

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() [1/2]	14
5.3.2.3 counter_ctriads() [2/2]	14
5.3.2.4 counter_degree()	14
5.3.2.5 counter_density()	14
5.3.2.6 counter_diff()	14
5.3.2.7 counter_edges()	15
5.3.2.8 counter_idegree() [1/2]	15
5.3.2.9 counter_idegree() [2/2]	15
5.3.2.10 counter_idegree15() [1/2]	15
5.3.2.11 counter_idegree15() [2/2]	15
5.3.2.12 counter_isolates() [1/2]	16
5.3.2.13 counter_isolates() [2/2]	16
5.3.2.14 counter_istar2() [1/2]	16
5.3.2.15 counter_istar2() [2/2]	16
5.3.2.16 counter_mutual()	16
5.3.2.17 counter_nodecov()	17
5.3.2.18 counter_nodeicov()	17
5.3.2.19 counter_nodematch()	17
5.3.2.20 counter_nodeocov()	17
5.3.2.21 counter_odegree() [1/2]	17
5.3.2.22 counter_odegree() [2/2]	18
5.3.2.23 counter_odegree15() [1/2]	18
5.3.2.24 counter_odegree15() [2/2]	18
5.3.2.25 counter_ostar2() [1/2]	18

5.3.2.26 counter_ostar2() [2/2]	18
5.3.2.27 counter_ttriads() [1/2]	19
5.3.2.28 counter_ttriads() [2/2]	19
5.3.2.29 NETWORK_COUNTER()	19
5.4 Phylo counters	19
5.4.1 Detailed Description	20
5.4.2 Function Documentation	20
5.4.2.1 counter_co_opt()	20
5.4.2.2 counter_cogain()	21
5.4.2.3 counter_gains()	21
5.4.2.4 counter_gains_from_0()	21
5.4.2.5 counter_gains_k_offspring()	22
5.4.2.6 counter_genes_changing()	22
5.4.2.7 counter_k_genes_changing()	22
5.4.2.8 counter_less_than_p_prop_genes_changing()	22
5.4.2.9 counter_longest()	23
5.4.2.10 counter_loss()	23
5.4.2.11 counter_maxfuns()	23
5.4.2.12 counter_neofun()	23
5.4.2.13 counter_neofun_a2b()	24
5.4.2.14 counter_overall_changes()	24
5.4.2.15 counter_overall_gains()	24
5.4.2.16 counter_overall_loss()	24
5.4.2.17 counter_pairwise_preserving()	25
5.4.2.18 counter_prop_genes_changing()	25
5.4.2.19 counter_subfun()	25
5.5 Phylo rules	25
5.5.1 Detailed Description	25
5.5.2 Function Documentation	26
5.5.2.1 rule_dyn_limit_changes()	26
<b>6 Namespace Documentation</b>	<b>27</b>
6.1 barry Namespace Reference	27
6.1.1 Detailed Description	27
6.2 barry::counters Namespace Reference	27
6.2.1 Detailed Description	27
6.3 barry::counters::network Namespace Reference	28
6.4 barry::counters::phylo Namespace Reference	28
6.5 CHECK Namespace Reference	28
6.5.1 Detailed Description	28
6.5.2 Variable Documentation	28
6.5.2.1 BOTH	28

6.5.2.2 NONE . . . . .	28
6.5.2.3 ONE . . . . .	28
6.5.2.4 TWO . . . . .	29
6.6 EXISTS Namespace Reference . . . . .	29
6.6.1 Detailed Description . . . . .	29
6.6.2 Variable Documentation . . . . .	29
6.6.2.1 AS_ONE . . . . .	29
6.6.2.2 AS_ZERO . . . . .	29
6.6.2.3 BOTH . . . . .	30
6.6.2.4 NONE . . . . .	30
6.6.2.5 ONE . . . . .	30
6.6.2.6 TWO . . . . .	30
6.6.2.7 UNKNOWN . . . . .	30
<b>7 Class Documentation</b>	<b>31</b>
7.1 BArray< Cell_Type, Data_Type > Class Template Reference . . . . .	31
7.1.1 Detailed Description . . . . .	33
7.1.2 Constructor & Destructor Documentation . . . . .	34
7.1.2.1 BArray() [1/6] . . . . .	34
7.1.2.2 BArray() [2/6] . . . . .	34
7.1.2.3 BArray() [3/6] . . . . .	34
7.1.2.4 BArray() [4/6] . . . . .	35
7.1.2.5 BArray() [5/6] . . . . .	35
7.1.2.6 BArray() [6/6] . . . . .	35
7.1.2.7 ~BArray() . . . . .	35
7.1.3 Member Function Documentation . . . . .	35
7.1.3.1 clear() . . . . .	35
7.1.3.2 col() . . . . .	36
7.1.3.3 D() [1/2] . . . . .	36
7.1.3.4 D() [2/2] . . . . .	36
7.1.3.5 default_val() . . . . .	36
7.1.3.6 flush_data() . . . . .	36
7.1.3.7 get_cell() . . . . .	36
7.1.3.8 get_col_vec() [1/2] . . . . .	37
7.1.3.9 get_col_vec() [2/2] . . . . .	37
7.1.3.10 get_entries() . . . . .	37
7.1.3.11 get_row_vec() [1/2] . . . . .	37
7.1.3.12 get_row_vec() [2/2] . . . . .	37
7.1.3.13 insert_cell() [1/3] . . . . .	38
7.1.3.14 insert_cell() [2/3] . . . . .	38
7.1.3.15 insert_cell() [3/3] . . . . .	38
7.1.3.16 is_dense() . . . . .	38

7.1.3.17 is_empty()	38
7.1.3.18 ncol()	39
7.1.3.19 nnozero()	39
7.1.3.20 nrow()	39
7.1.3.21 operator>() [1/2]	39
7.1.3.22 operator>() [2/2]	39
7.1.3.23 operator*=( )	39
7.1.3.24 operator+=( ) [1/3]	40
7.1.3.25 operator+=( ) [2/3]	40
7.1.3.26 operator+=( ) [3/3]	40
7.1.3.27 operator-=( ) [1/3]	40
7.1.3.28 operator-=( ) [2/3]	40
7.1.3.29 operator-=( ) [3/3]	40
7.1.3.30 operator/=( )	41
7.1.3.31 operator=( ) [1/2]	41
7.1.3.32 operator=( ) [2/2]	41
7.1.3.33 operator==( )	41
7.1.3.34 out_of_range()	41
7.1.3.35 print()	41
7.1.3.36 reserve()	42
7.1.3.37 resize()	42
7.1.3.38 rm_cell()	42
7.1.3.39 row()	42
7.1.3.40 set_data()	42
7.1.3.41 swap_cells()	43
7.1.3.42 swap_cols()	43
7.1.3.43 swap_rows()	43
7.1.3.44 toggle_cell()	43
7.1.3.45 toggle_lock()	44
7.1.3.46 transpose()	44
7.1.3.47 zero_col()	44
7.1.3.48 zero_row()	44
7.1.4 Friends And Related Function Documentation	44
7.1.4.1 BArrayCell< Cell_Type, Data_Type >	44
7.1.4.2 BArrayCell_const< Cell_Type, Data_Type >	45
7.1.5 Member Data Documentation	45
7.1.5.1 visited	45
7.2 BArrayCell< Cell_Type, Data_Type > Class Template Reference	45
7.2.1 Detailed Description	45
7.2.2 Constructor & Destructor Documentation	46
7.2.2.1 BArrayCell()	46
7.2.2.2 ~BArrayCell()	46

7.2.3 Member Function Documentation	46
7.2.3.1 operator Cell_Type()	46
7.2.3.2 operator*=( )	46
7.2.3.3 operator+=( )	47
7.2.3.4 operator-=( )	47
7.2.3.5 operator/=( )	47
7.2.3.6 operator=( )	47
7.2.3.7 operator==( )	47
7.3 BArrayCell_const< Cell_Type, Data_Type > Class Template Reference	48
7.3.1 Detailed Description	48
7.3.2 Constructor & Destructor Documentation	48
7.3.2.1 BArrayCell_const()	48
7.3.2.2 ~BArrayCell_const()	48
7.3.3 Member Function Documentation	49
7.3.3.1 operator Cell_Type()	49
7.3.3.2 operator!=( )	49
7.3.3.3 operator<( )	49
7.3.3.4 operator<=( )	49
7.3.3.5 operator==( )	49
7.3.3.6 operator>( )	50
7.3.3.7 operator>=( )	50
7.4 BArrayDense< Cell_Type, Data_Type > Class Template Reference	50
7.4.1 Detailed Description	52
7.4.2 Constructor & Destructor Documentation	53
7.4.2.1 BArrayDense() [1/6]	53
7.4.2.2 BArrayDense() [2/6]	53
7.4.2.3 BArrayDense() [3/6]	53
7.4.2.4 BArrayDense() [4/6]	54
7.4.2.5 BArrayDense() [5/6]	54
7.4.2.6 BArrayDense() [6/6]	54
7.4.2.7 ~BArrayDense()	54
7.4.3 Member Function Documentation	54
7.4.3.1 clear()	54
7.4.3.2 col() [1/2]	55
7.4.3.3 col() [2/2]	55
7.4.3.4 colsum()	55
7.4.3.5 D() [1/2]	55
7.4.3.6 D() [2/2]	55
7.4.3.7 default_val()	55
7.4.3.8 get_cell()	56
7.4.3.9 get_col_vec() [1/2]	56
7.4.3.10 get_col_vec() [2/2]	56

