH</a>	287
8.45	<a href="#">include/barry/rules-bones.hpp File Reference</a>	287
8.45.1	Function Documentation	288
8.45.1.1	<a href="#">rule_fun_default()</a>	288
8.46	<a href="#">include/barry/rules-meat.hpp File Reference</a>	289
8.47	<a href="#">include/barry/statscounter-bones.hpp File Reference</a>	289
8.48	<a href="#">include/barry/statscounter-meat.hpp File Reference</a>	291
8.48.1	Macro Definition Documentation	292
8.48.1.1	<a href="#">STATSCOUNTER_TEMPLATE</a>	292
8.48.1.2	<a href="#">STATSCOUNTER_TEMPLATE_ARGS</a>	292
8.48.1.3	<a href="#">STATSCOUNTER_TYPE</a>	292
8.48.2	Function Documentation	293
8.48.2.1	<a href="#">for()</a>	293
8.48.2.2	<a href="#">resize()</a>	293
8.48.2.3	<a href="#">STATSCOUNTER_TEMPLATE() [1/9]</a>	293
8.48.2.4	<a href="#">STATSCOUNTER_TEMPLATE() [2/9]</a>	293
8.48.2.5	<a href="#">STATSCOUNTER_TEMPLATE() [3/9]</a>	293
8.48.2.6	<a href="#">STATSCOUNTER_TEMPLATE() [4/9]</a>	294
8.48.2.7	<a href="#">STATSCOUNTER_TEMPLATE() [5/9]</a>	294
8.48.2.8	<a href="#">STATSCOUNTER_TEMPLATE() [6/9]</a>	294
8.48.2.9	<a href="#">STATSCOUNTER_TEMPLATE() [7/9]</a>	294
8.48.2.10	<a href="#">STATSCOUNTER_TEMPLATE() [8/9]</a>	294
8.48.2.11	<a href="#">STATSCOUNTER_TEMPLATE() [9/9]</a>	294
8.48.3	Variable Documentation	295
8.48.3.1	<a href="#">counter_deleted</a>	295

8.48.3.2 counters	295
8.48.3.3 counters_	295
8.48.3.4 f_	295
8.48.3.5 j	296
8.48.3.6 return	296
8.49 include/barry/statsdb.hpp File Reference	296
8.50 include/barry/support-bones.hpp File Reference	297
8.51 include/barry/support-meat.hpp File Reference	299
8.51.1 Macro Definition Documentation	300
8.51.1.1 BARRY_SUPPORT_MEAT_HPP	301
8.51.1.2 SUPPORT_TEMPLATE	301
8.51.1.3 SUPPORT_TEMPLATE_ARGS	301
8.51.1.4 SUPPORT_TYPE	301
8.51.2 Function Documentation	301
8.51.2.1 calc_backend()	302
8.51.2.2 for()	302
8.51.2.3 if() [1/3]	302
8.51.2.4 if() [2/3]	302
8.51.2.5 if() [3/3]	302
8.51.2.6 insert_cell()	302
8.51.2.7 rm_cell()	303
8.51.2.8 SUPPORT_TEMPLATE() [1/17]	303
8.51.2.9 SUPPORT_TEMPLATE() [2/17]	303
8.51.2.10 SUPPORT_TEMPLATE() [3/17]	303
8.51.2.11 SUPPORT_TEMPLATE() [4/17]	303
8.51.2.12 SUPPORT_TEMPLATE() [5/17]	304
8.51.2.13 SUPPORT_TEMPLATE() [6/17]	304
8.51.2.14 SUPPORT_TEMPLATE() [7/17]	304
8.51.2.15 SUPPORT_TEMPLATE() [8/17]	304
8.51.2.16 SUPPORT_TEMPLATE() [9/17]	304
8.51.2.17 SUPPORT_TEMPLATE() [10/17]	304
8.51.2.18 SUPPORT_TEMPLATE() [11/17]	305
8.51.2.19 SUPPORT_TEMPLATE() [12/17]	305
8.51.2.20 SUPPORT_TEMPLATE() [13/17]	305
8.51.2.21 SUPPORT_TEMPLATE() [14/17]	305
8.51.2.22 SUPPORT_TEMPLATE() [15/17]	305
8.51.2.23 SUPPORT_TEMPLATE() [16/17]	306
8.51.2.24 SUPPORT_TEMPLATE() [17/17]	306
8.51.3 Variable Documentation	306
8.51.3.1 array_bank	306
8.51.3.2 cfree	306
8.51.3.3 counters	306



8.51.3.4 counters_ . . . . .	307
8.51.3.5 delete_counters . . . . .	307
8.51.3.6 delete_rules . . . . .	307
8.51.3.7 delete_rules_dyn . . . . .	307
8.51.3.8 else . . . . .	307
8.51.3.9 f_ . . . . .	308
8.51.3.10 return . . . . .	308
8.51.3.11 rules . . . . .	308
8.51.3.12 rules_ . . . . .	308
8.51.3.13 rules_dyn . . . . .	308
8.51.3.14 stats_bank . . . . .	309
8.52 include/barry/typedefs.hpp File Reference . . . . .	309
8.52.1 Typedef Documentation . . . . .	311
8.52.1.1 Col_type . . . . .	311
8.52.1.2 Counter_fun_type . . . . .	311
8.52.1.3 Counts_type . . . . .	311
8.52.1.4 MapVec_type . . . . .	311
8.52.1.5 Row_type . . . . .	312
8.52.1.6 Rule_fun_type . . . . .	312
8.52.1.7 uint . . . . .	312
8.52.2 Function Documentation . . . . .	312
8.52.2.1 vec_equal() . . . . .	312
8.52.2.2 vec_equal_approx() . . . . .	313
8.52.2.3 vec_inner_prod() . . . . .	313
8.53 README.md File Reference . . . . .	313



# Chapter 1

## Main Page

### Barry: your to-go motif accountant

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

Among the key features included in `barry`, we have:

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

### Examples

#### Counting statistics in a graph

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

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

```

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

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

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

// Initializing the data. The program deals with 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

```

## Features

### Efficient memory usage

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

## Documentation

More information can be found in the Doxygen website [here](#) and in the PDF version of the documentation [here](#).

## Code of Conduct

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



## Chapter 2

# Module Index

### 2.1 Modules

Here is a list of all modules:

Counting . . . . .	11
Statistical Models . . . . .	11
Network counters . . . . .	12
Phylo counters . . . . .	17
Phylo rules . . . . .	23





## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">BArray&lt; Cell_Type, Data_Type &gt;</a>	
Baseline class for binary arrays	29
<a href="#">BArrayCell&lt; Cell_Type, Data_Type &gt;</a>	43
<a href="#">BArrayCell_const&lt; Cell_Type, Data_Type &gt;</a>	45
<a href="#">BArrayDense&lt; Cell_Type, Data_Type &gt;</a>	
Baseline class for binary arrays	48
<a href="#">BArrayDenseCell&lt; Cell_Type, Data_Type &gt;</a>	62
<a href="#">BArrayDenseCell_const&lt; Cell_Type, Data_Type &gt;</a>	64
<a href="#">BArrayRow&lt; Cell_Type, Data_Type &gt;</a>	67
<a href="#">BArrayRow_const&lt; Cell_Type, Data_Type &gt;</a>	69
<a href="#">BArrayVector&lt; Cell_Type, Data_Type &gt;</a>	
Row or column of a <a href="#">BArray</a>	71
<a href="#">BArrayVector_const&lt; Cell_Type, Data_Type &gt;</a>	75
<a href="#">Cell&lt; Cell_Type &gt;</a>	
Entries in <a href="#">BArray</a> . For now, it only has two members:	79
<a href="#">ConstBArrayRowIter&lt; Cell_Type, Data_Type &gt;</a>	84
<a href="#">Counter&lt; Array_Type, Data_Type &gt;</a>	
A counter function based on change statistics	86
<a href="#">Counters&lt; Array_Type, Data_Type &gt;</a>	
Vector of counters	91
<a href="#">Entries&lt; Cell_Type &gt;</a>	
A wrapper class to store <code>source</code> , <code>target</code> , <code>val</code> from a <a href="#">BArray</a> object	96
<a href="#">Flock</a>	
A <a href="#">Flock</a> is a group of <a href="#">Geese</a>	98
<a href="#">FreqTable&lt; T &gt;</a>	
Database of statistics	105
<a href="#">Geese</a>	
Annotated Phylo <a href="#">Model</a>	107
<a href="#">Model&lt; Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type &gt;</a>	
General framework for discrete exponential models. This class allows generating discrete exponential models in the form of a linear exponential model:	119
<a href="#">NetCounterData</a>	
Data class used to store arbitrary uint or double vectors	133
<a href="#">NetworkData</a>	
Data class for Networks	134

<a href="#">Node</a>	
A single node for the model . . . . .	137
<a href="#">NodeData</a>	
Data definition for the <code>PhyloArray</code> class . . . . .	143
<a href="#">PhyloCounterData</a> . . . . .	144
<a href="#">PhyloRuleDynData</a> . . . . .	147
<a href="#">PowerSet&lt; Array_Type, Data_Rule_Type &gt;</a>	
Powerset of a binary array . . . . .	149
<a href="#">Progress</a>	
A simple progress bar . . . . .	155
<a href="#">Rule&lt; Array_Type, Data_Type &gt;</a>	
Rule for determining if a cell should be included in a sequence . . . . .	156
<a href="#">Rules&lt; Array_Type, Data_Type &gt;</a>	
Vector of objects of class <a href="#">Rule</a> . . . . .	158
<a href="#">StatsCounter&lt; Array_Type, Data_Type &gt;</a>	
Count stats for a single Array . . . . .	162
<a href="#">Support&lt; Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type &gt;</a>	
Compute the support of sufficient statistics . . . . .	165
<a href="#">vecHasher&lt; T &gt;</a> . . . . .	176

## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

include/barry/barray-bones.hpp . . . . .	177
include/barry/barray-iterator.hpp . . . . .	178
include/barry/barray-meat-operators.hpp . . . . .	179
include/barry/barray-meat.hpp . . . . .	184
include/barry/barraycell-bones.hpp . . . . .	202
include/barry/barraycell-meat.hpp . . . . .	202
include/barry/barraydense-bones.hpp . . . . .	203
include/barry/barraydense-meat-operators.hpp . . . . .	205
include/barry/barraydense-meat.hpp . . . . .	209
include/barry/barraydensecell-bones.hpp . . . . .	225
include/barry/barraydensecell-meat.hpp . . . . .	225
include/barry/barrayrow-bones.hpp . . . . .	227
include/barry/barrayrow-meat.hpp . . . . .	228
include/barry/barrayvector-bones.hpp . . . . .	230
include/barry/barrayvector-meat.hpp . . . . .	231
include/barry/barry-configuration.hpp . . . . .	232
include/barry/barry-debug.hpp . . . . .	234
include/barry/barry.hpp . . . . .	234
include/barry/cell-bones.hpp . . . . .	237
include/barry/cell-meat.hpp . . . . .	238
include/barry/col-bones.hpp . . . . .	239
include/barry/counters-bones.hpp . . . . .	239
include/barry/counters-meat.hpp . . . . .	240
include/barry/model-bones.hpp . . . . .	272
include/barry/model-meat.hpp . . . . .	274
include/barry/powerset-bones.hpp . . . . .	285
include/barry/powerset-meat.hpp . . . . .	286
include/barry/progress.hpp . . . . .	287
include/barry/rules-bones.hpp . . . . .	287
include/barry/rules-meat.hpp . . . . .	289
include/barry/statscounter-bones.hpp . . . . .	289
include/barry/statscounter-meat.hpp . . . . .	291
include/barry/statsdb.hpp . . . . .	296
include/barry/support-bones.hpp . . . . .	297
include/barry/support-meat.hpp . . . . .	299

include/barry/typedefs.hpp	309
include/barry/counters/network-css.hpp	252
include/barry/counters/network.hpp	259
include/barry/counters/phylo.hpp	265
include/barry/models/geese.hpp	276
include/barry/models/geese/flock-bones.hpp	277
include/barry/models/geese/flock-meat.hpp	277
include/barry/models/geese/geese-bones.hpp	278
include/barry/models/geese/geese-meat-constructors.hpp	280
include/barry/models/geese/geese-meat-likelihood.hpp	280
include/barry/models/geese/geese-meat-likelihood_exhaust.hpp	281
include/barry/models/geese/geese-meat-predict.hpp	282
include/barry/models/geese/geese-meat-predict_exhaust.hpp	282
include/barry/models/geese/geese-meat-predict_sim.hpp	283
include/barry/models/geese/geese-meat-simulate.hpp	283
include/barry/models/geese/geese-meat.hpp	284
include/barry/models/geese/geese-node-bones.hpp	284

## 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.

## Functions

- `template<typename Tnet = Network>`  
`void counter\_edges (NetCounters< Tnet > *counters)`  
*Number of edges.*
- `template<typename Tnet = Network>`  
`void counter\_isolates (NetCounters< Tnet > *counters)`  
*Number of isolated vertices.*
- `template<typename Tnet = Network>`  
`void counter\_mutual (NetCounters< Tnet > *counters)`  
*Number of mutual ties.*
- `template<typename Tnet = Network>`  
`void counter\_istar2 (NetCounters< Tnet > *counters)`
- `template<typename Tnet = Network>`  
`void counter\_ostar2 (NetCounters< Tnet > *counters)`
- `template<typename Tnet = Network>`  
`void counter\_ttriads (NetCounters< Tnet > *counters)`
- `template<typename Tnet = Network>`  
`void counter\_ctriads (NetCounters< Tnet > *counters)`
- `template<typename Tnet = Network>`  
`void counter\_density (NetCounters< Tnet > *counters)`
- `template<typename Tnet = Network>`  
`void counter\_idegree15 (NetCounters< Tnet > *counters)`
- `template<typename Tnet = Network>`  
`void counter\_odegree15 (NetCounters< Tnet > *counters)`
- `template<typename Tnet = Network>`  
`void counter\_absdiff (NetCounters< Tnet > *counters, uint attr_id, double alpha=1.0)`  
*Sum of absolute attribute difference between ego and alter.*
- `template<typename Tnet = Network>`  
`void counter\_diff (NetCounters< Tnet > *counters, uint attr_id, double alpha=1.0, double tail_head=true)`  
*Sum of attribute difference between ego and alter to pow(alpha)*
- `NETWORK\_COUNTER (init_single_attr)`

- `template<typename Tnet = Network>`  
`void counter_nodeicov (NetCounters< Tnet > *counters, uint attr_id)`
- `template<typename Tnet = Network>`  
`void counter_nodeocov (NetCounters< Tnet > *counters, uint attr_id)`
- `template<typename Tnet = Network>`  
`void counter_nodecov (NetCounters< Tnet > *counters, uint attr_id)`
- `template<typename Tnet = Network>`  
`void counter_nodematch (NetCounters< Tnet > *counters, uint attr_id)`
- `template<typename Tnet = Network>`  
`void counter_idegree (NetCounters< Tnet > *counters, std::vector< uint > d)`  
*Counts number of vertices with a given in-degree.*
- `template<typename Tnet = Network>`  
`void counter_odegree (NetCounters< Tnet > *counters, std::vector< uint > d)`  
*Counts number of vertices with a given out-degree.*
- `template<typename Tnet = Network>`  
`void counter_degree (NetCounters< Tnet > *counters, std::vector< uint > d)`  
*Counts number of vertices with a given out-degree.*

### 5.3.1 Detailed Description

[Counters](#) for network models.

#### Parameters

<i>counters</i>	A pointer to a <code>NetCounters</code> object ( <code>Counters&lt;Network, NetCounterData&gt;</code> ).
-----------------	--

### 5.3.2 Function Documentation

#### 5.3.2.1 counter\_absdiff()

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

Sum of absolute attribute difference between ego and alter.

Definition at line 526 of file network.hpp.

#### 5.3.2.2 counter\_ctriads()

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

Definition at line 390 of file network.hpp.

### 5.3.2.3 counter\_degree()

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

Counts number of vertices with a given out-degree.

Definition at line 839 of file network.hpp.

### 5.3.2.4 counter\_density()

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

Definition at line 444 of file network.hpp.

### 5.3.2.5 counter\_diff()

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

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

Definition at line 571 of file network.hpp.

### 5.3.2.6 counter\_edges()

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

Number of edges.

Definition at line 138 of file network.hpp.



#### 5.3.2.7 counter\_iddegree()

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

Counts number of vertices with a given in-degree.

Definition at line 740 of file network.hpp.

#### 5.3.2.8 counter\_iddegree15()

```
template<typename Tnet = Network>
void counter_iddegree15 (
    NetCounters< Tnet > * counters ) [inline]
```

Definition at line 470 of file network.hpp.

#### 5.3.2.9 counter\_isolates()

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

Number of isolated vertices.

Definition at line 160 of file network.hpp.

#### 5.3.2.10 counter\_istar2()

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

Definition at line 253 of file network.hpp.

#### 5.3.2.11 counter\_mutual()

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

Number of mutual ties.

Definition at line 199 of file network.hpp.

#### 5.3.2.12 counter\_nodecov()

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

Definition at line 684 of file network.hpp.

#### 5.3.2.13 counter\_nodeicov()

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

Definition at line 634 of file network.hpp.

#### 5.3.2.14 counter\_nodematch()

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

Definition at line 709 of file network.hpp.

#### 5.3.2.15 counter\_nodeocov()

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

Definition at line 659 of file network.hpp.

#### 5.3.2.16 counter\_odegree()

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

Counts number of vertices with a given out-degree.

Definition at line 789 of file network.hpp.

#### 5.3.2.17 counter\_odegree15()

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

Definition at line 497 of file network.hpp.

#### 5.3.2.18 counter\_ostar2()

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

Definition at line 275 of file network.hpp.

#### 5.3.2.19 counter\_ttriads()

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

Definition at line 300 of file network.hpp.

#### 5.3.2.20 NETWORK\_COUNTER()

```
NETWORK_COUNTER (
    init_single_attr )
```

Definition at line 615 of file network.hpp.

## 5.4 Phylo counters

[Counters](#) for phylogenetic modeling.

## Functions

- void [counter\\_overall\\_gains](#) ([PhyloCounters](#) \*counters, unsigned int duplication=DEFAULT\_DUPLICATION)  
*Overall functional gains.*
- void [counter\\_gains](#) ([PhyloCounters](#) \*counters, std::vector< [uint](#) > nfun, unsigned int duplication=DEFAULT\_DUPLICATION)  
*Functional gains for a specific function (nfun).*
- void [counter\\_gains\\_k\\_offspring](#) ([PhyloCounters](#) \*counters, std::vector< [uint](#) > nfun, [uint](#) k=1u, unsigned int duplication=DEFAULT\_DUPLICATION)  
*k genes gain function nfun*
- void [counter\\_genes\\_changing](#) ([PhyloCounters](#) \*counters, unsigned int duplication=DEFAULT\_DUPLICATION)  
*Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)*
- void [counter\\_prop\\_genes\\_changing](#) ([PhyloCounters](#) \*counters, unsigned int duplication=DEFAULT\_DUPLICATION)  
*Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)*
- void [counter\\_overall\\_loss](#) ([PhyloCounters](#) \*counters, unsigned int duplication=DEFAULT\_DUPLICATION)  
*Overall functional loss.*
- void [counter\\_maxfuns](#) ([PhyloCounters](#) \*counters, [uint](#) lb, [uint](#) ub, unsigned int duplication=DEFAULT\_DUPLICATION)  
*Cap the number of functions per gene.*
- void [counter\\_loss](#) ([PhyloCounters](#) \*counters, std::vector< [uint](#) > nfun, unsigned int duplication=DEFAULT\_DUPLICATION)  
*Total count of losses for an specific function.*
- void [counter\\_overall\\_changes](#) ([PhyloCounters](#) \*counters, unsigned int duplication=DEFAULT\_DUPLICATION)  
*Total number of changes. Use this statistic to account for "preservation".*
- void [counter\\_subfun](#) ([PhyloCounters](#) \*counters, [uint](#) nfunA, [uint](#) nfunB, unsigned int duplication=DEFAULT\_DUPLICATION)  
*Total count of Sub-functionalization events.*
- void [counter\\_cogain](#) ([PhyloCounters](#) \*counters, [uint](#) nfunA, [uint](#) nfunB, unsigned int duplication=DEFAULT\_DUPLICATION)  
*Co-evolution (joint gain or loss)*
- void [counter\\_longest](#) ([PhyloCounters](#) \*counters, unsigned int duplication=DEFAULT\_DUPLICATION)  
*Longest branch mutates (either by gain or by loss)*
- void [counter\\_neofun](#) ([PhyloCounters](#) \*counters, [uint](#) nfunA, [uint](#) nfunB, unsigned int duplication=DEFAULT\_DUPLICATION)  
*Total number of neofunctionalization events.*
- void [counter\\_neofun\\_a2b](#) ([PhyloCounters](#) \*counters, [uint](#) nfunA, [uint](#) nfunB, unsigned int duplication=DEFAULT\_DUPLICATION)  
*Total number of neofunctionalization events.*
- void [counter\\_co\\_opt](#) ([PhyloCounters](#) \*counters, [uint](#) nfunA, [uint](#) nfunB, unsigned int duplication=DEFAULT\_DUPLICATION)  
*Function co-opting.*
- void [counter\\_k\\_genes\\_changing](#) ([PhyloCounters](#) \*counters, unsigned int k, unsigned int duplication=DEFAULT\_DUPLICATION)  
*Indicator function. Equals to one if k genes changed and zero otherwise.*

### 5.4.1 Detailed Description

[Counters](#) for phylogenetic modeling.

#### Parameters

<a href="#">counters</a>	A pointer to a <a href="#">PhyloCounters</a> object ( <a href="#">Counters</a> < <a href="#">PhyloArray</a> , <a href="#">PhyloCounterData</a> >).
--------------------------	--

### 5.4.2 Function Documentation

### 5.4.2.1 counter\_co\_opt()

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

Function co-opting.

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

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

This algorithm implements the change statistic.

Definition at line 1194 of file phylo.hpp.

### 5.4.2.2 counter\_cogain()

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

Co-evolution (joint gain or loss)

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

Definition at line 819 of file phylo.hpp.

### 5.4.2.3 counter\_gains()

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

Functional gains for a specific function (nfun).

Definition at line 192 of file phylo.hpp.

#### 5.4.2.4 counter\_gains\_k\_offspring()

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

k genes gain function nfun

Definition at line 238 of file phylo.hpp.

#### 5.4.2.5 counter\_genes\_changing()

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

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

Definition at line 311 of file phylo.hpp.

#### 5.4.2.6 counter\_k\_genes\_changing()

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

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

< How many genes diverge the parent

Definition at line 1293 of file phylo.hpp.

#### 5.4.2.7 counter\_longest()

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

Longest branch mutates (either by gain or by loss)

Definition at line 877 of file phylo.hpp.

#### 5.4.2.8 counter\_loss()

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

Total count of losses for an specific function.

Definition at line 622 of file phylo.hpp.

#### 5.4.2.9 counter\_maxfuns()

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

Cap the number of functions per gene.

Definition at line 540 of file phylo.hpp.

#### 5.4.2.10 counter\_neofun()

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

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 989 of file phylo.hpp.

#### 5.4.2.11 counter\_neofun\_a2b()

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

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 1075 of file phylo.hpp.

#### 5.4.2.12 counter\_overall\_changes()

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

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

Definition at line 672 of file phylo.hpp.

#### 5.4.2.13 counter\_overall\_gains()

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

Overall functional gains.

Total number of gains (irrespective of the function).

Definition at line 154 of file phylo.hpp.

#### 5.4.2.14 counter\_overall\_loss()

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

Overall functional loss.

Definition at line 497 of file phylo.hpp.

#### 5.4.2.15 counter\_prop\_genes\_changing()

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

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

Definition at line 383 of file phylo.hpp.



#### 5.4.2.16 counter\_subfun()

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

Total count of Sub-functionalization events.

It requires to specify data = {funA, funB}

Definition at line 732 of file phylo.hpp.

## 5.5 Phylo rules

[Rules](#) for phylogenetic modeling.

### Classes

- class [PhyloRuleDynData](#)

### Functions

- void [rule\\_dyn\\_limit\\_changes](#) ([PhyloSupport](#) \*support, [uint](#) pos, [uint](#) lb, [uint](#) ub, unsigned int duplication=[DEFAULT\\_DUPLICATION](#))  
*Overall functional gains.*

#### 5.5.1 Detailed Description

[Rules](#) for phylogenetic modeling.

##### Parameters

<i>rules</i>	A pointer to a <a href="#">PhyloRules</a> object ( <a href="#">Rules</a> < <a href="#">PhyloArray</a> , <a href="#">PhyloRuleData</a> >).
--------------	---

#### 5.5.2 Function Documentation

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

```
void rule_dyn_limit_changes (
    PhyloSupport * support,
    uint pos,
```

```
uint lb,  
uint ub,  
unsigned int duplication = DEFAULT_DUPLICATION ) [inline]
```

Overall functional gains.

#### Parameters

<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 1442 of file phylo.hpp.

## Chapter 6

# Namespace Documentation

### 6.1 `barry` Namespace Reference

`barry`: Your go-to motif accountant

#### Namespaces

- [counters](#)

*Tree class and 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 28 of file typedefs.hpp.

#### 6.5.2.2 NONE

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

Definition at line 29 of file typedefs.hpp.

#### 6.5.2.3 ONE

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

Definition at line 30 of file typedefs.hpp.

#### 6.5.2.4 TWO

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

Definition at line 31 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 46 of file typedefs.hpp.

#### 6.6.2.2 AS\_ZERO

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

Definition at line 45 of file typedefs.hpp.

#### 6.6.2.3 BOTH

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

Definition at line 39 of file typedefs.hpp.

#### 6.6.2.4 NONE

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

Definition at line 40 of file typedefs.hpp.

#### 6.6.2.5 ONE

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

Definition at line 41 of file typedefs.hpp.

#### 6.6.2.6 TWO

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

Definition at line 42 of file typedefs.hpp.

#### 6.6.2.7 UNKNOWN

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

Definition at line 44 of file typedefs.hpp.

## Chapter 7

# Class Documentation

### 7.1 BArray< Cell\_Type, Data\_Type > Class Template Reference

Baseline class for binary arrays.

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

#### Public Member Functions

- bool `operator==` (const BArray< Cell\_Type, Data\_Type > &Array\_)
- `~BArray` ()
- void `out_of_range` (uint i, uint j) const
- Cell\_Type `get_cell` (uint i, uint j, bool check\_bounds=true) const
- std::vector< Cell\_Type > `get_col_vec` (uint i, bool check\_bounds=true) const
- std::vector< Cell\_Type > `get_row_vec` (uint i, bool check\_bounds=true) const
- void `get_col_vec` (std::vector< Cell\_Type > \*x, uint i, bool check\_bounds=true) const
- void `get_row_vec` (std::vector< Cell\_Type > \*x, uint i, bool check\_bounds=true) const
- const Row\_type< Cell\_Type > & `row` (uint i, bool check\_bounds=true) const
- const Col\_type< Cell\_Type > & `col` (uint i, bool check\_bounds=true) const
- Entries< Cell\_Type > `get_entries` () const

*Get the edgelist.*

- void `transpose` ()
- void `clear` (bool hard=true)
- void `resize` (uint N\_, uint M\_)
- void `reserve` ()
- void `print` (const char \*fmt=nullptr,...) const

#### 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_ ↵</i> <i>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 31 of file barraycell-bones.hpp.

## 7.2.3 Member Function Documentation

### 7.2.3.1 operator Cell\_Type()

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

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

### 7.2.3.2 operator\*=( )

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

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

### 7.2.3.3 operator+=( )

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

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

#### 7.2.3.4 operator-=( )

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

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

#### 7.2.3.5 operator/=( )

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

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

#### 7.2.3.6 operator=( )

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

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

#### 7.2.3.7 operator==( )

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

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

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

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

## 7.3 BArrayCell\_const< Cell\_Type, Data\_Type > Class Template Reference

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

## Public Member Functions

- [BArrayCell\\_const](#) (const BArray< Cell\_Type, Data\_Type > \*Array\_, uint i\_, uint j\_, bool check\_bounds=true)
- [~BArrayCell\\_const](#) ()
- [operator Cell\\_Type](#) () const
- bool [operator==](#) (const Cell\_Type &val) const
- bool [operator!=](#) (const Cell\_Type &val) const
- bool [operator<](#) (const Cell\_Type &val) const
- bool [operator>](#) (const Cell\_Type &val) const
- bool [operator<=](#) (const Cell\_Type &val) const
- bool [operator>=](#) (const Cell\_Type &val) const

### 7.3.1 Detailed Description

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

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

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 BArrayCell\_const()

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

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

#### 7.3.2.2 ~BArrayCell\_const()

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

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

### 7.3.3 Member Function Documentation



#### 7.3.3.1 operator Cell\_Type()

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

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

#### 7.3.3.2 operator"!=(

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

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

#### 7.3.3.3 operator<(

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

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

#### 7.3.3.4 operator<=(

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

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

#### 7.3.3.5 operator==(

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

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

### 7.3.3.6 operator>()

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

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

### 7.3.3.7 operator>=()

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

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

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

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

## 7.4 BArrayDense< Cell\_Type, Data\_Type > Class Template Reference

Baseline class for binary arrays.

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

### Public Member Functions

- bool [operator==](#) (const BArrayDense< Cell\_Type, Data\_Type > &Array\_)
- [~BArrayDense](#) ()
- void [out\\_of\\_range](#) (uint i, uint j) const
- Cell\_Type [get\\_cell](#) (uint i, uint j, bool check\_bounds=true) const
- std::vector< Cell\_Type > [get\\_col\\_vec](#) (uint i, bool check\_bounds=true) const
- std::vector< Cell\_Type > [get\\_row\\_vec](#) (uint i, bool check\_bounds=true) const
- void [get\\_col\\_vec](#) (std::vector< Cell\_Type > \*x, uint i, bool check\_bounds=true) const
- void [get\\_row\\_vec](#) (std::vector< Cell\_Type > \*x, uint i, bool check\_bounds=true) const
- 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 <i>true</i> 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 <i>swap_cells</i> , check if either of both cells exists/don't exist.

- `BArrayDense< Cell_Type, Data_Type > & operator+= (const std::pair< uint, uint > &coords)`
- `BArrayDense< Cell_Type, Data_Type > & operator-= (const std::pair< uint, uint > &coords)`
- `BArrayDenseCell< Cell_Type, Data_Type > operator() (uint i, uint j, bool check_bounds=true)`
- `const BArrayDenseCell_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 BArrayDenseCell< Cell_Type, Data_Type >`
- `class BArrayDenseCell_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 64 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 67 of file barraydense-bones.hpp.

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

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

Edgelist with data.

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

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

Edgelist with no data (simpler)

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

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

Copy constructor.

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

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

Move operator.

#### 7.4.2.7 ~BArrayDense()

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

### 7.4.3 Member Function Documentation

#### 7.4.3.1 clear()

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

#### 7.4.3.2 col()

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

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

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

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

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

#### 7.4.3.5 default\_val()

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

#### 7.4.3.6 get\_cell()

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

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

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

**7.4.3.8 get\_col\_vec()** [2/2]

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

**7.4.3.9 get\_entries()**

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

Get the edgelist.

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

**Returns**

`Entries<Cell_Type>`

**7.4.3.10 get\_row\_vec()** [1/2]

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

**7.4.3.11 get\_row\_vec()** [2/2]

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

**7.4.3.12 insert\_cell()** [1/3]

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



**7.4.3.13 insert\_cell() [2/3]**

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

**7.4.3.14 insert\_cell() [3/3]**

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

**7.4.3.15 is\_empty()**

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

**7.4.3.16 ncol()**

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

**7.4.3.17 nnozero()**

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

**7.4.3.18 nrow()**

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

**7.4.3.19 operator()() [1/2]**

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

**7.4.3.20 operator()() [2/2]**

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

**7.4.3.21 operator\*=( )**

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

**7.4.3.22 operator+=( ) [1/3]**

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

**7.4.3.23 operator+=( ) [2/3]**

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

**7.4.3.24 operator+=( ) [3/3]**

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

**7.4.3.25 operator-=( ) [1/3]**

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

**7.4.3.26 operator-=( ) [2/3]**

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

**7.4.3.27 operator-=( ) [3/3]**

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

**7.4.3.28 operator/=( )**

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

**7.4.3.29 operator=( ) [1/2]**

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

Move assignment.

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

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

Assignment constructor.

#### 7.4.3.31 operator==()

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

#### 7.4.3.32 out\_of\_range()

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

#### 7.4.3.33 print()

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

#### 7.4.3.34 reserve()

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

#### 7.4.3.35 resize()

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

**7.4.3.36 rm\_cell()**

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

**7.4.3.37 row()**

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

**7.4.3.38 set\_data()**

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

Set the data object.

**Parameters**

<i>data_</i>	
<i>delete_ ↔ data_</i>	

**7.4.3.39 swap\_cells()**

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

#### 7.4.3.40 swap\_cols()

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

#### 7.4.3.41 swap\_rows()

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

#### 7.4.3.42 toggle\_cell()

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

#### 7.4.3.43 toggle\_lock()

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

#### 7.4.3.44 transpose()

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

#### 7.4.3.45 zero\_col()

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

#### 7.4.3.46 zero\_row()

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

### 7.4.4 Friends And Related Function Documentation

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

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

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

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

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

Definition at line 1 of file 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 49 of file bararraydense-bones.hpp.