7.4.3.11 <a href="#">get_data()</a> . . . . .	56
7.4.3.12 <a href="#">get_entries()</a> . . . . .	56
7.4.3.13 <a href="#">get_row_vec()</a> [1/2] . . . . .	57
7.4.3.14 <a href="#">get_row_vec()</a> [2/2] . . . . .	57
7.4.3.15 <a href="#">insert_cell()</a> [1/2] . . . . .	57
7.4.3.16 <a href="#">insert_cell()</a> [2/2] . . . . .	57
7.4.3.17 <a href="#">is_dense()</a> . . . . .	57
7.4.3.18 <a href="#">is_empty()</a> . . . . .	58
7.4.3.19 <a href="#">ncol()</a> . . . . .	58
7.4.3.20 <a href="#">nnozero()</a> . . . . .	58
7.4.3.21 <a href="#">nrow()</a> . . . . .	58
7.4.3.22 <a href="#">operator&gt;()</a> [1/2] . . . . .	58
7.4.3.23 <a href="#">operator&gt;()</a> [2/2] . . . . .	58
7.4.3.24 <a href="#">operator*=( )</a> . . . . .	59
7.4.3.25 <a href="#">operator+=( )</a> [1/3] . . . . .	59
7.4.3.26 <a href="#">operator+=( )</a> [2/3] . . . . .	59
7.4.3.27 <a href="#">operator+=( )</a> [3/3] . . . . .	59
7.4.3.28 <a href="#">operator-=( )</a> [1/3] . . . . .	59
7.4.3.29 <a href="#">operator-=( )</a> [2/3] . . . . .	59
7.4.3.30 <a href="#">operator-=( )</a> [3/3] . . . . .	60
7.4.3.31 <a href="#">operator/=( )</a> . . . . .	60
7.4.3.32 <a href="#">operator=()</a> [1/2] . . . . .	60
7.4.3.33 <a href="#">operator=()</a> [2/2] . . . . .	60
7.4.3.34 <a href="#">operator==( )</a> . . . . .	60
7.4.3.35 <a href="#">out_of_range()</a> . . . . .	60
7.4.3.36 <a href="#">print()</a> . . . . .	61
7.4.3.37 <a href="#">reserve()</a> . . . . .	61
7.4.3.38 <a href="#">resize()</a> . . . . .	61
7.4.3.39 <a href="#">rm_cell()</a> . . . . .	61
7.4.3.40 <a href="#">row()</a> [1/2] . . . . .	61
7.4.3.41 <a href="#">row()</a> [2/2] . . . . .	61
7.4.3.42 <a href="#">rowsum()</a> . . . . .	62
7.4.3.43 <a href="#">set_data()</a> . . . . .	62
7.4.3.44 <a href="#">swap_cells()</a> . . . . .	62
7.4.3.45 <a href="#">swap_cols()</a> . . . . .	62
7.4.3.46 <a href="#">swap_rows()</a> . . . . .	63
7.4.3.47 <a href="#">toggle_cell()</a> . . . . .	63
7.4.3.48 <a href="#">toggle_lock()</a> . . . . .	63
7.4.3.49 <a href="#">transpose()</a> . . . . .	63
7.4.3.50 <a href="#">zero_col()</a> . . . . .	63
7.4.3.51 <a href="#">zero_row()</a> . . . . .	64
7.4.4 <a href="#">Friends And Related Function Documentation</a> . . . . .	64



7.4.4.1 BArrayDenseCell< Cell_Type, Data_Type > . . . . .	64
7.4.4.2 BArrayDenseCol< Cell_Type, Data_Type > . . . . .	64
7.4.4.3 BArrayDenseCol_const< Cell_Type, Data_Type > . . . . .	64
7.4.4.4 BArrayDenseRow< Cell_Type, Data_Type > . . . . .	64
7.4.4.5 BArrayDenseRow_const< Cell_Type, Data_Type > . . . . .	65
7.4.5 Member Data Documentation . . . . .	65
7.4.5.1 visited . . . . .	65
7.5 BArrayDenseCell< Cell_Type, Data_Type > Class Template Reference . . . . .	65
7.5.1 Detailed Description . . . . .	66
7.5.2 Constructor & Destructor Documentation . . . . .	66
7.5.2.1 BArrayDenseCell() . . . . .	66
7.5.2.2 ~BArrayDenseCell() . . . . .	66
7.5.3 Member Function Documentation . . . . .	66
7.5.3.1 operator Cell_Type() . . . . .	66
7.5.3.2 operator*=( ) . . . . .	67
7.5.3.3 operator+=( ) . . . . .	67
7.5.3.4 operator-=( ) . . . . .	67
7.5.3.5 operator/=( ) . . . . .	67
7.5.3.6 operator=( ) . . . . .	67
7.5.3.7 operator==( ) . . . . .	68
7.5.4 Friends And Related Function Documentation . . . . .	68
7.5.4.1 BArrayDense< Cell_Type, Data_Type > . . . . .	68
7.5.4.2 BArrayDenseCol< Cell_Type, Data_Type > . . . . .	68
7.5.4.3 BArrayDenseCol_const< Cell_Type, Data_Type > . . . . .	68
7.6 BArrayDenseCell_const< Cell_Type, Data_Type > Class Template Reference . . . . .	69
7.6.1 Detailed Description . . . . .	69
7.7 BArrayDenseCol< Cell_Type, Data_Type > Class Template Reference . . . . .	69
7.7.1 Detailed Description . . . . .	69
7.7.2 Constructor & Destructor Documentation . . . . .	69
7.7.2.1 BArrayDenseCol() . . . . .	70
7.7.3 Member Function Documentation . . . . .	70
7.7.3.1 begin() . . . . .	70
7.7.3.2 end() . . . . .	70
7.7.3.3 operator()( ) . . . . .	70
7.7.3.4 size() . . . . .	70
7.7.4 Friends And Related Function Documentation . . . . .	71
7.7.4.1 BArrayDense< Cell_Type, Data_Type > . . . . .	71
7.7.4.2 BArrayDenseCell< Cell_Type, Data_Type > . . . . .	71
7.7.4.3 BArrayDenseCell_const< Cell_Type, Data_Type > . . . . .	71
7.8 BArrayDenseCol_const< Cell_Type, Data_Type > Class Template Reference . . . . .	71
7.8.1 Detailed Description . . . . .	72
7.8.2 Constructor & Destructor Documentation . . . . .	72

7.8.2.1 BArrayDenseCol_const()	72
7.8.3 Member Function Documentation	72
7.8.3.1 begin()	72
7.8.3.2 end()	72
7.8.3.3 operator>()	73
7.8.3.4 size()	73
7.8.4 Friends And Related Function Documentation	73
7.8.4.1 BArrayDenseCell< Cell_Type, Data_Type >	73
7.8.4.2 BArrayDenseCell_const< Cell_Type, Data_Type >	73
7.9 BArrayDenseRow< Cell_Type, Data_Type > Class Template Reference	73
7.9.1 Detailed Description	74
7.9.2 Constructor & Destructor Documentation	74
7.9.2.1 BArrayDenseRow()	74
7.9.3 Member Function Documentation	74
7.9.3.1 begin()	74
7.9.3.2 end()	75
7.9.3.3 operator>()	75
7.9.3.4 size()	75
7.9.4 Friends And Related Function Documentation	75
7.9.4.1 BArrayDense< Cell_Type, Data_Type >	75
7.9.4.2 BArrayDenseCell< Cell_Type, Data_Type >	75
7.9.4.3 BArrayDenseCell_const< Cell_Type, Data_Type >	76
7.10 BArrayDenseRow_const< Cell_Type, Data_Type > Class Template Reference	76
7.10.1 Detailed Description	76
7.10.2 Constructor & Destructor Documentation	76
7.10.2.1 BArrayDenseRow_const()	77
7.10.3 Member Function Documentation	77
7.10.3.1 begin()	77
7.10.3.2 end()	77
7.10.3.3 operator>()	77
7.10.3.4 size()	77
7.10.4 Friends And Related Function Documentation	78
7.10.4.1 BArrayDenseCell< Cell_Type, Data_Type >	78
7.10.4.2 BArrayDenseCell_const< Cell_Type, Data_Type >	78
7.11 BArrayRow< Cell_Type, Data_Type > Class Template Reference	78
7.11.1 Detailed Description	78
7.11.2 Constructor & Destructor Documentation	79
7.11.2.1 BArrayRow()	79
7.11.2.2 ~BArrayRow()	79
7.11.3 Member Function Documentation	79
7.11.3.1 operator BArrayRow< Cell_Type, Data_Type >()	79
7.11.3.2 operator*=( )	79