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

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

## 7.5 BArrayDenseCell< Cell\_Type, Data\_Type > Class Template Reference

```
#include <bararraydensecell-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 bararraydensecell-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 BArrayRow< Cell\_Type, Data\_Type > Class Template Reference

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

### Public Member Functions

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

### 7.7.1 Detailed Description

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

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

## 7.7.2 Constructor & Destructor Documentation

### 7.7.2.1 BArrayRow()

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

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

### 7.7.2.2 ~BArrayRow()

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

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

## 7.7.3 Member Function Documentation

### 7.7.3.1 operator BArrayRow< Cell\_Type, Data\_Type >()

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

### 7.7.3.2 operator\*=( )

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

### 7.7.3.3 operator+=( )

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

#### 7.7.3.4 operator==()

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

#### 7.7.3.5 operator/=(

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

#### 7.7.3.6 operator=()

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

#### 7.7.3.7 operator==(

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

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

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

## 7.8 BArrayRow\_const< Cell\_Type, Data\_Type > Class Template Reference

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

### Public Member Functions

- [BArrayRow\\_const](#) (const [BArray](#)< Cell\_Type, Data\_Type > \*Array\_, uint i\_, bool check\_bounds=true)
- [~BArrayRow\\_const](#) ()
- [operator BArrayRow\\_const< Cell\\_Type, Data\\_Type > \(\) const](#)
- bool [operator==](#) (const [BArrayRow\\_const](#)< Cell\_Type, Data\_Type > &val) const
- bool [operator!=](#) (const [BArrayRow\\_const](#)< Cell\_Type, Data\_Type > &val) const
- bool [operator<](#) (const [BArrayRow\\_const](#)< Cell\_Type, Data\_Type > &val) const
- bool [operator>](#) (const [BArrayRow\\_const](#)< Cell\_Type, Data\_Type > &val) const
- bool [operator<=](#) (const [BArrayRow\\_const](#)< Cell\_Type, Data\_Type > &val) const
- bool [operator>=](#) (const [BArrayRow\\_const](#)< Cell\_Type, Data\_Type > &val) const

### 7.8.1 Detailed Description

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

Definition at line 43 of file `barrayrow-bones.hpp`.

### 7.8.2 Constructor & Destructor Documentation

#### 7.8.2.1 BArrayRow\_const()

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

Definition at line 51 of file `barrayrow-bones.hpp`.

#### 7.8.2.2 ~BArrayRow\_const()

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

Definition at line 61 of file `barrayrow-bones.hpp`.

### 7.8.3 Member Function Documentation

#### 7.8.3.1 operator BArrayRow\_const< Cell\_Type, Data\_Type >()

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

#### 7.8.3.2 operator"!="()

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



**7.8.3.3 operator<()**

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

**7.8.3.4 operator<=()**

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

**7.8.3.5 operator==()**

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

**7.8.3.6 operator>()**

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

**7.8.3.7 operator>=()**

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

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

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

**7.9 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](#)

### 7.9.1 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](#).

### 7.9.2 Constructor & Destructor Documentation

#### 7.9.2.1 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 <a href="#">BArray</a> object
<i>dim_</i>	Dimension. 0 means row and 1 means column.
<i>i_</i>	Element to point.
<i>check_bounds</i>	When <a href="#">true</a> , check boundaries.

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

### 7.9.2.2 ~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.

## 7.9.3 Member Function Documentation

### 7.9.3.1 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.

### 7.9.3.2 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.

### 7.9.3.3 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.

#### 7.9.3.4 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.

#### 7.9.3.5 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.

#### 7.9.3.6 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.

#### 7.9.3.7 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.

#### 7.9.3.8 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.

#### 7.9.3.9 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.

#### 7.9.3.10 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.

#### 7.9.3.11 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.

#### 7.9.3.12 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](#)

## 7.10 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`

### 7.10.1 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`.

### 7.10.2 Constructor & Destructor Documentation

#### 7.10.2.1 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`.

#### 7.10.2.2 ~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`.

### 7.10.3 Member Function Documentation

#### 7.10.3.1 begin()

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

#### 7.10.3.2 end()

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

#### 7.10.3.3 is\_col()

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

#### 7.10.3.4 is\_row()

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

#### 7.10.3.5 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.

#### 7.10.3.6 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.

#### 7.10.3.7 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.

#### 7.10.3.8 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.

#### 7.10.3.9 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.

#### 7.10.3.10 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.



**7.10.3.11 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.

**7.10.3.12 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

**7.11 Cell< Cell\_Type > Class Template Reference**

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

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

**Public Member Functions**

- [Cell](#) ()
- [Cell](#) (Cell\_Type value\_, bool visited\_[\\_false](#), bool active\_[\\_true](#))
- [~Cell](#) ()
- [Cell](#) (const [Cell](#)< Cell\_Type > &arg)
- [Cell](#)< Cell\_Type > & [operator=](#) ([Cell](#)< Cell\_Type > &other)
- [Cell](#) ([Cell](#)< Cell\_Type > &&arg) [noexcept](#)
- [Cell](#)< Cell\_Type > & [operator=](#) ([Cell](#)< Cell\_Type > &&other) [noexcept](#)
- void [add](#) (Cell\_Type x)
- [operator Cell\\_Type](#) () const
- bool [operator==](#) (const [Cell](#)< Cell\_Type > &rhs) const
- bool [operator!=](#) (const [Cell](#)< Cell\_Type > &rhs) const
- void [add](#) (double x)
- void [add](#) (unsigned int x)
- void [add](#) (int x)
- [Cell](#) ()
- [Cell](#) ()
- [Cell](#) ()

**Public Attributes**

- Cell\_Type [value](#)
- bool [visited](#)
- bool [active](#)

### 7.11.1 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.

### 7.11.2 Constructor & Destructor Documentation

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

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

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

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

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

#### 7.11.2.3 ~Cell()

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

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

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

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

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

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

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

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

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

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

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

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

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

Definition at line 65 of file cell-meat.hpp.

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

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

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

### 7.11.3 Member Function Documentation

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

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

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

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

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

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

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

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

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

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

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

#### 7.11.3.5 operator Cell\_Type()

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

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

#### 7.11.3.6 operator"!="()

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

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

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

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

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

### 7.11.3.8 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.

### 7.11.3.9 operator==( )

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

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

## 7.11.4 Member Data Documentation

### 7.11.4.1 active

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

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

### 7.11.4.2 value

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

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

### 7.11.4.3 visited

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

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

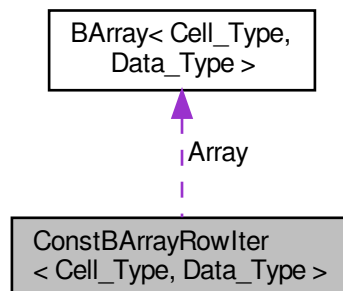
The documentation for this class was generated from the following files:

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

## 7.12 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`

### 7.12.1 Detailed Description

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

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

### 7.12.2 Constructor & Destructor Documentation

#### 7.12.2.1 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.

#### 7.12.2.2 ~ConstBArrayRowIter()

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

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

### 7.12.3 Member Data Documentation

#### 7.12.3.1 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.

#### 7.12.3.2 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.

### 7.12.3.3 `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`.

### 7.12.3.4 `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`

## 7.13 `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

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

- `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](#) = ""

### 7.13.1 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.

### 7.13.2 Constructor & Destructor Documentation

#### 7.13.2.1 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.

**7.13.2.2 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.

**7.13.2.3 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.

**7.13.2.4 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.

**7.13.2.5 ~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.

**7.13.3 Member Function Documentation**

#### 7.13.3.1 count()

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

#### 7.13.3.2 get\_description()

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

#### 7.13.3.3 get\_name()

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

#### 7.13.3.4 init()

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

#### 7.13.3.5 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.

#### 7.13.3.6 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.

## 7.13.4 Member Data Documentation

### 7.13.4.1 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.

### 7.13.4.2 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.

### 7.13.4.3 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.

### 7.13.4.4 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.

### 7.13.4.5 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.

## 7.13.4.6 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

## 7.14 Counters&lt; Array\_Type, Data\_Type &gt; 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**

## 7.14.1 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.

## 7.14.2 Constructor & Destructor Documentation

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

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

### 7.14.2.2 ~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.

### 7.14.2.3 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>	
—	

### 7.14.2.4 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>	
—	

### 7.14.3 Member Function Documentation

#### 7.14.3.1 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 )
```

#### 7.14.3.2 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 )
```

#### 7.14.3.3 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_ = "" )
```

#### 7.14.3.4 clear()

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

#### 7.14.3.5 get\_descriptions()

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

### 7.14.3.6 get\_names()

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

### 7.14.3.7 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>

### 7.14.3.8 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>&

### 7.14.3.9 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>\*

**7.14.3.10 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

**7.15 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](#)

**7.15.1 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 67 of file typedefs.hpp.

## 7.15.2 Constructor & Destructor Documentation

### 7.15.2.1 Entries() [1/2]

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

Definition at line 73 of file typedefs.hpp.

### 7.15.2.2 Entries() [2/2]

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

Definition at line 74 of file typedefs.hpp.

### 7.15.2.3 ~Entries()

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

Definition at line 81 of file typedefs.hpp.

## 7.15.3 Member Function Documentation

### 7.15.3.1 resize()

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

Definition at line 83 of file typedefs.hpp.

## 7.15.4 Member Data Documentation

### 7.15.4.1 source

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

Definition at line 69 of file typedefs.hpp.

### 7.15.4.2 target

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

Definition at line 70 of file typedefs.hpp.

### 7.15.4.3 val

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

Definition at line 71 of file typedefs.hpp.

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

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

## 7.16 Flock Class Reference

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

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

## Public Member Functions

- [Flock](#) ()
- [~Flock](#) ()
- unsigned int [add\\_data](#) (std::vector< std::vector< unsigned int > > &annotations, std::vector< unsigned int > &geneid, std::vector< int > &parent, std::vector< bool > &duplication)  
*Add a tree to the flock.*
- void [set\\_seed](#) (const unsigned int &s)  
*Set the seed of the model.*
- void [init](#) (unsigned int bar\_width=BARRY\_PROGRESS\_BAR\_WIDTH)
- phylocounters::PhyloCounters \* [get\\_counters](#) ()
- phylocounters::PhyloSupport \* [get\\_support](#) ()
- phylocounters::PhyloModel \* [get\\_model](#) ()
- double [likelihood\\_joint](#) (const std::vector< double > &par, bool as\_log=false, bool use\_reduced\_↔ sequence=true)  
*Returns the joint likelihood of the model.*
- [Geese](#) \* [operator\(\)](#) (unsigned int i, bool check\_bounds=true)  
*Access the i-th geese element.*

## Information about the model

- unsigned int [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()

### 7.16.1 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.

### 7.16.2 Constructor & Destructor Documentation

### 7.16.2.1 Flock()

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

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

### 7.16.2.2 ~Flock()

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

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

## 7.16.3 Member Function Documentation

### 7.16.3.1 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 <a href="#">Geese::Geese</a> .
<i>geneid</i>	see <a href="#">Geese</a> .
<i>parent</i>	see <a href="#">Geese</a> .
<i>duplication</i>	see <a href="#">Geese</a> .

#### Returns

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

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

### 7.16.3.2 colnames()

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

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

### 7.16.3.3 get\_counters()

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

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

### 7.16.3.4 get\_model()

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

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

### 7.16.3.5 get\_support()

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

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

### 7.16.3.6 init()

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

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

### 7.16.3.7 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 103 of file flock-meat.hpp.

**7.16.3.8 nfuncs()**

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

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

**7.16.3.9 nleafs()**

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

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

**7.16.3.10 nnodes()**

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

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

**7.16.3.11 nterms()**

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

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

**7.16.3.12 ntrees()**

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

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

**7.16.3.13 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 234 of file flock-meat.hpp.

### 7.16.3.14 parse\_polytomies()

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

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

### 7.16.3.15 print()

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

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

### 7.16.3.16 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-meat.hpp.

### 7.16.3.17 support\_size()

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

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

## 7.16.4 Member Data Documentation

### 7.16.4.1 dat

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

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

### 7.16.4.2 initialized

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

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

### 7.16.4.3 model

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

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

### 7.16.4.4 nfunctions

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

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

### 7.16.4.5 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-meat.hpp](#)

## 7.17 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

### 7.17.1 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.

### 7.17.2 Constructor & Destructor Documentation

#### 7.17.2.1 FreqTable()

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

Definition at line 28 of file statsdb.hpp.

#### 7.17.2.2 ~FreqTable()

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

Definition at line 29 of file statsdb.hpp.

### 7.17.3 Member Function Documentation

#### 7.17.3.1 add()

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

Definition at line 47 of file statsdb.hpp.

#### 7.17.3.2 as\_vector()

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

Definition at line 61 of file statsdb.hpp.

#### 7.17.3.3 clear()

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

Definition at line 83 of file statsdb.hpp.

#### 7.17.3.4 get\_data()

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

Definition at line 73 of file statsdb.hpp.

#### 7.17.3.5 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.

### 7.17.3.6 print()

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

Definition at line 102 of file statsdb.hpp.

### 7.17.3.7 reserve()

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

Definition at line 89 of file statsdb.hpp.

### 7.17.3.8 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

## 7.18 Geese Class Reference

Annotated Phyl<sup>o</sup> [Model](#).

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

## Public Member Functions

- `~Geese ()`
- `void init (unsigned int bar_width=BARRY_PROGRESS_BAR_WIDTH)`
- `void inherit_support (const Geese &model_, bool delete_support_=false)`
- `void calc_sequence (Node *n=nullptr)`
- `void calc_reduced_sequence ()`
- `double likelihood (const std::vector< double > &par, bool as_log=false, bool use_reduced_sequence=true)`
- `double likelihood_exhaust (const std::vector< double > &par)`
- `std::vector< double > get_probabilities () const`
- `void set_seed (const unsigned int &s)`
- `std::vector< std::vector< unsigned int > > simulate (const std::vector< double > &par)`
- `std::vector< std::vector< double > > observed_counts ()`
- `void print_observed_counts ()`
- `void print () const`  
*Prints information about the GEESE.*
- `void init_node (Node &n)`
- `void update_annotations (unsigned int nodeid, std::vector< unsigned int > newann)`
- `std::vector< std::vector< bool > > get_states () const`  
*Powerset of a gene's possible states.*
- `std::vector< unsigned int > get_annotated_nodes () const`  
*Returns the ids of the nodes with at least one annotation.*

### Construct a new Geese object

The model includes a total of  $N + 1$  nodes, the  $+ 1$  beign the root node.

#### Parameters

annotations	A vector of vectors with annotations. It should be of length $k$ (number of functions). Each vector should be of length $N$ (equal to the number of nodes, including interior). Possible values are 0, 1, and 9.
geneid	Id of the gene. It should be of length $N$ .
parent	Id of the parent gene. Also of length $N$
duplication	Logical scalar indicating the type of event (true: duplication, false: speciation.)

The ordering of the entries does not matter. Passing the nodes in post order or not makes no difference to the constructor.

- `Geese ()`
- `Geese (std::vector< std::vector< unsigned int > > &annotations, std::vector< unsigned int > &geneid, std::vector< int > &parent, std::vector< bool > &duplication)`
- `Geese (const Geese &model_, bool copy_data=true)`
- `Geese (Geese &&x) noexcept`
- `Geese & operator= (const Geese &model_)=delete`
- `Geese & operator= (Geese &&model_) noexcept=delete`

### Information about the model

#### Parameters

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

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

#### Returns

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

- *std::vector< std::vector< double > >* [predict](#) (**const** std::vector< double > &par, std::vector< std::vector< double > > \*res\_prob=nullptr, bool leave\_one\_out=false, bool only\_annotated=false, bool use\_reduced\_sequence=true)
- *std::vector< std::vector< double > >* [predict\\_backend](#) (**const** std::vector< double > &par, bool use\_reduced\_sequence, **const** std::vector< 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_engine = false`
- bool `delete_support = false`

### 7.18.1 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`.

### 7.18.2 Constructor & Destructor Documentation

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

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

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

#### 7.18.2.2 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`.



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

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

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

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

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

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

### 7.18.2.5 ~Geese()

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

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

## 7.18.3 Member Function Documentation

### 7.18.3.1 calc\_reduced\_sequence()

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

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

### 7.18.3.2 calc\_sequence()

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

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

### 7.18.3.3 colnames()

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

Names of the terms in the model.

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

### 7.18.3.4 get\_annotated\_nodes()

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

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

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

### 7.18.3.5 get\_counters()

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

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

### 7.18.3.6 get\_model()

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

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

### 7.18.3.7 get\_probabilities()

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

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

### 7.18.3.8 get\_rengine()

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

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

### 7.18.3.9 get\_states()

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

Powerset of a gene's possible states.

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

#### Returns

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

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

### 7.18.3.10 get\_support()

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

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

### 7.18.3.11 inherit\_support()

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

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

### 7.18.3.12 init()

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

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

### 7.18.3.13 init\_node()

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

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

#### 7.18.3.14 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.

#### 7.18.3.15 likelihood\_exhaust()

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

Definition at line 7 of file geese-meat-likelihood\_exhaust.hpp.

#### 7.18.3.16 nannotations()

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

Number of annotations.

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

#### 7.18.3.17 nfuncs()

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

Number of functions analyzed.

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

#### 7.18.3.18 nleafs()

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

Number of leaf.

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

### 7.18.3.19 nnodes()

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

Number of nodes (interior + leaf)

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

### 7.18.3.20 nterms()

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

Number of terms included.

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

### 7.18.3.21 observed\_counts()

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

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

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

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

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

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

### 7.18.3.24 parse\_polytomies()

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

Check polytomies and return the largest.

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

**7.18.3.25 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.

**7.18.3.26 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.

**7.18.3.27 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.

**7.18.3.28 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.

**7.18.3.29 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.

### 7.18.3.30 print()

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

Prints information about the GESE.

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

### 7.18.3.31 print\_observed\_counts()

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

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

### 7.18.3.32 set\_seed()

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

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

### 7.18.3.33 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.

### 7.18.3.34 support\_size()

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

Number of unique sets of sufficient stats.

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

#### 7.18.3.35 update\_annotations()

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

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

### 7.18.4 Member Data Documentation

#### 7.18.4.1 delete\_engine

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

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

#### 7.18.4.2 delete\_support

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

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

#### 7.18.4.3 initialized

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

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

#### 7.18.4.4 map\_to\_nodes

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

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



#### 7.18.4.5 nfunctions

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

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

#### 7.18.4.6 nodes

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

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

#### 7.18.4.7 reduced\_sequence

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

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

#### 7.18.4.8 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](#)

## 7.19 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 > &params={})
- Array\_Type `sample` (const uint &i, const std::vector< double > &params)
- double `conditional_prob` (const Array\_Type &Array\_, const std::vector< double > &params, unsigned int i, unsigned int j)  
*Conditional probability ("Gibbs sampler")*
- const std::mt19937 \* `get_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 `Counters` 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 `Rules` 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 >` &params, `const` `uint` &i, `bool` as\_log=`false`)
- double `likelihood` (`const` `std::vector< double >` &params, `const` `Array_Type` &Array\_, `int` i=-1, `bool` as\_log=`false`)
- double `likelihood` (`const` `std::vector< double >` &params, `const` `std::vector< double >` &target\_, `const` `uint` &i, `bool` as\_log=`false`)
- double `likelihood_total` (`const` `std::vector< double >` &params, `bool` as\_log=`false`)

### Extract elements by index

#### Parameters

i	Index relative to the array in the model.
params	A new vector of model parameters to compute the normalizing constant.
as_log	When <i>true</i> returns the logged version of the normalizing constant.

- double `get_norm_const` (`const` `std::vector< double >` &params, `const` `uint` &i, `bool` as\_log=`false`)
- `const` `std::vector< Array_Type >` \* `get_pset` (`const` `uint` &i)
- `const` `std::vector< std::vector< double > >` \* `get_pset_stats` (`const` `uint` &i)

### Size of the model

Number of different supports included in the model

This will return the size of *stats*.

#### Returns

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

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

## 7.19.1 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 <a href="#">BArray</a> object.
<i>Data_Counter_Type</i>	Any type.
<i>Data_Rule_Type</i>	Any type.

Definition at line 47 of file model-bones.hpp.

## 7.19.2 Constructor & Destructor Documentation

### 7.19.2.1 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 ( )
```

### 7.19.2.2 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_ )
```

### 7.19.2.3 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_ )
```

### 7.19.2.4 ~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.

## 7.19.3 Member Function Documentation

### 7.19.3.1 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.

### 7.19.3.2 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 )
```

### 7.19.3.3 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 )
```

#### 7.19.3.4 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 )
```

#### 7.19.3.5 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 )
```

#### 7.19.3.6 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 )
```

#### 7.19.3.7 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 )
```

#### 7.19.3.8 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 )
```

**7.19.3.9 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 )
```

**7.19.3.10 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 )
```

**7.19.3.11 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
```

**7.19.3.12 conditional\_prob()**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
double Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::conditional↵
_prob (
    const Array_Type & Array_,
    const std::vector< double > & params,
    unsigned int i,
    unsigned int j )
```

Conditional probability ("Gibbs sampler")

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

**Parameters**

<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

## 7.19.3.13 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_ )
```

## 7.19.3.14 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 ( )
```

## 7.19.3.15 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 )
```

## 7.19.3.16 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 )
```

**7.19.3.17 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 )
```

**7.19.3.18 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
```

**7.19.3.19 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 ( )
```

**7.19.3.20 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 ( )
```

**7.19.3.21 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 ( )
```

### 7.19.3.22 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 )
```

### 7.19.3.23 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 )
```

### 7.19.3.24 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 )
```

### 7.19.3.25 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 )
```

**7.19.3.26 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]
```

**7.19.3.27 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_ )
```

**7.19.3.28 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.

**7.19.3.29 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
```

**7.19.3.30 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 = {} )
```

### 7.19.3.31 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 )
```

### 7.19.3.32 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_ )
```

### 7.19.3.33 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_ )
```

### 7.19.3.34 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.

### 7.19.3.35 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_ )
```

**7.19.3.36 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_ )
```

**7.19.3.37 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.

**7.19.3.38 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]
```

**7.19.3.39 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]
```

**7.19.3.40 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]
```

### 7.19.3.41 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](#)

## 7.20 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](#)

### 7.20.1 Detailed Description

Data class used to store arbitrary uint or double vectors.

Definition at line 56 of file network.hpp.

### 7.20.2 Constructor & Destructor Documentation

#### 7.20.2.1 NetCounterData() [1/2]

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

Definition at line 62 of file network.hpp.

### 7.20.2.2 NetCounterData() [2/2]

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

Definition at line 63 of file network.hpp.

### 7.20.2.3 ~NetCounterData()

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

Definition at line 68 of file network.hpp.

## 7.20.3 Member Data Documentation

### 7.20.3.1 indices

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

Definition at line 59 of file network.hpp.

### 7.20.3.2 numbers

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

Definition at line 60 of file network.hpp.

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

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

## 7.21 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](#)

### 7.21.1 Detailed Description

Data class for Networks.

Details on the available counters for [NetworkData](#) can be found in the [Network counters](#) section.

This holds information about whether the graph is directed or not, and, if defined, vectors of node (vertex) attributes ([vertex\\_attr](#)).

Definition at line 19 of file network.hpp.

### 7.21.2 Constructor & Destructor Documentation

#### 7.21.2.1 NetworkData() [1/3]

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

Definition at line 25 of file network.hpp.

#### 7.21.2.2 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 33 of file network.hpp.

**7.21.2.3 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 45 of file network.hpp.

**7.21.2.4 ~NetworkData()**

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

Definition at line 51 of file network.hpp.

**7.21.3 Member Data Documentation****7.21.3.1 directed**

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

Definition at line 22 of file network.hpp.

### 7.21.3.2 vertex\_attr

```
std::vector< std::vector< double > > NetworkData::vertex_attr
```

Definition at line 23 of file network.hpp.

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

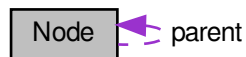
- include/barry/counters/[network.hpp](#)

## 7.22 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.*

### 7.22.1 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`.

### 7.22.2 Constructor & Destructor Documentation

#### 7.22.2.1 `Node()` [1/5]

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

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

### 7.22.2.2 Node() [2/5]

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

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

### 7.22.2.3 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.

### 7.22.2.4 Node() [4/5]

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

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

### 7.22.2.5 Node() [5/5]

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

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

### 7.22.2.6 ~Node()

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

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

## 7.22.3 Member Function Documentation

### 7.22.3.1 `get_parent()`

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

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

### 7.22.3.2 `is_leaf()`

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

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

### 7.22.3.3 `noffspring()`

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

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

## 7.22.4 Member Data Documentation

### 7.22.4.1 `annotations`

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

Observed annotations (only defined for [Geese](#))

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

### 7.22.4.2 `array`

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

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

### 7.22.4.3 arrays

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

Arrays given all possible states.

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

### 7.22.4.4 duplication

```
bool Node::duplication
```

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

### 7.22.4.5 id

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

Id of the node (as specified in the input)

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

### 7.22.4.6 narray

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

ID of the array in the model.

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

### 7.22.4.7 offspring

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

Offspring nodes.

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

#### 7.22.4.8 ord

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

Order in which the node was created.

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

#### 7.22.4.9 parent

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

Parent node.

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

#### 7.22.4.10 probability

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

The probability of observing each state.

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

#### 7.22.4.11 subtree\_prob

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

Induced subtree probabilities.

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

#### 7.22.4.12 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](#)



## 7.23 NodeData Class Reference

Data definition for the `PhyloArray` class.

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

### Public Member Functions

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

### Public Attributes

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

### 7.23.1 Detailed Description

Data definition for the `PhyloArray` class.

Details about the available counters for `PhyloArray` objects can be found in the [Phylo counters](#) section.

This holds basic information about a given node.

Definition at line 38 of file `phylo.hpp`.

### 7.23.2 Constructor & Destructor Documentation

#### 7.23.2.1 NodeData()

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

Definition at line 58 of file `phylo.hpp`.

### 7.23.3 Member Data Documentation

### 7.23.3.1 blengths

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

Branch length.

Definition at line 44 of file phylo.hpp.

### 7.23.3.2 duplication

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

Definition at line 54 of file phylo.hpp.

### 7.23.3.3 states

```
std::vector< bool > NodeData::states = {}
```

State of the parent node.

Definition at line 49 of file phylo.hpp.

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

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

## 7.24 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](#) ()

## 7.24.1 Detailed Description

Definition at line 69 of file phylo.hpp.

## 7.24.2 Constructor & Destructor Documentation

### 7.24.2.1 PhyloCounterData()

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

Definition at line 75 of file phylo.hpp.

## 7.24.3 Member Function Documentation

### 7.24.3.1 at()

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

Definition at line 80 of file phylo.hpp.

### 7.24.3.2 begin()

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

Definition at line 87 of file phylo.hpp.

### 7.24.3.3 empty()

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

Definition at line 90 of file phylo.hpp.

#### 7.24.3.4 end()

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

Definition at line 88 of file phylo.hpp.

#### 7.24.3.5 get\_counters()

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

Definition at line 91 of file phylo.hpp.

#### 7.24.3.6 operator()()

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

Definition at line 81 of file phylo.hpp.

#### 7.24.3.7 push\_back()

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

Definition at line 83 of file phylo.hpp.

#### 7.24.3.8 reserve()

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

Definition at line 82 of file phylo.hpp.

#### 7.24.3.9 shrink\_to\_fit()

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

Definition at line 84 of file phylo.hpp.

### 7.24.3.10 size()

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

Definition at line 85 of file phylo.hpp.

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

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

## 7.25 PhyloRuleDynData Class Reference

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

### Public Member Functions

- [PhyloRuleDynData](#) ([const](#) std::vector< double > \*counts\_, [uint](#) pos\_, [uint](#) lb\_, [uint](#) ub\_, [uint](#) duplication\_)
- [~PhyloRuleDynData](#) ()

### Public Attributes

- [const](#) std::vector< double > \* [counts](#)
- [uint](#) [pos](#)
- [uint](#) [lb](#)
- [uint](#) [ub](#)
- [uint](#) [duplication](#)

### 7.25.1 Detailed Description

Definition at line 1414 of file phylo.hpp.

### 7.25.2 Constructor & Destructor Documentation

#### 7.25.2.1 PhyloRuleDynData()

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

Definition at line 1421 of file phylo.hpp.

### 7.25.2.2 ~PhyloRuleDynData()

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

Definition at line 1430 of file phylo.hpp.

## 7.25.3 Member Data Documentation

### 7.25.3.1 counts

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

Definition at line 1416 of file phylo.hpp.

### 7.25.3.2 duplication

```
uint PhyloRuleDynData::duplication
```

Definition at line 1420 of file phylo.hpp.

### 7.25.3.3 lb

```
uint PhyloRuleDynData::lb
```

Definition at line 1418 of file phylo.hpp.

### 7.25.3.4 pos

```
uint PhyloRuleDynData::pos
```

Definition at line 1417 of file phylo.hpp.

### 7.25.3.5 ub

```
uint PhyloRuleDynData::ub
```

Definition at line 1419 of file phylo.hpp.

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

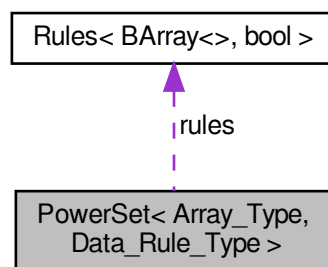
- [include/barry/counters/phylo.hpp](#)

## 7.26 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](#)

### 7.26.1 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.

### 7.26.2 Constructor & Destructor Documentation

#### 7.26.2.1 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.

#### 7.26.2.2 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.



**7.26.2.3 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.

**7.26.2.4 ~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.

**7.26.3 Member Function Documentation****7.26.3.1 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.

**7.26.3.2 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.

**7.26.3.3 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.

#### 7.26.3.4 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.

#### 7.26.3.5 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.

#### 7.26.3.6 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.

#### 7.26.3.7 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.

#### 7.26.3.8 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.

### 7.26.3.9 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.

### 7.26.3.10 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.

### 7.26.3.11 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.

### 7.26.3.12 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.

## 7.26.4 Member Data Documentation

### 7.26.4.1 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.

#### 7.26.4.2 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.

#### 7.26.4.3 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.

#### 7.26.4.4 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.

#### 7.26.4.5 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.

#### 7.26.4.6 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.

#### 7.26.4.7 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.

#### 7.26.4.8 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

## 7.27 Progress Class Reference

A simple progress bar.

```
#include <progress.hpp>
```

### Public Member Functions

- [Progress](#) (int n\_, int width\_)
- [~Progress](#) ()
- void [next](#) ()
- void [end](#) ()

#### 7.27.1 Detailed Description

A simple progress bar.

Definition at line 11 of file progress.hpp.

#### 7.27.2 Constructor & Destructor Documentation

### 7.27.2.1 Progress()

```
Progress::Progress (
    int n_,
    int width_ ) [inline]
```

Definition at line 30 of file progress.hpp.

### 7.27.2.2 ~Progress()

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

Definition at line 23 of file progress.hpp.

## 7.27.3 Member Function Documentation

### 7.27.3.1 end()

```
void Progress::end ( ) [inline]
```

Definition at line 52 of file progress.hpp.

### 7.27.3.2 next()

```
void Progress::next ( ) [inline]
```

Definition at line 41 of file progress.hpp.

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

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

## 7.28 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*

<code>fun_</code>	<i>A function of type <code>Rule_fun_type</code>.</i>
<code>dat_</code>	<i>Data pointer to be passed to <code>fun_</code></i>
<code>delete_↔ dat_</code>	<i>When <code>true</code>, the <code>Rule</code> destructor will delete the pointer, if defined.</i>

- `Rule()`
- `Rule(Rule_fun_type< Array_Type, Data_Type > fun_, Data_Type *dat_ = nullptr, bool delete_dat_ = false)`

**7.28.1 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 <code>BArray</code> .
<i>Data_Type</i>	Any type.

Definition at line 22 of file `rules-bones.hpp`.

**7.28.2 Constructor & Destructor Documentation****7.28.2.1 Rule() [1/2]**

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

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

**7.28.2.2 Rule() [2/2]**

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Rule< Array_Type, Data_Type >::Rule (
    Rule_fun_type< Array_Type, Data_Type > fun_,
    Data_Type * dat_ = nullptr,
    bool delete_dat_ = false ) [inline]
```

Definition at line 42 of file `rules-bones.hpp`.

### 7.28.2.3 ~Rule()

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

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

## 7.28.3 Member Function Documentation

### 7.28.3.1 D()

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

Read/Write access to the data.

### 7.28.3.2 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

## 7.29 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)

**7.29.1 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 <a href="#">BArray</a>
<i>Data_Type</i>	Any type.

Definition at line 68 of file rules-bones.hpp.

**7.29.2 Constructor & Destructor Documentation****7.29.2.1 Rules() [1/2]**

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::Rules ( ) [inline]
```

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

**7.29.2.2 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.

### 7.29.2.3 ~Rules()

```
template<typename Array_Type , typename Data_Type >
Rules< Array_Type, Data_Type >::~~Rules ( ) [inline]
```

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

## 7.29.3 Member Function Documentation

### 7.29.3.1 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.

### 7.29.3.2 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.

### 7.29.3.3 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.

### 7.29.3.4 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.

### 7.29.3.5 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 <a href="#">BArray</a> .
<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.

### 7.29.3.6 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 <a href="#">BArray</a> 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.

### 7.29.3.7 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.

### 7.29.3.8 size()

```
template<typename Array_Type , typename Data_Type >
uint Rules< Array_Type, Data_Type >::size ( ) const [inline], [noexcept]
```

Definition at line 85 of file rules-bones.hpp.

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

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

## 7.30 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

### 7.30.1 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.

## 7.30.2 Constructor & Destructor Documentation

### 7.30.2.1 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

<a href="#">Array</a> ↔	A const pointer to a <a href="#">BArray</a> .
—	

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

### 7.30.2.2 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.

### 7.30.2.3 ~StatsCounter()

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

## 7.30.3 Member Function Documentation

### 7.30.3.1 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_ )
```

**7.30.3.2 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_ )
```

**7.30.3.3 count\_all()**

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
std::vector< double > StatsCounter< Array_Type, Data_Type >::count_all ( )
```

**7.30.3.4 count\_current()**

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
void StatsCounter< Array_Type, Data_Type >::count_current (
    uint i,
    uint j )
```

**7.30.3.5 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.

**7.30.3.6 get\_counters()**

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

**7.30.3.7 get\_descriptions()**

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

### 7.30.3.8 get\_names()

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

### 7.30.3.9 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

<code>Array_↔</code> —	A pointer to an array of class Array_Type.
---------------------------	--

### 7.30.3.10 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](#)

## 7.31 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`

### 7.31.1 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`.

### 7.31.2 Constructor & Destructor Documentation

#### 7.31.2.1 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`.

**7.31.2.2 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.

**7.31.2.3 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.

**7.31.2.4 ~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.

**7.31.3 Member Function Documentation****7.31.3.1 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_ )
```

### 7.31.3.2 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_ )
```

### 7.31.3.3 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_ )
```

### 7.31.3.4 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_ )
```

### 7.31.3.5 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_ )
```

### 7.31.3.6 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_ )
```

### 7.31.3.7 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.

#### 7.31.3.8 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 )
```

#### 7.31.3.9 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.

#### 7.31.3.10 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
```

#### 7.31.3.11 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
```

### 7.31.3.12 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.

### 7.31.3.13 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
```

### 7.31.3.14 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).

### 7.31.3.15 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).

### 7.31.3.16 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 )
```

#### 7.31.3.17 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
```

#### 7.31.3.18 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
( )
```

#### 7.31.3.19 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_ )
```

#### 7.31.3.20 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_ )
```

#### 7.31.3.21 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_ )
```

### 7.31.3.22 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_ )
```

## 7.31.4 Member Data Documentation

### 7.31.4.1 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.

### 7.31.4.2 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.

### 7.31.4.3 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.



#### 7.31.4.4 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.

#### 7.31.4.5 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.

#### 7.31.4.6 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.

#### 7.31.4.7 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.

#### 7.31.4.8 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.

#### 7.31.4.9 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.

#### 7.31.4.10 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](#)

## 7.32 vecHasher< T > Struct Template Reference

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

### Public Member Functions

- `std::size_t operator() (std::vector< T > const &dat) const noexcept`

#### 7.32.1 Detailed Description

```
template<typename T>
struct vecHasher< T >
```

Definition at line 94 of file typedefs.hpp.

#### 7.32.2 Member Function Documentation

##### 7.32.2.1 operator()()

```
template<typename T >
std::size_t vecHasher< T >::operator() (
    std::vector< T > const & dat ) const [inline], [noexcept]
```

Definition at line 95 of file typedefs.hpp.

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

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

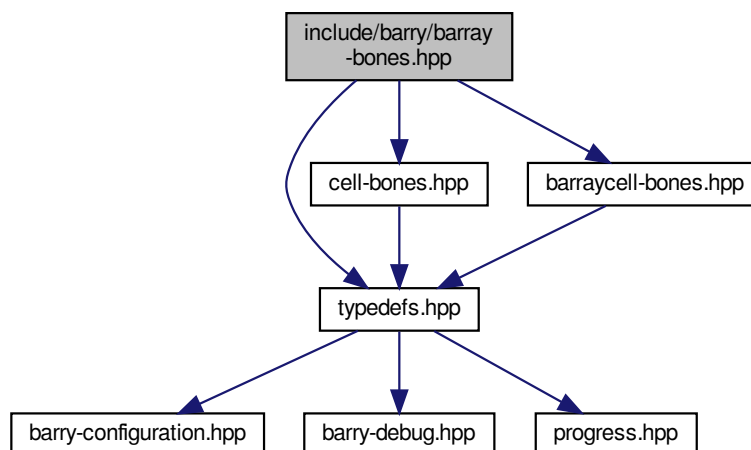
## Chapter 8

# File Documentation

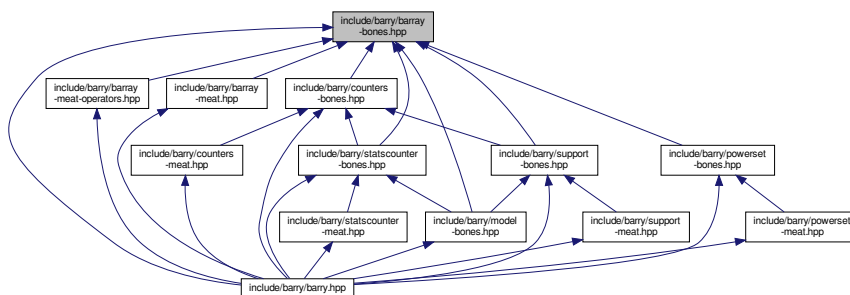
### 8.1 include/barry/barray-bones.hpp File Reference

```
#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

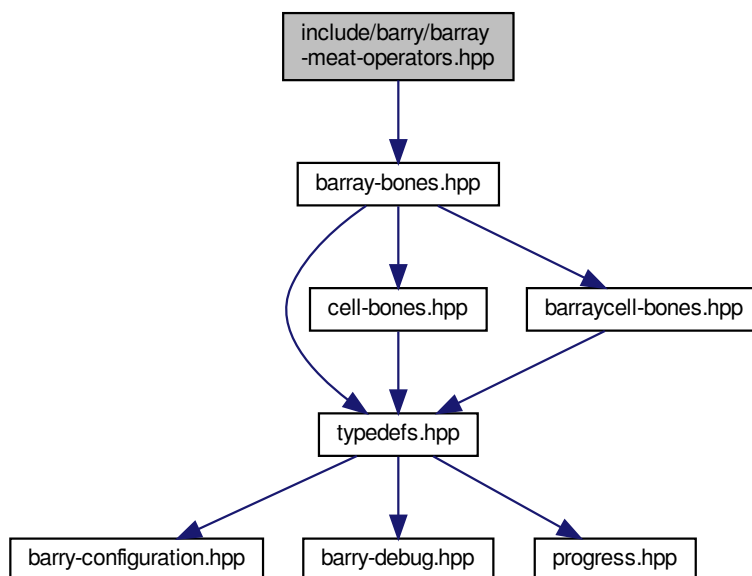
### Classes

- class [ConstBArrayRowIter< Cell\\_Type, Data\\_Type >](#)

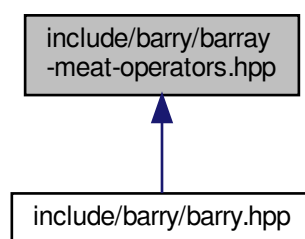
## 8.3 include/barry/barray-meat-operators.hpp File Reference

```
#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 BARRAY_TYPE() BArray<Cell_Type, Data_Type>`
- `#define BARRAY_TEMPLATE_ARGS() <typename Cell_Type, typename Data_Type>`
- `#define BARRAY_TEMPLATE(a, b) template BARRAY_TEMPLATE_ARGS() inline a BARRAY_TYPE()::b`
- `#define ROW(a) this->el_ij[a]`
- `#define COL(a) this->el_ji[a]`

## Functions

- template `BARRAY_TEMPLATE_ARGS` () inline void `checkdim_(const BARRAY_TYPE()` &lhs
- template `const BARRAY_TYPE` () &rhs)
- `BARRAY_TEMPLATE` (`BARRAY_TYPE()`&, `operator+=`)(`const BArray`< `Cell_Type`
- `for` (`uint i=0`;`i`< `nrow()`;`++i`) `for`(`uint j=0`;`j`=`el[POS(i, j)]`
- `j`< `ncol()`;`++j`) `this-> operator()` (`i, j`)+
- `BARRAY_TEMPLATE` (`BARRAY_TYPE()`&, `operator+=`)(`const Cell_Type` &rhs)
- `BARRAY_TEMPLATE` (`BARRAY_TYPE()`&, `operator-=`)(`const BArray`< `Cell_Type`
- `BARRAY_TEMPLATE` (`BARRAY_TYPE()`&, `operator-=`)(`const Cell_Type` &rhs)
- `BARRAY_TEMPLATE` (`BARRAY_TYPE()`&, `operator*=`)(`const Cell_Type` &rhs)
- `BARRAY_TEMPLATE` (`BARRAY_TYPE()`&, `operator/=`)(`const Cell_Type` &rhs)

## Variables

- `Data_Type` & `rhs`
- `return` \* `this`

## 8.3.1 Macro Definition Documentation

### 8.3.1.1 BARRAY\_TEMPLATE

```
#define BARRAY_TEMPLATE(
    a,
    b )  template BARRAY_TEMPLATE_ARGS() inline a BARRAY_TYPE():b
```

Definition at line 11 of file `barray-meat-operators.hpp`.