7.11.3.3 operator+=()	79
7.11.3.4 operator-=()	80
7.11.3.5 operator/=(())	80
7.11.3.6 operator=()	80
7.11.3.7 operator==(())	80
7.12 BArrayRow_const< Cell_Type, Data_Type > Class Template Reference	80
7.12.1 Detailed Description	81
7.12.2 Constructor & Destructor Documentation	81
7.12.2.1 BArrayRow_const()	81
7.12.2.2 ~BArrayRow_const()	81
7.12.3 Member Function Documentation	81
7.12.3.1 operator BArrayRow_const< Cell_Type, Data_Type >()	81
7.12.3.2 operator!=(())	81
7.12.3.3 operator<()	82
7.12.3.4 operator<=()	82
7.12.3.5 operator==(())	82
7.12.3.6 operator>()	82
7.12.3.7 operator>=()	82
7.13 BArrayVector< Cell_Type, Data_Type > Class Template Reference	82
7.13.1 Detailed Description	83
7.13.2 Constructor & Destructor Documentation	83
7.13.2.1 BArrayVector()	83
7.13.2.2 ~BArrayVector()	84
7.13.3 Member Function Documentation	84
7.13.3.1 begin()	84
7.13.3.2 end()	84
7.13.3.3 is_col()	84
7.13.3.4 is_row()	85
7.13.3.5 operator std::vector< Cell_Type >()	85
7.13.3.6 operator*=(())	85
7.13.3.7 operator+=()	85
7.13.3.8 operator-=()	85
7.13.3.9 operator/=(())	86
7.13.3.10 operator=()	86
7.13.3.11 operator==(())	86
7.13.3.12 size()	86
7.14 BArrayVector_const< Cell_Type, Data_Type > Class Template Reference	86
7.14.1 Detailed Description	87
7.14.2 Constructor & Destructor Documentation	87
7.14.2.1 BArrayVector_const()	87
7.14.2.2 ~BArrayVector_const()	87
7.14.3 Member Function Documentation	88

7.14.3.1 begin()	88
7.14.3.2 end()	88
7.14.3.3 is_col()	88
7.14.3.4 is_row()	88
7.14.3.5 operator std::vector< Cell_Type >()	88
7.14.3.6 operator!=(())	89
7.14.3.7 operator<()	89
7.14.3.8 operator<=()	89
7.14.3.9 operator==(())	89
7.14.3.10 operator>()	89
7.14.3.11 operator>=()	90
7.14.3.12 size()	90
7.15 Cell< Cell_Type > Class Template Reference	90
7.15.1 Detailed Description	91
7.15.2 Constructor & Destructor Documentation	91
7.15.2.1 Cell() [1/7]	91
7.15.2.2 Cell() [2/7]	91
7.15.2.3 ~Cell()	91
7.15.2.4 Cell() [3/7]	92
7.15.2.5 Cell() [4/7]	92
7.15.2.6 Cell() [5/7]	92
7.15.2.7 Cell() [6/7]	92
7.15.2.8 Cell() [7/7]	92
7.15.3 Member Function Documentation	92
7.15.3.1 add() [1/4]	93
7.15.3.2 add() [2/4]	93
7.15.3.3 add() [3/4]	93
7.15.3.4 add() [4/4]	93
7.15.3.5 operator Cell_Type()	93
7.15.3.6 operator!=(())	93
7.15.3.7 operator=() [1/2]	94
7.15.3.8 operator=() [2/2]	94
7.15.3.9 operator==(())	94
7.15.4 Member Data Documentation	94
7.15.4.1 active	94
7.15.4.2 value	94
7.15.4.3 visited	95
7.16 Cell_const< Cell_Type > Class Template Reference	95
7.16.1 Detailed Description	95
7.17 ConstBArrayRowIter< Cell_Type, Data_Type > Class Template Reference	95
7.17.1 Detailed Description	96
7.17.2 Constructor & Destructor Documentation	96

7.17.2.1 ConstBArrayRowIter()	96
7.17.2.2 ~ConstBArrayRowIter()	96
7.17.3 Member Data Documentation	96
7.17.3.1 Array	97
7.17.3.2 current_col	97
7.17.3.3 current_row	97
7.17.3.4 iter	97
7.18 Counter< Array_Type, Data_Type > Class Template Reference	97
7.18.1 Detailed Description	98
7.18.2 Constructor & Destructor Documentation	98
7.18.2.1 Counter() [1/4]	99
7.18.2.2 Counter() [2/4]	99
7.18.2.3 Counter() [3/4]	99
7.18.2.4 Counter() [4/4]	99
7.18.2.5 ~Counter()	99
7.18.3 Member Function Documentation	100
7.18.3.1 count()	100
7.18.3.2 get_description()	100
7.18.3.3 get_name()	100
7.18.3.4 init()	100
7.18.3.5 operator=() [1/2]	100
7.18.3.6 operator=() [2/2]	101
7.18.4 Member Data Documentation	101
7.18.4.1 count_fun	101
7.18.4.2 data	101
7.18.4.3 delete_data	101
7.18.4.4 desc	101
7.18.4.5 init_fun	102
7.18.4.6 name	102
7.19 Counters< Array_Type, Data_Type > Class Template Reference	102
7.19.1 Detailed Description	103
7.19.2 Constructor & Destructor Documentation	103
7.19.2.1 Counters() [1/3]	103
7.19.2.2 ~Counters()	103
7.19.2.3 Counters() [2/3]	103
7.19.2.4 Counters() [3/3]	104
7.19.3 Member Function Documentation	104
7.19.3.1 add_counter() [1/3]	104
7.19.3.2 add_counter() [2/3]	104
7.19.3.3 add_counter() [3/3]	104
7.19.3.4 clear()	105
7.19.3.5 get_descriptions()	105

7.19.3.6 <code>get_names()</code>	105
7.19.3.7 <code>operator=()</code> [1/2]	105
7.19.3.8 <code>operator=()</code> [2/2]	105
7.19.3.9 <code>operator[]()</code>	106
7.19.3.10 <code>size()</code>	106
7.20 Entries< Cell_Type > Class Template Reference	107
7.20.1 Detailed Description	107
7.20.2 Constructor & Destructor Documentation	107
7.20.2.1 <code>Entries()</code> [1/2]	107
7.20.2.2 <code>Entries()</code> [2/2]	108
7.20.2.3 <code>~Entries()</code>	108
7.20.3 Member Function Documentation	108
7.20.3.1 <code>resize()</code>	108
7.20.4 Member Data Documentation	108
7.20.4.1 <code>source</code>	108
7.20.4.2 <code>target</code>	108
7.20.4.3 <code>val</code>	109
7.21 Flock Class Reference	109
7.21.1 Detailed Description	110
7.21.2 Constructor & Destructor Documentation	110
7.21.2.1 <code>Flock()</code>	110
7.21.2.2 <code>~Flock()</code>	110
7.21.3 Member Function Documentation	110
7.21.3.1 <code>add_data()</code>	110
7.21.3.2 <code>colnames()</code>	111
7.21.3.3 <code>get_counters()</code>	111
7.21.3.4 <code>get_model()</code>	111
7.21.3.5 <code>get_stats_support()</code>	111
7.21.3.6 <code>get_stats_target()</code>	112
7.21.3.7 <code>get_support_fun()</code>	112
7.21.3.8 <code>init()</code>	112
7.21.3.9 <code>likelihood_joint()</code>	112
7.21.3.10 <code>nfuncs()</code>	113
7.21.3.11 <code>nleafs()</code>	113
7.21.3.12 <code>nnodes()</code>	113
7.21.3.13 <code>nterms()</code>	113
7.21.3.14 <code>ntrees()</code>	113
7.21.3.15 <code>operator()()</code>	113
7.21.3.16 <code>parse_polytomies()</code>	114
7.21.3.17 <code>print()</code>	114
7.21.3.18 <code>set_seed()</code>	114
7.21.3.19 <code>support_size()</code>	115

7.21.4 Member Data Documentation	115
7.21.4.1 dat	115
7.21.4.2 initialized	115
7.21.4.3 model	115
7.21.4.4 nfunctions	115
7.21.4.5 rengine	116
7.22 FreqTable< T > Class Template Reference	116
7.22.1 Detailed Description	116
7.22.2 Constructor & Destructor Documentation	116
7.22.2.1 FreqTable()	117
7.22.2.2 ~FreqTable()	117
7.22.3 Member Function Documentation	117
7.22.3.1 add()	117
7.22.3.2 as_vector()	117
7.22.3.3 clear()	117
7.22.3.4 get_data()	118
7.22.3.5 get_index()	118
7.22.3.6 make_hash()	118
7.22.3.7 print()	118
7.22.3.8 reserve()	118
7.22.3.9 size()	119
7.23 Geese Class Reference	119
7.23.1 Detailed Description	121
7.23.2 Constructor & Destructor Documentation	122
7.23.2.1 Geese() [1/4]	122
7.23.2.2 Geese() [2/4]	122
7.23.2.3 Geese() [3/4]	122
7.23.2.4 Geese() [4/4]	122
7.23.2.5 ~Geese()	122
7.23.3 Member Function Documentation	123
7.23.3.1 calc_reduced_sequence()	123
7.23.3.2 calc_sequence()	123
7.23.3.3 colnames()	123
7.23.3.4 get_annotated_nodes()	123
7.23.3.5 get_counters()	123
7.23.3.6 get_model()	124
7.23.3.7 get_probabilities()	124
7.23.3.8 get_rengine()	124
7.23.3.9 get_states()	124
7.23.3.10 get_support_fun()	124
7.23.3.11 inherit_support()	125
7.23.3.12 init()	125

7.23.3.13	<a href="#">init_node()</a>	125
7.23.3.14	<a href="#">likelihood()</a>	125
7.23.3.15	<a href="#">likelihood_exhaust()</a>	125
7.23.3.16	<a href="#">nannotations()</a>	126
7.23.3.17	<a href="#">nfuncs()</a>	126
7.23.3.18	<a href="#">nleaves()</a>	126
7.23.3.19	<a href="#">nnodes()</a>	126
7.23.3.20	<a href="#">nterms()</a>	126
7.23.3.21	<a href="#">observed_counts()</a>	127
7.23.3.22	<a href="#">operator=()</a> [1/2]	127
7.23.3.23	<a href="#">operator=()</a> [2/2]	127
7.23.3.24	<a href="#">parse_polytomies()</a>	127
7.23.3.25	<a href="#">predict()</a>	127
7.23.3.26	<a href="#">predict_backend()</a>	128
7.23.3.27	<a href="#">predict_exhaust()</a>	128
7.23.3.28	<a href="#">predict_exhaust_backend()</a>	128
7.23.3.29	<a href="#">predict_sim()</a>	128
7.23.3.30	<a href="#">print()</a>	128
7.23.3.31	<a href="#">print_observed_counts()</a>	129
7.23.3.32	<a href="#">set_seed()</a>	129
7.23.3.33	<a href="#">simulate()</a>	129
7.23.3.34	<a href="#">support_size()</a>	129
7.23.3.35	<a href="#">update_annotations()</a>	129
7.23.4	<a href="#">Member Data Documentation</a>	129
7.23.4.1	<a href="#">delete_engine</a>	130
7.23.4.2	<a href="#">delete_support</a>	130
7.23.4.3	<a href="#">initialized</a>	130
7.23.4.4	<a href="#">map_to_nodes</a>	130
7.23.4.5	<a href="#">nfunctions</a>	130
7.23.4.6	<a href="#">nodes</a>	130
7.23.4.7	<a href="#">pset_loc</a>	131
7.23.4.8	<a href="#">reduced_sequence</a>	131
7.23.4.9	<a href="#">sequence</a>	131
7.24	<a href="#">Model&lt; Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type &gt; Class Template Reference</a>	131
7.24.1	<a href="#">Detailed Description</a>	134
7.24.2	<a href="#">Constructor &amp; Destructor Documentation</a>	134
7.24.2.1	<a href="#">Model()</a> [1/3]	134
7.24.2.2	<a href="#">Model()</a> [2/3]	134
7.24.2.3	<a href="#">Model()</a> [3/3]	135
7.24.2.4	<a href="#">~Model()</a>	135
7.24.3	<a href="#">Member Function Documentation</a>	135



7.24.3.1 add_array()	135
7.24.3.2 add_counter() [1/3]	135
7.24.3.3 add_counter() [2/3]	136
7.24.3.4 add_counter() [3/3]	136
7.24.3.5 add_rule() [1/3]	136
7.24.3.6 add_rule() [2/3]	136
7.24.3.7 add_rule() [3/3]	136
7.24.3.8 add_rule_dyn() [1/3]	137
7.24.3.9 add_rule_dyn() [2/3]	137
7.24.3.10 add_rule_dyn() [3/3]	137
7.24.3.11 colnames()	137
7.24.3.12 conditional_prob()	137
7.24.3.13 gen_key()	138
7.24.3.14 get_arrays2support()	138
7.24.3.15 get_counters()	138
7.24.3.16 get_norm_const()	138
7.24.3.17 get_pset()	139
7.24.3.18 get_pset_arrays()	139
7.24.3.19 get_pset_probs()	139
7.24.3.20 get_pset_stats() [1/2]	139
7.24.3.21 get_pset_stats() [2/2]	139
7.24.3.22 get_rengine()	140
7.24.3.23 get_rules()	140
7.24.3.24 get_rules_dyn()	140
7.24.3.25 get_stats_support()	140
7.24.3.26 get_stats_target()	140
7.24.3.27 get_support_fun()	141
7.24.3.28 likelihood() [1/3]	141
7.24.3.29 likelihood() [2/3]	141
7.24.3.30 likelihood() [3/3]	141
7.24.3.31 likelihood_total()	142
7.24.3.32 nterms()	142
7.24.3.33 operator=()	142
7.24.3.34 print()	142
7.24.3.35 print_stats()	142
7.24.3.36 sample() [1/2]	143
7.24.3.37 sample() [2/2]	143
7.24.3.38 set_counters()	143
7.24.3.39 set_keygen()	143
7.24.3.40 set_rengine()	143
7.24.3.41 set_rules()	144
7.24.3.42 set_rules_dyn()	144

---

7.24.3.43	set_seed()	144
7.24.3.44	size()	144
7.24.3.45	size_unique()	144
7.24.3.46	store_psets()	145
7.24.3.47	support_size()	145
7.25	NetCounterData Class Reference	145
7.25.1	Detailed Description	145
7.25.2	Constructor & Destructor Documentation	145
7.25.2.1	NetCounterData() [1/2]	146
7.25.2.2	NetCounterData() [2/2]	146
7.25.2.3	~NetCounterData()	146
7.25.3	Member Data Documentation	146
7.25.3.1	indices	146
7.25.3.2	numbers	146
7.26	NetworkData Class Reference	147
7.26.1	Detailed Description	147
7.26.2	Constructor & Destructor Documentation	147
7.26.2.1	NetworkData() [1/3]	147
7.26.2.2	NetworkData() [2/3]	147
7.26.2.3	NetworkData() [3/3]	148
7.26.2.4	~NetworkData()	148
7.26.3	Member Data Documentation	148
7.26.3.1	directed	148
7.26.3.2	vertex_attr	149
7.27	Node Class Reference	149
7.27.1	Detailed Description	150
7.27.2	Constructor & Destructor Documentation	150
7.27.2.1	Node() [1/5]	150
7.27.2.2	Node() [2/5]	151
7.27.2.3	Node() [3/5]	151
7.27.2.4	Node() [4/5]	151
7.27.2.5	Node() [5/5]	151
7.27.2.6	~Node()	151
7.27.3	Member Function Documentation	151
7.27.3.1	get_parent()	152
7.27.3.2	is_leaf()	152
7.27.3.3	noffspring()	152
7.27.4	Member Data Documentation	152
7.27.4.1	annotations	152
7.27.4.2	array	152
7.27.4.3	arrays	153
7.27.4.4	duplication	153

7.27.4.5 id . . . . .	153
7.27.4.6 narray . . . . .	153
7.27.4.7 offspring . . . . .	153
7.27.4.8 ord . . . . .	154
7.27.4.9 parent . . . . .	154
7.27.4.10 probability . . . . .	154
7.27.4.11 subtree_prob . . . . .	154
7.27.4.12 visited . . . . .	154
7.28 NodeData Class Reference . . . . .	155
7.28.1 Detailed Description . . . . .	155
7.28.2 Constructor & Destructor Documentation . . . . .	155
7.28.2.1 NodeData() . . . . .	155
7.28.3 Member Data Documentation . . . . .	155
7.28.3.1 blengths . . . . .	156
7.28.3.2 duplication . . . . .	156
7.28.3.3 states . . . . .	156
7.29 PhyloCounterData Class Reference . . . . .	156
7.29.1 Detailed Description . . . . .	157
7.29.2 Constructor & Destructor Documentation . . . . .	157
7.29.2.1 PhyloCounterData() . . . . .	157
7.29.3 Member Function Documentation . . . . .	157
7.29.3.1 at() . . . . .	157
7.29.3.2 begin() . . . . .	157
7.29.3.3 empty() . . . . .	157
7.29.3.4 end() . . . . .	158
7.29.3.5 get_counters() . . . . .	158
7.29.3.6 operator>() . . . . .	158
7.29.3.7 operator[]() . . . . .	158
7.29.3.8 push_back() . . . . .	158
7.29.3.9 reserve() . . . . .	158
7.29.3.10 shrink_to_fit() . . . . .	159
7.29.3.11 size() . . . . .	159
7.30 PhyloRuleDynData Class Reference . . . . .	159
7.30.1 Detailed Description . . . . .	159
7.30.2 Constructor & Destructor Documentation . . . . .	159
7.30.2.1 PhyloRuleDynData() . . . . .	160
7.30.2.2 ~PhyloRuleDynData() . . . . .	160
7.30.3 Member Data Documentation . . . . .	160
7.30.3.1 counts . . . . .	160
7.30.3.2 duplication . . . . .	160
7.30.3.3 lb . . . . .	160
7.30.3.4 pos . . . . .	161

7.30.3.5 ub	161
7.31 PowerSet< Array_Type, Data_Rule_Type > Class Template Reference	161
7.31.1 Detailed Description	162
7.31.2 Constructor & Destructor Documentation	163
7.31.2.1 PowerSet() [1/3]	163
7.31.2.2 PowerSet() [2/3]	163
7.31.2.3 PowerSet() [3/3]	163
7.31.2.4 ~PowerSet()	163
7.31.3 Member Function Documentation	164
7.31.3.1 add_rule() [1/3]	164
7.31.3.2 add_rule() [2/3]	164
7.31.3.3 add_rule() [3/3]	164
7.31.3.4 begin()	164
7.31.3.5 calc()	164
7.31.3.6 end()	165
7.31.3.7 get_data()	165
7.31.3.8 get_data_ptr()	165
7.31.3.9 init_support()	165
7.31.3.10 operator[]()	165
7.31.3.11 reset()	166
7.31.3.12 size()	166
7.31.4 Member Data Documentation	166
7.31.4.1 coordinates_free	166
7.31.4.2 coordinates_locked	166
7.31.4.3 data	166
7.31.4.4 EmptyArray	167
7.31.4.5 M	167
7.31.4.6 N	167
7.31.4.7 n_free	167
7.31.4.8 n_locked	167
7.31.4.9 rules	167
7.31.4.10 rules_deleted	168
7.32 Progress Class Reference	168
7.32.1 Detailed Description	168
7.32.2 Constructor & Destructor Documentation	168
7.32.2.1 Progress()	168
7.32.2.2 ~Progress()	169
7.32.3 Member Function Documentation	169
7.32.3.1 end()	169
7.32.3.2 next()	169
7.33 Rule< Array_Type, Data_Type > Class Template Reference	169
7.33.1 Detailed Description	170