### 8.3.1.2 BARRAY\_TEMPLATE\_ARGS

```
template BARRAY_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

Definition at line 9 of file `barray-meat-operators.hpp`.

### 8.3.1.3 BARRAY\_TYPE

```
template Data_Type BARRAY_TYPE( ) BArray<Cell_Type, Data_Type>
```

Definition at line 7 of file `barray-meat-operators.hpp`.

#### 8.3.1.4 BARRY\_BARRAY\_MEAT\_OPERATORS\_HPP

```
#define BARRY_BARRAY_MEAT_OPERATORS_HPP 1
```

Definition at line 5 of file barray-meat-operators.hpp.

#### 8.3.1.5 COL

```
#define COL(  
    a ) this->el\_ji[a]
```

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

#### 8.3.1.6 ROW

```
#define ROW(  
    a ) this->el\_ij[a]
```

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

### 8.3.2 Function Documentation

#### 8.3.2.1 BARRAY\_TEMPLATE() [1/6]

```
BARRAY_TEMPLATE (  
    BARRAY\_TYPE() & ,  
    operator* ) const &
```

Definition at line 88 of file barray-meat-operators.hpp.

#### 8.3.2.2 BARRAY\_TEMPLATE() [2/6]

```
BARRAY_TEMPLATE (  
    BARRAY\_TYPE() & ,  
    operator+ ) const
```

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

```
BARRAY_TEMPLATE (
    BARRAY_TYPE() & ,
    operator+ ) const &
```

Definition at line 46 of file barray-meat-operators.hpp.

### 8.3.2.4 BARRAY\_TEMPLATE() [4/6]

```
BARRAY_TEMPLATE (
    BARRAY_TYPE() & ,
    operator- ) const
```

### 8.3.2.5 BARRAY\_TEMPLATE() [5/6]

```
BARRAY_TEMPLATE (
    BARRAY_TYPE() & ,
    operator- ) const &
```

Definition at line 75 of file barray-meat-operators.hpp.

### 8.3.2.6 BARRAY\_TEMPLATE() [6/6]

```
BARRAY_TEMPLATE (
    BARRAY_TYPE() & ,
    operator/ ) const &
```

Definition at line 105 of file barray-meat-operators.hpp.

### 8.3.2.7 BARRAY\_TEMPLATE\_ARGS()

```
template BARRAY_TEMPLATE_ARGS ( ) const &
```

### 8.3.2.8 BARRAY\_TYPE()

```
template const BARRAY_TYPE ( ) &
```

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



### 8.3.2.9 for()

```
for ( ) = el[POS(i, j)] [pure virtual]
```

Definition at line 66 of file barray-meat-operators.hpp.

### 8.3.2.10 operator()

```
j< ncol(); ++j) this-> operator() (
    i ,
    j )
```

## 8.3.3 Variable Documentation

### 8.3.3.1 rhs

Data\_Type & rhs

#### Initial value:

```
{
    checkdim_(*this, rhs)
```

Definition at line 33 of file barray-meat-operators.hpp.

### 8.3.3.2 this

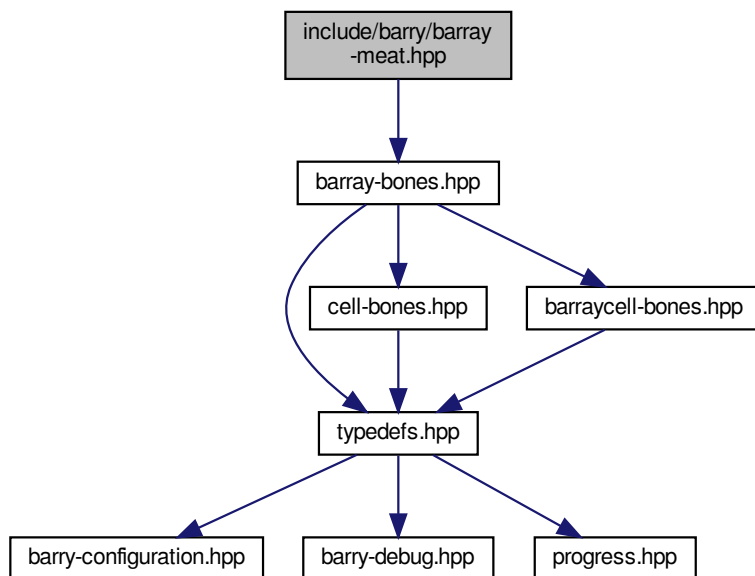
```
return * this
```

Definition at line 43 of file barray-meat-operators.hpp.

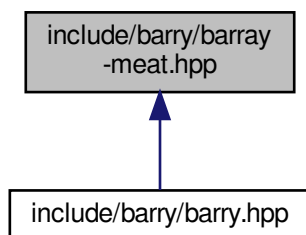
## 8.4 include/barry/barray-meat.hpp File Reference

```
#include "barray-bones.hpp"
```

Include dependency graph for barray-meat.hpp:



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



### Macros

- `#define BARRAY_TYPE() BArray<Cell_Type, Data_Type>`
- `#define BARRAY_TEMPLATE_ARGS() <typename Cell_Type, typename Data_Type>`
- `#define BARRAY_TEMPLATE(a, b) template BARRAY_TEMPLATE_ARGS() inline a BARRAY_TYPE()::b`
- `#define ROW(a) this->el_ij[a]`
- `#define COL(a) this->el_ji[a]`

## Functions

- [BARRAY\\_TEMPLATE](#) ([BArray](#))([uint N\\_](#)
- [el\\_ij](#) [resize](#) ([N](#))
- [el\\_ji](#) [resize](#) ([M](#))
- [for](#) ([uint i=0u](#); [i](#) < [source.size\(\)](#); ++[i](#))
- [Data\\_Type](#) [bool M](#) ([Array\\_.M](#))
- [BARRAY\\_TEMPLATE](#) ([BARRAY\\_TYPE](#)() &, [operator=](#))([const BArray](#) < [Cell\\_Type](#)
- [BARRAY\\_TEMPLATE](#) ([BArray](#))([BARRAY\\_TYPE](#)() &&[x](#)) [noexcept](#)
- [BARRAY\\_TEMPLATE](#) ([BARRAY\\_TYPE](#)() &, [operator=](#))([BARRAY\\_TYPE](#)() &&[x](#)) [noexcept](#)
- [BARRAY\\_TEMPLATE](#) ([bool](#), [operator=](#))([const BARRAY\\_TYPE](#)() &[Array\\_](#))
- [BARRAY\\_TEMPLATE](#) ([BArray](#))()
- [BARRAY\\_TEMPLATE](#) ([void](#), [set\\_data](#))([Data\\_Type \\*data\\_](#)
- [BARRAY\\_TEMPLATE](#) ([Data\\_Type \\*](#), [D](#))()
- [BARRAY\\_TEMPLATE](#) ([void](#), [out\\_of\\_range](#))([uint i](#)
- [BARRAY\\_TEMPLATE](#) ([Cell\\_Type](#), [get\\_cell](#))([uint i](#)
- [if](#) ([ROW](#)([i](#)).size() == 0u) [return](#)([Cell\\_Type](#)) 0.0
- [if](#) ([search](#) != [ROW](#)([i](#)).end()) [return](#) [search](#) -> [second.value](#)
- [return](#) ([Cell\\_Type](#)) 0.0
- [BARRAY\\_TEMPLATE](#) ([std::vector](#) < [Cell\\_Type](#) >, [get\\_row\\_vec](#))([uint i](#)
- [std::vector](#) < [Cell\\_Type](#) > [ans](#) ([ncol\(\)](#), ([Cell\\_Type](#)) [false](#))
- [for](#) ([const auto](#) &[iter](#) : [row](#)([i](#), [false](#))) [ans](#)[[iter.first](#)]
- [BARRAY\\_TEMPLATE](#) ([void](#), [get\\_row\\_vec](#))([std](#)
- [BARRAY\\_TEMPLATE](#) ([BARRAY\\_TYPE](#)() &, [operator=](#))([const std](#)
- [BARRAY\\_TEMPLATE](#) ([void](#), [insert\\_cell](#))([uint i](#)
- [if](#) ([check\\_exists](#))
- [COL](#) ([j](#)).emplace([i](#)
- & [ROW](#) ([i](#))[[j](#)])
- [BARRAY\\_TEMPLATE](#) ([void](#), [swap\\_cells](#))([uint i0](#)
- [if](#) ([report](#) != nullptr) (\*[report](#))
- [if](#) ([check0](#) & [check1](#))
- [else if](#) (![check0](#) & [check1](#))
- [else if](#) ([check0](#) & ![check1](#))
- [BARRAY\\_TEMPLATE](#) ([void](#), [toggle\\_cell](#))([uint i](#)
- [BARRAY\\_TEMPLATE](#) ([void](#), [swap\\_rows](#))([uint i0](#)
- [if](#) ([ROW](#)([i0](#)).size() == 0u) [move0](#)
- [if](#) ([ROW](#)([i1](#)).size() == 0u) [move1](#)
- [if](#) (![move0](#) && ![move1](#)) [return](#)
- [ROW](#) ([i0](#)).swap([ROW](#)([i1](#)))
- [BARRAY\\_TEMPLATE](#) ([void](#), [swap\\_cols](#))([uint j0](#)
- [if](#) ([COL](#)([j0](#)).size() == 0u) [check0](#)
- [if](#) ([COL](#)([j1](#)).size() == 0u) [check1](#)
- [if](#) ([check0](#) && [check1](#))
- [else if](#) ([check0](#) && ![check1](#))
- [else if](#) (![check0](#) && [check1](#))
- [BARRAY\\_TEMPLATE](#) ([void](#), [zero\\_row](#))([uint i](#)
- [for](#) ([auto row](#) = [row0.begin\(\)](#); [row](#) != [row0.end\(\)](#); ++[row](#)) [rm\\_cell](#)([i](#)
- [BARRAY\\_TEMPLATE](#) ([void](#), [zero\\_col](#))([uint j](#)
- [if](#) ([COL](#)([j](#)).size() == 0u) [return](#)
- [BARRAY\\_TEMPLATE](#) ([void](#), [transpose](#))()
- [BARRAY\\_TEMPLATE](#) ([void](#), [clear](#))([bool hard](#))
- [BARRAY\\_TEMPLATE](#) ([void](#), [resize](#))([uint N\\_](#)
- [if](#) ([M](#) < [M](#)) [for](#) ([uint j](#) = [N\\_](#)

## Variables

- `uint M_`
- `uint const std::vector< uint > & source`
- `uint const std::vector< uint > const std::vector< uint > & target`
- `uint const std::vector< uint > const std::vector< uint > const std::vector< Cell_Type > & value`
- `uint const std::vector< uint > const std::vector< uint > const std::vector< Cell_Type > bool add`
- `if(source.size() !=value.size()) throw std N = N_`
- `M = M_`
- `return`
- `Data_Type & Array_`
- `Data_Type bool copy_data`
- `bool delete_data_`
- `data = data_`
- `delete_data = delete_data_`
- `uint j const`
- `uint j`
- `auto search = ROW(i).find(j)`
- `return ans`
- `uint const Cell< Cell_Type > & v`
- `uint const Cell< Cell_Type > bool check_bounds`
- `uint const Cell< Cell_Type > bool bool check_exists`
- `else`
- `NCells`
- `uint j0`
- `uint uint i1`
- `uint uint uint j1`
- `uint uint uint bool int int * report`
- `auto row0 = ROW(i)`
- `row first`
- `row false`
- `auto col0 = COL(j)`

## 8.4.1 Macro Definition Documentation

### 8.4.1.1 BARRAY\_TEMPLATE

```
#define BARRAY_TEMPLATE(
    a,
    b )  template BARRAY_TEMPLATE_ARGS() inline a BARRAY_TYPE()::b
```

Definition at line 11 of file `barray-meat.hpp`.

### 8.4.1.2 BARRAY\_TEMPLATE\_ARGS

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

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

### 8.4.1.3 BARRAY\_TYPE

```
#define BARRAY_TYPE( ) BArray<Cell_Type, Data_Type>
```

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

### 8.4.1.4 COL

```
#define COL(  
    a ) this->el_ji[a]
```

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

### 8.4.1.5 ROW

```
#define ROW(  
    a ) this->el_ij[a]
```

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

## 8.4.2 Function Documentation

### 8.4.2.1 ans()

```
std::vector< Cell_Type > ans (  
    ncol() ,  
    (Cell_Type) false )
```

### 8.4.2.2 BARRAY\_TEMPLATE() [1/23]

```
BARRAY_TEMPLATE (  
    BArray ) && [noexcept]
```

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

**8.4.2.3 BARRAY\_TEMPLATE()** [2/23]

```
BARRAY_TEMPLATE (
    BArray )
```

**8.4.2.4 BARRAY\_TEMPLATE()** [3/23]

```
BARRAY_TEMPLATE (
    ~ BArray )
```

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

**8.4.2.5 BARRAY\_TEMPLATE()** [4/23]

```
BARRAY_TEMPLATE (
    BARRAY_TYPE() & ,
    operator- ) const
```

Definition at line 580 of file barray-meat.hpp.

**8.4.2.6 BARRAY\_TEMPLATE()** [5/23]

```
BARRAY_TEMPLATE (
    BARRAY_TYPE() & ,
    operator ) && [noexcept]
```

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

**8.4.2.7 BARRAY\_TEMPLATE()** [6/23]

```
BARRAY_TEMPLATE (
    BARRAY_TYPE() & ,
    operator ) const
```

**8.4.2.8 BARRAY\_TEMPLATE()** [7/23]

```
BARRAY_TEMPLATE (
    bool ,
    operator == ) const &
```

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

#### 8.4.2.9 BARRAY\_TEMPLATE() [8/23]

```
BARRAY_TEMPLATE (
    Cell_Type ,
    get_cell )
```

#### 8.4.2.10 BARRAY\_TEMPLATE() [9/23]

```
BARRAY_TEMPLATE (
    Data_Type * ,
    D )
```

Definition at line 355 of file barray-meat.hpp.

#### 8.4.2.11 BARRAY\_TEMPLATE() [10/23]

```
BARRAY_TEMPLATE (
    std::vector< Cell_Type > ,
    get_row_vec )
```

#### 8.4.2.12 BARRAY\_TEMPLATE() [11/23]

```
BARRAY_TEMPLATE (
    void ,
    clear )
```

Definition at line 1113 of file barray-meat.hpp.

#### 8.4.2.13 BARRAY\_TEMPLATE() [12/23]

```
BARRAY_TEMPLATE (
    void ,
    get_row_vec )
```

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

**8.4.2.14 BARRAY\_TEMPLATE()** [13/23]

```
BARRAY_TEMPLATE (
    void ,
    insert_cell )
```

**8.4.2.15 BARRAY\_TEMPLATE()** [14/23]

```
BARRAY_TEMPLATE (
    void ,
    out_of_range )
```

**8.4.2.16 BARRAY\_TEMPLATE()** [15/23]

```
BARRAY_TEMPLATE (
    void ,
    resize )
```

**8.4.2.17 BARRAY\_TEMPLATE()** [16/23]

```
BARRAY_TEMPLATE (
    void ,
    set_data )
```

**8.4.2.18 BARRAY\_TEMPLATE()** [17/23]

```
BARRAY_TEMPLATE (
    void ,
    swap_cells )
```

**8.4.2.19 BARRAY\_TEMPLATE()** [18/23]

```
BARRAY_TEMPLATE (
    void ,
    swap_cols )
```



**8.4.2.20 BARRAY\_TEMPLATE()** [19/23]

```
BARRAY_TEMPLATE (
    void ,
    swap_rows )
```

**8.4.2.21 BARRAY\_TEMPLATE()** [20/23]

```
BARRAY_TEMPLATE (
    void ,
    toggle_cell )
```

**8.4.2.22 BARRAY\_TEMPLATE()** [21/23]

```
BARRAY_TEMPLATE (
    void ,
    transpose )
```

Definition at line 1052 of file barray-meat.hpp.

**8.4.2.23 BARRAY\_TEMPLATE()** [22/23]

```
BARRAY_TEMPLATE (
    void ,
    zero_col )
```

**8.4.2.24 BARRAY\_TEMPLATE()** [23/23]

```
BARRAY_TEMPLATE (
    void ,
    zero_row )
```

**8.4.2.25 COL()**

```
COL (
    j )
```

**8.4.2.26 for()** [1/3]

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

**8.4.2.27 for()** [2/3]

```
for (
    const auto &iter : rowi, false )
```

**8.4.2.28 for()** [3/3]

```
for ( )
```

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

**8.4.2.29 if()** [1/17]

```
else if (
    !check0 && check1 )
```

Definition at line 991 of file barray-meat.hpp.

**8.4.2.30 if()** [2/17]

```
else if (
    !check0 & check1 )
```

Definition at line 839 of file barray-meat.hpp.

**8.4.2.31 if()** [3/17]

```
if (
    !move0 && ! move1 )
```

**8.4.2.32 if()** [4/17]

```
else if (
    check0 &! check1 )
```

Definition at line 847 of file barray-meat.hpp.

**8.4.2.33 if()** [5/17]

```
else if (
    check0 &&! check1 )
```

Definition at line 982 of file barray-meat.hpp.

**8.4.2.34 if()** [6/17]

```
if (
    check0 && check1 )
```

Definition at line 955 of file barray-meat.hpp.

**8.4.2.35 if()** [7/17]

```
if (
    check0 & check1 )
```

Definition at line 821 of file barray-meat.hpp.

**8.4.2.36 if()** [8/17]

```
else if (
    check_exists == CHECK::BOTH )
```

Definition at line 662 of file barray-meat.hpp.

**8.4.2.37 if()** [9/17]

```
if (
    COL(j).size() == 0u )
```

**8.4.2.38 if() [10/17]**

```
if (
    COL(j0).size() == 0u )
```

**8.4.2.39 if() [11/17]**

```
if (
    COL(j1).size() == 0u )
```

**8.4.2.40 if() [12/17]**

```
else if ( ) = N_
```

Definition at line 80 of file barray-meat.hpp.

**8.4.2.41 if() [13/17]**

```
if (
    report !    = nullptr )
```

**8.4.2.42 if() [14/17]**

```
if (
    ROW(i).size() == 0u )
```

**8.4.2.43 if() [15/17]**

```
if (
    ROW(i0).size() == 0u )
```

**8.4.2.44 if() [16/17]**

```
if (
    ROW(i1).size() == 0u )
```

**8.4.2.45 if()** [17/17]

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

**8.4.2.46 M()**

```
Data_Type bool M (
    Array_ M )
```

Definition at line 130 of file barray-meat.hpp.

**8.4.2.47 resize()** [1/2]

```
el_ji resize (
    M )
```

**8.4.2.48 resize()** [2/2]

```
el_ij resize (
    N )
```

**8.4.2.49 return()**

```
return (
    Cell_Type )
```

**8.4.2.50 ROW()** [1/2]

```
& ROW (
    i )
```

**8.4.2.51 ROW()** [2/2]

```
ROW (
    i0 )
```

### 8.4.3 Variable Documentation

#### 8.4.3.1 add

```
uint const std::vector< uint > const std::vector< uint > bool add
```

**Initial value:**

```
{  
    if (source.size() != target.size())  
        throw std::length_error("-source- and -target- don't match on length.")
```

Definition at line 28 of file barray-meat.hpp.

#### 8.4.3.2 ans

```
return ans
```

Definition at line 432 of file barray-meat.hpp.

#### 8.4.3.3 Array\_

```
Data_Type & Array_
```

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

#### 8.4.3.4 check\_bounds

```
bool check_bounds
```

**Initial value:**

```
{  
    if (check_bounds) {  
        out_of_range(i0,0u);  
        out_of_range(i1,0u);  
    }  
  
    bool move0=true, move1=true
```

Definition at line 655 of file barray-meat.hpp.

#### 8.4.3.5 check\_exists

```
uint bool int check_exists
```

##### Initial value:

```
{  
    if (check_bounds)  
        out_of_range(i, j)
```

Definition at line 656 of file barray-meat.hpp.

#### 8.4.3.6 col0

```
auto col0 = COL(j)
```

Definition at line 1044 of file barray-meat.hpp.

#### 8.4.3.7 const

```
bool check_bounds const
```

##### Initial value:

```
{  
    if (i >= N)  
        throw std::range_error("The row is out of range.")
```

Definition at line 385 of file barray-meat.hpp.

#### 8.4.3.8 copy\_data

```
Data_Type bool copy_data
```

Definition at line 129 of file barray-meat.hpp.

#### 8.4.3.9 data

```
data = data_
```

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

#### 8.4.3.10 delete\_data

```
delete_data = delete_data_
```

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

#### 8.4.3.11 delete\_data\_

```
bool delete_data_
```

##### Initial value:

```
{  
    if ((data != nullptr) && delete_data)  
        delete data
```

Definition at line 342 of file barray-meat.hpp.

#### 8.4.3.12 else

```
else
```

##### Initial value:

```
{  
    ROW(i).insert(std::pair< uint, Cell<Cell_Type>>(j, v))
```

Definition at line 686 of file barray-meat.hpp.

#### 8.4.3.13 false

```
row false
```

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

#### 8.4.3.14 first

```
row first
```

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



**8.4.3.15 i1**

```
uint i1
```

Definition at line 759 of file barray-meat.hpp.

**8.4.3.16 j**

```
uint j
```

**Initial value:**

```
{  
    if (init_fun == nullptr)  
        return 0.0
```

Definition at line 397 of file barray-meat.hpp.

**8.4.3.17 j0**

```
uint j0
```

Definition at line 758 of file barray-meat.hpp.

**8.4.3.18 j1**

```
uint j1
```

Definition at line 759 of file barray-meat.hpp.

**8.4.3.19 M**

```
M = M_
```

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

#### 8.4.3.20 M\_

```
uint M_
```

##### Initial value:

```
{  
  
    if (N_ < N)  
        for (uint i = N_; i < N; ++i)  
            zero_row(i, false)
```

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

#### 8.4.3.21 N

```
if (source.size() != target.size()) throw std::if (source.size() != value.size()) throw std::N =  
N_
```

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

#### 8.4.3.22 NCells

```
NCells
```

Definition at line 690 of file barray-meat.hpp.

#### 8.4.3.23 report

```
uint uint uint bool int int* report
```

Definition at line 762 of file barray-meat.hpp.

#### 8.4.3.24 return

```
return
```

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

#### 8.4.3.25 row0

```
auto row0 = ROW(i)
```

Definition at line 1023 of file barray-meat.hpp.

#### 8.4.3.26 search

```
auto search = ROW(i).find(j)
```

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

#### 8.4.3.27 source

```
uint const std::vector< uint > & source
```

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

#### 8.4.3.28 target

```
uint const std::vector< uint > const std::vector< uint > & target
```

Definition at line 26 of file barray-meat.hpp.

#### 8.4.3.29 v

```
uint Cell_Type v
```

Definition at line 654 of file barray-meat.hpp.

#### 8.4.3.30 value

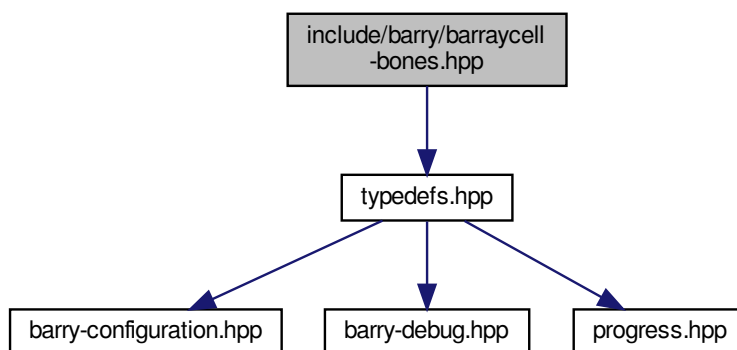
```
uint const std::vector< uint > const std::vector< uint > const std::vector< Cell_Type >&  
value
```

Definition at line 27 of file barray-meat.hpp.

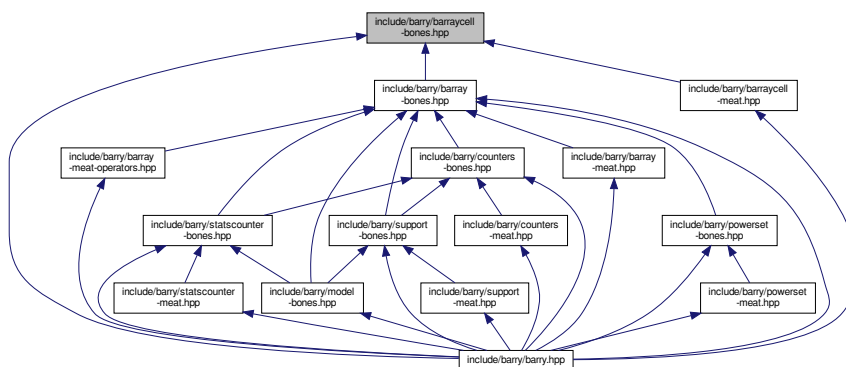
## 8.5 include/barry/barraycell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barraycell-bones.hpp:



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



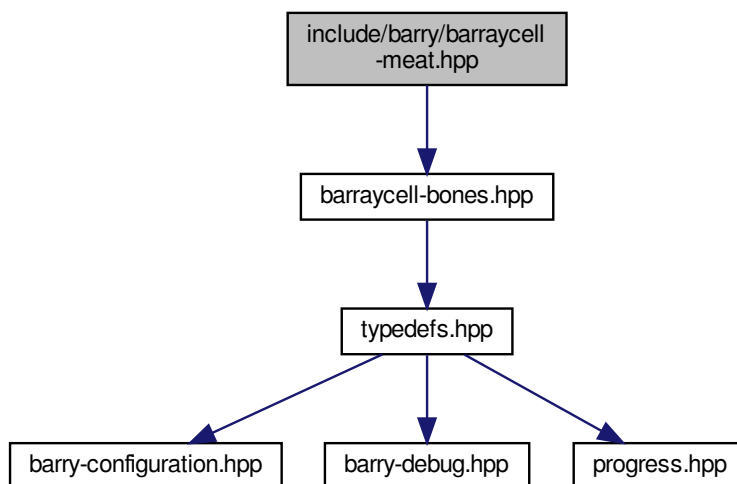
### Classes

- class [BArrayCell< Cell\\_Type, Data\\_Type >](#)
- class [BArrayCell\\_const< Cell\\_Type, Data\\_Type >](#)

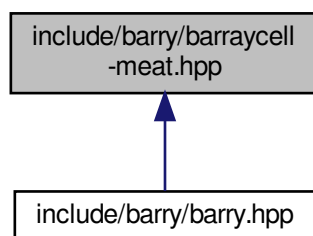
## 8.6 include/barry/barraycell-meal.hpp File Reference

```
#include "barraycell-bones.hpp"
```

Include dependency graph for barraycell-meat.hpp:



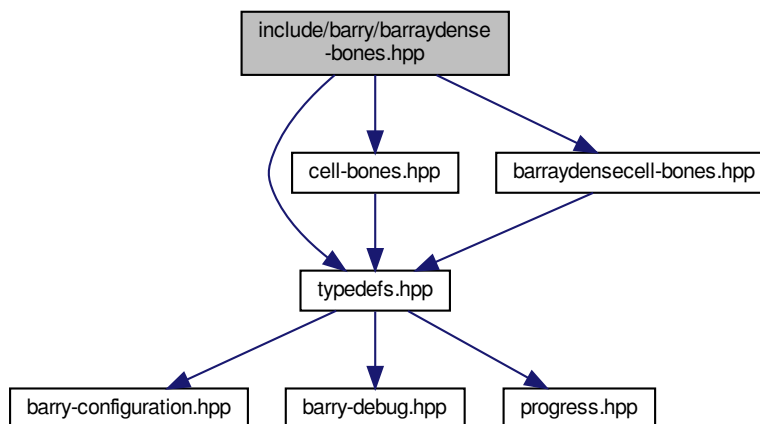
This graph shows which files directly or indirectly include this file:



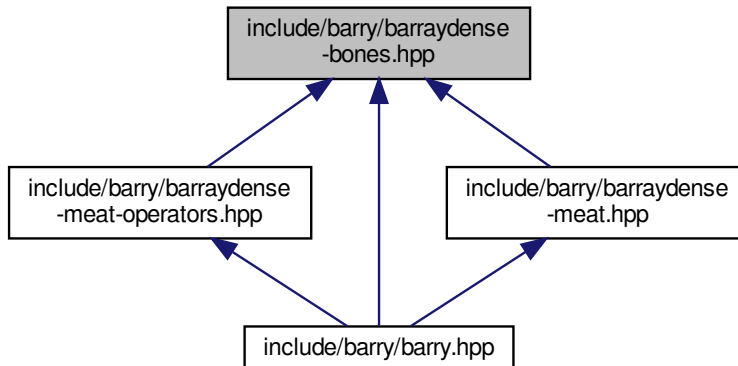
## 8.7 include/barry/barraydense-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "cell-bones.hpp"
#include "barraydensecell-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.

## Macros

- `#define BARRY_BARRAYDENSE_BONES_HPP` 1

## 8.7.1 Macro Definition Documentation

### 8.7.1.1 BARRY\_BARRAYDENSE\_BONES\_HPP

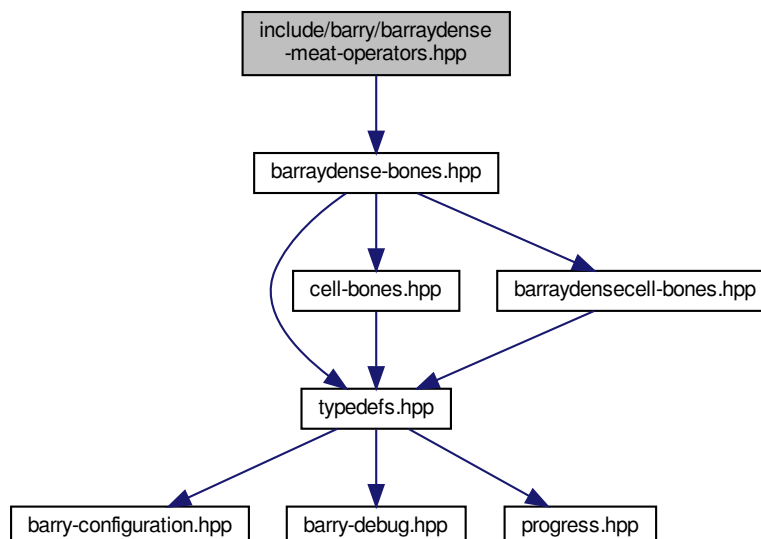
```
#define BARRY_BARRAYDENSE_BONES_HPP 1
```

Definition at line 8 of file barraydense-bones.hpp.

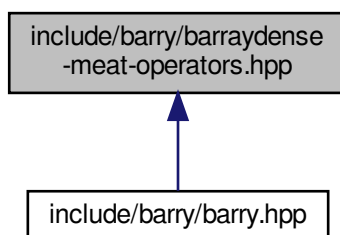
## 8.8 include/barry/barraydense-meat-operators.hpp File Reference

```
#include "barraydense-bones.hpp"
```

Include dependency graph for barraydense-meat-operators.hpp:



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



## Macros

- `#define BARRY_BARRAY_MEAT_OPERATORS_HPP 1`
- `#define BDENSE_TYPE() BArrayDense<Cell_Type, Data_Type>`
- `#define BDENSE_TEMPLATE_ARGS() <typename Cell_Type, typename Data_Type>`
- `#define BDENSE_TEMPLATE(a, b) template BDENSE_TEMPLATE_ARGS() inline a BDENSE_TYPE():b`
- `#define ROW(a) this->el_ij[a]`
- `#define COL(a) this->el_ji[a]`
- `#define POS(a, b) (b)*N + (a)`
- `#define POS_N(a, b, c) (b)*(c) + (a)`

## Functions

- `template BDENSE_TEMPLATE_ARGS() inline void checkdim_(const BDENSE_TYPE() &lhs`
- `template const BDENSE_TYPE() &rhs)`
- `BDENSE_TEMPLATE(BDENSE_TYPE() &, operator+=)(const BDENSE_TYPE() &rhs)`
- `BDENSE_TEMPLATE(BDENSE_TYPE() &, operator-=)(const BDENSE_TYPE() &rhs)`
- `BDENSE_TEMPLATE(BDENSE_TYPE() &, operator*=)(const Cell_Type &rhs)`
- `BDENSE_TEMPLATE(BDENSE_TYPE() &, operator/=)(const Cell_Type &rhs)`

## 8.8.1 Macro Definition Documentation

### 8.8.1.1 BARRY\_BARRAY\_MEAT\_OPERATORS\_HPP

```
#define BARRY_BARRAY_MEAT_OPERATORS_HPP 1
```

Definition at line 5 of file `barraydense-meat-operators.hpp`.

### 8.8.1.2 BDENSE\_TEMPLATE

```
#define BDENSE_TEMPLATE(  
    a,  
    b ) template BDENSE_TEMPLATE_ARGS() inline a BDENSE_TYPE():b
```

Definition at line 11 of file `barraydense-meat-operators.hpp`.

### 8.8.1.3 BDENSE\_TEMPLATE\_ARGS

```
template BDENSE_TEMPLATE_ARGS( ) <typename Cell_Type, typename Data_Type>
```

Definition at line 9 of file `barraydense-meat-operators.hpp`.



#### 8.8.1.4 BDENSE\_TYPE

```
template Data_Type BDENSE_TYPE( ) BArrayDense<Cell_Type, Data_Type>
```

Definition at line 7 of file barraydense-meat-operators.hpp.

#### 8.8.1.5 COL

```
#define COL(  
    a ) this->el_ji[a]
```

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

#### 8.8.1.6 POS

```
#define POS(  
    a,  
    b ) (b)*N + (a)
```

Definition at line 16 of file barraydense-meat-operators.hpp.

#### 8.8.1.7 POS\_N

```
#define POS_N(  
    a,  
    b,  
    c ) (b)*(c) + (a)
```

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

#### 8.8.1.8 ROW

```
#define ROW(  
    a ) this->el_ij[a]
```

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

### 8.8.2 Function Documentation

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

```
BDENSE_TEMPLATE (
    BDENSE_TYPE() & ,
    operator* ) const &
```

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

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

```
BDENSE_TEMPLATE (
    BDENSE_TYPE() & ,
    operator+ ) const &
```

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

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

```
BDENSE_TEMPLATE (
    BDENSE_TYPE() & ,
    operator- ) const &
```

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

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