7.33.2 Constructor & Destructor Documentation	170
7.33.2.1 Rule() [1/2]	170
7.33.2.2 Rule() [2/2]	170
7.33.2.3 ~Rule()	171
7.33.3 Member Function Documentation	171
7.33.3.1 D()	171
7.33.3.2 operator()()	171
7.34 Rules< Array_Type, Data_Type > Class Template Reference	171
7.34.1 Detailed Description	172
7.34.2 Constructor & Destructor Documentation	172
7.34.2.1 Rules() [1/2]	172
7.34.2.2 Rules() [2/2]	172
7.34.2.3 ~Rules()	173
7.34.3 Member Function Documentation	173
7.34.3.1 add_rule() [1/3]	173
7.34.3.2 add_rule() [2/3]	173
7.34.3.3 add_rule() [3/3]	173
7.34.3.4 clear()	173
7.34.3.5 get_seq()	173
7.34.3.6 operator()()	174
7.34.3.7 operator=()	174
7.34.3.8 size()	175
7.35 StatsCounter< Array_Type, Data_Type > Class Template Reference	175
7.35.1 Detailed Description	175
7.35.2 Constructor & Destructor Documentation	176
7.35.2.1 StatsCounter() [1/3]	176
7.35.2.2 StatsCounter() [2/3]	176
7.35.2.3 StatsCounter() [3/3]	176
7.35.2.4 ~StatsCounter()	176
7.35.3 Member Function Documentation	177
7.35.3.1 add_counter() [1/2]	177
7.35.3.2 add_counter() [2/2]	177
7.35.3.3 count_all()	177
7.35.3.4 count_current()	177
7.35.3.5 count_init()	177
7.35.3.6 get_counters()	178
7.35.3.7 get_descriptions()	178
7.35.3.8 get_names()	178
7.35.3.9 reset_array()	178
7.35.3.10 set_counters()	178
7.36 Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type > Class Template Reference	179

7.36.1 Detailed Description	180
7.36.2 Constructor & Destructor Documentation	181
7.36.2.1 Support() [1/3]	181
7.36.2.2 Support() [2/3]	181
7.36.2.3 Support() [3/3]	181
7.36.2.4 ~Support()	181
7.36.3 Member Function Documentation	182
7.36.3.1 add_counter() [1/2]	182
7.36.3.2 add_counter() [2/2]	182
7.36.3.3 add_rule() [1/2]	182
7.36.3.4 add_rule() [2/2]	182
7.36.3.5 add_rule_dyn() [1/2]	182
7.36.3.6 add_rule_dyn() [2/2]	183
7.36.3.7 calc()	183
7.36.3.8 eval_rules_dyn()	183
7.36.3.9 get_counters()	183
7.36.3.10 get_counts()	184
7.36.3.11 get_current_stats()	184
7.36.3.12 get_data()	184
7.36.3.13 get_rules()	184
7.36.3.14 get_rules_dyn()	184
7.36.3.15 init_support()	185
7.36.3.16 print()	185
7.36.3.17 reset_array() [1/2]	185
7.36.3.18 reset_array() [2/2]	185
7.36.3.19 set_counters()	185
7.36.3.20 set_rules()	186
7.36.3.21 set_rules_dyn()	186
7.36.4 Member Data Documentation	186
7.36.4.1 change_stats	186
7.36.4.2 coordiantes_n_free	186
7.36.4.3 coordiantes_n_locked	186
7.36.4.4 coordinates_free	187
7.36.4.5 coordinates_locked	187
7.36.4.6 current_stats	187
7.36.4.7 delete_counters	187
7.36.4.8 delete_rules	187
7.36.4.9 delete_rules_dyn	188
7.36.4.10 hashes	188
7.36.4.11 hashes_initialized	188
7.36.4.12 M	188
7.36.4.13 max_num_elements	188

7.36.4.14 N	189
7.36.4.15 n_counters	189
7.37 vecHasher< T > Struct Template Reference	189
7.37.1 Detailed Description	189
7.37.2 Member Function Documentation	189
7.37.2.1 operator>()	189
<b>8 File Documentation</b>	<b>191</b>
8.1 include/barry/barray-bones.hpp File Reference	191
8.1.1 Macro Definition Documentation	192
8.1.1.1 BARRAY_BONES_HPP	192
8.2 include/barry/barray-iterator.hpp File Reference	192
8.3 include/barry/barray-meat-operators.hpp File Reference	193
8.3.1 Macro Definition Documentation	194
8.3.1.1 BARRAY_TEMPLATE	194
8.3.1.2 BARRAY_TEMPLATE_ARGS	194
8.3.1.3 BARRAY_TYPE	194
8.3.1.4 BARRY_BARRAY_MEAT_OPERATORS_HPP	195
8.3.1.5 COL	195
8.3.1.6 ROW	195
8.3.2 Function Documentation	195
8.3.2.1 BARRAY_TEMPLATE() [1/6]	195
8.3.2.2 BARRAY_TEMPLATE() [2/6]	195
8.3.2.3 BARRAY_TEMPLATE() [3/6]	196
8.3.2.4 BARRAY_TEMPLATE() [4/6]	196
8.3.2.5 BARRAY_TEMPLATE() [5/6]	196
8.3.2.6 BARRAY_TEMPLATE() [6/6]	196
8.3.2.7 BARRAY_TEMPLATE_ARGS()	196
8.3.2.8 BARRAY_TYPE()	196
8.3.2.9 for()	197
8.3.2.10 operator>()	197
8.3.3 Variable Documentation	197
8.3.3.1 rhs	197
8.3.3.2 this	197
8.4 include/barry/barray-meat.hpp File Reference	198
8.4.1 Macro Definition Documentation	200
8.4.1.1 BARRAY_TEMPLATE	200
8.4.1.2 BARRAY_TEMPLATE_ARGS	200
8.4.1.3 BARRAY_TYPE	201
8.4.1.4 COL	201
8.4.1.5 ROW	201
8.4.2 Function Documentation	201

8.4.2.1 ans()	201
8.4.2.2 BARRAY_TEMPLATE() [1/23]	201
8.4.2.3 BARRAY_TEMPLATE() [2/23]	202
8.4.2.4 BARRAY_TEMPLATE() [3/23]	202
8.4.2.5 BARRAY_TEMPLATE() [4/23]	202
8.4.2.6 BARRAY_TEMPLATE() [5/23]	202
8.4.2.7 BARRAY_TEMPLATE() [6/23]	202
8.4.2.8 BARRAY_TEMPLATE() [7/23]	202
8.4.2.9 BARRAY_TEMPLATE() [8/23]	203
8.4.2.10 BARRAY_TEMPLATE() [9/23]	203
8.4.2.11 BARRAY_TEMPLATE() [10/23]	203
8.4.2.12 BARRAY_TEMPLATE() [11/23]	203
8.4.2.13 BARRAY_TEMPLATE() [12/23]	203
8.4.2.14 BARRAY_TEMPLATE() [13/23]	204
8.4.2.15 BARRAY_TEMPLATE() [14/23]	204
8.4.2.16 BARRAY_TEMPLATE() [15/23]	204
8.4.2.17 BARRAY_TEMPLATE() [16/23]	204
8.4.2.18 BARRAY_TEMPLATE() [17/23]	204
8.4.2.19 BARRAY_TEMPLATE() [18/23]	204
8.4.2.20 BARRAY_TEMPLATE() [19/23]	205
8.4.2.21 BARRAY_TEMPLATE() [20/23]	205
8.4.2.22 BARRAY_TEMPLATE() [21/23]	205
8.4.2.23 BARRAY_TEMPLATE() [22/23]	205
8.4.2.24 BARRAY_TEMPLATE() [23/23]	205
8.4.2.25 COL()	205
8.4.2.26 for() [1/3]	206
8.4.2.27 for() [2/3]	206
8.4.2.28 for() [3/3]	206
8.4.2.29 if() [1/17]	206
8.4.2.30 if() [2/17]	206
8.4.2.31 if() [3/17]	206
8.4.2.32 if() [4/17]	207
8.4.2.33 if() [5/17]	207
8.4.2.34 if() [6/17]	207
8.4.2.35 if() [7/17]	207
8.4.2.36 if() [8/17]	207
8.4.2.37 if() [9/17]	207
8.4.2.38 if() [10/17]	208
8.4.2.39 if() [11/17]	208
8.4.2.40 if() [12/17]	208
8.4.2.41 if() [13/17]	208
8.4.2.42 if() [14/17]	208



8.4.2.43 if() [15/17]	208
8.4.2.44 if() [16/17]	208
8.4.2.45 if() [17/17]	209
8.4.2.46 M()	209
8.4.2.47 resize() [1/2]	209
8.4.2.48 resize() [2/2]	209
8.4.2.49 return()	209
8.4.2.50 ROW() [1/2]	209
8.4.2.51 ROW() [2/2]	209
8.4.3 Variable Documentation	210
8.4.3.1 add	210
8.4.3.2 ans	210
8.4.3.3 Array_	210
8.4.3.4 check_bounds	210
8.4.3.5 check_exists	211
8.4.3.6 col0	211
8.4.3.7 const	211
8.4.3.8 copy_data	211
8.4.3.9 data	211
8.4.3.10 delete_data	212
8.4.3.11 delete_data_	212
8.4.3.12 else	212
8.4.3.13 false	212
8.4.3.14 first	212
8.4.3.15 i1	213
8.4.3.16 j	213
8.4.3.17 j0	213
8.4.3.18 j1	213
8.4.3.19 M	213
8.4.3.20 M_	214
8.4.3.21 N	214
8.4.3.22 NCells	214
8.4.3.23 report	214
8.4.3.24 return	214
8.4.3.25 row0	215
8.4.3.26 search	215
8.4.3.27 source	215
8.4.3.28 target	215
8.4.3.29 v	215
8.4.3.30 value	215
8.5 include/barry/barraycell-bones.hpp File Reference	216
8.6 include/barry/barraycell-meat.hpp File Reference	216

8.7 include/barry/barraydense-bones.hpp File Reference . . . . .	217
8.7.1 Macro Definition Documentation . . . . .	219
8.7.1.1 BARRY_BARRAYDENSE_BONES_HPP . . . . .	219
8.8 include/barry/barraydense-meat-operators.hpp File Reference . . . . .	219
8.8.1 Macro Definition Documentation . . . . .	220
8.8.1.1 BARRY_BARRAYDENSE_MEAT_OPERATORS_HPP . . . . .	220
8.8.1.2 BDENSE_TEMPLATE . . . . .	220
8.8.1.3 BDENSE_TEMPLATE_ARGS . . . . .	220
8.8.1.4 BDENSE_TYPE . . . . .	221
8.8.1.5 COL . . . . .	221
8.8.1.6 POS . . . . .	221
8.8.1.7 POS_N . . . . .	221
8.8.1.8 ROW . . . . .	221
8.8.2 Function Documentation . . . . .	221
8.8.2.1 BDENSE_TEMPLATE() [1/4] . . . . .	222
8.8.2.2 BDENSE_TEMPLATE() [2/4] . . . . .	222
8.8.2.3 BDENSE_TEMPLATE() [3/4] . . . . .	222
8.8.2.4 BDENSE_TEMPLATE() [4/4] . . . . .	222
8.8.2.5 BDENSE_TEMPLATE_ARGS() . . . . .	222
8.8.2.6 BDENSE_TYPE() . . . . .	222
8.9 include/barry/barraydense-meat.hpp File Reference . . . . .	223
8.9.1 Macro Definition Documentation . . . . .	225
8.9.1.1 BDENSE_TEMPLATE . . . . .	225
8.9.1.2 BDENSE_TEMPLATE_ARGS . . . . .	226
8.9.1.3 BDENSE_TYPE . . . . .	226
8.9.1.4 COL . . . . .	226
8.9.1.5 POS . . . . .	226
8.9.1.6 POS_N . . . . .	226
8.9.1.7 ROW . . . . .	227
8.9.1.8 ZERO_CELL . . . . .	227
8.9.2 Function Documentation . . . . .	227
8.9.2.1 ans() . . . . .	227
8.9.2.2 BDENSE_TEMPLATE() [1/37] . . . . .	227
8.9.2.3 BDENSE_TEMPLATE() [2/37] . . . . .	227
8.9.2.4 BDENSE_TEMPLATE() [3/37] . . . . .	227
8.9.2.5 BDENSE_TEMPLATE() [4/37] . . . . .	228
8.9.2.6 BDENSE_TEMPLATE() [5/37] . . . . .	228
8.9.2.7 BDENSE_TEMPLATE() [6/37] . . . . .	228
8.9.2.8 BDENSE_TEMPLATE() [7/37] . . . . .	228
8.9.2.9 BDENSE_TEMPLATE() [8/37] . . . . .	228
8.9.2.10 BDENSE_TEMPLATE() [9/37] . . . . .	229
8.9.2.11 BDENSE_TEMPLATE() [10/37] . . . . .	229

8.9.2.12 BDENSE_TEMPLATE() [11/37]	229
8.9.2.13 BDENSE_TEMPLATE() [12/37]	229
8.9.2.14 BDENSE_TEMPLATE() [13/37]	229
8.9.2.15 BDENSE_TEMPLATE() [14/37]	230
8.9.2.16 BDENSE_TEMPLATE() [15/37]	230
8.9.2.17 BDENSE_TEMPLATE() [16/37]	230
8.9.2.18 BDENSE_TEMPLATE() [17/37]	230
8.9.2.19 BDENSE_TEMPLATE() [18/37]	230
8.9.2.20 BDENSE_TEMPLATE() [19/37]	231
8.9.2.21 BDENSE_TEMPLATE() [20/37]	231
8.9.2.22 BDENSE_TEMPLATE() [21/37]	231
8.9.2.23 BDENSE_TEMPLATE() [22/37]	231
8.9.2.24 BDENSE_TEMPLATE() [23/37]	231
8.9.2.25 BDENSE_TEMPLATE() [24/37]	232
8.9.2.26 BDENSE_TEMPLATE() [25/37]	232
8.9.2.27 BDENSE_TEMPLATE() [26/37]	232
8.9.2.28 BDENSE_TEMPLATE() [27/37]	232
8.9.2.29 BDENSE_TEMPLATE() [28/37]	232
8.9.2.30 BDENSE_TEMPLATE() [29/37]	232
8.9.2.31 BDENSE_TEMPLATE() [30/37]	233
8.9.2.32 BDENSE_TEMPLATE() [31/37]	233
8.9.2.33 BDENSE_TEMPLATE() [32/37]	233
8.9.2.34 BDENSE_TEMPLATE() [33/37]	233
8.9.2.35 BDENSE_TEMPLATE() [34/37]	233
8.9.2.36 BDENSE_TEMPLATE() [35/37]	233
8.9.2.37 BDENSE_TEMPLATE() [36/37]	234
8.9.2.38 BDENSE_TEMPLATE() [37/37]	234
8.9.2.39 for()	234
8.9.2.40 if() [1/4]	234
8.9.2.41 if() [2/4]	234
8.9.2.42 if() [3/4]	234
8.9.2.43 if() [4/4]	235
8.9.2.44 insert_cell() [1/2]	235
8.9.2.45 insert_cell() [2/2]	235
8.9.2.46 M()	235
8.9.2.47 resize() [1/6]	235
8.9.2.48 resize() [2/6]	236
8.9.2.49 resize() [3/6]	236
8.9.2.50 resize() [4/6]	236
8.9.2.51 resize() [5/6]	236
8.9.2.52 resize() [6/6]	236
8.9.2.53 rm_cell() [1/3]	236

8.9.2.54 <a href="#">rm_cell()</a> [2/3]	237
8.9.2.55 <a href="#">rm_cell()</a> [3/3]	237
8.9.2.56 <a href="#">va_end()</a>	237
8.9.2.57 <a href="#">va_start()</a>	237
8.9.2.58 <a href="#">vprintf()</a>	237
8.9.3 Variable Documentation	237
8.9.3.1 <a href="#">add</a>	238
8.9.3.2 <a href="#">ans</a>	238
8.9.3.3 <a href="#">check_bounds</a>	238
8.9.3.4 <a href="#">check_exists</a>	238
8.9.3.5 <a href="#">col</a>	239
8.9.3.6 <a href="#">const</a>	239
8.9.3.7 <a href="#">copy_data</a>	239
8.9.3.8 <a href="#">data</a>	239
8.9.3.9 <a href="#">delete_data</a>	239
8.9.3.10 <a href="#">delete_data_</a>	240
8.9.3.11 <a href="#">el</a>	240
8.9.3.12 <a href="#">el_colsums</a>	240
8.9.3.13 <a href="#">el_rowsums</a>	240
8.9.3.14 <a href="#">else</a>	240
8.9.3.15 <a href="#">false</a>	241
8.9.3.16 <a href="#">i1</a>	241
8.9.3.17 <a href="#">j</a>	241
8.9.3.18 <a href="#">j0</a>	241
8.9.3.19 <a href="#">j1</a>	241
8.9.3.20 <a href="#">M</a>	241
8.9.3.21 <a href="#">M_</a>	242
8.9.3.22 <a href="#">N</a>	242
8.9.3.23 <a href="#">report</a>	242
8.9.3.24 <a href="#">return</a>	242
8.9.3.25 <a href="#">source</a>	242
8.9.3.26 <a href="#">target</a>	243
8.9.3.27 <a href="#">v</a>	243
8.9.3.28 <a href="#">val0</a>	243
8.9.3.29 <a href="#">val1</a>	243
8.9.3.30 <a href="#">value</a>	243
8.10 <a href="#">include/barry/barraydensecell-bones.hpp</a> File Reference	244
8.10.1 Macro Definition Documentation	244
8.10.1.1 <a href="#">POS</a>	245
8.11 <a href="#">include/barry/barraydensecell-meat.hpp</a> File Reference	245
8.11.1 Macro Definition Documentation	246
8.11.1.1 <a href="#">POS</a>	246