```
BDENSE_TEMPLATE (
    BDENSE_TYPE() & ,
    operator/ ) const &
```

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

#### 8.8.2.5 BDENSE\_TEMPLATE\_ARGS()

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

#### 8.8.2.6 BDENSE\_TYPE()

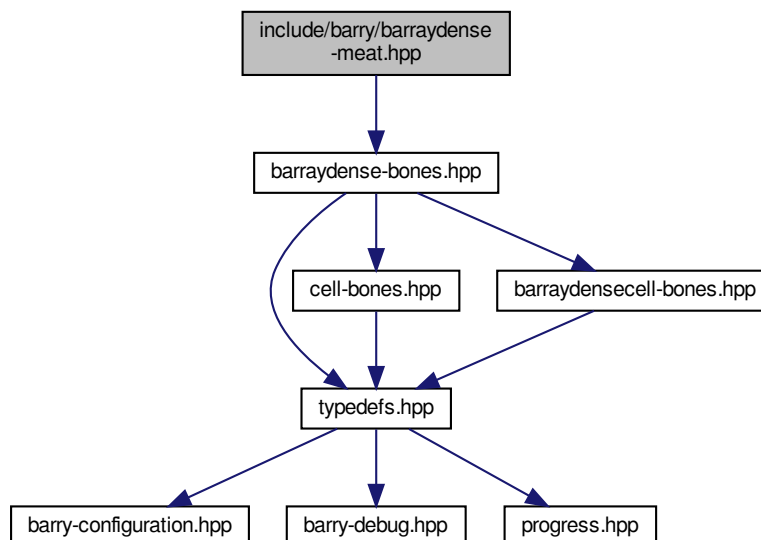
```
template const BDENSE_TYPE ( ) &
```

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

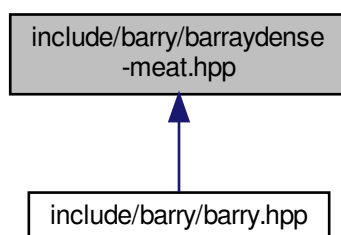
## 8.9 include/barry/barraydense-meat.hpp File Reference

```
#include "barraydense-bones.hpp"
```

Include dependency graph for barraydense-meat.hpp:



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



### Macros

- `#define BDENSE_TYPE() BArrayDense<Cell_Type, Data_Type>`
- `#define BDENSE_TEMPLATE_ARGS() <typename Cell_Type, typename Data_Type>`
- `#define BDENSE_TEMPLATE(a, b) template BDENSE_TEMPLATE_ARGS() inline a BDENSE_TYPE()::b`
- `#define ROW(a) this->el_ij[a]`
- `#define COL(a) this->el_ji[a]`
- `#define POS(a, b) (b)*N + (a)`
- `#define POS_N(a, b, c) (b)*(c) + (a)`
- `#define ZERO_CELL Cell< Cell_Type > ( (Cell_Type) 0.0, false, false)`

## Functions

- [BDENSE\\_TEMPLATE](#) ([, BArrayDense](#))([uint N\\_](#)
- [el resize](#) ([N, M](#))
- [for](#) ([uint i=0u;i< source.size\(\);++i](#))
- [el resize](#) ([N \\*M](#))
- [BDENSE\\_TEMPLATE](#) ([, BArrayDense](#))([const BDENSE\\_TYPE\(\)](#) &[Array\\_](#)
- [bool M](#) ([Array\\_.M](#))
- [BDENSE\\_TEMPLATE](#) ([BDENSE\\_TYPE\(\)](#) &, [operator=](#))([const BDENSE\\_TYPE\(\)](#) &[Array\\_](#))
- [BDENSE\\_TEMPLATE](#) ([, BArrayDense](#))([BDENSE\\_TYPE\(\)](#) &&[x](#)) [noexcept](#)
- [BDENSE\\_TEMPLATE](#) ([BDENSE\\_TYPE\(\)](#) &, [operator=](#))([BDENSE\\_TYPE\(\)](#) &&[x](#)) [noexcept](#)
- [BDENSE\\_TEMPLATE](#) ([bool, operator=](#))([const BDENSE\\_TYPE\(\)](#) &[Array\\_](#))
- [BDENSE\\_TEMPLATE](#) ([, ~BArrayDense](#))()
- [BDENSE\\_TEMPLATE](#) ([void, set\\_data](#))([Data\\_Type \\*data\\_](#)
- [BDENSE\\_TEMPLATE](#) ([Data\\_Type \\*, D](#))()
- [BDENSE\\_TEMPLATE](#) ([const Data\\_Type \\*, D](#))() [const](#)
- [BDENSE\\_TEMPLATE](#) ([void, out\\_of\\_range](#))([uint i](#)
- [BDENSE\\_TEMPLATE](#) ([Cell\\_Type, get\\_cell](#))([uint i](#)
- [BDENSE\\_TEMPLATE](#) ([std::vector< Cell\\_Type >, get\\_row\\_vec](#))([uint i](#)
- [std::vector< Cell\\_Type >](#) [ans](#) ([ncol\(\)](#), [static\\_cast< Cell\\_Type >](#) ([false](#)))
- [BDENSE\\_TEMPLATE](#) ([void, get\\_row\\_vec](#))([std](#)
- [BDENSE\\_TEMPLATE](#) ([BDENSE\\_TYPE\(\)](#) &, [operator=](#))([const std](#)
- [BDENSE\\_TEMPLATE](#) ([void, insert\\_cell](#))([uint i](#)
- [if](#) ([check\\_exists](#))
- [BDENSE\\_TEMPLATE](#) ([void, swap\\_cells](#))([uint i0](#)
- [if](#) ([report !=nullptr](#))([\\*report](#))
- [if](#) ([check0 &check1](#))
- [else if](#) ([!check0 &check1](#))
- [else if](#) ([check0 &!check1](#))
- [BDENSE\\_TEMPLATE](#) ([void, toggle\\_cell](#))([uint i](#)
- [BDENSE\\_TEMPLATE](#) ([void, swap\\_rows](#))([uint i0](#)
- [BDENSE\\_TEMPLATE](#) ([void, swap\\_cols](#))([uint j0](#)
- [BDENSE\\_TEMPLATE](#) ([void, zero\\_row](#))([uint i](#)
- [if](#) ([NCells==0u](#)) [return](#)
- [BDENSE\\_TEMPLATE](#) ([void, zero\\_col](#))([uint j](#)
- [BDENSE\\_TEMPLATE](#) ([void, transpose](#))()
- [BDENSE\\_TEMPLATE](#) ([void, clear](#))([bool hard](#))
- [BDENSE\\_TEMPLATE](#) ([void, resize](#))([uint N\\_](#)
- [el resize](#) ([N\\_ \\*M\\_, ZERO\\_CELL](#))
- [BDENSE\\_TEMPLATE](#) ([void, reserve](#))()
- [BDENSE\\_TEMPLATE](#) ([void, print](#))() [const](#)

## Variables

- [uint M\\_](#)
- [uint const](#) [std::vector< uint >](#) & [source](#)
- [uint const](#) [std::vector< uint >](#) [const](#) [std::vector< uint >](#) & [target](#)
- [uint const](#) [std::vector< uint >](#) [const](#) [std::vector< uint >](#) [const](#) [std::vector< Cell\\_Type >](#) & [value](#)
- [uint const](#) [std::vector< uint >](#) [const](#) [std::vector< uint >](#) [const](#) [std::vector< Cell\\_Type >](#) [bool add](#)
- [if](#)([source.size\(\)](#) !=[value.size\(\)](#)) [throw std](#) [N](#) = [N\\_](#)
- [M](#) = [M\\_](#)
- [return](#)
- [bool](#) [copy\\_data](#)
- [bool](#) [delete\\_data\\_](#)

- `data = data_`
- `delete_data = delete_data_`
- `uint j const`
- `uint j`
- `return ans`
- `uint const Cell< Cell_Type > & v`
- `uint const Cell< Cell_Type > bool check_bounds`
- `uint const Cell< Cell_Type > bool bool check_exists`
- `else`
- `el [POS(i, j)] = tmp`
- `NCells`
- `uint j0`
- `uint uint i1`
- `uint uint uint j1`
- `uint uint uint bool int int * report`
- `col`
- `false`

## 8.9.1 Macro Definition Documentation

### 8.9.1.1 BDENSE\_TEMPLATE

```
#define BDENSE_TEMPLATE(
    a,
    b )  template BDENSE_TEMPLATE_ARGS() inline a BDENSE_TYPE()::b
```

Definition at line 11 of file barraydense-meat.hpp.

### 8.9.1.2 BDENSE\_TEMPLATE\_ARGS

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

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

### 8.9.1.3 BDENSE\_TYPE

```
#define BDENSE_TYPE( ) BArrayDense<Cell_Type, Data_Type>
```

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

#### 8.9.1.4 COL

```
#define COL(  
    a )  this->el_ji[a]
```

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

#### 8.9.1.5 POS

```
#define POS(  
    a,  
    b )  (b)*N + (a)
```

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

#### 8.9.1.6 POS\_N

```
#define POS_N(  
    a,  
    b,  
    c )  (b)*(c) + (a)
```

Definition at line 17 of file baraydense-meat.hpp.

#### 8.9.1.7 ROW

```
#define ROW(  
    a )  this->el_ij[a]
```

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

#### 8.9.1.8 ZERO\_CELL

```
#define ZERO_CELL Cell< Cell_Type >( (Cell_Type) 0.0, false, false)
```

Definition at line 22 of file baraydense-meat.hpp.

### 8.9.2 Function Documentation

### 8.9.2.1 ans()

```
std::vector< Cell_Type > ans (
    ncol() ,
    static_cast< Cell_Type > false )
```

### 8.9.2.2 BDENSE\_TEMPLATE() [1/27]

```
BDENSE_TEMPLATE (
    BArrayDense ) && [noexcept]
```

Definition at line 209 of file barraydense-meat.hpp.

### 8.9.2.3 BDENSE\_TEMPLATE() [2/27]

```
BDENSE_TEMPLATE (
    BArrayDense ) const &
```

### 8.9.2.4 BDENSE\_TEMPLATE() [3/27]

```
BDENSE_TEMPLATE (
    BArrayDense )
```

### 8.9.2.5 BDENSE\_TEMPLATE() [4/27]

```
BDENSE_TEMPLATE (
    ~ BArrayDense )
```

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

### 8.9.2.6 BDENSE\_TEMPLATE() [5/27]

```
BDENSE_TEMPLATE (
    BDENSE_TYPE() & ,
    operator- ) const
```

Definition at line 519 of file barraydense-meat.hpp.

#### 8.9.2.7 BDENSE\_TEMPLATE() [6/27]

```
BDENSE_TEMPLATE (
    BDENSE_TYPE() & ,
    operator ) && [noexcept]
```

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

#### 8.9.2.8 BDENSE\_TEMPLATE() [7/27]

```
BDENSE_TEMPLATE (
    BDENSE_TYPE() & ,
    operator ) const &
```

Definition at line 162 of file bararraydense-meat.hpp.

#### 8.9.2.9 BDENSE\_TEMPLATE() [8/27]

```
BDENSE_TEMPLATE (
    bool ,
    operator == ) const &
```

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

#### 8.9.2.10 BDENSE\_TEMPLATE() [9/27]

```
BDENSE_TEMPLATE (
    Cell_Type ,
    get_cell )
```

#### 8.9.2.11 BDENSE\_TEMPLATE() [10/27]

```
BDENSE_TEMPLATE (
    const Data_Type * ,
    D ) const
```

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



**8.9.2.12 BDENSE\_TEMPLATE()** [11/27]

```
BDENSE_TEMPLATE (
    Data_Type * ,
    D )
```

Definition at line 306 of file barraydense-meat.hpp.

**8.9.2.13 BDENSE\_TEMPLATE()** [12/27]

```
BDENSE_TEMPLATE (
    std::vector< Cell_Type > ,
    get_row_vec )
```

**8.9.2.14 BDENSE\_TEMPLATE()** [13/27]

```
BDENSE_TEMPLATE (
    void ,
    clear )
```

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

**8.9.2.15 BDENSE\_TEMPLATE()** [14/27]

```
BDENSE_TEMPLATE (
    void ,
    get_row_vec )
```

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

**8.9.2.16 BDENSE\_TEMPLATE()** [15/27]

```
BDENSE_TEMPLATE (
    void ,
    insert_cell )
```

**8.9.2.17 BDENSE\_TEMPLATE()** [16/27]

```
BDENSE_TEMPLATE (
    void ,
    out_of_range )
```

**8.9.2.18 BDENSE\_TEMPLATE()** [17/27]

```
BDENSE_TEMPLATE (
    void ,
    print ) const
```

Definition at line 1000 of file baraydense-meat.hpp.

**8.9.2.19 BDENSE\_TEMPLATE()** [18/27]

```
BDENSE_TEMPLATE (
    void ,
    reserve )
```

Definition at line 990 of file baraydense-meat.hpp.

**8.9.2.20 BDENSE\_TEMPLATE()** [19/27]

```
BDENSE_TEMPLATE (
    void ,
    resize )
```

**8.9.2.21 BDENSE\_TEMPLATE()** [20/27]

```
BDENSE_TEMPLATE (
    void ,
    set_data )
```

**8.9.2.22 BDENSE\_TEMPLATE()** [21/27]

```
BDENSE_TEMPLATE (
    void ,
    swap_cells )
```

**8.9.2.23 BDENSE\_TEMPLATE()** [22/27]

```
BDENSE_TEMPLATE (
    void ,
    swap_cols )
```

**8.9.2.24 BDENSE\_TEMPLATE()** [23/27]

```
BDENSE_TEMPLATE (
    void ,
    swap_rows )
```

**8.9.2.25 BDENSE\_TEMPLATE()** [24/27]

```
BDENSE_TEMPLATE (
    void ,
    toggle_cell )
```

**8.9.2.26 BDENSE\_TEMPLATE()** [25/27]

```
BDENSE_TEMPLATE (
    void ,
    transpose )
```

Definition at line 897 of file barraydense-meat.hpp.

**8.9.2.27 BDENSE\_TEMPLATE()** [26/27]

```
BDENSE_TEMPLATE (
    void ,
    zero_col )
```

**8.9.2.28 BDENSE\_TEMPLATE()** [27/27]

```
BDENSE_TEMPLATE (
    void ,
    zero_row )
```

**8.9.2.29 for()**

```
for ( )
```

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

**8.9.2.30 if() [1/6]**

```
else if (
    !check0 & check1 )
```

Definition at line 747 of file `barraydense-meat.hpp`.

**8.9.2.31 if() [2/6]**

```
else if (
    check0 & ! check1 )
```

Definition at line 755 of file `barraydense-meat.hpp`.

**8.9.2.32 if() [3/6]**

```
if (
    check0 & check1 )
```

Definition at line 729 of file `barraydense-meat.hpp`.

**8.9.2.33 if() [4/6]**

```
if (
    check_exists == CHECK::BOTH )
```

Definition at line 587 of file `barraydense-meat.hpp`.

**8.9.2.34 if() [5/6]**

```
if (
    NCells == 0u )
```

**8.9.2.35 if() [6/6]**

```
if (
    report !      = nullptr )
```

**8.9.2.36 M()**

```
bool M (
    Array_  M )
```

Definition at line 129 of file barraydense-meat.hpp.

**8.9.2.37 resize() [1/3]**

```
el resize (
    N * M )
```

**8.9.2.38 resize() [2/3]**

```
el resize (
    N ,
    M )
```

**8.9.2.39 resize() [3/3]**

```
el resize (
    N_ * M_,
    ZERO_CELL )
```

**8.9.3 Variable Documentation****8.9.3.1 add**

```
uint const std::vector< uint > const std::vector< uint > bool add
```

**Initial value:**

```
{
    if (source.size() != target.size())
        throw std::length_error("-source- and -target- don't match on length.")
```

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

### 8.9.3.2 ans

```
return ans
```

Definition at line 355 of file bararraydense-meat.hpp.

### 8.9.3.3 check\_bounds

```
bool check_bounds
```

**Initial value:**

```
{  
    if (check_bounds)  
    {  
        out_of_range(i0,0u);  
        out_of_range(i1,0u);  
    }  
  
    if (NCells == 0u)  
        return
```

Definition at line 580 of file bararraydense-meat.hpp.

### 8.9.3.4 check\_exists

```
uint bool int check_exists
```

**Initial value:**

```
{  
    if (check_bounds)  
        out_of_range(i,j)
```

Definition at line 581 of file bararraydense-meat.hpp.

### 8.9.3.5 col

```
col
```

Definition at line 871 of file bararraydense-meat.hpp.

### 8.9.3.6 const

```
bool check_bounds const
```

**Initial value:**

```
{  
    if (i >= N)  
        throw std::range_error("The row is out of range.")
```

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

### 8.9.3.7 copy\_data

```
bool copy_data
```

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

### 8.9.3.8 data

```
data = data_
```

Definition at line 299 of file barraydense-meat.hpp.

### 8.9.3.9 delete\_data

```
delete_data = delete_data_
```

Definition at line 300 of file barraydense-meat.hpp.

### 8.9.3.10 delete\_data\_

```
bool delete_data_
```

#### Initial value:

```
{  
    if ((data != nullptr) && delete_data)  
        delete data
```

Definition at line 293 of file barraydense-meat.hpp.

### 8.9.3.11 el

```
return el = tmp
```

Definition at line 604 of file barraydense-meat.hpp.

**8.9.3.12 else**

else

**Initial value:**

```
{  
    Cell< Cell_Type > tmp (v)
```

Definition at line 601 of file bararraydense-meat.hpp.

**8.9.3.13 false**

false

Definition at line 871 of file bararraydense-meat.hpp.

**8.9.3.14 i1**

```
uint i1
```

Definition at line 665 of file bararraydense-meat.hpp.

**8.9.3.15 j**

```
j
```

Definition at line 330 of file bararraydense-meat.hpp.

**8.9.3.16 j0**

```
uint j0
```

Definition at line 664 of file bararraydense-meat.hpp.

**8.9.3.17 j1**

```
uint j1
```

Definition at line 665 of file bararraydense-meat.hpp.



### 8.9.3.18 M

M = M\_

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

### 8.9.3.19 M\_

uint M\_

**Initial value:**

```
{  
    if (NCells == 0u)  
    {  
        el.resize(N_ * M_);  
        N = N_;  
        M = M_;  
        return;  
    }  
  
    std::vector< Cell< Cell_Type > > el_tmp(std::move(el))
```

Definition at line 27 of file barraydense-meat.hpp.

### 8.9.3.20 N

N = N\_

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

### 8.9.3.21 NCells

NCells

Definition at line 605 of file barraydense-meat.hpp.

### 8.9.3.22 report

uint uint uint bool int int\* report

Definition at line 668 of file barraydense-meat.hpp.

#### 8.9.3.23 return

```
return
```

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

#### 8.9.3.24 source

```
uint const std::vector< uint >& source
```

Definition at line 28 of file bararraydense-meat.hpp.

#### 8.9.3.25 target

```
uint const std::vector< uint > const std::vector< uint >& target
```

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

#### 8.9.3.26 v

```
uint Cell_Type v
```

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

#### 8.9.3.27 value

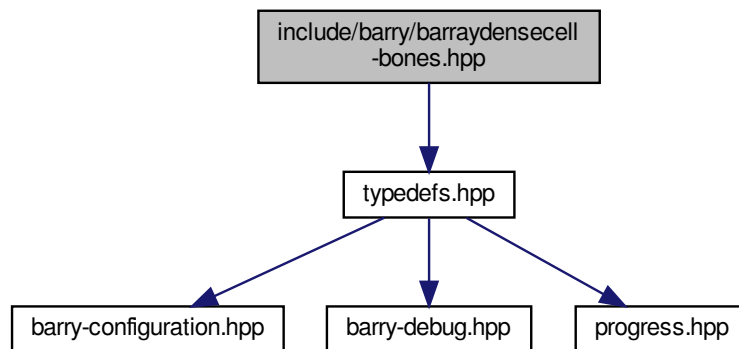
```
return el [POS(i, j)] value
```

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

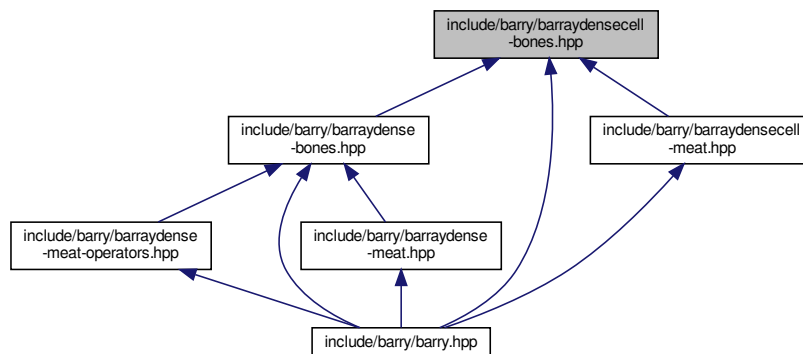
## 8.10 include/barry/barraydensecell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barraydensecell-bones.hpp:



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



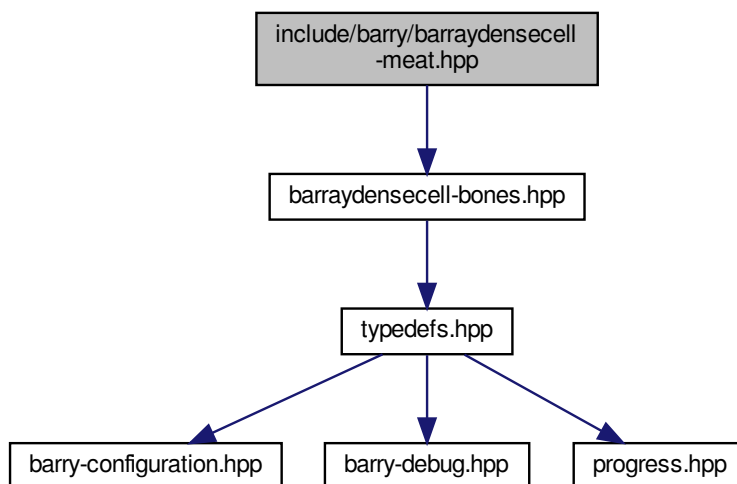
### Classes

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

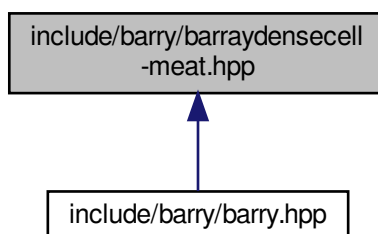
## 8.11 include/barry/barraydensecell-meal.hpp File Reference

```
#include "barraydensecell-bones.hpp"
```

Include dependency graph for `barraydensecell-meat.hpp`:



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



## Macros

- `#define POS(a, b) (a) + (b) * Array->N`

### 8.11.1 Macro Definition Documentation

## 8.11.1.1 POS

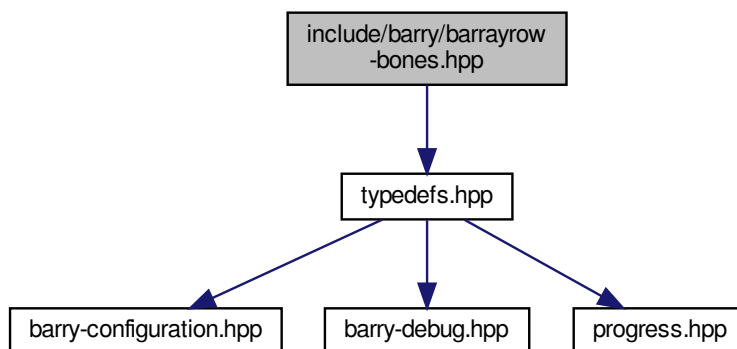
```
#define POS(
    a,
    b ) (a) + (b) * Array->N
```

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

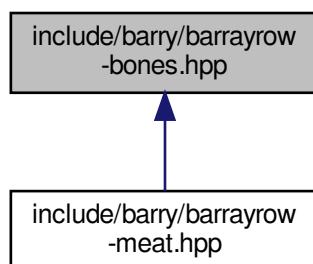
## 8.12 include/barry/barrayrow-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barrayrow-bones.hpp:



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



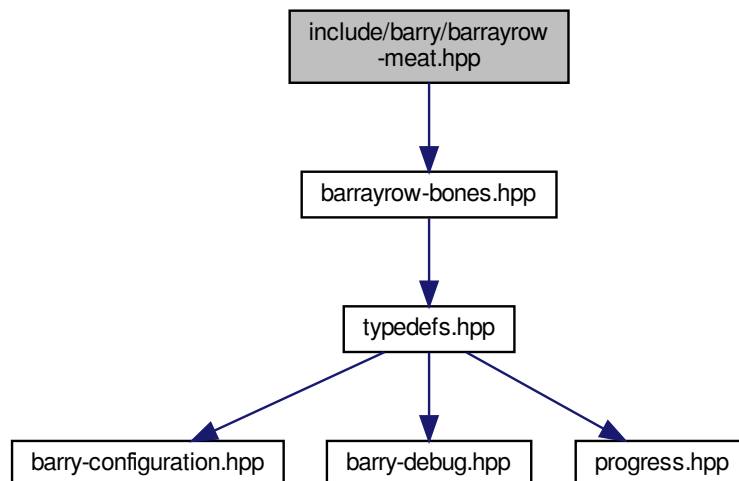
## Classes

- class [BArrayRow< Cell\\_Type, Data\\_Type >](#)
- class [BArrayRow\\_const< Cell\\_Type, Data\\_Type >](#)

## 8.13 include/barry/barrayrow-meat.hpp File Reference

```
#include "barrayrow-bones.hpp"
```

Include dependency graph for barrayrow-meat.hpp:



### Macros

- `#define BARRY_BARRAYROW_MEAT_HPP 1`
- `#define BROW_TYPE() BArrayRow<Cell_Type, Data_Type>`
- `#define BROW_TEMPLATE_ARGS() <typename Cell_Type, typename Data_Type>`
- `#define BROW_TEMPLATE(a, b) template BROW_TEMPLATE_ARGS() inline a BROW_TYPE()::b`

### Functions

- `BROW_TEMPLATE (void, operator=)(const BROW_TYPE() &val)`
- `BROW_TEMPLATE (void, operator+=)(const BROW_TYPE() &val)`
- `BROW_TEMPLATE (void, operator-=)(const BROW_TYPE() &val)`
- `BROW_TEMPLATE (void, operator*=)(const BROW_TYPE() &val)`
- `BROW_TEMPLATE (void, operator/=)(const BROW_TYPE() &val)`

### 8.13.1 Macro Definition Documentation

#### 8.13.1.1 BARRY\_BARRAYROW\_MEAT\_HPP

```
#define BARRY_BARRAYROW_MEAT_HPP 1
```

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

### 8.13.1.2 BROW\_TEMPLATE

```
#define BROW_TEMPLATE(  
    a,  
    b )  template BROW_TEMPLATE_ARGS() inline a BROW_TYPE():b
```

Definition at line 10 of file barrayrow-meat.hpp.

### 8.13.1.3 BROW\_TEMPLATE\_ARGS

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

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

### 8.13.1.4 BROW\_TYPE

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

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

## 8.13.2 Function Documentation

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

```
BROW_TEMPLATE (  
    void ,  
    operator* ) const &
```

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

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

```
BROW_TEMPLATE (  
    void ,  
    operator+ ) const &
```

Definition at line 27 of file barrayrow-meat.hpp.

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

```
BROW_TEMPLATE (
    void ,
    operator- ) const &
```

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

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

```
BROW_TEMPLATE (
    void ,
    operator/ ) const &
```

Definition at line 57 of file barrow-meat.hpp.

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

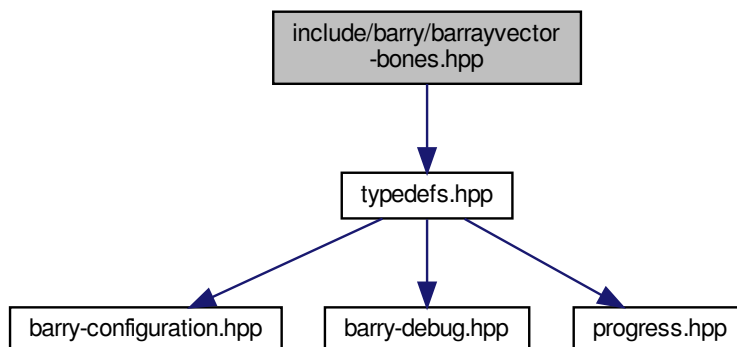
```
BROW_TEMPLATE (
    void ,
    operator ) const &
```

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

## 8.14 include/barry/barrayvector-bones.hpp File Reference

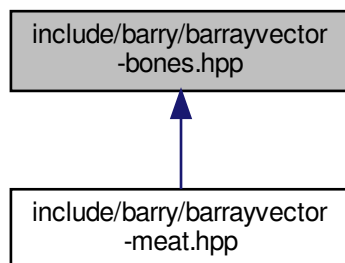
```
#include "typedefs.hpp"
```

Include dependency graph for barrayvector-bones.hpp:





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



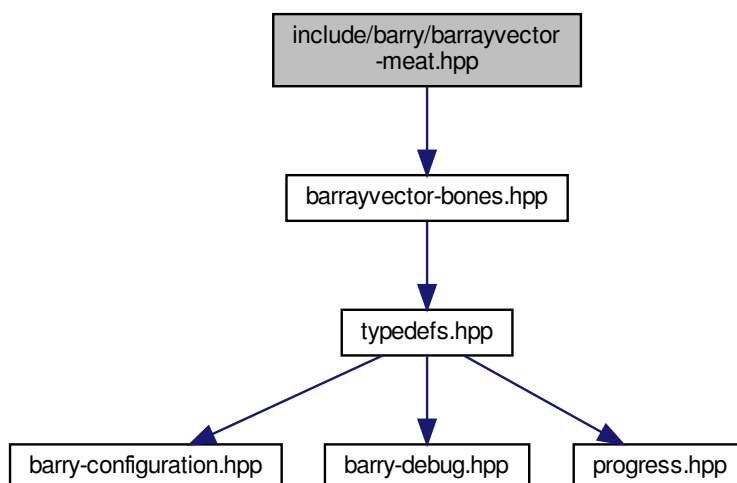
## Classes

- class [BArrayVector< Cell\\_Type, Data\\_Type >](#)  
*Row or column of a [BArray](#)*
- class [BArrayVector\\_const< Cell\\_Type, Data\\_Type >](#)

## 8.15 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`

### 8.15.1 Macro Definition Documentation

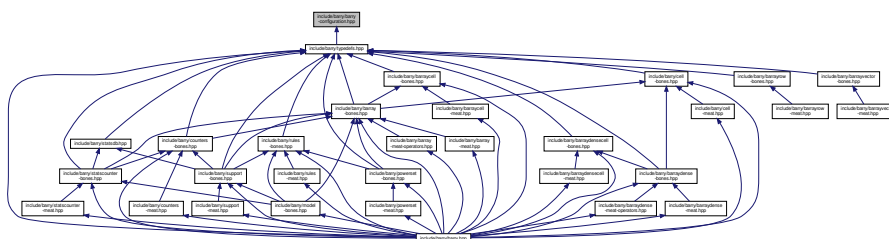
#### 8.15.1.1 BARRY\_BARRAYVECTOR\_MEAT\_HPP

```
#define BARRY_BARRAYVECTOR_MEAT_HPP 1
```

Definition at line 4 of file `barrayvector-meat.hpp`.

## 8.16 include/barry/barry-configuration.hpp File Reference

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



## Configuration MACROS

These are mostly related to performance. The definitions follow:

- `BARRY_USE_UNORDERED_MAP` If specified, then barry is compiled using `std::unordered_map`. Otherwise it will use `std::map` for the arrays.
- `BARRY_USE_SAFE_EXP` When specified, it will multiply all likelihoods in `Model` by  $(1/100)/(1/100)$  so that numerical overflows are avoided.
- `BARRY_USE_ISFINITE` When specified, it will introduce a macro that checks whether the likelihood is finite or not.
- `printf_barry` If not specified, will be defined as `printf`.
- `BARRY_DEBUG_LEVEL`, when defined, will make things verbose.
- `#define BARRY_SAFE_EXP -100.0`
- `#define BARRY_ISFINITE(a)`
- `#define BARRY_CHECK_SUPPORT(x, maxs)`
- `#define printf_barry printf`
- `#define BARRY_MAX_NUM_ELEMENTS static_cast< unsigned int >(UINT_MAX/2u)`
- `template<typename Ta, typename Tb >`  
`using Map = std::map< Ta, Tb >`

## 8.16.1 Macro Definition Documentation

### 8.16.1.1 BARRY\_CHECK\_SUPPORT

```
#define BARRY_CHECK_SUPPORT(  
    x,  
    maxs )
```

Definition at line 47 of file barry-configuration.hpp.

### 8.16.1.2 BARRY\_ISFINITE

```
#define BARRY_ISFINITE(  
    a )
```

Definition at line 40 of file barry-configuration.hpp.

### 8.16.1.3 BARRY\_MAX\_NUM\_ELEMENTS

```
#define BARRY_MAX_NUM_ELEMENTS static_cast< unsigned int >(UINT_MAX/2u)
```

Definition at line 55 of file barry-configuration.hpp.

### 8.16.1.4 BARRY\_SAFE\_EXP

```
#define BARRY_SAFE_EXP -100.0
```

Definition at line 33 of file barry-configuration.hpp.

### 8.16.1.5 printf\_barry

```
#define printf_barry printf
```

Definition at line 51 of file barry-configuration.hpp.

## 8.16.2 Typedef Documentation

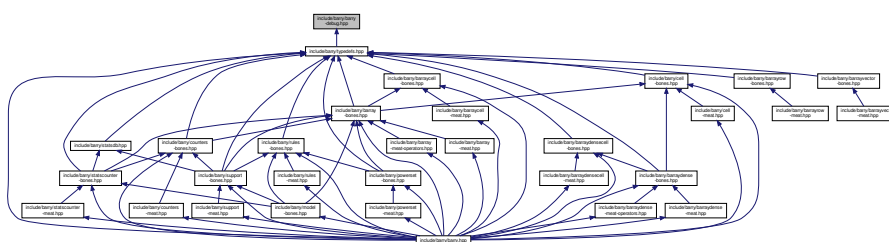
### 8.16.2.1 Map

```
template<typename Ta , typename Tb >
using Map = std::map<Ta,Tb>
```

Definition at line 27 of file barry-configuration.hpp.

## 8.17 include/barry/barry-debug.hpp File Reference

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



## Macros

- #define BARRY\_DEBUG\_LEVEL 0

### 8.17.1 Macro Definition Documentation

#### 8.17.1.1 BARRY\_DEBUG\_LEVEL

```
#define BARRY_DEBUG_LEVEL 0
```

Definition at line 5 of file barry-debug.hpp.

## 8.18 include/barry/barry.hpp File Reference

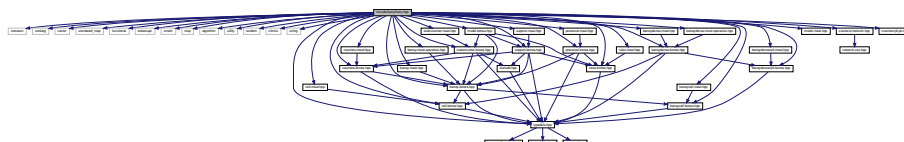
```
#include <iostream>
#include <cstdlib>
#include <vector>
#include <unordered_map>
#include <functional>
#include <stdexcept>
#include <cmath>
#include <map>
#include <algorithm>
#include <utility>
```

```

#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 "barraydense-bones.hpp"
#include "barraydensecell-bones.hpp"
#include "barraydense-meat.hpp"
#include "barraydensecell-meat.hpp"
#include "barraydense-meat-operators.hpp"
#include "counters-bones.hpp"
#include "counters-meat.hpp"
#include "statscounter-bones.hpp"
#include "statscounter-meat.hpp"
#include "support-bones.hpp"
#include "support-meat.hpp"
#include "powerset-bones.hpp"
#include "powerset-meat.hpp"
#include "model-bones.hpp"
#include "model-meat.hpp"
#include "rules-bones.hpp"
#include "rules-meat.hpp"
#include "counters/network.hpp"
#include "counters/phylo.hpp"

```

Include 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)`

## 8.18.1 Macro Definition Documentation

### 8.18.1.1 BARRY\_HPP

```
#define BARRY_HPP
```

Definition at line 20 of file barry.hpp.

### 8.18.1.2 BARRY\_VERSION

```
#define BARRY_VERSION 0.1
```

Definition at line 22 of file barry.hpp.

### 8.18.1.3 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 79 of file barry.hpp.

### 8.18.1.4 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 82 of file barry.hpp.

#### 8.18.1.5 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 86 of file barry.hpp.

#### 8.18.1.6 RULE\_LAMBDA

```
#define RULE_LAMBDA(  
    a )
```

**Value:**

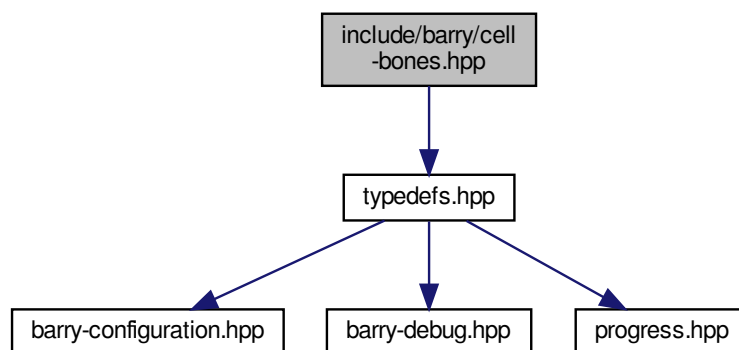
```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \
Rule_fun_type<Array_Type, Data_Type> a = \
[] (const Array_Type & Array, uint i, uint j, Data_Type * data)
```

Definition at line 89 of file barry.hpp.

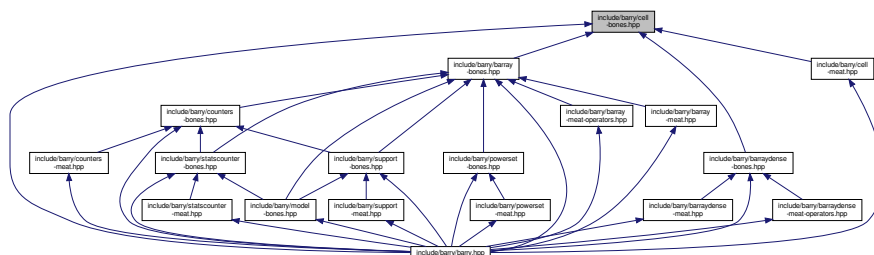
## 8.19 include/barry/cell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for cell-bones.hpp:



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



## Classes

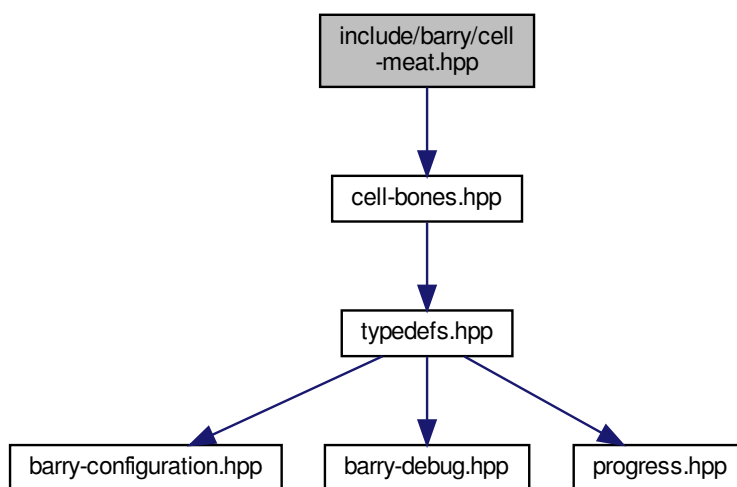
- class [Cell< Cell\\_Type >](#)

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

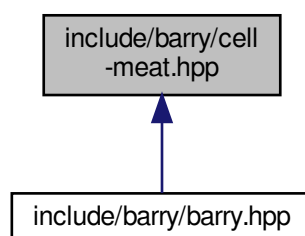
## 8.20 include/barry/cell-meat.hpp File Reference

```
#include "cell-bones.hpp"
```

Include dependency graph for cell-meat.hpp:



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





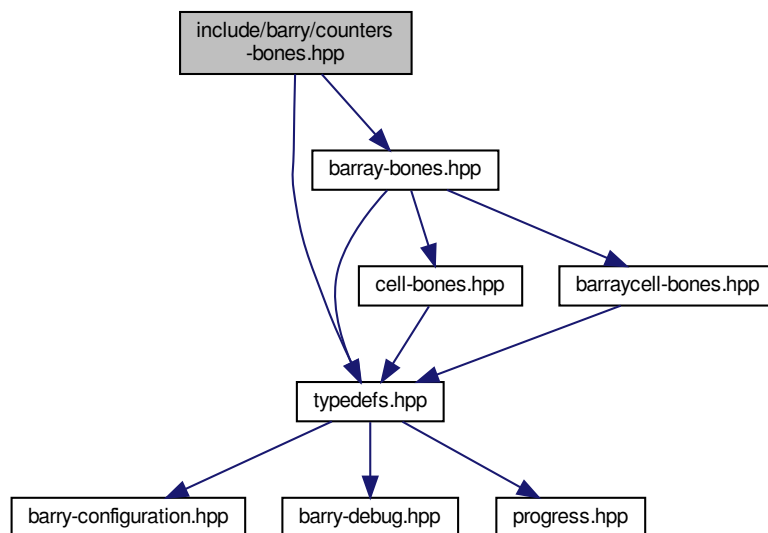
## 8.21 include/barry/col-bones.hpp File Reference

## 8.22 include/barry/counters-bones.hpp File Reference

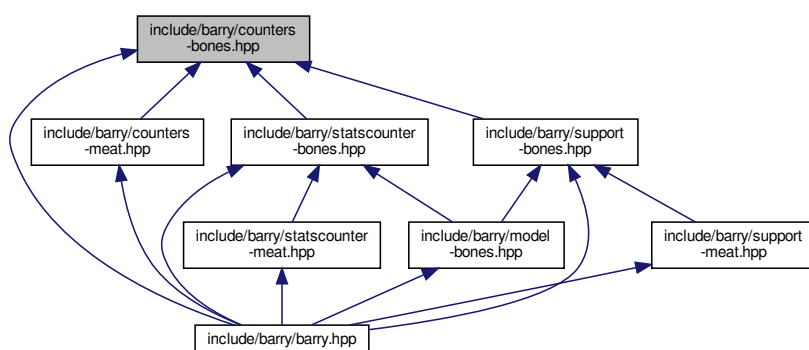
```
#include "typedefs.hpp"
```

```
#include "barray-bones.hpp"
```

Include dependency graph for counters-bones.hpp:



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



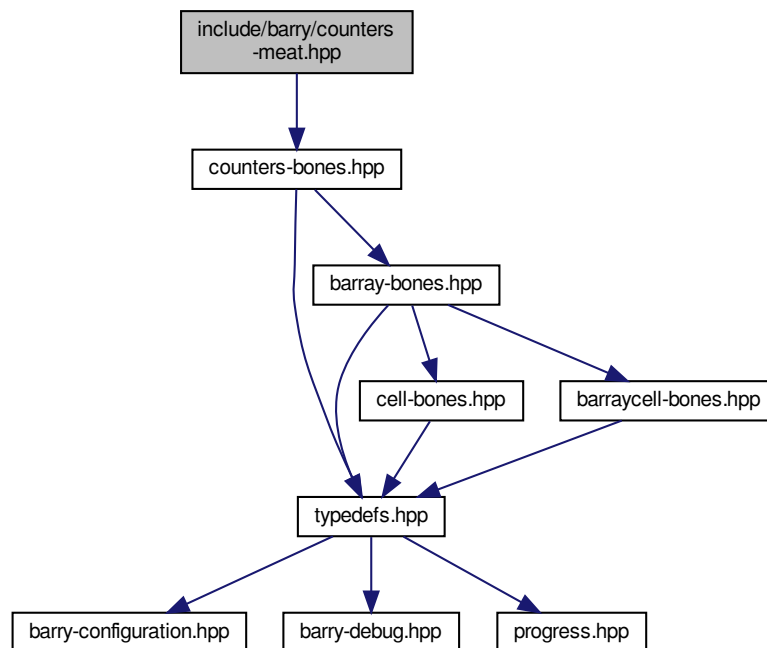
## Classes

- class `Counter< Array_Type, Data_Type >`  
A counter function based on change statistics.
- class `Counters< Array_Type, Data_Type >`  
Vector of counters.

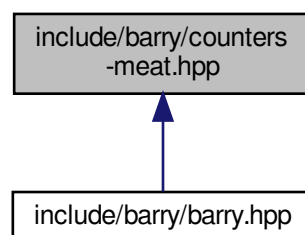
## 8.23 include/barry/counters-meat.hpp File Reference

```
#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\_

## 8.23.1 Macro Definition Documentation

### 8.23.1.1 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.

### 8.23.1.2 COUNTER\_TEMPLATE\_ARGS

```
#define COUNTER_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

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

### 8.23.1.3 COUNTER\_TYPE

```
#define COUNTER_TYPE( ) Counter<Array_Type,Data_Type>
```

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

### 8.23.1.4 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.

### 8.23.1.5 COUNTERS\_TEMPLATE\_ARGS

```
#define COUNTERS_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

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

### 8.23.1.6 COUNTERS\_TYPE

```
#define COUNTERS_TYPE( ) Counters<Array_Type,Data_Type>
```

Definition at line 149 of file counters-meat.hpp.

## 8.23.2 Function Documentation

### 8.23.2.1 count\_fun()

```
return count_fun (
    Array ,
    i ,
    j ,
    data )
```

### 8.23.2.2 COUNTER\_TEMPLATE() [1/7]

```
COUNTER_TEMPLATE (
    Counter ) const
```

### 8.23.2.3 COUNTER\_TEMPLATE() [2/7]

```
COUNTER_TEMPLATE (
    COUNTER\_TYPE() & ,
    operator )
```

### 8.23.2.4 COUNTER\_TEMPLATE() [3/7]

```
COUNTER_TEMPLATE (
    COUNTER\_TYPE() ,
    operator ) const
```

#### 8.23.2.5 COUNTER\_TEMPLATE() [4/7]

```
COUNTER_TEMPLATE (
    double ,
    count ) &
```

[< Move assignment](#)

#### 8.23.2.6 COUNTER\_TEMPLATE() [5/7]

```
COUNTER_TEMPLATE (
    double ,
    init ) &
```

#### 8.23.2.7 COUNTER\_TEMPLATE() [6/7]

```
COUNTER_TEMPLATE (
    std::string ,
    get_description ) const
```

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

#### 8.23.2.8 COUNTER\_TEMPLATE() [7/7]

```
COUNTER_TEMPLATE (
    std::string ,
    get_name ) const
```

Definition at line 137 of file counters-meat.hpp.

#### 8.23.2.9 COUNTERS\_TEMPLATE() [1/8]

```
COUNTERS_TEMPLATE (
    Counters )
```

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

**8.23.2.10 COUNTERS\_TEMPLATE()** [2/8]

```
COUNTERS_TEMPLATE (
    COUNTER_TYPE() & ,
    operator [] )
```

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

**8.23.2.11 COUNTERS\_TEMPLATE()** [3/8]

```
COUNTERS_TEMPLATE (
    COUNTERS_TYPE() & ,
    operator )
```

**8.23.2.12 COUNTERS\_TEMPLATE()** [4/8]

```
COUNTERS_TEMPLATE (
    COUNTERS_TYPE() ,
    operator ) const
```

**8.23.2.13 COUNTERS\_TEMPLATE()** [5/8]

```
COUNTERS_TEMPLATE (
    std::vector< std::string > ,
    get_descriptions ) const
```

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

**8.23.2.14 COUNTERS\_TEMPLATE()** [6/8]

```
COUNTERS_TEMPLATE (
    std::vector< std::string > ,
    get_names ) const
```

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

**8.23.2.15 COUNTERS\_TEMPLATE()** [7/8]

```
COUNTERS_TEMPLATE (
    void ,
    add_counter )
```

**8.23.2.16 COUNTERS\_TEMPLATE()** [8/8]

```
COUNTERS_TEMPLATE (
    void ,
    clear )
```

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

**8.23.2.17 data()**

```
Data_Type&& counter_ data (
    std::move(counter_.data) )
```

**8.23.2.18 delete\_data()** [1/3]

```
Data_Type&& counter_ delete_data (
    std::move(counter_.delete_data) )
```

**8.23.2.19 delete\_data()** [2/3]

```
Data_Type&& counters_ delete_data (
    std::move(counters_.delete_data) )
```

**8.23.2.20 delete\_data()** [3/3]

```
Data_Type Data_Type delete_data (
    true )
```



**8.23.2.21 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.

**8.23.2.22 delete\_to\_be\_deleted() [2/2]**

```
Data_Type Data_Type delete_to_be_deleted (
    true )
```

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

**8.23.2.23 desc()**

```
Data_Type&& counter_ desc (
    std::move(counter_.desc) )
```

Move constructor.

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

**8.23.2.24 init\_fun() [1/3]**

```
return init_fun (
    Array ,
    i ,
    j ,
    data )
```

**8.23.2.25 init\_fun() [2/3]**

```
Data_Type init_fun (
    counter_. init_fun )
```

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

**8.23.2.26 init\_fun() [3/3]**

```
Data_Type&& counter_ init_fun (
    std::move(counter_.init_fun) )
```

**8.23.2.27 name()**

```
Data_Type&& counter_ name (
    std::move(counter_.name) )
```

**8.23.2.28 push\_back() [1/2]**

```
data push_back (
    new Counter< Array_Type, Data_Type > count_fun_, init_fun_, data_, delete_data_,
    name_, desc_ )
```

**8.23.2.29 push\_back() [2/2]**

```
data push_back (
    new Counter< Array_Type, Data_Type > counter )
```

**8.23.2.30 to\_be\_deleted() [1/2]**

```
Data_Type Data_Type to_be_deleted (
    new std::vector< uint > 0u )
```

**8.23.2.31 to\_be\_deleted() [2/2]**

```
Data_Type&& counters_ to_be_deleted (
    std::move(counters_.to_be_deleted) )
```

**8.23.3 Variable Documentation**

### 8.23.3.1 count\_fun\_

Data\_Type count\_fun\_

Definition at line 291 of file counters-meat.hpp.

### 8.23.3.2 counter

Data\_Type \* counter

#### Initial value:

```
{  
    to_be_deleted->push_back(data->size())
```

Definition at line 273 of file counters-meat.hpp.

### 8.23.3.3 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.

### 8.23.3.4 data\_

Data\_Type Counter\_fun\_type<Array\_Type,Data\_Type> Data\_Type\* data\_

Definition at line 293 of file counters-meat.hpp.

### 8.23.3.5 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.

### 8.23.3.6 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.

### 8.23.3.7 i

```
uint i
```

Definition at line 117 of file counters-meat.hpp.

### 8.23.3.8 init\_fun\_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> init_fun_
```

Definition at line 292 of file counters-meat.hpp.

### 8.23.3.9 j

```
uint uint j
```

#### Initial value:

```
{  
    if (count_fun == nullptr)  
        return 0.0
```

Definition at line 117 of file counters-meat.hpp.

### 8.23.3.10 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.

### 8.23.3.11 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.

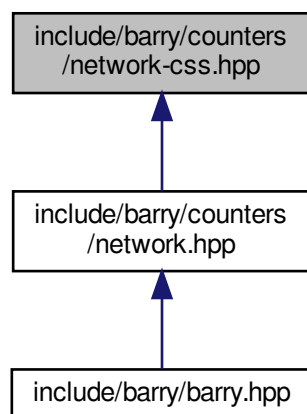
### 8.23.3.12 return

return

Definition at line 279 of file counters-meat.hpp.

## 8.24 include/barry/counters/network-css.hpp File Reference

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



### Macros

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

### Functions

- `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 (commission)*
- `void counter_css_completely_false_recip_omiss (NetCounters *counters, uint netsize, const std::vector< uint > &end_)`  
*Counts completely false reciprocity (omission)*

- void `counter_css_mixed_recip` (`NetCounters` \*counters, `uint` netsize, `const` `std::vector`< `uint` > &end\_)  
*Counts mixed reciprocity errors.*
- void `counter_css_census01` (`NetCounters` \*counters, `uint` netsize, `const` `std::vector`< `uint` > &end\_)
- void `counter_css_census02` (`NetCounters` \*counters, `uint` netsize, `const` `std::vector`< `uint` > &end\_)
- void `counter_css_census03` (`NetCounters` \*counters, `uint` netsize, `const` `std::vector`< `uint` > &end\_)
- void `counter_css_census04` (`NetCounters` \*counters, `uint` netsize, `const` `std::vector`< `uint` > &end\_)
- void `counter_css_census05` (`NetCounters` \*counters, `uint` netsize, `const` `std::vector`< `uint` > &end\_)
- void `counter_css_census06` (`NetCounters` \*counters, `uint` netsize, `const` `std::vector`< `uint` > &end\_)
- void `counter_css_census07` (`NetCounters` \*counters, `uint` netsize, `const` `std::vector`< `uint` > &end\_)
- void `counter_css_census08` (`NetCounters` \*counters, `uint` netsize, `const` `std::vector`< `uint` > &end\_)
- void `counter_css_census09` (`NetCounters` \*counters, `uint` netsize, `const` `std::vector`< `uint` > &end\_)
- void `counter_css_census10` (`NetCounters` \*counters, `uint` netsize, `const` `std::vector`< `uint` > &end\_)

## 8.24.1 Macro Definition Documentation

### 8.24.1.1 CSS\_APPEND

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

#### Value:

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

Definition at line 42 of file network-css.hpp.

### 8.24.1.2 CSS\_CASE\_ELSE

```
#define CSS_CASE_ELSE( )
```

Definition at line 27 of file network-css.hpp.

### 8.24.1.3 CSS\_CASE\_PERCEIVED

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

Definition at line 20 of file network-css.hpp.

#### 8.24.1.4 CSS\_CASE\_TRUTH

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

Definition at line 13 of file network-css.hpp.

#### 8.24.1.5 CSS\_CHECK\_SIZE

```
#define CSS_CHECK_SIZE( )
```

**Value:**

```
for (uint i = 0u; i < end_.size(); ++i) {\
  if (i == 0u) continue; \
  else if (end_[i] < end_[i-1u]) \
    throw std::logic_error("Endpoints should be specified in order.");}
```

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

#### 8.24.1.6 CSS\_CHECK\_SIZE\_INIT

```
#define CSS_CHECK_SIZE_INIT( )
```

**Value:**

```
/* The indices fall within the network */ \
if ((data->indices.at(0) > Array.ncol()) \
| (data->indices.at(2) > Array.ncol())) \
  throw std::range_error("The network does not match the prescribed size.");
```

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

#### 8.24.1.7 CSS\_NET\_COUNTER\_LAMBDA\_INIT

```
#define CSS_NET_COUNTER_LAMBDA_INIT( )
```

**Value:**

```
NETWORK_COUNTER_LAMBDA(tmp_init) {\
  CSS_CHECK_SIZE_INIT() \
  return 0.0; \
};
```

Definition at line 49 of file network-css.hpp.



### 8.24.1.8 CSS\_PERCEIVED\_CELLS

```
#define CSS_PERCEIVED_CELLS( )
```

**Value:**

```
double tji = static_cast<double>(Array(j - s, i - s, false)); \
double pji = static_cast<double>(Array(j, i, false)); \
double tij = static_cast<double>(Array(i - s, j - s, false));
```

Definition at line 21 of file network-css.hpp.

### 8.24.1.9 CSS\_SIZE

```
#define CSS_SIZE( )
```

**Value:**

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

Definition at line 7 of file network-css.hpp.

### 8.24.1.10 CSS\_TRUE\_CELLS

```
#define CSS_TRUE_CELLS( )
```

**Value:**

```
double tji = static_cast<double>(Array(j, i, false)); \
double pij = static_cast<double>(Array(i + s, j + s, false)); \
double pji = static_cast<double>(Array(j + s, i + s, false));
```

Definition at line 14 of file network-css.hpp.

## 8.24.2 Function Documentation

### 8.24.2.1 counter\_css\_census01()

```
void counter_css_census01 (
    NetCounters * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

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

#### 8.24.2.2 counter\_css\_census02()

```
void counter_css_census02 (
    NetCounters * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 314 of file network-css.hpp.

#### 8.24.2.3 counter\_css\_census03()

```
void counter_css_census03 (
    NetCounters * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 353 of file network-css.hpp.

#### 8.24.2.4 counter\_css\_census04()

```
void counter_css_census04 (
    NetCounters * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 392 of file network-css.hpp.

#### 8.24.2.5 counter\_css\_census05()

```
void counter_css_census05 (
    NetCounters * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 431 of file network-css.hpp.

#### 8.24.2.6 counter\_css\_census06()

```
void counter_css_census06 (
    NetCounters * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 470 of file network-css.hpp.

#### 8.24.2.7 counter\_css\_census07()

```
void counter_css_census07 (
    NetCounters * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 509 of file network-css.hpp.

#### 8.24.2.8 counter\_css\_census08()

```
void counter_css_census08 (
    NetCounters * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 548 of file network-css.hpp.

#### 8.24.2.9 counter\_css\_census09()

```
void counter_css_census09 (
    NetCounters * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 587 of file network-css.hpp.

#### 8.24.2.10 counter\_css\_census10()

```
void counter_css_census10 (
    NetCounters * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 626 of file network-css.hpp.

#### 8.24.2.11 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 154 of file network-css.hpp.

#### 8.24.2.12 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 194 of file network-css.hpp.

#### 8.24.2.13 counter\_css\_mixed\_recip()

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

Counts mixed reciprocity errors.

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

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

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

Counts errors of commission.

##### Parameters

<i>netsize</i>	Size of the reference (true) network
<i>end_</i> ↵	Vector indicating one past the ending index of each network. (see details)
—	

The *end\_* parameter should be of length *N* of *networks* - 1. It is assumed that the first network ends at *netsize*.

Definition at line 63 of file network-css.hpp.

#### 8.24.2.15 counter\_css\_partially\_false\_recip\_omiss()

```
void counter_css_partially_false_recip_omiss (
    NetCounters * counters,
```

```
uint netsize,  
const std::vector< uint > & end_ ) [inline]
```

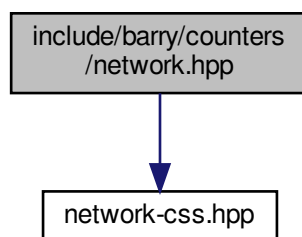
Counts errors of omission.

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

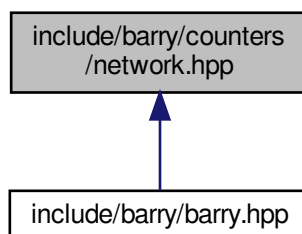
## 8.25 include/barry/counters/network.hpp File Reference

```
#include "network-css.hpp"
```

Include dependency graph for network.hpp:



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



## Classes

- class [NetworkData](#)  
*Data class for Networks.*
- class [NetCounterData](#)  
*Data class used to store arbitrary uint or double vectors.*

## Macros

- #define `NET_C_DATA_IDX(i)` (`data->indices[i]`)
- #define `NET_C_DATA_NUM(i)` (`data->numbers[i]`)

### Macros for defining counters

- #define `NETWORK_COUNTER(a)`
- #define `NETWORK_COUNTER_LAMBDA(a)`

### Macros for defining rules

- #define `NETWORK_RULE(a)`
- #define `NETWORK_RULE_LAMBDA(a)`

## Typedefs

### Convenient typedefs for network objects.

- typedef `BArray< double, NetworkData > Network`
- typedef `BArrayDense< double, NetworkData > NetworkDense`
- template<typename Tnet = Network>  
using `NetCounter` = `Counter< Tnet, NetCounterData >`
- template<typename Tnet = Network>  
using `NetCounters` = `Counters< Tnet, NetCounterData >`
- template<typename Tnet = Network>  
using `NetSupport` = `Support< Tnet, NetCounterData >`
- template<typename Tnet = Network>  
using `NetStatsCounter` = `StatsCounter< Tnet, NetCounterData >`
- template<typename Tnet >  
using `NetModel` = `Model< Tnet, NetCounterData >`
- template<typename Tnet = Network>  
using `NetRule` = `Rule< Tnet, bool >`
- template<typename Tnet = Network>  
using `NetRules` = `Rules< Tnet, bool >`

## Functions

- template<typename Tnet = Network>  
void `counter_edges` (`NetCounters< Tnet > *counters`)  
*Number of edges.*
- template<typename Tnet = Network>  
void `counter_isolates` (`NetCounters< Tnet > *counters`)  
*Number of isolated vertices.*
- template<typename Tnet = Network>  
void `counter_mutual` (`NetCounters< Tnet > *counters`)  
*Number of mutual ties.*
- template<typename Tnet = Network>  
void `counter_istar2` (`NetCounters< Tnet > *counters`)
- template<typename Tnet = Network>  
void `counter_ostar2` (`NetCounters< Tnet > *counters`)

- `template<typename Tnet = Network>`  
`void counter_ttriads (NetCounters< Tnet > *counters)`
- `template<typename Tnet = Network>`  
`void counter_ctriads (NetCounters< Tnet > *counters)`
- `template<typename Tnet = Network>`  
`void counter_density (NetCounters< Tnet > *counters)`
- `template<typename Tnet = Network>`  
`void counter_idegree15 (NetCounters< Tnet > *counters)`
- `template<typename Tnet = Network>`  
`void counter_odegree15 (NetCounters< Tnet > *counters)`
- `template<typename Tnet = Network>`  
`void counter_absdiff (NetCounters< Tnet > *counters, uint attr_id, double alpha=1.0)`  
*Sum of absolute attribute difference between ego and alter.*
- `template<typename Tnet = Network>`  
`void counter_diff (NetCounters< Tnet > *counters, uint attr_id, double alpha=1.0, double tail_head=true)`  
*Sum of attribute difference between ego and alter to pow(alpha)*
- `NETWORK_COUNTER` (init\_single\_attr)
- `template<typename Tnet = Network>`  
`void counter_nodeicov (NetCounters< Tnet > *counters, uint attr_id)`
- `template<typename Tnet = Network>`  
`void counter_nodeocov (NetCounters< Tnet > *counters, uint attr_id)`
- `template<typename Tnet = Network>`  
`void counter_nodecov (NetCounters< Tnet > *counters, uint attr_id)`
- `template<typename Tnet = Network>`  
`void counter_nodematch (NetCounters< Tnet > *counters, uint attr_id)`
- `template<typename Tnet = Network>`  
`void counter_idegree (NetCounters< Tnet > *counters, std::vector< uint > d)`  
*Counts number of vertices with a given in-degree.*
- `template<typename Tnet = Network>`  
`void counter_odegree (NetCounters< Tnet > *counters, std::vector< uint > d)`  
*Counts number of vertices with a given out-degree.*
- `template<typename Tnet = Network>`  
`void counter_degree (NetCounters< Tnet > *counters, std::vector< uint > d)`  
*Counts number of vertices with a given out-degree.*

## Rules for network models

### Parameters

rules	A pointer to a <i>NetRules</i> object ( <i>Rules</i> < <i>Network</i> , <i>bool</i> >).
-------	---

- `template<typename Tnet = Network>`  
`void rules_zerodiag (NetRules< Tnet > *rules)`  
*Number of edges.*

## 8.25.1 Macro Definition Documentation

### 8.25.1.1 NET\_C\_DATA\_IDX

```
#define NET_C_DATA_IDX(  
    i ) (data->indices[i])
```

Definition at line 74 of file network.hpp.

### 8.25.1.2 NET\_C\_DATA\_NUM

```
#define NET_C_DATA_NUM(  
    i ) (data->numbers[i])
```

Definition at line 75 of file network.hpp.

### 8.25.1.3 NETWORK\_COUNTER

```
#define NETWORK_COUNTER(  
    a )
```

**Value:**

```
template<typename Tnet = Network>\  
inline double (a) (const Tnet & Array, uint i, uint j, NetCounterData * data)
```

Function for definition of a network counter function

Definition at line 104 of file network.hpp.

### 8.25.1.4 NETWORK\_COUNTER\_LAMBDA

```
#define NETWORK_COUNTER_LAMBDA(  
    a )
```

**Value:**

```
Counter_fun_type<Tnet, NetCounterData> a = \  
    [] (const Tnet & Array, uint i, uint j, NetCounterData * data)
```

Lambda function for definition of a network counter function

Definition at line 109 of file network.hpp.



### 8.25.1.5 NETWORK\_RULE

```
#define NETWORK_RULE(  
    a )
```

**Value:**

```
template<typename Tnet = Network>\  
inline bool (a) (const Tnet & Array, uint i, uint j, bool * data)
```

Function for definition of a network counter function

Definition at line 119 of file network.hpp.

### 8.25.1.6 NETWORK\_RULE\_LAMBDA

```
#define NETWORK_RULE_LAMBDA(  
    a )
```

**Value:**

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

Lambda function for definition of a network counter function

Definition at line 124 of file network.hpp.

## 8.25.2 Typedef Documentation

### 8.25.2.1 NetCounter

```
template<typename Tnet = Network>  
using NetCounter = Counter<Tnet, NetCounterData >
```

Definition at line 85 of file network.hpp.

### 8.25.2.2 NetCounters

```
template<typename Tnet = Network>  
using NetCounters = Counters<Tnet, NetCounterData>
```

Definition at line 87 of file network.hpp.

### 8.25.2.3 NetModel

```
template<typename Tnet >  
using NetModel = Model<Tnet, NetCounterData>
```

Definition at line 93 of file network.hpp.

### 8.25.2.4 NetRule

```
template<typename Tnet = Network>  
using NetRule = Rule<Tnet, bool>
```

Definition at line 95 of file network.hpp.

### 8.25.2.5 NetRules

```
template<typename Tnet = Network>  
using NetRules = Rules<Tnet, bool>
```

Definition at line 97 of file network.hpp.

### 8.25.2.6 NetStatsCounter

```
template<typename Tnet = Network>  
using NetStatsCounter = StatsCounter<Tnet, NetCounterData>
```

Definition at line 91 of file network.hpp.

### 8.25.2.7 NetSupport

```
template<typename Tnet = Network>  
using NetSupport = Support<Tnet, NetCounterData >
```

Definition at line 89 of file network.hpp.

### 8.25.2.8 Network

```
typedef BArray<double, NetworkData> Network
```

Definition at line 82 of file network.hpp.

### 8.25.2.9 NetworkDense

```
typedef BArrayDense<double, NetworkData> NetworkDense
```

Definition at line 83 of file network.hpp.

## 8.25.3 Function Documentation

### 8.25.3.1 rules\_zerodiag()

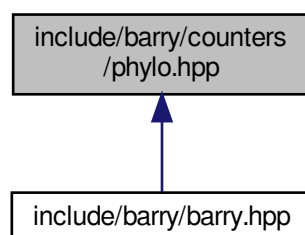
```
template<typename Tnet = Network>
void rules_zerodiag (
    NetRules< Tnet > * rules ) [inline]
```

Number of edges.

Definition at line 895 of file network.hpp.

## 8.26 include/barry/counters/phylo.hpp File Reference

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



## Classes

- class [NodeData](#)  
*Data definition for the `PhyloArray` class.*
- class [PhyloCounterData](#)
- class [PhyloRuleDynData](#)

## Macros

- `#define DEFAULT_DUPLICATION 1u`
- `#define DUPL_SPEC 0u`
- `#define DUPL_DUPL 1u`
- `#define DUPL_EITH 2u`
- `#define MAKE_DUPL_VARS()`
- `#define IS_EITHER() (DATA_AT == DUPL_EITH)`
- `#define IS_DUPLICATION() ((DATA_AT == DUPL_DUPL) & (DPL))`
- `#define IS_SPECIATION() ((DATA_AT == DUPL_SPEC) & (!DPL))`
- `#define IF_MATCHES()`
- `#define IF_NOTMATCHES()`
- `#define PHYLO_COUNTER_LAMBDA(a)`  
*Extension of a simple counter.*
- `#define PHYLO_RULE_DYN_LAMBDA(a)`
- `#define PHYLO_CHECK_MISSING()`

## Typedefs

- `typedef std::vector< std::pair< uint, uint > > PhyloRuleData`

### Convenient typedefs for Node objects.

- `typedef BArray< uint, NodeData > PhyloArray`
- `typedef Counter< PhyloArray, PhyloCounterData > PhyloCounter`
- `typedef Counters< PhyloArray, PhyloCounterData > PhyloCounters`
- `typedef Rule< PhyloArray, PhyloRuleData > PhyloRule`
- `typedef Rules< PhyloArray, PhyloRuleData > PhyloRules`
- `typedef Rule< PhyloArray, PhyloRuleDynData > PhyloRuleDyn`
- `typedef Rules< PhyloArray, PhyloRuleDynData > PhyloRulesDyn`
- `typedef Support< PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport`
- `typedef StatsCounter< PhyloArray, PhyloCounterData > PhyloStatsCounter`
- `typedef Model< PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel`
- `typedef PowerSet< PhyloArray, PhyloRuleData > PhyloPowerSet`

## Functions

- `std::string get_last_name (unsigned int d)`
- `void counter_overall_gains (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)`  
*Overall functional gains.*
- `void counter_gains (PhyloCounters *counters, std::vector< uint > nfun, unsigned int duplication=DEFAULT_DUPLICATION)`  
*Functional gains for a specific function (nfun).*
- `void counter_gains_k_offspring (PhyloCounters *counters, std::vector< uint > nfun, uint k=1u, unsigned int duplication=DEFAULT_DUPLICATION)`  
*k genes gain function nfun*
- `void counter_genes_changing (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)`  
*Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)*
- `void counter_prop_genes_changing (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)`  
*Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)*
- `void counter_overall_loss (PhyloCounters *counters, unsigned int duplication=DEFAULT_DUPLICATION)`

- Overall functional loss.*

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

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

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

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

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

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

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

*Total number of neofunctionalization events.*
- void `counter_neofun_a2b` (`PhyloCounters *counters`, `uint nfunA`, `uint nfunB`, unsigned int duplication=`DEFAULT_DUPLICATION`)

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

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

*Indicator function. Equals to one if k genes changed and zero otherwise.*
- void `rule_dyn_limit_changes` (`PhyloSupport *support`, `uint pos`, `uint lb`, `uint ub`, unsigned int duplication=`DEFAULT_DUPLICATION`)

*Overall functional gains.*

## 8.26.1 Macro Definition Documentation

### 8.26.1.1 DEFAULT\_DUPLICATION

```
#define DEFAULT_DUPLICATION 1u
```

Definition at line 5 of file phylo.hpp.

### 8.26.1.2 DUPL\_DUPL

```
#define DUPL_DUPL 1u
```

Definition at line 7 of file phylo.hpp.

### 8.26.1.3 DUPL\_EITH

```
#define DUPL_EITH 2u
```

Definition at line 8 of file phylo.hpp.

#### 8.26.1.4 DUPL\_SPEC

```
#define DUPL_SPEC 0u
```

Definition at line 6 of file phylo.hpp.

#### 8.26.1.5 IF\_MATCHES

```
#define IF_MATCHES( )
```

**Value:**

```
MAKE_DUPL_VARS() \
if (IS_EITHER() | IS_DUPLICATION() | IS_SPECIATION())
```

Definition at line 19 of file phylo.hpp.

#### 8.26.1.6 IF\_NOTMATCHES

```
#define IF_NOTMATCHES( )
```

**Value:**

```
MAKE_DUPL_VARS() \
if (!IS_EITHER() & !IS_DUPLICATION() & !IS_SPECIATION())
```

Definition at line 21 of file phylo.hpp.

#### 8.26.1.7 IS\_DUPLICATION

```
#define IS_DUPLICATION( ) ((DATA_AT == DUPL_DUPL) & (DPL))
```

Definition at line 16 of file phylo.hpp.

#### 8.26.1.8 IS\_EITHER

```
#define IS_EITHER( ) (DATA_AT == DUPL_EITH)
```

Definition at line 15 of file phylo.hpp.

### 8.26.1.9 IS\_SPECIATION

```
#define IS_SPECIATION( ) ((DATA_AT == DUPL_SPEC) & (!DPL))
```

Definition at line 17 of file phylo.hpp.

### 8.26.1.10 MAKE\_DUPL\_VARS

```
#define MAKE_DUPL_VARS( )
```

**Value:**

```
bool DPL = Array.D()->duplication; \
unsigned int DATA_AT = data->at(0u);
```

Definition at line 11 of file phylo.hpp.

### 8.26.1.11 PHYLO\_CHECK\_MISSING

```
#define PHYLO_CHECK_MISSING( )
```

**Value:**

```
if (Array.D() == nullptr) \
throw std::logic_error("The array data is nullptr."); \
if (data == nullptr) \
throw std::logic_error("The counter/rule data is nullptr.")
```

Definition at line 136 of file phylo.hpp.

### 8.26.1.12 PHYLO\_COUNTER\_LAMBDA

```
#define PHYLO_COUNTER_LAMBDA(
    a )
```

**Value:**

```
Counter_fun_type<PhyloArray, PhyloCounterData> a = \
[] (const PhyloArray & Array, uint i, uint j, PhyloCounterData * data)
```

Extension of a simple counter.

It allows specifying extra arguments, in particular, the corresponding sets of rows to which this statistic may be relevant. This could be important in the case of, for example, counting correlation type statistics between function 1 and 2, and between function 1 and 3.

Definition at line 130 of file phylo.hpp.

### 8.26.1.13 PHYLO\_RULE\_DYN\_LAMBDA

```
#define PHYLO_RULE_DYN_LAMBDA(  
    a )
```

**Value:**

```
Rule_fun_type<PhyloArray, PhyloRuleDynData> a = \  
[] (const PhyloArray & Array, uint i, uint j, PhyloRuleDynData * data)
```

Definition at line 133 of file phylo.hpp.

## 8.26.2 Typedef Documentation

### 8.26.2.1 PhyloArray

```
typedef BArray<uint, NodeData> PhyloArray
```

Definition at line 103 of file phylo.hpp.

### 8.26.2.2 PhyloCounter

```
typedef Counter<PhyloArray, PhyloCounterData > PhyloCounter
```

Definition at line 104 of file phylo.hpp.

### 8.26.2.3 PhyloCounters

```
typedef Counters< PhyloArray, PhyloCounterData> PhyloCounters
```

Definition at line 105 of file phylo.hpp.

### 8.26.2.4 PhyloModel

```
typedef Model<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel
```

Definition at line 115 of file phylo.hpp.



### 8.26.2.5 PhyloPowerSet

```
typedef PowerSet<PhyloArray, PhyloRuleData> PhyloPowerSet
```

Definition at line 116 of file phylo.hpp.

### 8.26.2.6 PhyloRule

```
typedef Rule<PhyloArray,PhyloRuleData> PhyloRule
```

Definition at line 107 of file phylo.hpp.

### 8.26.2.7 PhyloRuleData