8.12 include/barry/barraydensecol-bones.hpp File Reference	246
8.12.1 Macro Definition Documentation	247
8.12.1.1 POS	247
8.12.1.2 POS_N	247
8.12.1.3 ZERO_CELL	247
8.13 include/barry/barraydenserow-bones.hpp File Reference	247
8.13.1 Macro Definition Documentation	248
8.13.1.1 POS	248
8.13.1.2 POS_N	248
8.13.1.3 ZERO_CELL	248
8.14 include/barry/barrayrow-bones.hpp File Reference	249
8.15 include/barry/barrayrow-meat.hpp File Reference	249
8.15.1 Macro Definition Documentation	250
8.15.1.1 BARRY_BARRAYROW_MEAT_HPP	250
8.15.1.2 BROW_TEMPLATE	251
8.15.1.3 BROW_TEMPLATE_ARGS	251
8.15.1.4 BROW_TYPE	251
8.15.2 Function Documentation	251
8.15.2.1 BROW_TEMPLATE() [1/5]	251
8.15.2.2 BROW_TEMPLATE() [2/5]	251
8.15.2.3 BROW_TEMPLATE() [3/5]	252
8.15.2.4 BROW_TEMPLATE() [4/5]	252
8.15.2.5 BROW_TEMPLATE() [5/5]	252
8.16 include/barry/barrayvector-bones.hpp File Reference	252
8.17 include/barry/barrayvector-meat.hpp File Reference	253
8.17.1 Macro Definition Documentation	254
8.17.1.1 BARRY_BARRAYVECTOR_MEAT_HPP	254
8.18 include/barry/barry-configuration.hpp File Reference	254
8.18.1 Macro Definition Documentation	255
8.18.1.1 BARRY_CHECK_SUPPORT	255
8.18.1.2 BARRY_ISFINITE	255
8.18.1.3 BARRY_MAX_NUM_ELEMENTS	255
8.18.1.4 BARRY_SAFE_EXP	255
8.18.1.5 printf_barry	255
8.18.2 Typedef Documentation	255
8.18.2.1 Map	256
8.19 include/barry/barry-debug.hpp File Reference	256
8.19.1 Macro Definition Documentation	256
8.19.1.1 BARRY_DEBUG_LEVEL	256
8.20 include/barry/barry-macros.hpp File Reference	256
8.20.1 Macro Definition Documentation	257
8.20.1.1 BARRY_ONE	257

8.20.1.2 BARRY_ONE_DENSE	257
8.20.1.3 BARRY_UNUSED	257
8.20.1.4 BARRY_ZERO	257
8.20.1.5 BARRY_ZERO_DENSE	258
8.21 include/barry/barry.hpp File Reference	258
8.21.1 Macro Definition Documentation	259
8.21.1.1 BARRY_HPP	259
8.21.1.2 BARRY_VERSION	259
8.21.1.3 COUNTER_FUNCTION	260
8.21.1.4 COUNTER_LAMBDA	260
8.21.1.5 RULE_FUNCTION	260
8.21.1.6 RULE_LAMBDA	260
8.22 include/barry/cell-bones.hpp File Reference	261
8.23 include/barry/cell-meat.hpp File Reference	261
8.24 include/barry/col-bones.hpp File Reference	262
8.25 include/barry/counters-bones.hpp File Reference	262
8.26 include/barry/counters-meat.hpp File Reference	264
8.26.1 Macro Definition Documentation	266
8.26.1.1 COUNTER_TEMPLATE	266
8.26.1.2 COUNTER_TEMPLATE_ARGS	266
8.26.1.3 COUNTER_TYPE	266
8.26.1.4 COUNTERS_TEMPLATE	266
8.26.1.5 COUNTERS_TEMPLATE_ARGS	266
8.26.1.6 COUNTERS_TYPE	267
8.26.2 Function Documentation	267
8.26.2.1 count_fun()	267
8.26.2.2 COUNTER_TEMPLATE() [1/7]	267
8.26.2.3 COUNTER_TEMPLATE() [2/7]	267
8.26.2.4 COUNTER_TEMPLATE() [3/7]	267
8.26.2.5 COUNTER_TEMPLATE() [4/7]	268
8.26.2.6 COUNTER_TEMPLATE() [5/7]	268
8.26.2.7 COUNTER_TEMPLATE() [6/7]	268
8.26.2.8 COUNTER_TEMPLATE() [7/7]	268
8.26.2.9 COUNTERS_TEMPLATE() [1/8]	268
8.26.2.10 COUNTERS_TEMPLATE() [2/8]	269
8.26.2.11 COUNTERS_TEMPLATE() [3/8]	269
8.26.2.12 COUNTERS_TEMPLATE() [4/8]	269
8.26.2.13 COUNTERS_TEMPLATE() [5/8]	269
8.26.2.14 COUNTERS_TEMPLATE() [6/8]	269
8.26.2.15 COUNTERS_TEMPLATE() [7/8]	270
8.26.2.16 COUNTERS_TEMPLATE() [8/8]	270
8.26.2.17 data()	270

8.26.2.18 delete_data() [1/3]	270
8.26.2.19 delete_data() [2/3]	270
8.26.2.20 delete_data() [3/3]	270
8.26.2.21 delete_to_be_deleted() [1/2]	271
8.26.2.22 delete_to_be_deleted() [2/2]	271
8.26.2.23 desc()	271
8.26.2.24 init_fun() [1/3]	271
8.26.2.25 init_fun() [2/3]	271
8.26.2.26 init_fun() [3/3]	272
8.26.2.27 name()	272
8.26.2.28 push_back() [1/2]	272
8.26.2.29 push_back() [2/2]	272
8.26.2.30 to_be_deleted() [1/2]	272
8.26.2.31 to_be_deleted() [2/2]	272
8.26.3 Variable Documentation	272
8.26.3.1 count_fun_	273
8.26.3.2 counter	273
8.26.3.3 counter_	273
8.26.3.4 data_	273
8.26.3.5 delete_data_	274
8.26.3.6 desc_	274
8.26.3.7 i	274
8.26.3.8 init_fun_	274
8.26.3.9 j	274
8.26.3.10 name_	275
8.26.3.11 noexcept	275
8.26.3.12 return	275
8.27 include/barry/counters/network-css.hpp File Reference	276
8.27.1 Macro Definition Documentation	277
8.27.1.1 CSS_APPEND	277
8.27.1.2 CSS_CASE_ELSE	278
8.27.1.3 CSS_CASE_PERCEIVED	278
8.27.1.4 CSS_CASE_TRUTH	278
8.27.1.5 CSS_CHECK_SIZE	278
8.27.1.6 CSS_CHECK_SIZE_INIT	278
8.27.1.7 CSS_NET_COUNTER_LAMBDA_INIT	279
8.27.1.8 CSS_PERCEIVED_CELLS	279
8.27.1.9 CSS_SIZE	279
8.27.1.10 CSS_TRUE_CELLS	279
8.27.2 Function Documentation	279
8.27.2.1 counter_css_census01()	280
8.27.2.2 counter_css_census02()	280

8.27.2.3 counter_css_census03()	280
8.27.2.4 counter_css_census04()	280
8.27.2.5 counter_css_census05()	281
8.27.2.6 counter_css_census06()	281
8.27.2.7 counter_css_census07()	281
8.27.2.8 counter_css_census08()	281
8.27.2.9 counter_css_census09()	282
8.27.2.10 counter_css_census10()	282
8.27.2.11 counter_css_completely_false_recip_comiss()	282
8.27.2.12 counter_css_completely_false_recip_omiss()	282
8.27.2.13 counter_css_mixed_recip()	283
8.27.2.14 counter_css_partially_false_recip_commi()	283
8.27.2.15 counter_css_partially_false_recip_omiss()	283
8.28 include/barry/counters/network.hpp File Reference	284
8.28.1 Macro Definition Documentation	286
8.28.1.1 BARRY_ZERO_NETWORK	287
8.28.1.2 BARRY_ZERO_NETWORK_DENSE	287
8.28.1.3 NET_C_DATA_IDX	287
8.28.1.4 NET_C_DATA_NUM	287
8.28.1.5 NETWORK_COUNTER	287
8.28.1.6 NETWORK_COUNTER_LAMBDA	288
8.28.1.7 NETWORK_RULE	288
8.28.1.8 NETWORK_RULE_LAMBDA	288
8.28.1.9 NETWORKDENSE_COUNTER_LAMBDA	288
8.28.2 Typedef Documentation	289
8.28.2.1 NetCounter	289
8.28.2.2 NetCounters	289
8.28.2.3 NetModel	289
8.28.2.4 NetRule	289
8.28.2.5 NetRules	289
8.28.2.6 NetStatsCounter	290
8.28.2.7 NetSupport	290
8.28.2.8 Network	290
8.28.2.9 NetworkDense	290
8.28.3 Function Documentation	290
8.28.3.1 rules_zerodiag()	290
8.29 include/barry/counters/phylo.hpp File Reference	291
8.29.1 Macro Definition Documentation	293
8.29.1.1 DEFAULT_DUPLICATION	293
8.29.1.2 DUPL_DUPL	293
8.29.1.3 DUPL_EITH	293
8.29.1.4 DUPL_SPEC	293