```
typedef std::vector< std::pair< uint, uint > > PhyloRuleData
```

Definition at line 96 of file phylo.hpp.

### 8.26.2.8 PhyloRuleDyn

```
typedef Rule<PhyloArray,PhyloRuleDynData> PhyloRuleDyn
```

Definition at line 110 of file phylo.hpp.

### 8.26.2.9 PhyloRules

```
typedef Rules<PhyloArray,PhyloRuleData> PhyloRules
```

Definition at line 108 of file phylo.hpp.

### 8.26.2.10 PhyloRulesDyn

```
typedef Rules<PhyloArray,PhyloRuleDynData> PhyloRulesDyn
```

Definition at line 111 of file phylo.hpp.

### 8.26.2.11 PhyloStatsCounter

```
typedef StatsCounter<PhyloArray, PhyloCounterData> PhyloStatsCounter
```

Definition at line 114 of file phylo.hpp.

### 8.26.2.12 PhyloSupport

```
typedef Support<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport
```

Definition at line 113 of file phylo.hpp.

## 8.26.3 Function Documentation

### 8.26.3.1 get\_last\_name()

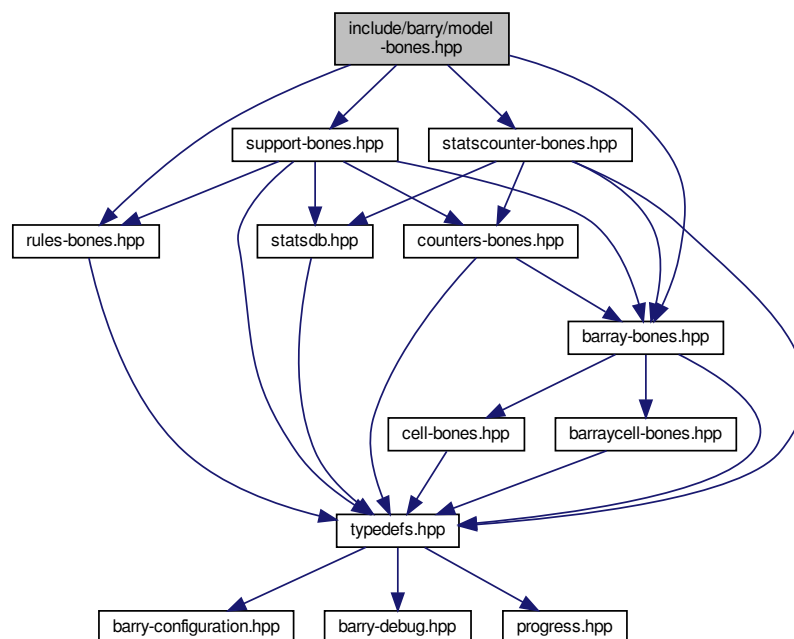
```
std::string get_last_name (
    unsigned int d ) [inline]
```

Definition at line 141 of file phylo.hpp.

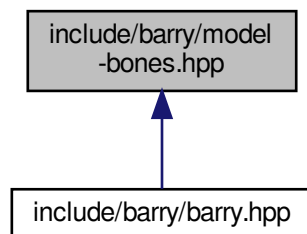
## 8.27 include/barry/model-bones.hpp File Reference

```
#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)*

### 8.27.1 Function Documentation

#### 8.27.1.1 [keygen\\_default\(\)](#)

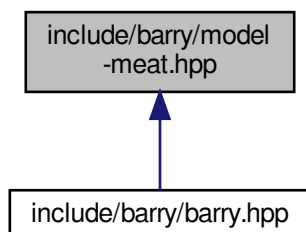
```
template<typename Array_Type >
std::vector< double > keygen_default (
    const Array_Type & Array_ ) [inline]
```

Array Hasher class (used for computing support)

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

## 8.28 include/barry/model-meat.hpp File Reference

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



### Macros

- `#define MODEL_TYPE()`
- `#define MODEL_TEMPLATE_ARGS()`
- `#define MODEL_TEMPLATE(a, b) template MODEL_TEMPLATE_ARGS() inline a MODEL_TYPE()::b`

### Functions

- `double update_normalizing_constant(const std::vector< double > &params, const Counts_type &support)`
- `double likelihood_(const std::vector< double > &target_stats, const std::vector< double > &params, const double normalizing_constant, bool log_=false)`
- `MODEL_TEMPLATE(, Model)()`
- `MODEL_TEMPLATE(, Model)(const MODEL_TYPE() &Model_)`

### 8.28.1 Macro Definition Documentation

#### 8.28.1.1 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.

### 8.28.1.2 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.

### 8.28.1.3 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.

## 8.28.2 Function Documentation

### 8.28.2.1 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.

### 8.28.2.2 MODEL\_TEMPLATE() [1/2]

```
MODEL_TEMPLATE (  
    Model )
```

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

### 8.28.2.3 MODEL\_TEMPLATE() [2/2]

```
MODEL_TEMPLATE (
    Model ) const &
```

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

### 8.28.2.4 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.

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

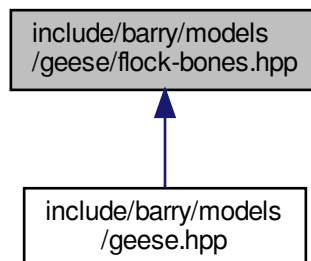
```
#include "geese/geese-node-bones.hpp"
#include "geese/geese-bones.hpp"
#include "geese/geese-meat.hpp"
#include "geese/geese-meat-constructors.hpp"
#include "geese/geese-meat-likelihood.hpp"
#include "geese/geese-meat-likelihood_exhaust.hpp"
#include "geese/geese-meat-simulate.hpp"
#include "geese/geese-meat-predict.hpp"
#include "geese/geese-meat-predict_exhaust.hpp"
#include "geese/geese-meat-predict_sim.hpp"
#include "geese/flock-bones.hpp"
#include "geese/flock-meat.hpp"
```

Include dependency graph for geese.hpp:



## 8.30 include/barry/models/geese/flock-bones.hpp File Reference

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



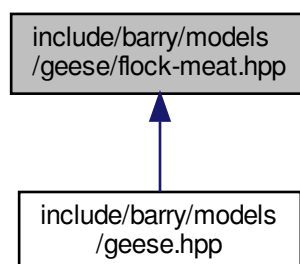
### Classes

- class [Flock](#)

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

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

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



## 8.32 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)

#### 8.32.1 Macro Definition Documentation



### 8.32.1.1 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.

## 8.32.2 Function Documentation

### 8.32.2.1 keygen\_full()

```
std::vector< double > keygen_full (   
    const phylocounters::PhyloArray & array ) [inline]
```

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

### 8.32.2.2 RULE\_FUNCTION()

```
RULE_FUNCTION (   
    rule_empty_free )
```

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

### 8.32.2.3 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.

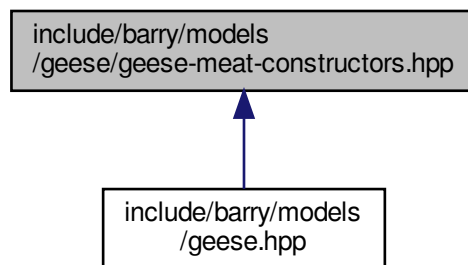
### 8.32.2.4 vector\_caster()

```
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.

### 8.33 include/barry/models/geese/geese-meat-constructors.hpp File Reference

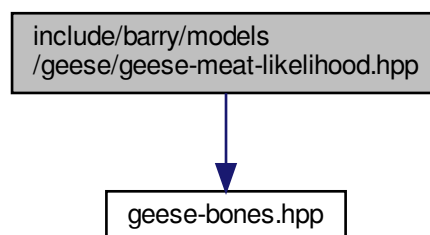
This graph shows which files directly or indirectly include this file:



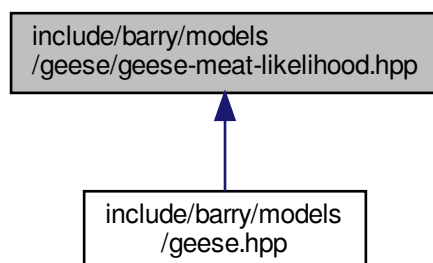
### 8.34 include/barry/models/geese/geese-meat-likelihood.hpp File Reference

```
#include "geese-bones.hpp"
```

Include dependency graph for `geese-meat-likelihood.hpp`:

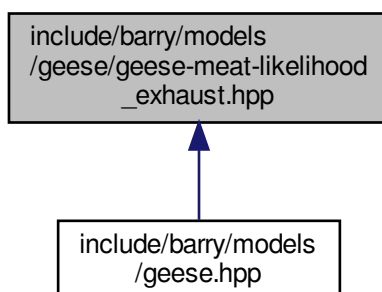


This graph shows which files directly or indirectly include this file:



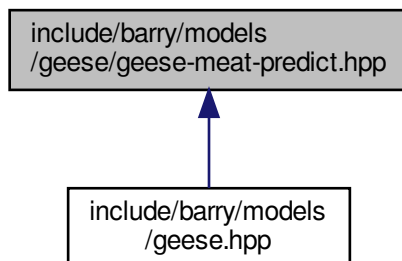
### 8.35 include/barry/models/geese/geese-meat-likelihood\_exhaust.hpp File Reference

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



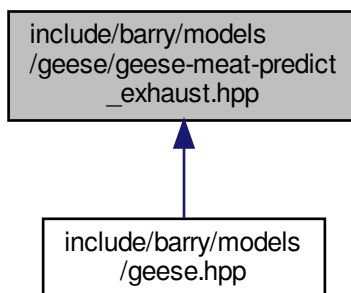
### 8.36 include/barry/models/geese/geese-meat-predict.hpp File Reference

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



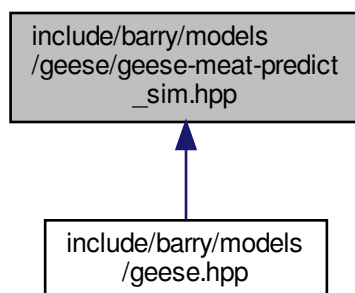
### 8.37 include/barry/models/geese/geese-meat-predict\_exhaust.hpp File Reference

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



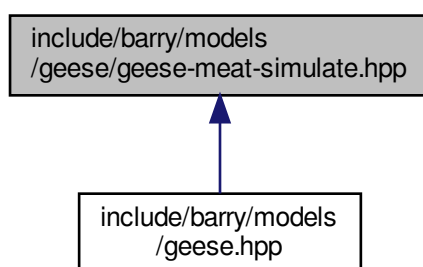
## 8.38 include/barry/models/geese/geese-meat-predict\_sim.hpp File Reference

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



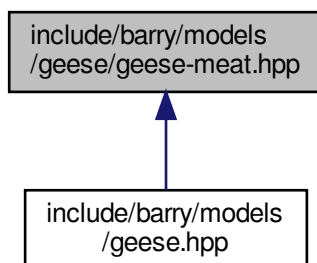
## 8.39 include/barry/models/geese/geese-meat-simulate.hpp File Reference

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



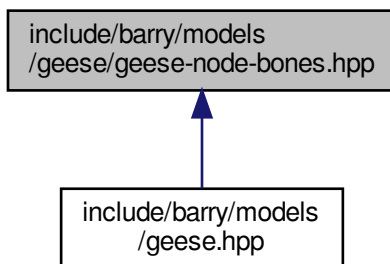
## 8.40 include/barry/models/geese/geese-meat.hpp File Reference

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



## 8.41 include/barry/models/geese/geese-node-bones.hpp File Reference

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



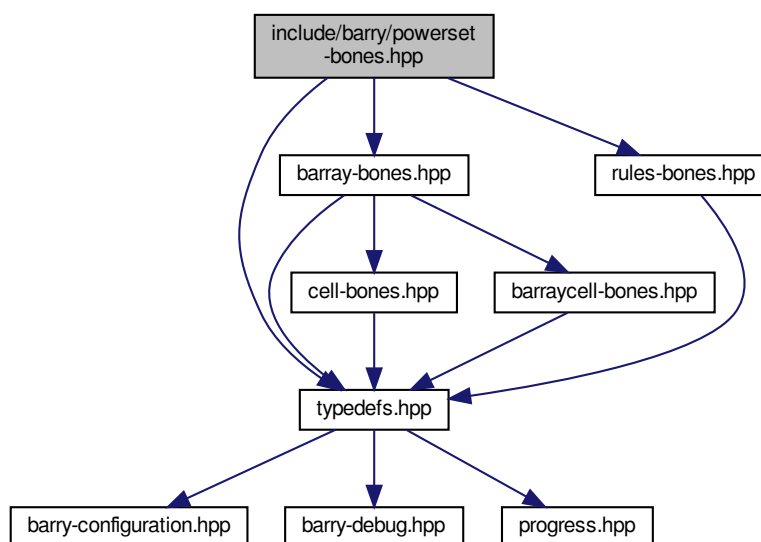
### Classes

- class [Node](#)  
*A single node for the model.*

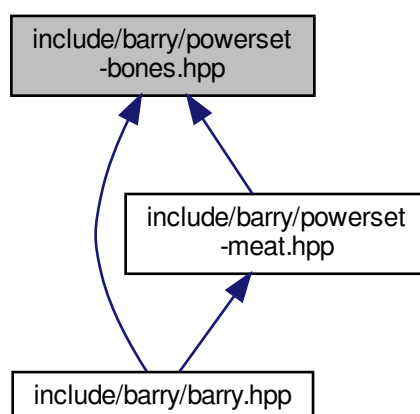
## 8.42 include/barry/powerset-bones.hpp File Reference

```
#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

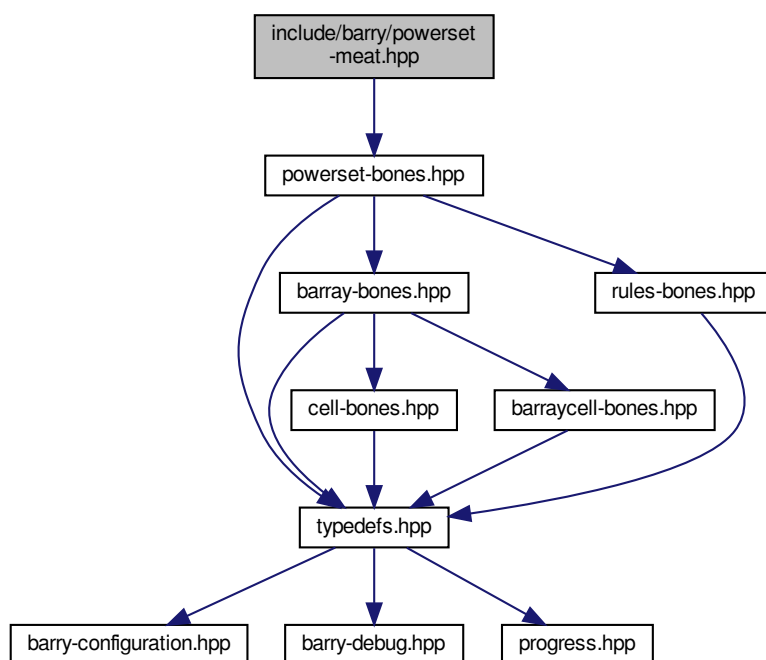
- class [PowerSet< Array\\_Type, Data\\_Rule\\_Type >](#)

*Powerset of a binary array.*

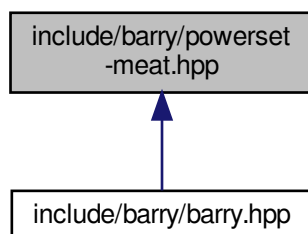
## 8.43 include/barry/powerset-meat.hpp File Reference

```
#include "powerset-bones.hpp"
```

Include dependency graph for powerset-meat.hpp:



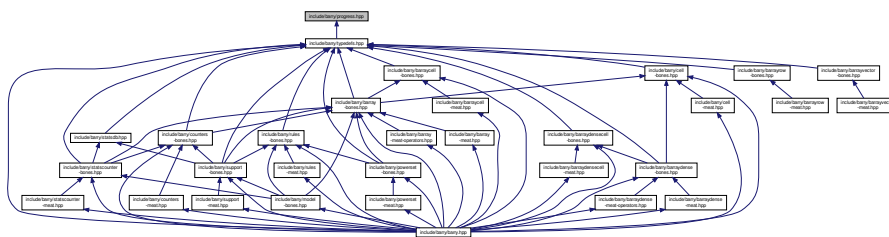
This graph shows which files directly or indirectly include this file:





#### 8.44 include/barry/progress.hpp File Reference

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



## Classes

- class **Progress**  
*A simple progress bar.*

## Macros

- #define BARRY\_PROGRESS\_BAR\_WIDTH 80

### 8.44.1 Macro Definition Documentation

#### 8.44.1.1 BARRY\_PROGRESS\_BAR\_WIDTH

```
#define BARRY_PROGRESS_BAR_WIDTH 80
```

Definition at line 5 of file progress.hpp.

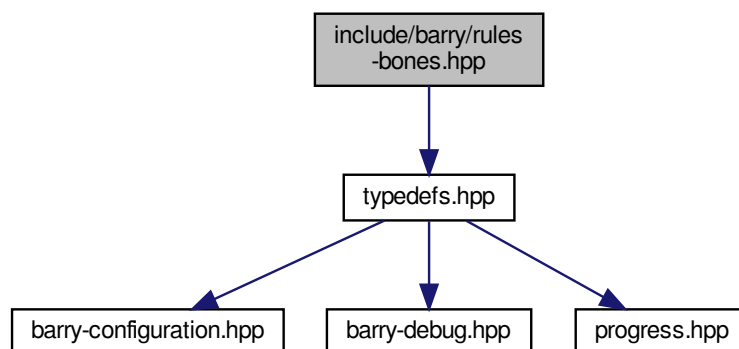
#### 8.45 include/barry/rules-bones.hpp File Reference

```
#include "typedefs.hpp"
```

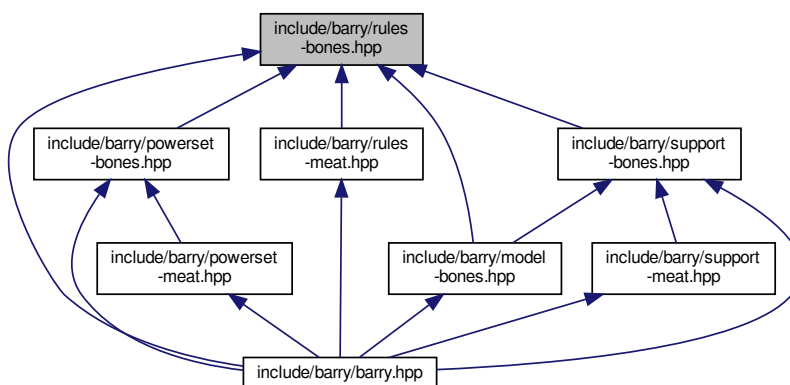
```

#include "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)`

### 8.45.1 Function Documentation

#### 8.45.1.1 rule\_fun\_default()

```

template<typename Array_Type , typename Data_Type >
bool rule_fun_default (
    const Array_Type * array,
    uint i,
    uint j,
    Data_Type * dat )

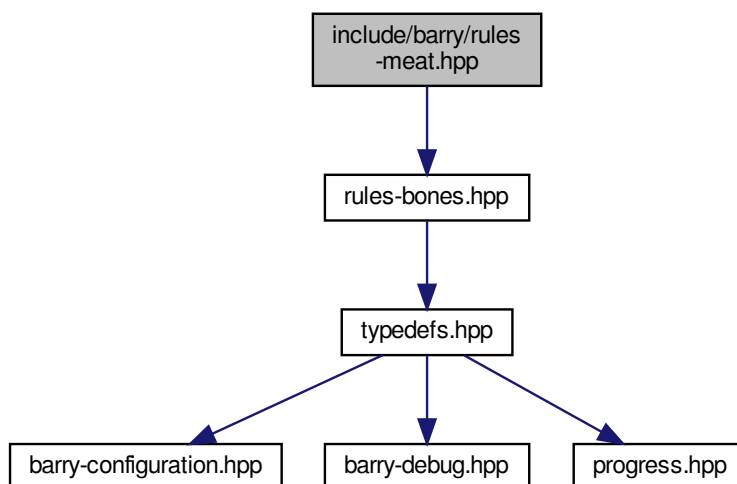
```

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

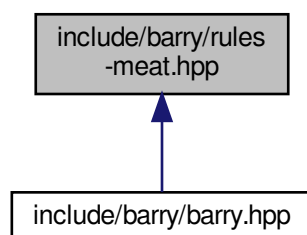
## 8.46 include/barry/rules-meat.hpp File Reference

```
#include "rules-bones.hpp"
```

Include dependency graph for rules-meat.hpp:



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

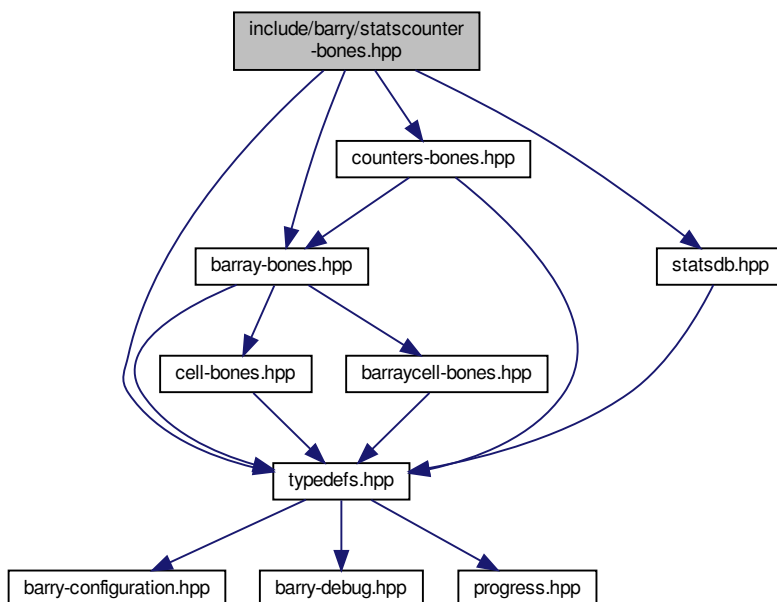


## 8.47 include/barry/statscounter-bones.hpp File Reference

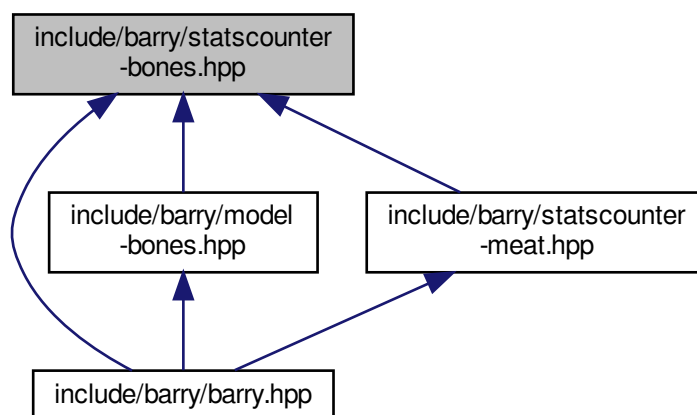
```
#include "typedefs.hpp"  
#include "barray-bones.hpp"  
#include "statsdb.hpp"
```

```
#include "counters-bones.hpp"
```

Include dependency graph for statscounter-bones.hpp:



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



## Classes

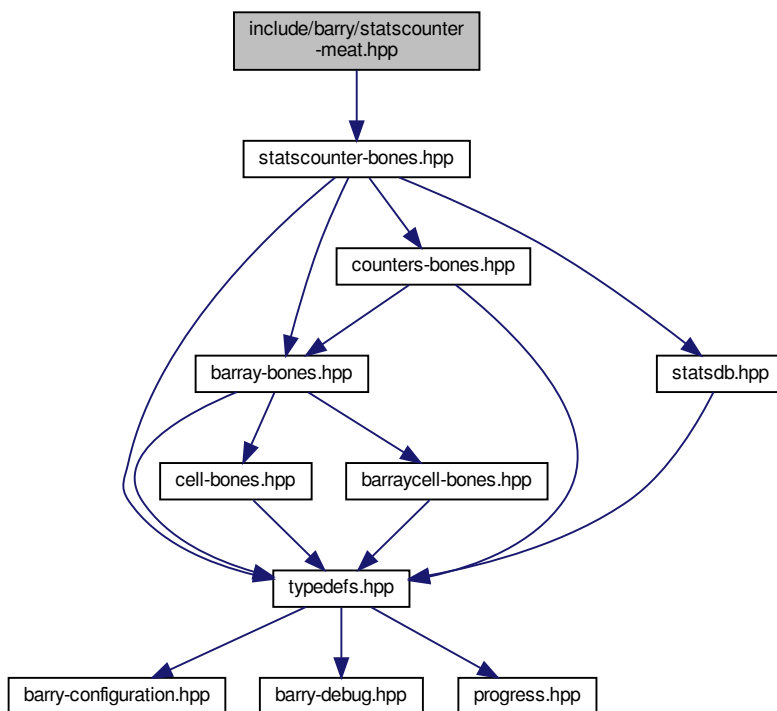
- class [StatsCounter< Array\\_Type, Data\\_Type >](#)

*Count stats for a single Array.*

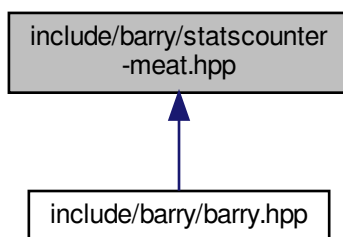
## 8.48 include/barry/statscounter-meat.hpp File Reference

```
#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](#)

## 8.48.1 Macro Definition Documentation

### 8.48.1.1 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.

### 8.48.1.2 STATSCOUNTER\_TEMPLATE\_ARGS

```
template STATSCOUNTER\_TEMPLATE\_ARGS( ) <typename Array_Type, typename Data_Type>
```

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

### 8.48.1.3 STATSCOUNTER\_TYPE

```
template Data\_Type \* STATSCOUNTER\_TYPE( ) StatsCounter<Array\_Type,Data\_Type>
```

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

## 8.48.2 Function Documentation

### 8.48.2.1 for()

```
for (
    uint n = 0; n < counters->size(); ++n ) = change_stats[pos][n]
```

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

### 8.48.2.2 resize()

```
current_stats resize (
    counters-> size(),
    0. 0 )
```

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

```
STATSCOUNTER_TEMPLATE (
    ~ StatsCounter )
```

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

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

```
STATSCOUNTER_TEMPLATE (
    std::vector< double > ,
    count_all )
```

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

### 8.48.2.5 STATSCOUNTER\_TEMPLATE() [3/9]

```
STATSCOUNTER_TEMPLATE (
    std::vector< std::string > ,
    get_descriptions ) const
```

Definition at line 171 of file statscounter-meat.hpp.

#### 8.48.2.6 STATSCOUNTER\_TEMPLATE() [4/9]

```
STATSCOUNTER_TEMPLATE (
    std::vector< std::string > ,
    get_names ) const
```

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

#### 8.48.2.7 STATSCOUNTER\_TEMPLATE() [5/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    add_counter )
```

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

```
STATSCOUNTER_TEMPLATE (
    void ,
    count_current )
```

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

```
STATSCOUNTER_TEMPLATE (
    void ,
    count_init )
```

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

```
STATSCOUNTER_TEMPLATE (
    void ,
    reset_array ) const
```

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

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

```
STATSCOUNTER_TEMPLATE (
    void ,
    set_counters )
```



### 8.48.3 Variable Documentation

#### 8.48.3.1 counter\_deleted

```
counter_deleted = true
```

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

#### 8.48.3.2 counters

```
counters = counters_
```

Definition at line 53 of file statscounter-meat.hpp.

#### 8.48.3.3 counters\_

```
Data_Type* counters_
```

**Initial value:**

```
{  
  
    if (!counter_deleted)  
        delete counters
```

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

#### 8.48.3.4 f\_

```
Data_Rule_Dyn_Type f_
```

**Initial value:**

```
{  
  
    counters->add_counter(f_)
```

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

### 8.48.3.5 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.

### 8.48.3.6 return

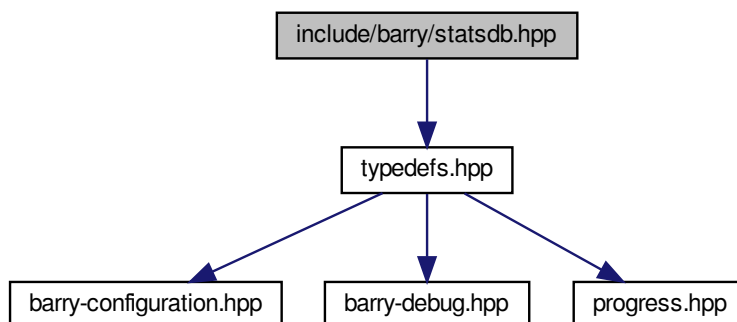
```
return
```

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

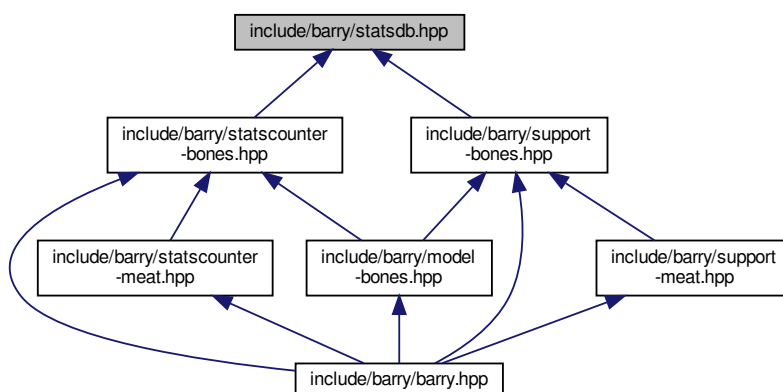
## 8.49 include/barry/statsdb.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for statsdb.hpp:



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



## Classes

- class [FreqTable< T >](#)  
*Database of statistics.*

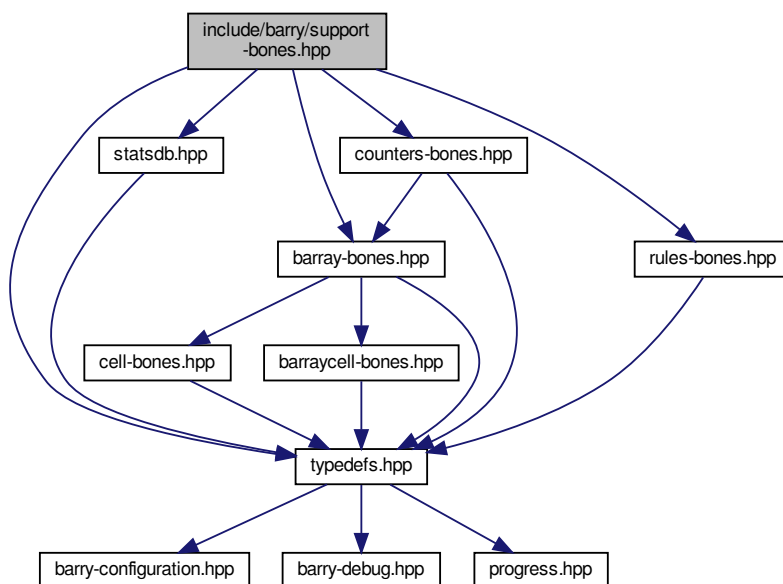
## 8.50 include/barry/support-bones.hpp File Reference

```

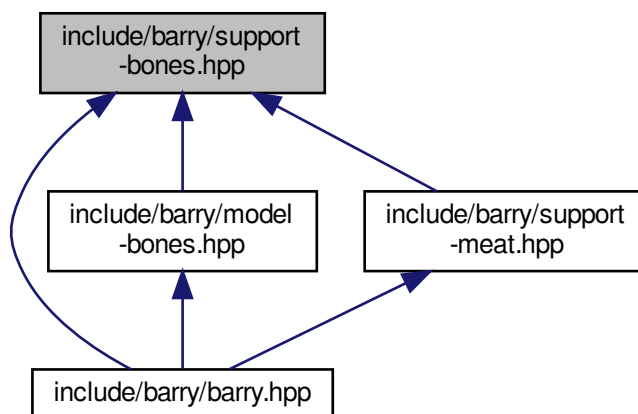
#include "typedefs.hpp"
#include "barray-bones.hpp"
#include "statsdb.hpp"
#include "counters-bones.hpp"
#include "rules-bones.hpp"

```

Include dependency graph for support-bones.hpp:



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



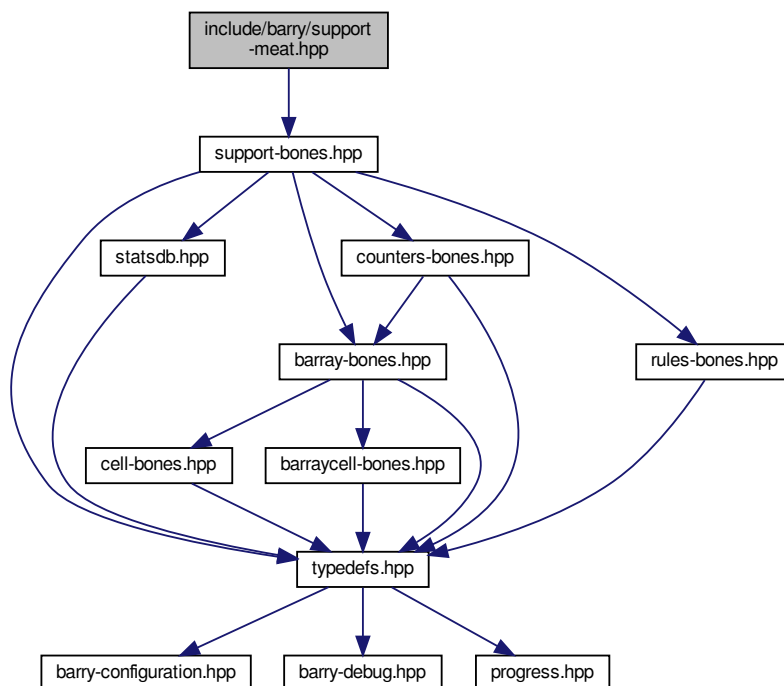
## Classes

- class [Support< Array\\_Type, Data\\_Counter\\_Type, Data\\_Rule\\_Type, Data\\_Rule\\_Dyn\\_Type >](#)  
*Compute the support of sufficient statistics.*

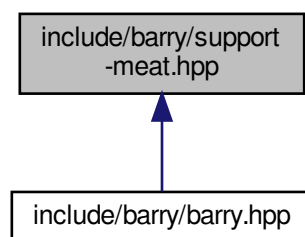
## 8.51 include/barry/support-meat.hpp File Reference

```
#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_`

### 8.51.1 Macro Definition Documentation

### 8.51.1.1 BARRY\_SUPPORT\_MEAT\_HPP

```
#define BARRY_SUPPORT_MEAT_HPP 1
```

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

### 8.51.1.2 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.

### 8.51.1.3 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.

### 8.51.1.4 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.

## 8.51.2 Function Documentation

### 8.51.2.1 calc\_backend()

```
calc_backend (
    pos+ 1u,
    array_bank ,
    stats_bank )
```

### 8.51.2.2 for()

```
for (
    uint n = 0u; n < counters->size(); ++n )
```

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

### 8.51.2.3 if() [1/3]

```
if (
    array_bank !    = nullptr ) -> push_back(EmptyArray)
```

### 8.51.2.4 if() [2/3]

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

Definition at line 158 of file support-meat.hpp.

### 8.51.2.5 if() [3/3]

```
if (
    stats_bank !    = nullptr ) -> push_back(current_stats)
```

### 8.51.2.6 insert\_cell()

```
EmptyArray insert_cell (
    cfree. first,
    cfree. second,
    EmptyArray.default_val(). value,
    false ,
    false )
```



### 8.51.2.7 rm\_cell()

```
EmptyArray rm_cell (
    cfree.  first,
    cfree.  second,
    false ,
    false )
```

### 8.51.2.8 SUPPORT\_TEMPLATE() [1/17]

```
SUPPORT_TEMPLATE (
    bool ,
    eval_rules_dyn ) const
```

Definition at line 330 of file support-meat.hpp.

### 8.51.2.9 SUPPORT\_TEMPLATE() [2/17]

```
SUPPORT_TEMPLATE (
    const FreqTable<> & ,
    get_data ) const
```

Definition at line 379 of file support-meat.hpp.

### 8.51.2.10 SUPPORT\_TEMPLATE() [3/17]

```
SUPPORT_TEMPLATE (
    const MapVec_type<> * ,
    get_counts_ptr ) const
```

Definition at line 358 of file support-meat.hpp.

### 8.51.2.11 SUPPORT\_TEMPLATE() [4/17]

```
SUPPORT_TEMPLATE (
    Counts_type ,
    get_counts ) const
```

Definition at line 352 of file support-meat.hpp.

**8.51.2.12 SUPPORT\_TEMPLATE() [5/17]**

```
SUPPORT_TEMPLATE (
    std::vector< double > * ,
    get_current_stats )
```

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

**8.51.2.13 SUPPORT\_TEMPLATE() [6/17]**

```
SUPPORT_TEMPLATE (
    void ,
    add_counter )
```

**8.51.2.14 SUPPORT\_TEMPLATE() [7/17]**

```
SUPPORT_TEMPLATE (
    void ,
    add_rule )
```

**8.51.2.15 SUPPORT\_TEMPLATE() [8/17]**

```
SUPPORT_TEMPLATE (
    void ,
    add_rule_dyn )
```

**8.51.2.16 SUPPORT\_TEMPLATE() [9/17]**

```
SUPPORT_TEMPLATE (
    void ,
    calc )
```

Definition at line 207 of file support-meat.hpp.

**8.51.2.17 SUPPORT\_TEMPLATE() [10/17]**

```
SUPPORT_TEMPLATE (
    void ,
    calc_backend )
```

**8.51.2.18 SUPPORT\_TEMPLATE()** [11/17]

```
SUPPORT_TEMPLATE (
    void ,
    init_support )
```

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

**8.51.2.19 SUPPORT\_TEMPLATE()** [12/17]

```
SUPPORT_TEMPLATE (
    void ,
    print ) const
```

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

**8.51.2.20 SUPPORT\_TEMPLATE()** [13/17]

```
SUPPORT_TEMPLATE (
    void ,
    reset_array )
```

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

**8.51.2.21 SUPPORT\_TEMPLATE()** [14/17]

```
SUPPORT_TEMPLATE (
    void ,
    reset_array ) const &
```

Definition at line 97 of file support-meat.hpp.

**8.51.2.22 SUPPORT\_TEMPLATE()** [15/17]

```
SUPPORT_TEMPLATE (
    void ,
    set_counters )
```

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

```
SUPPORT_TEMPLATE (
    void ,
    set_rules )
```

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

```
SUPPORT_TEMPLATE (
    void ,
    set_rules_dyn )
```

### 8.51.3 Variable Documentation

#### 8.51.3.1 array\_bank

```
std::vector< Array_Type >* array_bank
```

Definition at line 109 of file support-meat.hpp.

#### 8.51.3.2 cfree

```
const std::pair<uint,uint>& cfree = coordinates_free[pos]
```

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

#### 8.51.3.3 counters

```
counters = counters_
```

Definition at line 258 of file support-meat.hpp.

#### 8.51.3.4 counters\_

Data\_Counter\_Type\* counters\_

##### Initial value:

```
{  
  
    if (delete_counters)  
        delete counters
```

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

#### 8.51.3.5 delete\_counters

delete\_counters = false

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

#### 8.51.3.6 delete\_rules

delete\_rules = false

Definition at line 291 of file support-meat.hpp.

#### 8.51.3.7 delete\_rules\_dyn

delete\_rules\_dyn = false

Definition at line 323 of file support-meat.hpp.

#### 8.51.3.8 else

else

##### Initial value:

```
{  
    data.add(current_stats)
```

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

### 8.51.3.9 f\_

Data\_Rule\_Dyn\_Type f\_

#### Initial value:

```
{  
    counters->add_counter(f_)
```

Definition at line 233 of file support-meat.hpp.

### 8.51.3.10 return

```
return
```

Definition at line 203 of file support-meat.hpp.

### 8.51.3.11 rules

```
rules = rules_
```

Definition at line 292 of file support-meat.hpp.

### 8.51.3.12 rules\_

Data\_Rule\_Dyn\_Type \* rules\_

#### Initial value:

```
{  
  
    if (delete_rules)  
        delete rules
```

Definition at line 285 of file support-meat.hpp.

### 8.51.3.13 rules\_dyn

```
rules_dyn = rules_
```

Definition at line 324 of file support-meat.hpp.

## 8.51.3.14 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.

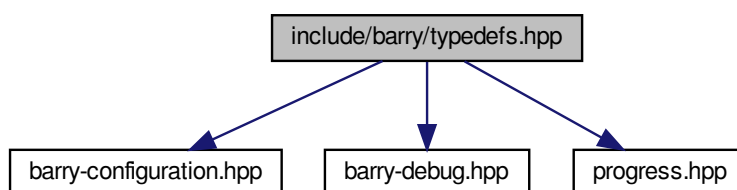
## 8.52 include/barry/typedefs.hpp File Reference

```
#include "barry-configuration.hpp"
```

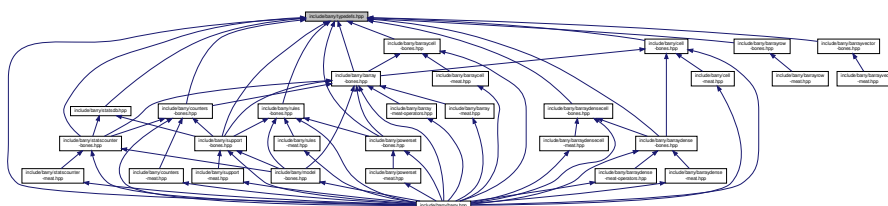
```
#include "barry-debug.hpp"
```

```
#include "progress.hpp"
```

Include dependency graph for typedefs.hpp:



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



## Classes

- class [Entries< Cell\\_Type >](#)

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

- struct [vecHasher< T >](#)

## Namespaces

- [CHECK](#)  
*Integer constants used to specify which cell should be check.*
- [EXISTS](#)  
*Integer constants used to specify which cell should be check to exist or not.*

## Typedefs

- typedef unsigned int [uint](#)
- typedef std::vector< std::pair< std::vector< double >, [uint](#) > > [Counts\\_type](#)
- template<typename Cell\_Type >  
using [Row\\_type](#) = [Map](#)< [uint](#), [Cell](#)< Cell\_Type > >
- template<typename Cell\_Type >  
using [Col\\_type](#) = [Map](#)< [uint](#), [Cell](#)< Cell\_Type > \* >
- template<typename Ta = double, typename Tb = uint>  
using [MapVec\\_type](#) = std::unordered\_map< std::vector< Ta >, Tb, [vecHasher](#)< Ta > >
  
- template<typename Array\_Type , typename Data\_Type >  
using [Counter\\_fun\\_type](#) = std::function< double([const](#) Array\_Type &, [uint](#), [uint](#), Data\_Type \*)>  
*[Counter](#) and rule functions.*
- template<typename Array\_Type , typename Data\_Type >  
using [Rule\\_fun\\_type](#) = std::function< bool([const](#) Array\_Type &, [uint](#), [uint](#), Data\_Type \*)>

## Functions

- template<typename T >  
T [vec\\_inner\\_prod](#) ([const](#) std::vector< T > &a, [const](#) std::vector< T > &b)
  
- template<typename T >  
bool [vec\\_equal](#) ([const](#) std::vector< T > &a, [const](#) std::vector< T > &b)  
*Compares if -a- and -b- are equal.*
- template<typename T >  
bool [vec\\_equal\\_approx](#) ([const](#) std::vector< T > &a, [const](#) std::vector< T > &b, double eps=1e-10)

## Variables

- [const](#) int [CHECK::BOTH](#) = -1
- [const](#) int [CHECK::NONE](#) = 0
- [const](#) int [CHECK::ONE](#) = 1
- [const](#) int [CHECK::TWO](#) = 2
- [const](#) int [EXISTS::BOTH](#) = -1
- [const](#) int [EXISTS::NONE](#) = 0
- [const](#) int [EXISTS::ONE](#) = 1
- [const](#) int [EXISTS::TWO](#) = 1
- [const](#) int [EXISTS::UNKNOWN](#) = -1
- [const](#) int [EXISTS::AS\\_ZERO](#) = 0
- [const](#) int [EXISTS::AS\\_ONE](#) = 1



## 8.52.1 Typedef Documentation

### 8.52.1.1 Col\_type

```
template<typename Cell_Type >
using Col_type = Map< uint, Cell<Cell_Type>* >
```

Definition at line 59 of file typedefs.hpp.

### 8.52.1.2 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 <a href="#">BArray</a>
<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 132 of file typedefs.hpp.

### 8.52.1.3 Counts\_type

```
typedef std::vector< std::pair< std::vector<double>, uint > > Counts_type
```

Definition at line 52 of file typedefs.hpp.

### 8.52.1.4 MapVec\_type

```
template<typename Ta = double, typename Tb = uint>
using MapVec_type = std::unordered_map< std::vector< Ta >, Tb, vecHasher<Ta> >
```

Definition at line 113 of file typedefs.hpp.

### 8.52.1.5 Row\_type

```
template<typename Cell_Type >
using Row_type = Map< uint, Cell<Cell_Type> >
```

Definition at line 56 of file typedefs.hpp.

### 8.52.1.6 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 135 of file typedefs.hpp.

### 8.52.1.7 uint

```
typedef unsigned int uint
```

Definition at line 18 of file typedefs.hpp.

## 8.52.2 Function Documentation

### 8.52.2.1 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 146 of file typedefs.hpp.

### 8.52.2.2 `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 164 of file typedefs.hpp.

### 8.52.2.3 `vec_inner_prod()`

```
template<typename T >
T vec_inner_prod (
    const std::vector< T > & a,
    const std::vector< T > & b ) [inline]
```

Definition at line 184 of file typedefs.hpp.

## 8.53 README.md File Reference



# Index

- ~BArray
  - BArray< Cell\_Type, Data\_Type >, [33](#)
- ~BArrayCell
  - BArrayCell< Cell\_Type, Data\_Type >, [44](#)
- ~BArrayCell\_const
  - BArrayCell\_const< Cell\_Type, Data\_Type >, [46](#)
- ~BArrayDense
  - BArrayDense< Cell\_Type, Data\_Type >, [52](#)
- ~BArrayDenseCell
  - BArrayDenseCell< Cell\_Type, Data\_Type >, [63](#)
- ~BArrayDenseCell\_const
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, [65](#)
- ~BArrayRow
  - BArrayRow< Cell\_Type, Data\_Type >, [68](#)
- ~BArrayRow\_const
  - BArrayRow\_const< Cell\_Type, Data\_Type >, [70](#)
- ~BArrayVector
  - BArrayVector< Cell\_Type, Data\_Type >, [73](#)
- ~BArrayVector\_const
  - BArrayVector\_const< Cell\_Type, Data\_Type >, [76](#)
- ~Cell
  - Cell< Cell\_Type >, [80](#)
- ~ConstBArrayRowIter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [85](#)
- ~Counter
  - Counter< Array\_Type, Data\_Type >, [88](#)
- ~Counters
  - Counters< Array\_Type, Data\_Type >, [92](#)
- ~Entries
  - Entries< Cell\_Type >, [97](#)
- ~Flock
  - Flock, [100](#)
- ~FreqTable
  - FreqTable< T >, [105](#)
- ~Geese
  - Geese, [111](#)
- ~Model
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [123](#)
- ~NetCounterData
  - NetCounterData, [134](#)
- ~NetworkData
  - NetworkData, [136](#)
- ~Node
  - Node, [139](#)
- ~PhyloRuleDynData
  - PhyloRuleDynData, [147](#)
- ~PowerSet
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [151](#)
- ~Progress
  - Progress, [156](#)
- ~Rule
  - Rule< Array\_Type, Data\_Type >, [157](#)
- ~Rules
  - Rules< Array\_Type, Data\_Type >, [159](#)
- ~StatsCounter
  - StatsCounter< Array\_Type, Data\_Type >, [163](#)
- ~Support
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [168](#)
- active
  - Cell< Cell\_Type >, [83](#)
- add
  - barray-meat.hpp, [196](#)
  - barraydense-meat.hpp, [219](#)
  - Cell< Cell\_Type >, [81](#), [82](#)
  - FreqTable< T >, [106](#)
- add\_array
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [124](#)
- add\_counter
  - Counters< Array\_Type, Data\_Type >, [93](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [124](#)
  - StatsCounter< Array\_Type, Data\_Type >, [163](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [168](#)
- add\_data
  - Flock, [100](#)
- add\_rule
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [125](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [151](#)
  - Rules< Array\_Type, Data\_Type >, [160](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [169](#)
- add\_rule\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [125](#), [126](#)

- Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
169
- annotations
  - Node, 140
- ans
  - barray-meat.hpp, 187, 196
  - barraydense-meat.hpp, 212, 219
- Array
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, 85
- array
  - Node, 140
- Array\_
  - barray-meat.hpp, 196
- array\_bank
  - support-meat.hpp, 306
- arrays
  - Node, 140
- AS\_ONE
  - EXISTS, 27
- as\_vector
  - FreqTable< T >, 106
- AS\_ZERO
  - EXISTS, 27
- at
  - PhyloCounterData, 145
- 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, 178
- barray-meat-operators.hpp
  - BARRAY\_TEMPLATE, 180–182
  - BARRAY\_TEMPLATE\_ARGS, 180, 182
  - BARRAY\_TYPE, 180, 182
  - BARRY\_BARRAY\_MEAT\_OPERATORS\_HPP,  
180
  - COL, 181
  - for, 182
  - operator(), 183
  - rhs, 183
  - ROW, 181
  - this, 183
- barray-meat.hpp
  - add, 196
  - ans, 187, 196
  - Array\_, 196
  - BARRAY\_TEMPLATE, 186–191
  - BARRAY\_TEMPLATE\_ARGS, 186
  - BARRAY\_TYPE, 186
  - check\_bounds, 196
  - check\_exists, 196
  - COL, 187, 191
  - col0, 197
  - const, 197
  - copy\_data, 197
  - data, 197
  - delete\_data, 197
  - delete\_data\_, 198
  - else, 198
  - false, 198
  - first, 198
  - for, 191, 192
  - i1, 198
  - if, 192–194
  - j, 199
  - j0, 199
  - j1, 199
  - M, 195, 199
  - M\_, 199
  - N, 200
  - NCells, 200
  - report, 200
  - resize, 195
  - return, 195, 200

- ROW, [187](#), [195](#)
  - row0, [200](#)
  - search, [201](#)
  - source, [201](#)
  - target, [201](#)
  - v, [201](#)
  - value, [201](#)
- BARRAY\_BONES\_HPP
  - barray-bones.hpp, [178](#)
- BARRAY\_TEMPLATE
  - barray-meat-operators.hpp, [180–182](#)
  - barray-meat.hpp, [186–191](#)
- BARRAY\_TEMPLATE\_ARGS
  - barray-meat-operators.hpp, [180](#), [182](#)
  - barray-meat.hpp, [186](#)
- BARRAY\_TYPE
  - barray-meat-operators.hpp, [180](#), [182](#)
  - barray-meat.hpp, [186](#)
- BArrayCell
  - BArrayCell< Cell\_Type, Data\_Type >, [43](#)
- BArrayCell< Cell\_Type, Data\_Type >, [43](#)
  - ~BArrayCell, [44](#)
  - BArray< Cell\_Type, Data\_Type >, [42](#)
  - BArrayCell, [43](#)
  - 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](#)
  - 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](#)
  - BArrayDense, [51](#), [52](#)
  - BArrayDenseCell< Cell\_Type, Data\_Type >, [61](#)
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, [61](#)
  - clear, [52](#)
  - col, [52](#)
  - D, [53](#)
  - default\_val, [53](#)
  - get\_cell, [53](#)
  - get\_col\_vec, [53](#)
  - get\_entries, [54](#)
  - get\_row\_vec, [54](#)
  - insert\_cell, [54](#), [55](#)
  - is\_empty, [55](#)
  - ncol, [55](#)
  - nnozero, [55](#)
  - nrow, [55](#)
  - operator\*=, [56](#)
  - operator(), [56](#)
  - operator+=, [56](#)
  - operator-=, [57](#)
  - operator/=: [57](#)
  - operator=: [57](#)
  - operator==, [58](#)
  - out\_of\_range, [58](#)
  - print, [58](#)
  - reserve, [58](#)
  - resize, [58](#)
  - rm\_cell, [58](#)
  - row, [59](#)
  - set\_data, [59](#)
  - swap\_cells, [59](#)
  - swap\_cols, [59](#)
  - swap\_rows, [60](#)
  - toggle\_cell, [60](#)
  - toggle\_lock, [60](#)
  - transpose, [60](#)
  - visited, [61](#)
  - zero\_col, [60](#)
  - zero\_row, [61](#)
- barraydense-bones.hpp
  - BARRY\_BARRAYDENSE\_BONES\_HPP, [205](#)
- barraydense-meat-operators.hpp
  - BARRY\_BARRAY\_MEAT\_OPERATORS\_HPP, [206](#)
  - BDENSE\_TEMPLATE, [206–208](#)
  - BDENSE\_TEMPLATE\_ARGS, [206](#), [208](#)
  - BDENSE\_TYPE, [206](#), [208](#)
  - COL, [207](#)
  - POS, [207](#)
  - POS\_N, [207](#)
  - ROW, [207](#)
- barraydense-meat.hpp
  - add, [219](#)
  - ans, [212](#), [219](#)
  - BDENSE\_TEMPLATE, [211](#), [213–217](#)
  - BDENSE\_TEMPLATE\_ARGS, [211](#)
  - BDENSE\_TYPE, [211](#)
  - check\_bounds, [220](#)
  - check\_exists, [220](#)
  - COL, [211](#)
  - col, [220](#)
  - const, [220](#)
  - copy\_data, [220](#)
  - data, [221](#)
  - delete\_data, [221](#)
  - delete\_data\_, [221](#)
  - el, [221](#)

- else, [221](#)
- false, [222](#)
- for, [217](#)
- i1, [222](#)
- if, [218](#)
- j, [222](#)
- j0, [222](#)
- j1, [222](#)
- M, [219](#), [222](#)
- M\_, [223](#)
- N, [223](#)
- NCells, [223](#)
- POS, [212](#)
- POS\_N, [212](#)
- report, [223](#)
- resize, [219](#)
- return, [223](#)
- ROW, [212](#)
- source, [224](#)
- target, [224](#)
- v, [224](#)
- value, [224](#)
- ZERO\_CELL, [212](#)
- BArrayDenseCell
  - BArrayDenseCell< Cell\_Type, Data\_Type >, [62](#)
- BArrayDenseCell< Cell\_Type, Data\_Type >, [62](#)
  - ~BArrayDenseCell, [63](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [61](#)
  - BArrayDenseCell, [62](#)
  - operator Cell\_Type, [63](#)
  - operator\*=[, 63](#)
  - operator+=[, 63](#)
  - operator-=[, 63](#)
  - operator/=[, 64](#)
  - operator=[, 64](#)
  - operator==[, 64](#)
- barraydensecell-meat.hpp
  - POS, [226](#)
- BArrayDenseCell\_const
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, [65](#)
- BArrayDenseCell\_const< Cell\_Type, Data\_Type >, [64](#)
  - ~BArrayDenseCell\_const, [65](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [61](#)
  - BArrayDenseCell\_const, [65](#)
  - operator Cell\_Type, [65](#)
  - operator!=[, 66](#)
  - operator<[, 66](#)
  - operator<=[, 66](#)
  - operator>[, 66](#)
  - operator>=[, 67](#)
  - operator==[, 66](#)
- BArrayRow
  - BArrayRow< Cell\_Type, Data\_Type >, [68](#)
- BArrayRow< Cell\_Type, Data\_Type >, [67](#)
  - ~BArrayRow, [68](#)
  - BArrayRow, [68](#)
  - operator BArrayRow< Cell\_Type, Data\_Type >, [68](#)
- operator\*=[, 68](#)
- operator+=[, 68](#)
- operator-=[, 68](#)
- operator/=[, 69](#)
- operator=[, 69](#)
- operator==[, 69](#)
- barrayrow-meat.hpp
  - BARRY\_BARRAYROW\_MEAT\_HPP, [228](#)
  - BROW\_TEMPLATE, [228–230](#)
  - BROW\_TEMPLATE\_ARGS, [229](#)
  - BROW\_TYPE, [229](#)
- BArrayRow\_const
  - BArrayRow\_const< Cell\_Type, Data\_Type >, [70](#)
- BArrayRow\_const< Cell\_Type, Data\_Type >, [69](#)
  - ~BArrayRow\_const, [70](#)
  - BArrayRow\_const, [70](#)
  - operator BArrayRow\_const< Cell\_Type, Data\_Type >, [70](#)
  - operator!=[, 70](#)
  - operator<[, 70](#)
  - operator<=[, 71](#)
  - operator>[, 71](#)
  - operator>=[, 71](#)
  - operator==[, 71](#)
- BArrayVector
  - BArrayVector< Cell\_Type, Data\_Type >, [72](#)
- BArrayVector< Cell\_Type, Data\_Type >, [71](#)
  - ~BArrayVector, [73](#)
  - BArrayVector, [72](#)
  - begin, [73](#)
  - end, [73](#)
  - is\_col, [73](#)
  - is\_row, [73](#)
  - operator std::vector< Cell\_Type >, [74](#)
  - operator\*=[, 74](#)
  - operator+=[, 74](#)
  - operator-=[, 74](#)
  - operator/=[, 74](#)
  - operator=[, 75](#)
  - operator==[, 75](#)
  - size, [75](#)
- barrayvector-meat.hpp
  - BARRY\_BARRAYVECTOR\_MEAT\_HPP, [232](#)
- BArrayVector\_const
  - BArrayVector\_const< Cell\_Type, Data\_Type >, [76](#)
- BArrayVector\_const< Cell\_Type, Data\_Type >, [75](#)
  - ~BArrayVector\_const, [76](#)
  - BArrayVector\_const, [76](#)
  - begin, [77](#)
  - end, [77](#)
  - is\_col, [77](#)
  - is\_row, [77](#)
  - operator std::vector< Cell\_Type >, [77](#)
  - operator!=[, 77](#)
  - operator<[, 78](#)
  - operator<=[, 78](#)
  - operator>[, 78](#)
  - operator>=[, 78](#)



- operator==, 78
- size, 79
- barry, 25
- barry-configuration.hpp
  - BARRY\_CHECK\_SUPPORT, 233
  - BARRY\_ISFINITE, 233
  - BARRY\_MAX\_NUM\_ELEMENTS, 233
  - BARRY\_SAFE\_EXP, 233
  - Map, 233
  - printf\_barry, 233
- barry-debug.hpp
  - BARRY\_DEBUG\_LEVEL, 234
- barry.hpp
  - BARRY\_HPP, 236
  - BARRY\_VERSION, 236
  - COUNTER\_FUNCTION, 236
  - COUNTER\_LAMBDA, 236
  - RULE\_FUNCTION, 236
  - RULE\_LAMBDA, 237
- barry::counters, 25
- barry::counters::network, 26
- barry::counters::phylo, 26
- BARRY\_BARRAY\_MEAT\_OPERATORS\_HPP
  - barray-meat-operators.hpp, 180
  - barraydense-meat-operators.hpp, 206
- BARRY\_BARRAYDENSE\_BONES\_HPP
  - barraydense-bones.hpp, 205
- BARRY\_BARRAYROW\_MEAT\_HPP
  - barrayrow-meat.hpp, 228
- BARRY\_BARRAYVECTOR\_MEAT\_HPP
  - barrayvector-meat.hpp, 232
- BARRY\_CHECK\_SUPPORT
  - barry-configuration.hpp, 233
- BARRY\_DEBUG\_LEVEL
  - barry-debug.hpp, 234
- BARRY\_HPP
  - barry.hpp, 236
- BARRY\_ISFINITE
  - barry-configuration.hpp, 233
- BARRY\_MAX\_NUM\_ELEMENTS
  - barry-configuration.hpp, 233
- BARRY\_PROGRESS\_BAR\_WIDTH
  - progress.hpp, 287
- BARRY\_SAFE\_EXP
  - barry-configuration.hpp, 233
- BARRY\_SUPPORT\_MEAT\_HPP
  - support-meat.hpp, 300
- BARRY\_VERSION
  - barry.hpp, 236
- BDENSE\_TEMPLATE
  - barraydense-meat-operators.hpp, 206–208
  - barraydense-meat.hpp, 211, 213–217
- BDENSE\_TEMPLATE\_ARGS
  - barraydense-meat-operators.hpp, 206, 208
  - barraydense-meat.hpp, 211
- BDENSE\_TYPE
  - barraydense-meat-operators.hpp, 206, 208
  - barraydense-meat.hpp, 211
- begin
  - BArrayVector< Cell\_Type, Data\_Type >, 73
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 77
  - PhyloCounterData, 145
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 151
- blengths
  - NodeData, 143
- BOTH
  - CHECK, 26
  - EXISTS, 27
- BROW\_TEMPLATE
  - barrayrow-meat.hpp, 228–230
- BROW\_TEMPLATE\_ARGS
  - barrayrow-meat.hpp, 229
- BROW\_TYPE
  - barrayrow-meat.hpp, 229
- calc
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 152
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 169
- calc\_backend
  - support-meat.hpp, 301
- calc\_reduced\_sequence
  - Geese, 111
- calc\_sequence
  - Geese, 111
- Cell
  - Cell< Cell\_Type >, 80, 81
- Cell< Cell\_Type >, 79
  - ~Cell, 80
  - active, 83
  - add, 81, 82
  - Cell, 80, 81
  - operator Cell\_Type, 82
  - operator!=, 82
  - operator=, 82, 83
  - operator==, 83
  - value, 83
  - visited, 83
- cfree
  - support-meat.hpp, 306
- change\_stats
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 174
- CHECK, 26
  - BOTH, 26
  - NONE, 26
  - ONE, 26
  - TWO, 26
- check\_bounds
  - barray-meat.hpp, 196
  - barraydense-meat.hpp, 220
- check\_exists
  - barray-meat.hpp, 196
  - barraydense-meat.hpp, 220
- clear

- BArray< Cell\_Type, Data\_Type >, 33
- BArrayDense< Cell\_Type, Data\_Type >, 52
- Counters< Array\_Type, Data\_Type >, 93
- FreqTable< T >, 106
- Rules< Array\_Type, Data\_Type >, 160
- COL
  - barray-meat-operators.hpp, 181
  - barray-meat.hpp, 187, 191
  - barraydense-meat-operators.hpp, 207
  - barraydense-meat.hpp, 211
- col
  - BArray< Cell\_Type, Data\_Type >, 33
  - BArrayDense< Cell\_Type, Data\_Type >, 52
  - barraydense-meat.hpp, 220
- col0
  - barray-meat.hpp, 197
- Col\_type
  - typedefs.hpp, 311
- colnames
  - Flock, 100
  - Geese, 111
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 126
- conditional\_prob
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 126
- const
  - barray-meat.hpp, 197
  - barraydense-meat.hpp, 220
- ConstBArrayRowIter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, 85
- ConstBArrayRowIter< Cell\_Type, Data\_Type >, 84
  - ~ConstBArrayRowIter, 85
  - Array, 85
  - ConstBArrayRowIter, 85
  - current\_col, 85
  - current\_row, 85
  - iter, 86
- coordinates\_free
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 153
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 174
- coordinates\_locked
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 153
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 174
- copy\_data
  - barray-meat.hpp, 197
  - barraydense-meat.hpp, 220
- count
  - Counter< Array\_Type, Data\_Type >, 88
- count\_all
  - StatsCounter< Array\_Type, Data\_Type >, 164
- count\_current
  - StatsCounter< Array\_Type, Data\_Type >, 164
- count\_fun
  - Counter< Array\_Type, Data\_Type >, 90
  - counters-meat.hpp, 243
- count\_fun\_
  - counters-meat.hpp, 248
- count\_init
  - StatsCounter< Array\_Type, Data\_Type >, 164
- Counter
  - Counter< Array\_Type, Data\_Type >, 87, 88
- counter
  - counters-meat.hpp, 249
- Counter< Array\_Type, Data\_Type >, 86
  - ~Counter, 88
  - count, 88
  - count\_fun, 90
  - Counter, 87, 88
  - data, 90
  - delete\_data, 90
  - desc, 90
  - get\_description, 89
  - get\_name, 89
  - init, 89
  - init\_fun, 90
  - name, 90
  - operator=, 89
- counter\_
  - counters-meat.hpp, 249
- counter\_absdiff
  - Network counters, 13
- counter\_co\_opt
  - Phylo counters, 18
- counter\_cogain
  - Phylo counters, 19
- counter\_css\_census01
  - network-css.hpp, 255
- counter\_css\_census02
  - network-css.hpp, 255
- counter\_css\_census03
  - network-css.hpp, 256
- counter\_css\_census04
  - network-css.hpp, 256
- counter\_css\_census05
  - network-css.hpp, 256
- counter\_css\_census06
  - network-css.hpp, 256
- counter\_css\_census07
  - network-css.hpp, 256
- counter\_css\_census08
  - network-css.hpp, 257
- counter\_css\_census09
  - network-css.hpp, 257
- counter\_css\_census10
  - network-css.hpp, 257
- counter\_css\_completely\_false\_recip\_comiss
  - network-css.hpp, 257
- counter\_css\_completely\_false\_recip\_omiss
  - network-css.hpp, 257

- counter\_css\_mixed\_recip
  - network-css.hpp, 258
- counter\_css\_partially\_false\_recip\_commi
  - network-css.hpp, 258
- counter\_css\_partially\_false\_recip\_omiss
  - network-css.hpp, 258
- counter\_ctriads
  - Network counters, 13
- counter\_degree
  - Network counters, 13
- counter\_deleted
  - statscounter-meat.hpp, 295
- counter\_density
  - Network counters, 14
- counter\_diff
  - Network counters, 14
- counter\_edges
  - Network counters, 14
- Counter\_fun\_type
  - typedefs.hpp, 311
- COUNTER\_FUNCTION
  - barry.hpp, 236
- counter\_gains
  - Phylo counters, 19
- counter\_gains\_k\_offspring
  - Phylo counters, 19
- counter\_genes\_changing
  - Phylo counters, 20
- counter\_idegree
  - Network counters, 14
- counter\_idegree15
  - Network counters, 15
- counter\_isolates
  - Network counters, 15
- counter\_istar2
  - Network counters, 15
- counter\_k\_genes\_changing
  - Phylo counters, 20
- COUNTER\_LAMBDA
  - barry.hpp, 236
- counter\_longest
  - Phylo counters, 20
- counter\_loss
  - Phylo counters, 20
- counter\_maxfuns
  - Phylo counters, 21
- counter\_mutual
  - Network counters, 15
- counter\_neofun
  - Phylo counters, 21
- counter\_neofun\_a2b
  - Phylo counters, 21
- counter\_nodecov
  - Network counters, 15
- counter\_nodeicov
  - Network counters, 16
- counter\_nodematch
  - Network counters, 16
- counter\_nodecov
  - Network counters, 16
- counter\_odegree
  - Network counters, 16
- counter\_odegree15
  - Network counters, 16
- 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, 242–244
- COUNTER\_TEMPLATE\_ARGS
  - counters-meat.hpp, 242
- counter\_ttriads
  - Network counters, 17
- COUNTER\_TYPE
  - counters-meat.hpp, 242
- Counters
  - Counters< Array\_Type, Data\_Type >, 92
- counters
  - statscounter-meat.hpp, 295
  - support-meat.hpp, 306
- Counters< Array\_Type, Data\_Type >, 91
  - ~Counters, 92
  - add\_counter, 93
  - clear, 93
  - Counters, 92
  - get\_descriptions, 93
  - get\_names, 93
  - operator=, 94
  - operator[], 94
  - size, 96
- counters-meat.hpp
  - count\_fun, 243
  - count\_fun\_, 248
  - counter, 249
  - counter\_, 249
  - COUNTER\_TEMPLATE, 242–244
  - COUNTER\_TEMPLATE\_ARGS, 242
  - COUNTER\_TYPE, 242
  - COUNTERS\_TEMPLATE, 242, 244–246
  - COUNTERS\_TEMPLATE\_ARGS, 242
  - COUNTERS\_TYPE, 242
  - data, 246
  - data\_, 249
  - delete\_data, 246
  - delete\_data\_, 249
  - delete\_to\_be\_deleted, 246, 247
  - desc, 247

- desc\_, 250
- i, 250
- init\_fun, 247
- init\_fun\_, 250
- j, 250
- name, 248
- name\_, 250
- noexcept, 251
- push\_back, 248
- return, 251
- to\_be\_deleted, 248
- counters\_
  - statscounter-meat.hpp, 295
  - support-meat.hpp, 306
- COUNTERS\_TEMPLATE
  - counters-meat.hpp, 242, 244–246
- COUNTERS\_TEMPLATE\_ARGS
  - counters-meat.hpp, 242
- COUNTERS\_TYPE
  - counters-meat.hpp, 242
- Counting, 11
- counts
  - PhyloRuleDynData, 148
- Counts\_type
  - typedefs.hpp, 311
- CSS\_APPEND
  - network-css.hpp, 253
- CSS\_CASE\_ELSE
  - network-css.hpp, 253
- CSS\_CASE\_PERCEIVED
  - network-css.hpp, 253
- CSS\_CASE\_TRUTH
  - network-css.hpp, 253
- CSS\_CHECK\_SIZE
  - network-css.hpp, 254
- CSS\_CHECK\_SIZE\_INIT
  - network-css.hpp, 254
- CSS\_NET\_COUNTER\_LAMBDA\_INIT
  - network-css.hpp, 254
- CSS\_PERCEIVED\_CELLS
  - network-css.hpp, 254
- CSS\_SIZE
  - network-css.hpp, 255
- CSS\_TRUE\_CELLS
  - network-css.hpp, 255
- current\_col
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, 85
- current\_row
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, 85
- current\_stats
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 174
- D
  - BArray< Cell\_Type, Data\_Type >, 34
  - BArrayDense< Cell\_Type, Data\_Type >, 53
  - Rule< Array\_Type, Data\_Type >, 158
- dat
  - Flock, 104
- data
  - barray-meat.hpp, 197
  - barraydense-meat.hpp, 221
  - Counter< Array\_Type, Data\_Type >, 90
  - counters-meat.hpp, 246
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 154
- data\_
  - counters-meat.hpp, 249
- DEFAULT\_DUPLICATION
  - phylo.hpp, 267
- 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 >, 175
  - support-meat.hpp, 307
- delete\_data
  - barray-meat.hpp, 197
  - barraydense-meat.hpp, 221
  - Counter< Array\_Type, Data\_Type >, 90
  - counters-meat.hpp, 246
- delete\_data\_
  - barray-meat.hpp, 198
  - barraydense-meat.hpp, 221
  - counters-meat.hpp, 249
- delete\_rengine
  - Geese, 118
- delete\_rules
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 175
  - support-meat.hpp, 307
- delete\_rules\_dyn
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 175
  - support-meat.hpp, 307
- delete\_support
  - Geese, 118
- delete\_to\_be\_deleted
  - counters-meat.hpp, 246, 247
- desc
  - Counter< Array\_Type, Data\_Type >, 90
  - counters-meat.hpp, 247
- desc\_
  - counters-meat.hpp, 250
- directed
  - NetworkData, 136
- DUPL\_DUPL
  - phylo.hpp, 267
- DUPL\_EITH
  - phylo.hpp, 267
- DUPL\_SPEC
  - phylo.hpp, 267
- duplication

- Node, [141](#)
- NodeData, [144](#)
- PhyloRuleDynData, [148](#)
- el
  - barraydense-meat.hpp, [221](#)
- else
  - barray-meat.hpp, [198](#)
  - barraydense-meat.hpp, [221](#)
  - support-meat.hpp, [307](#)
- empty
  - PhyloCounterData, [145](#)
- EmptyArray
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [154](#)
- end
  - BArrayVector< Cell\_Type, Data\_Type >, [73](#)
  - BArrayVector\_const< Cell\_Type, Data\_Type >, [77](#)
  - PhyloCounterData, [145](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [152](#)
  - Progress, [156](#)
- Entries
  - Entries< Cell\_Type >, [97](#)
- Entries< Cell\_Type >, [96](#)
  - ~Entries, [97](#)
  - Entries, [97](#)
  - resize, [97](#)
  - source, [98](#)
  - target, [98](#)
  - val, [98](#)
- eval\_rules\_dyn
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [171](#)
- EXISTS, [27](#)
  - AS\_ONE, [27](#)
  - AS\_ZERO, [27](#)
  - BOTH, [27](#)
  - NONE, [28](#)
  - ONE, [28](#)
  - TWO, [28](#)
  - UNKNOWN, [28](#)
- f\_
  - statscounter-meat.hpp, [295](#)
  - support-meat.hpp, [307](#)
- false
  - barray-meat.hpp, [198](#)
  - barraydense-meat.hpp, [222](#)
- first
  - barray-meat.hpp, [198](#)
- Flock, [98](#)
  - ~Flock, [100](#)
  - add\_data, [100](#)
  - colnames, [100](#)
  - dat, [104](#)
  - Flock, [99](#)
  - get\_counters, [100](#)
  - get\_model, [101](#)
  - get\_support, [101](#)
  - init, [101](#)
  - initialized, [104](#)
  - likelihood\_joint, [101](#)
  - model, [104](#)
  - nfunctions, [104](#)
  - nfuncs, [102](#)
  - nleaves, [102](#)
  - nnodes, [102](#)
  - nterms, [102](#)
  - ntrees, [102](#)
  - operator(), [102](#)
  - parse\_polytomies, [103](#)
  - print, [103](#)
  - rengine, [104](#)
  - set\_seed, [103](#)
  - support\_size, [103](#)
- flush\_data
  - BArray< Cell\_Type, Data\_Type >, [34](#)
- for
  - barray-meat-operators.hpp, [182](#)
  - barray-meat.hpp, [191](#), [192](#)
  - barraydense-meat.hpp, [217](#)
  - statscounter-meat.hpp, [293](#)
  - support-meat.hpp, [302](#)
- FreqTable
  - FreqTable< T >, [105](#)
- FreqTable< T >, [105](#)
  - ~FreqTable, [105](#)
  - add, [106](#)
  - as\_vector, [106](#)
  - clear, [106](#)
  - FreqTable, [105](#)
  - get\_data, [106](#)
  - get\_data\_ptr, [106](#)
  - print, [106](#)
  - reserve, [107](#)
  - size, [107](#)
- Geese, [107](#)
  - ~Geese, [111](#)
  - calc\_reduced\_sequence, [111](#)
  - calc\_sequence, [111](#)
  - colnames, [111](#)
  - delete\_engine, [118](#)
  - delete\_support, [118](#)
  - Geese, [110](#), [111](#)
  - get\_annotated\_nodes, [112](#)
  - get\_counters, [112](#)
  - get\_model, [112](#)
  - get\_probabilities, [112](#)
  - get\_rengine, [112](#)
  - get\_states, [112](#)
  - get\_support, [113](#)
  - inherit\_support, [113](#)
  - init, [113](#)
  - init\_node, [113](#)
  - initialized, [118](#)
  - likelihood, [113](#)
  - likelihood\_exhaust, [114](#)

- map\_to\_nodes, 118
- nannotations, 114
- nfunctions, 118
- nfuncs, 114
- nleafs, 114
- nnodes, 114
- nodes, 119
- nterms, 115
- observed\_counts, 115
- operator=, 115
- parse\_polytomies, 115
- predict, 115
- predict\_backend, 116
- predict\_exhaust, 116
- predict\_exhaust\_backend, 116
- predict\_sim, 116
- print, 116
- print\_observed\_counts, 117
- reduced\_sequence, 119
- sequence, 119
- set\_seed, 117
- simulate, 117
- support\_size, 117
- update\_annotations, 117
- geese-bones.hpp
  - INITIALIZED, 278
  - keygen\_full, 279
  - RULE\_FUNCTION, 279
  - vec\_diff, 279
  - vector\_caster, 279
- gen\_key
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 127
- get\_annotated\_nodes
  - Geese, 112
- get\_cell
  - BArray< Cell\_Type, Data\_Type >, 34
  - BArrayDense< Cell\_Type, Data\_Type >, 53
- get\_col\_vec
  - BArray< Cell\_Type, Data\_Type >, 34, 35
  - BArrayDense< Cell\_Type, Data\_Type >, 53
- get\_counters
  - Flock, 100
  - Geese, 112
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 127
  - PhyloCounterData, 146
  - StatsCounter< Array\_Type, Data\_Type >, 164
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 171
- get\_counts
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 171
- get\_counts\_ptr
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 171
- get\_current\_stats
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 171
- get\_data
  - FreqTable< T >, 106
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 152
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 172
- get\_data\_ptr
  - FreqTable< T >, 106
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 152
- get\_description
  - Counter< Array\_Type, Data\_Type >, 89
- get\_descriptions
  - Counters< Array\_Type, Data\_Type >, 93
  - StatsCounter< Array\_Type, Data\_Type >, 164
- get\_entries
  - BArray< Cell\_Type, Data\_Type >, 35
  - BArrayDense< Cell\_Type, Data\_Type >, 54
- get\_last\_name
  - phylo.hpp, 272
- get\_model
  - Flock, 101
  - Geese, 112
- get\_name
  - Counter< Array\_Type, Data\_Type >, 89
- get\_names
  - Counters< Array\_Type, Data\_Type >, 93
  - StatsCounter< Array\_Type, Data\_Type >, 164
- get\_norm\_const
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 127
- get\_parent
  - Node, 139
- get\_probabilities
  - Geese, 112
- get\_pset
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 127
- get\_pset\_stats
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 127
- get\_rengine
  - Geese, 112
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 128
- get\_row\_vec
  - BArray< Cell\_Type, Data\_Type >, 35
  - BArrayDense< Cell\_Type, Data\_Type >, 54

- get\_rules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 128
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 172
- get\_rules\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 128
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 172
- get\_seq
  - Rules< Array\_Type, Data\_Type >, 160
- get\_states
  - Geese, 112
- get\_support
  - Flock, 101
  - Geese, 113
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 128
- i
  - counters-meat.hpp, 250
- i1
  - barray-meat.hpp, 198
  - barraydense-meat.hpp, 222
- id
  - Node, 141
- if
  - barray-meat.hpp, 192–194
  - barraydense-meat.hpp, 218
  - support-meat.hpp, 302
- IF\_MATCHES
  - phylo.hpp, 268
- IF\_NOTMATCHES
  - phylo.hpp, 268
- include/barry/barray-bones.hpp, 177
- include/barry/barray-iterator.hpp, 178
- include/barry/barray-meat-operators.hpp, 179
- include/barry/barray-meat.hpp, 184
- include/barry/barraycell-bones.hpp, 202
- include/barry/barraycell-meat.hpp, 202
- include/barry/barraydense-bones.hpp, 203
- include/barry/barraydense-meat-operators.hpp, 205
- include/barry/barraydense-meat.hpp, 209
- include/barry/barraydensecell-bones.hpp, 225
- include/barry/barraydensecell-meat.hpp, 225
- include/barry/barrayrow-bones.hpp, 227
- include/barry/barrayrow-meat.hpp, 228
- include/barry/barrayvector-bones.hpp, 230
- include/barry/barrayvector-meat.hpp, 231
- include/barry/barry-configuration.hpp, 232
- include/barry/barry-debug.hpp, 234
- include/barry/barry.hpp, 234
- include/barry/cell-bones.hpp, 237
- include/barry/cell-meat.hpp, 238
- include/barry/col-bones.hpp, 239
- include/barry/counters-bones.hpp, 239
- include/barry/counters-meat.hpp, 240
- include/barry/counters/network-css.hpp, 252
- include/barry/counters/network.hpp, 259
- include/barry/counters/phylo.hpp, 265
- include/barry/model-bones.hpp, 272
- include/barry/model-meat.hpp, 274
- include/barry/models/geese.hpp, 276
- include/barry/models/geese/flock-bones.hpp, 277
- include/barry/models/geese/flock-meat.hpp, 277
- include/barry/models/geese/geese-bones.hpp, 278
- include/barry/models/geese/geese-meat-constructors.hpp, 280
- include/barry/models/geese/geese-meat-likelihood.hpp, 280
- include/barry/models/geese/geese-meat-likelihood\_exhaust.hpp, 281
- include/barry/models/geese/geese-meat-predict.hpp, 282
- include/barry/models/geese/geese-meat-predict\_exhaust.hpp, 282
- include/barry/models/geese/geese-meat-predict\_sim.hpp, 283
- include/barry/models/geese/geese-meat-simulate.hpp, 283
- include/barry/models/geese/geese-meat.hpp, 284
- include/barry/models/geese/geese-node-bones.hpp, 284
- include/barry/powerset-bones.hpp, 285
- include/barry/powerset-meat.hpp, 286
- include/barry/progress.hpp, 287
- include/barry/rules-bones.hpp, 287
- include/barry/rules-meat.hpp, 289
- include/barry/statscounter-bones.hpp, 289
- include/barry/statscounter-meat.hpp, 291
- include/barry/statsdb.hpp, 296
- include/barry/support-bones.hpp, 297
- include/barry/support-meat.hpp, 299
- include/barry/typedefs.hpp, 309
- indices
  - NetCounterData, 134
- inherit\_support
  - Geese, 113
- init
  - Counter< Array\_Type, Data\_Type >, 89
  - Flock, 101
  - Geese, 113
- init\_fun
  - Counter< Array\_Type, Data\_Type >, 90
  - counters-meat.hpp, 247
- init\_fun\_
  - counters-meat.hpp, 250
- init\_node
  - Geese, 113
- init\_support
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 152



- Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
172
- INITIALIZED
  - geese-bones.hpp, 278
- initialized
  - Flock, 104
  - Geese, 118
- insert\_cell
  - BArray< Cell\_Type, Data\_Type >, 35, 36
  - BArrayDense< Cell\_Type, Data\_Type >, 54, 55
  - support-meat.hpp, 302
- is\_col
  - BArrayVector< Cell\_Type, Data\_Type >, 73
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 77
- IS\_DUPLICATION
  - phylo.hpp, 268
- IS\_EITHER
  - phylo.hpp, 268
- is\_empty
  - BArray< Cell\_Type, Data\_Type >, 36
  - BArrayDense< Cell\_Type, Data\_Type >, 55
- is\_leaf
  - Node, 140
- is\_row
  - BArrayVector< Cell\_Type, Data\_Type >, 73
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 77
- IS\_SPECIATION
  - phylo.hpp, 268
- iter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, 86
- j
  - barray-meat.hpp, 199
  - barraydense-meat.hpp, 222
  - counters-meat.hpp, 250
  - statscounter-meat.hpp, 295
- j0
  - barray-meat.hpp, 199
  - barraydense-meat.hpp, 222
- j1
  - barray-meat.hpp, 199
  - barraydense-meat.hpp, 222
- keygen\_default
  - model-bones.hpp, 273
- keygen\_full
  - geese-bones.hpp, 279
- lb
  - PhyloRuleDynData, 148
- likelihood
  - Geese, 113
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
128, 129
- likelihood\_
  - model-meat.hpp, 275
- likelihood\_exhaust
  - Geese, 114
- likelihood\_joint
  - Flock, 101
- likelihood\_total
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
129
- M
  - barray-meat.hpp, 195, 199
  - barraydense-meat.hpp, 219, 222
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 154
  - Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
175
- M\_
  - barray-meat.hpp, 199
  - barraydense-meat.hpp, 223
- MAKE\_DUPL\_VARS
  - phylo.hpp, 269
- Map
  - barry-configuration.hpp, 233
- map\_to\_nodes
  - Geese, 118
- MapVec\_type
  - typedefs.hpp, 311
- max\_num\_elements
  - Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
175
- Model
  - Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
123
- model
  - Flock, 104
- Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type,  
Data\_Rule\_Dyn\_Type >, 119
  - ~Model, 123
  - add\_array, 124
  - add\_counter, 124
  - add\_rule, 125
  - add\_rule\_dyn, 125, 126
  - colnames, 126
  - conditional\_prob, 126
  - gen\_key, 127
  - get\_counters, 127
  - get\_norm\_const, 127
  - get\_pset, 127
  - get\_pset\_stats, 127
  - get\_engine, 128
  - get\_rules, 128
  - get\_rules\_dyn, 128
  - get\_support, 128
  - likelihood, 128, 129
  - likelihood\_total, 129
  - Model, 123
  - nterms, 129
  - operator=, 130



- print, [130](#)
- print\_stats, [130](#)
- sample, [130](#)
- set\_counters, [131](#)
- set\_keygen, [131](#)
- set\_rengine, [131](#)
- set\_rules, [131](#)
- set\_rules\_dyn, [131](#)
- set\_seed, [132](#)
- size, [132](#)
- size\_unique, [132](#)
- store\_psets, [132](#)
- support\_size, [132](#)
- model-bones.hpp
  - keygen\_default, [273](#)
- model-meat.hpp
  - likelihood\_, [275](#)
  - MODEL\_TEMPLATE, [274](#), [275](#)
  - MODEL\_TEMPLATE\_ARGS, [274](#)
  - MODEL\_TYPE, [275](#)
  - update\_normalizing\_constant, [276](#)
- MODEL\_TEMPLATE
  - model-meat.hpp, [274](#), [275](#)
- MODEL\_TEMPLATE\_ARGS
  - model-meat.hpp, [274](#)
- MODEL\_TYPE
  - model-meat.hpp, [275](#)
- N
  - barray-meat.hpp, [200](#)
  - barraydense-meat.hpp, [223](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [154](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [176](#)
- name
  - Counter< Array\_Type, Data\_Type >, [90](#)
  - counters-meat.hpp, [248](#)
- name\_
  - counters-meat.hpp, [250](#)
- nannotations
  - Geese, [114](#)
- narray
  - Node, [141](#)
- NCells
  - barray-meat.hpp, [200](#)
  - barraydense-meat.hpp, [223](#)
- ncol
  - BArray< Cell\_Type, Data\_Type >, [36](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [55](#)
- NET\_C\_DATA\_IDX
  - network.hpp, [261](#)
- NET\_C\_DATA\_NUM
  - network.hpp, [262](#)
- NetCounter
  - network.hpp, [263](#)
- NetCounterData, [133](#)
  - ~NetCounterData, [134](#)
  - indices, [134](#)
  - NetCounterData, [133](#)
  - numbers, [134](#)
- NetCounters
  - network.hpp, [263](#)
- NetModel
  - network.hpp, [263](#)
- NetRule
  - network.hpp, [264](#)
- NetRules
  - network.hpp, [264](#)
- NetStatsCounter
  - network.hpp, [264](#)
- NetSupport
  - network.hpp, [264](#)
- Network
  - network.hpp, [264](#)
- Network counters, [12](#)
  - counter\_absdiff, [13](#)
  - counter\_ctriads, [13](#)
  - counter\_degree, [13](#)
  - counter\_density, [14](#)
  - counter\_diff, [14](#)
  - counter\_edges, [14](#)
  - counter\_iddegree, [14](#)
  - counter\_iddegree15, [15](#)
  - counter\_isolates, [15](#)
  - counter\_istar2, [15](#)
  - counter\_mutual, [15](#)
  - counter\_nodecov, [15](#)
  - counter\_nodeicov, [16](#)
  - counter\_nodematch, [16](#)
  - counter\_nodeocov, [16](#)
  - counter\_odegree, [16](#)
  - counter\_odegree15, [16](#)
  - counter\_ostar2, [17](#)
  - counter\_ttriads, [17](#)
  - NETWORK\_COUNTER, [17](#)
- network-css.hpp
  - counter\_css\_census01, [255](#)
  - counter\_css\_census02, [255](#)
  - counter\_css\_census03, [256](#)
  - counter\_css\_census04, [256](#)
  - counter\_css\_census05, [256](#)
  - counter\_css\_census06, [256](#)
  - counter\_css\_census07, [256](#)
  - counter\_css\_census08, [257](#)
  - counter\_css\_census09, [257](#)
  - counter\_css\_census10, [257](#)
  - counter\_css\_completely\_false\_recip\_comiss, [257](#)
  - counter\_css\_completely\_false\_recip\_omiss, [257](#)
  - counter\_css\_mixed\_recip, [258](#)
  - counter\_css\_partially\_false\_recip\_commi, [258](#)
  - counter\_css\_partially\_false\_recip\_omiss, [258](#)
  - CSS\_APPEND, [253](#)
  - CSS\_CASE\_ELSE, [253](#)
  - CSS\_CASE\_PERCEIVED, [253](#)
  - CSS\_CASE\_TRUTH, [253](#)
  - CSS\_CHECK\_SIZE, [254](#)

- CSS\_CHECK\_SIZE\_INIT, 254
- CSS\_NET\_COUNTER\_LAMBDA\_INIT, 254
- CSS\_PERCEIVED\_CELLS, 254
- CSS\_SIZE, 255
- CSS\_TRUE\_CELLS, 255
- network.hpp
  - NET\_C\_DATA\_IDX, 261
  - NET\_C\_DATA\_NUM, 262
  - NetCounter, 263
  - NetCounters, 263
  - NetModel, 263
  - NetRule, 264
  - NetRules, 264
  - NetStatsCounter, 264
  - NetSupport, 264
  - Network, 264
  - NETWORK\_COUNTER, 262
  - NETWORK\_COUNTER\_LAMBDA, 262
  - NETWORK\_RULE, 262
  - NETWORK\_RULE\_LAMBDA, 263
  - NetworkDense, 264
  - rules\_zerodiag, 265
- NETWORK\_COUNTER
  - Network counters, 17
  - network.hpp, 262
- NETWORK\_COUNTER\_LAMBDA
  - network.hpp, 262
- NETWORK\_RULE
  - network.hpp, 262
- NETWORK\_RULE\_LAMBDA
  - network.hpp, 263
- NetworkData, 134
  - ~NetworkData, 136
  - directed, 136
  - NetworkData, 135, 136
  - vertex\_attr, 136
- NetworkDense
  - network.hpp, 264
- next
  - Progress, 156
- nfunctions
  - Flock, 104
  - Geese, 118
- nfuncs
  - Flock, 102
  - Geese, 114
- nleafs
  - Flock, 102
  - Geese, 114
- nnodes
  - Flock, 102
  - Geese, 114
- nnozero
  - BArray< Cell\_Type, Data\_Type >, 36
  - BArrayDense< Cell\_Type, Data\_Type >, 55
- Node, 137
  - ~Node, 139
  - annotations, 140
  - array, 140
  - arrays, 140
  - duplication, 141
  - get\_parent, 139
  - id, 141
  - is\_leaf, 140
  - narray, 141
  - Node, 138, 139
  - noffspring, 140
  - offspring, 141
  - ord, 141
  - parent, 142
  - probability, 142
  - subtree\_prob, 142
  - visited, 142
- NodeData, 143
  - blengths, 143
  - duplication, 144
  - NodeData, 143
  - states, 144
- nodes
  - Geese, 119
- noexcept
  - counters-meat.hpp, 251
- noffspring
  - Node, 140
- NONE
  - CHECK, 26
  - EXISTS, 28
- nrow
  - BArray< Cell\_Type, Data\_Type >, 37
  - BArrayDense< Cell\_Type, Data\_Type >, 55
- nterms
  - Flock, 102
  - Geese, 115
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 129
- ntrees
  - Flock, 102
- numbers
  - NetCounterData, 134
- observed\_counts
  - Geese, 115
- offspring
  - Node, 141
- ONE
  - CHECK, 26
  - EXISTS, 28
- operator BArrayRow< Cell\_Type, Data\_Type >
  - BArrayRow< Cell\_Type, Data\_Type >, 68
- operator BArrayRow\_const< Cell\_Type, Data\_Type >
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 70
- operator Cell\_Type
  - BArrayCell< Cell\_Type, Data\_Type >, 44
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 46
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 63

- BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 65
- Cell< Cell\_Type >, 82
- operator std::vector< Cell\_Type >
  - BArrayVector< Cell\_Type, Data\_Type >, 74
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 77
- operator!=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 47
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 66
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 70
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 77
  - Cell< Cell\_Type >, 82
- operator<
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 47
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 66
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 70
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 78
- operator<=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 47
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 66
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 71
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 78
- operator>
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 47
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 66
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 71
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 78
- operator>=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 48
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 67
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 71
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 78
- operator\*=
  - BArray< Cell\_Type, Data\_Type >, 37
  - BArrayCell< Cell\_Type, Data\_Type >, 44
  - BArrayDense< Cell\_Type, Data\_Type >, 56
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 63
  - BArrayRow< Cell\_Type, Data\_Type >, 68
  - BArrayVector< Cell\_Type, Data\_Type >, 74
- operator()
  - BArray< Cell\_Type, Data\_Type >, 37
  - barray-meat-operators.hpp, 183
  - BArrayDense< Cell\_Type, Data\_Type >, 56
  - Flock, 102
  - PhyloCounterData, 146
  - Rule< Array\_Type, Data\_Type >, 158
  - Rules< Array\_Type, Data\_Type >, 161
  - vecHasher< T >, 176
- operator+=
  - BArray< Cell\_Type, Data\_Type >, 37, 38
  - BArrayCell< Cell\_Type, Data\_Type >, 44
  - BArrayDense< Cell\_Type, Data\_Type >, 56
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 63
- BArrayRow< Cell\_Type, Data\_Type >, 68
- BArrayVector< Cell\_Type, Data\_Type >, 74
- operator=
  - BArray< Cell\_Type, Data\_Type >, 38
  - BArrayCell< Cell\_Type, Data\_Type >, 44
  - BArrayDense< Cell\_Type, Data\_Type >, 57
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 63
  - BArrayRow< Cell\_Type, Data\_Type >, 68
  - BArrayVector< Cell\_Type, Data\_Type >, 74
- operator/=
  - BArray< Cell\_Type, Data\_Type >, 38
  - BArrayCell< Cell\_Type, Data\_Type >, 45
  - BArrayDense< Cell\_Type, Data\_Type >, 57
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 64
  - BArrayRow< Cell\_Type, Data\_Type >, 69
  - BArrayVector< Cell\_Type, Data\_Type >, 74
- operator=
  - BArray< Cell\_Type, Data\_Type >, 38, 39
  - BArrayCell< Cell\_Type, Data\_Type >, 45
  - BArrayDense< Cell\_Type, Data\_Type >, 57
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 64
  - BArrayRow< Cell\_Type, Data\_Type >, 69
  - BArrayVector< Cell\_Type, Data\_Type >, 75
  - Cell< Cell\_Type >, 82, 83
  - Counter< Array\_Type, Data\_Type >, 89
  - Counters< Array\_Type, Data\_Type >, 94
  - Geese, 115
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 130
  - Rules< Array\_Type, Data\_Type >, 161
- operator==
  - BArray< Cell\_Type, Data\_Type >, 39
  - BArrayCell< Cell\_Type, Data\_Type >, 45
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 47
  - BArrayDense< Cell\_Type, Data\_Type >, 58
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 64
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 66
  - BArrayRow< Cell\_Type, Data\_Type >, 69
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 71
  - BArrayVector< Cell\_Type, Data\_Type >, 75
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 78
  - Cell< Cell\_Type >, 83
- operator[]
  - Counters< Array\_Type, Data\_Type >, 94
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 153
- ord
  - Node, 141
- out\_of\_range
  - BArray< Cell\_Type, Data\_Type >, 39
  - BArrayDense< Cell\_Type, Data\_Type >, 58
- parent
  - Node, 142
- parse\_polytomies
  - Flock, 103
  - Geese, 115
- Phylo counters, 17

- counter\_co\_opt, 18
- counter\_cogain, 19
- counter\_gains, 19
- counter\_gains\_k\_offspring, 19
- counter\_genes\_changing, 20
- counter\_k\_genes\_changing, 20
- counter\_longest, 20
- counter\_loss, 20
- counter\_maxfun, 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
  - DEFAULT\_DUPLICATION, 267
  - DUPL\_DUPL, 267
  - DUPL\_EITH, 267
  - DUPL\_SPEC, 267
  - get\_last\_name, 272
  - IF\_MATCHES, 268
  - IF\_NOTMATCHES, 268
  - IS\_DUPLICATION, 268
  - IS\_EITHER, 268
  - IS\_SPECIATION, 268
  - MAKE\_DUPL\_VARS, 269
  - PHYLO\_CHECK\_MISSING, 269
  - PHYLO\_COUNTER\_LAMBDA, 269
  - PHYLO\_RULE\_DYN\_LAMBDA, 269
  - PhyloArray, 270
  - PhyloCounter, 270
  - PhyloCounters, 270
  - PhyloModel, 270
  - PhyloPowerSet, 270
  - PhyloRule, 271
  - PhyloRuleData, 271
  - PhyloRuleDyn, 271
  - PhyloRules, 271
  - PhyloRulesDyn, 271
  - PhyloStatsCounter, 271
  - PhyloSupport, 272
- PHYLO\_CHECK\_MISSING
  - phylo.hpp, 269
- PHYLO\_COUNTER\_LAMBDA
  - phylo.hpp, 269
- PHYLO\_RULE\_DYN\_LAMBDA
  - phylo.hpp, 269
- PhyloArray
  - phylo.hpp, 270
- PhyloCounter
  - phylo.hpp, 270
- PhyloCounterData, 144
  - at, 145
  - begin, 145
  - empty, 145
  - end, 145
  - get\_counters, 146
  - operator(), 146
  - PhyloCounterData, 145
  - push\_back, 146
  - reserve, 146
  - shrink\_to\_fit, 146
  - size, 146
- PhyloCounters
  - phylo.hpp, 270
- PhyloModel
  - phylo.hpp, 270
- PhyloPowerSet
  - phylo.hpp, 270
- PhyloRule
  - phylo.hpp, 271
- PhyloRuleData
  - phylo.hpp, 271
- PhyloRuleDyn
  - phylo.hpp, 271
- PhyloRuleDynData, 147
  - ~PhyloRuleDynData, 147
  - counts, 148
  - duplication, 148
  - lb, 148
  - PhyloRuleDynData, 147
  - pos, 148
  - ub, 148
- PhyloRules
  - phylo.hpp, 271
- PhyloRulesDyn
  - phylo.hpp, 271
- PhyloStatsCounter
  - phylo.hpp, 271
- PhyloSupport
  - phylo.hpp, 272
- POS
  - barraydense-meat-operators.hpp, 207
  - barraydense-meat.hpp, 212
  - barraydensecell-meat.hpp, 226
- pos
  - PhyloRuleDynData, 148
- POS\_N
  - barraydense-meat-operators.hpp, 207
  - barraydense-meat.hpp, 212
- PowerSet
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 150
- PowerSet< Array\_Type, Data\_Rule\_Type >, 149
  - ~PowerSet, 151
  - add\_rule, 151
  - begin, 151
  - calc, 152
  - coordinates\_free, 153
  - coordinates\_locked, 153
  - data, 154
  - EmptyArray, 154
  - end, 152

- get\_data, 152
- get\_data\_ptr, 152
- init\_support, 152
- M, 154
- N, 154
- operator[], 153
- PowerSet, 150
- reset, 153
- rules, 154
- rules\_deleted, 155
- size, 153
- predict
  - Geese, 115
- predict\_backend
  - Geese, 116
- predict\_exhaust
  - Geese, 116
- predict\_exhaust\_backend
  - Geese, 116
- predict\_sim
  - Geese, 116
- print
  - BArray< Cell\_Type, Data\_Type >, 39
  - BArrayDense< Cell\_Type, Data\_Type >, 58
  - Flock, 103
  - FreqTable< T >, 106
  - Geese, 116
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 130
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 172
- print\_observed\_counts
  - Geese, 117
- print\_stats
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 130
- printf\_barry
  - barry-configuration.hpp, 233
- probability
  - Node, 142
- Progress, 155
  - ~Progress, 156
  - end, 156
  - next, 156
  - Progress, 155
- progress.hpp
  - BARRY\_PROGRESS\_BAR\_WIDTH, 287
- push\_back
  - counters-meat.hpp, 248
  - PhyloCounterData, 146
- README.md, 313
- reduced\_sequence
  - Geese, 119
- rengine
  - Flock, 104
- report
  - barray-meat.hpp, 200
  - barraydense-meat.hpp, 223
- reserve
  - BArray< Cell\_Type, Data\_Type >, 39
  - BArrayDense< Cell\_Type, Data\_Type >, 58
  - FreqTable< T >, 107
  - PhyloCounterData, 146
- reset
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 153
- reset\_array
  - StatsCounter< Array\_Type, Data\_Type >, 165
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 173
- resize
  - BArray< Cell\_Type, Data\_Type >, 39
  - barray-meat.hpp, 195
  - BArrayDense< Cell\_Type, Data\_Type >, 58
  - barraydense-meat.hpp, 219
  - Entries< Cell\_Type >, 97
  - statscounter-meat.hpp, 293
- return
  - barray-meat.hpp, 195, 200
  - barraydense-meat.hpp, 223
  - counters-meat.hpp, 251
  - statscounter-meat.hpp, 296
  - support-meat.hpp, 308
- rhs
  - barray-meat-operators.hpp, 183
- rm\_cell
  - BArray< Cell\_Type, Data\_Type >, 40
  - BArrayDense< Cell\_Type, Data\_Type >, 58
  - support-meat.hpp, 302
- ROW
  - barray-meat-operators.hpp, 181
  - barray-meat.hpp, 187, 195
  - barraydense-meat-operators.hpp, 207
  - barraydense-meat.hpp, 212
- row
  - BArray< Cell\_Type, Data\_Type >, 40
  - BArrayDense< Cell\_Type, Data\_Type >, 59
- row0
  - barray-meat.hpp, 200
- Row\_type
  - typedefs.hpp, 311
- Rule
  - Rule< Array\_Type, Data\_Type >, 157
- Rule< Array\_Type, Data\_Type >, 156
  - ~Rule, 157
  - D, 158
  - operator(), 158
  - Rule, 157
- rule\_dyn\_limit\_changes
  - Phylo rules, 23
- rule\_fun\_default
  - rules-bones.hpp, 288
- Rule\_fun\_type

- typedefs.hpp, 312
- RULE\_FUNCTION
  - barry.hpp, 236
  - geese-bones.hpp, 279
- RULE\_LAMBDA
  - barry.hpp, 237
- Rules
  - Rules< Array\_Type, Data\_Type >, 159
- rules
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 154
  - support-meat.hpp, 308
- Rules< Array\_Type, Data\_Type >, 158
  - ~Rules, 159
  - add\_rule, 160
  - clear, 160
  - get\_seq, 160
  - operator(), 161
  - operator=, 161
  - Rules, 159
  - size, 161
- rules-bones.hpp
  - rule\_fun\_default, 288
- rules\_
  - support-meat.hpp, 308
- rules\_deleted
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 155
- rules\_dyn
  - support-meat.hpp, 308
- rules\_zerodiag
  - network.hpp, 265
- sample
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 130
- search
  - barray-meat.hpp, 201
- sequence
  - Geese, 119
- set\_counters
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 131
  - StatsCounter< Array\_Type, Data\_Type >, 165
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 173
- set\_data
  - BArray< Cell\_Type, Data\_Type >, 40
  - BArrayDense< Cell\_Type, Data\_Type >, 59
- set\_keygen
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 131
- set\_engine
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 131
- set\_rules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 131
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 173
- set\_rules\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 131
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 173
- set\_seed
  - Flock, 103
  - Geese, 117
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 132
- shrink\_to\_fit
  - PhyloCounterData, 146
- simulate
  - Geese, 117
- size
  - BArrayVector< Cell\_Type, Data\_Type >, 75
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 79
  - Counters< Array\_Type, Data\_Type >, 96
  - FreqTable< T >, 107
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 132
  - PhyloCounterData, 146
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 153
  - Rules< Array\_Type, Data\_Type >, 161
- size\_unique
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 132
- source
  - barray-meat.hpp, 201
  - barraydense-meat.hpp, 224
  - Entries< Cell\_Type >, 98
- states
  - NodeData, 144
- Statistical Models, 11
- stats\_bank
  - support-meat.hpp, 308
- StatsCounter
  - StatsCounter< Array\_Type, Data\_Type >, 163
- StatsCounter< Array\_Type, Data\_Type >, 162
  - ~StatsCounter, 163
  - add\_counter, 163
  - count\_all, 164
  - count\_current, 164
  - count\_init, 164
  - get\_counters, 164
  - get\_descriptions, 164
  - get\_names, 164

- reset\_array, [165](#)
- set\_counters, [165](#)
- StatsCounter, [163](#)
- statscounter-meat.hpp
  - counter\_deleted, [295](#)
  - counters, [295](#)
  - counters\_, [295](#)
  - f\_, [295](#)
  - for, [293](#)
  - j, [295](#)
  - resize, [293](#)
  - return, [296](#)
  - STATSCOUNTER\_TEMPLATE, [292–294](#)
  - STATSCOUNTER\_TEMPLATE\_ARGS, [292](#)
  - STATSCOUNTER\_TYPE, [292](#)
- STATSCOUNTER\_TEMPLATE
  - statscounter-meat.hpp, [292–294](#)
- STATSCOUNTER\_TEMPLATE\_ARGS
  - statscounter-meat.hpp, [292](#)
- STATSCOUNTER\_TYPE
  - statscounter-meat.hpp, [292](#)
- store\_psets
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [132](#)
- subtree\_prob
  - Node, [142](#)
- Support
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [167, 168](#)
- Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [165](#)
  - ~Support, [168](#)
  - add\_counter, [168](#)
  - add\_rule, [169](#)
  - add\_rule\_dyn, [169](#)
  - calc, [169](#)
  - change\_stats, [174](#)
  - coordinates\_free, [174](#)
  - coordinates\_locked, [174](#)
  - current\_stats, [174](#)
  - delete\_counters, [175](#)
  - delete\_rules, [175](#)
  - delete\_rules\_dyn, [175](#)
  - eval\_rules\_dyn, [171](#)
  - get\_counters, [171](#)
  - get\_counts, [171](#)
  - get\_counts\_ptr, [171](#)
  - get\_current\_stats, [171](#)
  - get\_data, [172](#)
  - get\_rules, [172](#)
  - get\_rules\_dyn, [172](#)
  - init\_support, [172](#)
  - M, [175](#)
  - max\_num\_elements, [175](#)
  - N, [176](#)
  - print, [172](#)
  - reset\_array, [173](#)
  - set\_counters, [173](#)
  - set\_rules, [173](#)
  - set\_rules\_dyn, [173](#)
  - Support, [167, 168](#)
- support-meat.hpp
  - array\_bank, [306](#)
  - BARRY\_SUPPORT\_MEAT\_HPP, [300](#)
  - calc\_backend, [301](#)
  - cfree, [306](#)
  - counters, [306](#)
  - counters\_, [306](#)
  - delete\_counters, [307](#)
  - delete\_rules, [307](#)
  - delete\_rules\_dyn, [307](#)
  - else, [307](#)
  - f\_, [307](#)
  - for, [302](#)
  - if, [302](#)
  - insert\_cell, [302](#)
  - return, [308](#)
  - rm\_cell, [302](#)
  - rules, [308](#)
  - rules\_, [308](#)
  - rules\_dyn, [308](#)
  - stats\_bank, [308](#)
  - SUPPORT\_TEMPLATE, [301, 303–306](#)
  - SUPPORT\_TEMPLATE\_ARGS, [301](#)
  - SUPPORT\_TYPE, [301](#)
- support\_size
  - Flock, [103](#)
  - Geese, [117](#)
- Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [132](#)
  - SUPPORT\_TEMPLATE
    - support-meat.hpp, [301, 303–306](#)
  - SUPPORT\_TEMPLATE\_ARGS
    - support-meat.hpp, [301](#)
  - SUPPORT\_TYPE
    - support-meat.hpp, [301](#)
- swap\_cells
  - BArray< Cell\_Type, Data\_Type >, [40](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [59](#)
- swap\_cols
  - BArray< Cell\_Type, Data\_Type >, [41](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [59](#)
- swap\_rows
  - BArray< Cell\_Type, Data\_Type >, [41](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [60](#)
- target
  - barray-meat.hpp, [201](#)
  - barraydense-meat.hpp, [224](#)
  - Entries< Cell\_Type >, [98](#)
- this
  - barray-meat-operators.hpp, [183](#)
- to\_be\_deleted
  - counters-meat.hpp, [248](#)

toggle\_cell  
     BArray< Cell\_Type, Data\_Type >, [41](#)  
     BArrayDense< Cell\_Type, Data\_Type >, [60](#)  
 toggle\_lock  
     BArray< Cell\_Type, Data\_Type >, [41](#)  
     BArrayDense< Cell\_Type, Data\_Type >, [60](#)  
 transpose  
     BArray< Cell\_Type, Data\_Type >, [41](#)  
     BArrayDense< Cell\_Type, Data\_Type >, [60](#)  
 TWO  
     CHECK, [26](#)  
     EXISTS, [28](#)  
 typedefs.hpp  
     Col\_type, [311](#)  
     Counter\_fun\_type, [311](#)  
     Counts\_type, [311](#)  
     MapVec\_type, [311](#)  
     Row\_type, [311](#)  
     Rule\_fun\_type, [312](#)  
     uint, [312](#)  
     vec\_equal, [312](#)  
     vec\_equal\_approx, [312](#)  
     vec\_inner\_prod, [313](#)  
  
 ub  
     PhyloRuleDynData, [148](#)  
 uint  
     typedefs.hpp, [312](#)  
 UNKNOWN  
     EXISTS, [28](#)  
 update\_annotations  
     Geese, [117](#)  
 update\_normalizing\_constant  
     model-meat.hpp, [276](#)  
  
 v  
     barray-meat.hpp, [201](#)  
     barraydense-meat.hpp, [224](#)  
 val  
     Entries< Cell\_Type >, [98](#)  
 value  
     barray-meat.hpp, [201](#)  
     barraydense-meat.hpp, [224](#)  
     Cell< Cell\_Type >, [83](#)  
 vec\_diff  
     geese-bones.hpp, [279](#)  
 vec\_equal  
     typedefs.hpp, [312](#)  
 vec\_equal\_approx  
     typedefs.hpp, [312](#)  
 vec\_inner\_prod  
     typedefs.hpp, [313](#)  
 vecHasher< T >, [176](#)  
     operator(), [176](#)  
 vector\_caster  
     geese-bones.hpp, [279](#)  
 vertex\_attr  
     NetworkData, [136](#)  
 visited  
  
     BArray< Cell\_Type, Data\_Type >, [42](#)  
     BArrayDense< Cell\_Type, Data\_Type >, [61](#)  
     Cell< Cell\_Type >, [83](#)  
     Node, [142](#)  
  
 ZERO\_CELL  
     barraydense-meat.hpp, [212](#)  
 zero\_col  
     BArray< Cell\_Type, Data\_Type >, [42](#)  
     BArrayDense< Cell\_Type, Data\_Type >, [60](#)  
 zero\_row  
     BArray< Cell\_Type, Data\_Type >, [42](#)  
     BArrayDense< Cell\_Type, Data\_Type >, [61](#)