8.29.1.5 IF_MATCHES . . . . .	294
8.29.1.6 IF_NOTMATCHES . . . . .	294
8.29.1.7 IS_DUPLICATION . . . . .	294
8.29.1.8 IS_EITHER . . . . .	294
8.29.1.9 IS_SPECIATION . . . . .	294
8.29.1.10 MAKE_DUPL_VARS . . . . .	295
8.29.1.11 PHYLO_CHECK_MISSING . . . . .	295
8.29.1.12 PHYLO_COUNTER_LAMBDA . . . . .	295
8.29.1.13 PHYLO_RULE_DYN_LAMBDA . . . . .	295
8.29.2 Typedef Documentation . . . . .	296
8.29.2.1 PhyloArray . . . . .	296
8.29.2.2 PhyloCounter . . . . .	296
8.29.2.3 PhyloCounters . . . . .	296
8.29.2.4 PhyloModel . . . . .	296
8.29.2.5 PhyloPowerSet . . . . .	296
8.29.2.6 PhyloRule . . . . .	297
8.29.2.7 PhyloRuleData . . . . .	297
8.29.2.8 PhyloRuleDyn . . . . .	297
8.29.2.9 PhyloRules . . . . .	297
8.29.2.10 PhyloRulesDyn . . . . .	297
8.29.2.11 PhyloStatsCounter . . . . .	297
8.29.2.12 PhyloSupport . . . . .	298
8.29.3 Function Documentation . . . . .	298
8.29.3.1 get_last_name() . . . . .	298
8.30 include/barry/model-bones.hpp File Reference . . . . .	298
8.30.1 Function Documentation . . . . .	299
8.30.1.1 keygen_default() . . . . .	299
8.31 include/barry/model-meat.hpp File Reference . . . . .	300
8.31.1 Macro Definition Documentation . . . . .	300
8.31.1.1 MODEL_TEMPLATE . . . . .	300
8.31.1.2 MODEL_TEMPLATE_ARGS . . . . .	301
8.31.1.3 MODEL_TYPE . . . . .	301
8.31.2 Function Documentation . . . . .	301
8.31.2.1 likelihood_() . . . . .	301
8.31.2.2 MODEL_TEMPLATE() [1/2] . . . . .	301
8.31.2.3 MODEL_TEMPLATE() [2/2] . . . . .	302
8.31.2.4 update_normalizing_constant() . . . . .	302
8.32 include/barry/models/geese.hpp File Reference . . . . .	302
8.33 include/barry/models/geese/flock-bones.hpp File Reference . . . . .	303
8.34 include/barry/models/geese/flock-meat.hpp File Reference . . . . .	303
8.35 include/barry/models/geese/geese-bones.hpp File Reference . . . . .	304
8.35.1 Macro Definition Documentation . . . . .	304

8.35.1.1 INITIALIZED . . . . .	305
8.35.2 Function Documentation . . . . .	305
8.35.2.1 keygen_full() . . . . .	305
8.35.2.2 RULE_FUNCTION() . . . . .	305
8.35.2.3 vec_diff() . . . . .	305
8.35.2.4 vector_caster() . . . . .	305
8.36 include/barry/models/geese/geese-meat-constructors.hpp File Reference . . . . .	306
8.37 include/barry/models/geese/geese-meat-likelihood.hpp File Reference . . . . .	306
8.38 include/barry/models/geese/geese-meat-likelihood_exhaust.hpp File Reference . . . . .	307
8.39 include/barry/models/geese/geese-meat-predict.hpp File Reference . . . . .	308
8.40 include/barry/models/geese/geese-meat-predict_exhaust.hpp File Reference . . . . .	308
8.41 include/barry/models/geese/geese-meat-predict_sim.hpp File Reference . . . . .	309
8.42 include/barry/models/geese/geese-meat-simulate.hpp File Reference . . . . .	309
8.43 include/barry/models/geese/geese-meat.hpp File Reference . . . . .	310
8.44 include/barry/models/geese/geese-node-bones.hpp File Reference . . . . .	310
8.45 include/barry/powerset-bones.hpp File Reference . . . . .	311
8.46 include/barry/powerset-meat.hpp File Reference . . . . .	312
8.47 include/barry/progress.hpp File Reference . . . . .	313
8.47.1 Macro Definition Documentation . . . . .	313
8.47.1.1 BARRY_PROGRESS_BAR_WIDTH . . . . .	313
8.48 include/barry/rules-bones.hpp File Reference . . . . .	313
8.48.1 Function Documentation . . . . .	314
8.48.1.1 rule_fun_default() . . . . .	314
8.49 include/barry/rules-meat.hpp File Reference . . . . .	315
8.50 include/barry/statscounter-bones.hpp File Reference . . . . .	315
8.51 include/barry/statscounter-meat.hpp File Reference . . . . .	317
8.51.1 Macro Definition Documentation . . . . .	318
8.51.1.1 STATSCOUNTER_TEMPLATE . . . . .	318
8.51.1.2 STATSCOUNTER_TEMPLATE_ARGS . . . . .	318
8.51.1.3 STATSCOUNTER_TYPE . . . . .	319
8.51.2 Function Documentation . . . . .	319
8.51.2.1 clear() . . . . .	319
8.51.2.2 for() . . . . .	319
8.51.2.3 resize() . . . . .	319
8.51.2.4 STATSCOUNTER_TEMPLATE() [1/9] . . . . .	319
8.51.2.5 STATSCOUNTER_TEMPLATE() [2/9] . . . . .	319
8.51.2.6 STATSCOUNTER_TEMPLATE() [3/9] . . . . .	320
8.51.2.7 STATSCOUNTER_TEMPLATE() [4/9] . . . . .	320
8.51.2.8 STATSCOUNTER_TEMPLATE() [5/9] . . . . .	320
8.51.2.9 STATSCOUNTER_TEMPLATE() [6/9] . . . . .	320
8.51.2.10 STATSCOUNTER_TEMPLATE() [7/9] . . . . .	320
8.51.2.11 STATSCOUNTER_TEMPLATE() [8/9] . . . . .	320

8.51.2.12 STATSCOUNTER_TEMPLATE() [9/9]	321
8.51.3 Variable Documentation	321
8.51.3.1 counter	321
8.51.3.2 counter_deleted	321
8.51.3.3 counters	321
8.51.3.4 counters_	321
8.51.3.5 current_stats	322
8.51.3.6 EmptyArray	322
8.51.3.7 f_	322
8.51.3.8 j	322
8.51.3.9 return	322
8.52 include/barry/statsdb.hpp File Reference	323
8.53 include/barry/support-bones.hpp File Reference	323
8.54 include/barry/support-meat.hpp File Reference	325
8.54.1 Macro Definition Documentation	326
8.54.1.1 BARRY_SUPPORT_MEAT_HPP	327
8.54.1.2 SUPPORT_TEMPLATE	327
8.54.1.3 SUPPORT_TEMPLATE_ARGS	327
8.54.1.4 SUPPORT_TYPE	327
8.54.2 Function Documentation	327
8.54.2.1 calc_backend_dense()	328
8.54.2.2 calc_backend_sparse()	328
8.54.2.3 for()	328
8.54.2.4 if() [1/4]	328
8.54.2.5 if() [2/4]	328
8.54.2.6 if() [3/4]	328
8.54.2.7 if() [4/4]	329
8.54.2.8 insert_cell() [1/2]	329
8.54.2.9 insert_cell() [2/2]	329
8.54.2.10 rm_cell()	329
8.54.2.11 SUPPORT_TEMPLATE() [1/17]	329
8.54.2.12 SUPPORT_TEMPLATE() [2/17]	330
8.54.2.13 SUPPORT_TEMPLATE() [3/17]	330
8.54.2.14 SUPPORT_TEMPLATE() [4/17]	330
8.54.2.15 SUPPORT_TEMPLATE() [5/17]	330
8.54.2.16 SUPPORT_TEMPLATE() [6/17]	330
8.54.2.17 SUPPORT_TEMPLATE() [7/17]	331
8.54.2.18 SUPPORT_TEMPLATE() [8/17]	331
8.54.2.19 SUPPORT_TEMPLATE() [9/17]	331
8.54.2.20 SUPPORT_TEMPLATE() [10/17]	331
8.54.2.21 SUPPORT_TEMPLATE() [11/17]	331
8.54.2.22 SUPPORT_TEMPLATE() [12/17]	331

8.54.2.23 SUPPORT_TEMPLATE() [13/17]	332
8.54.2.24 SUPPORT_TEMPLATE() [14/17]	332
8.54.2.25 SUPPORT_TEMPLATE() [15/17]	332
8.54.2.26 SUPPORT_TEMPLATE() [16/17]	332
8.54.2.27 SUPPORT_TEMPLATE() [17/17]	332
8.54.3 Variable Documentation	332
8.54.3.1 array_bank	333
8.54.3.2 change_stats_different	333
8.54.3.3 coord_i	333
8.54.3.4 coord_j	333
8.54.3.5 counters	333
8.54.3.6 counters_	333
8.54.3.7 delete_counters	334
8.54.3.8 delete_rules	334
8.54.3.9 delete_rules_dyn	334
8.54.3.10 else	334
8.54.3.11 f_	334
8.54.3.12 hashes	335
8.54.3.13 return	335
8.54.3.14 rules	335
8.54.3.15 rules_	335
8.54.3.16 rules_dyn	335
8.54.3.17 stats_bank	336
8.54.3.18 tmp_chng	336
8.55 include/barry/typedefs.hpp File Reference	336
8.55.1 Typedef Documentation	338
8.55.1.1 Col_type	338
8.55.1.2 Counter_fun_type	338
8.55.1.3 Counts_type	339
8.55.1.4 MapVec_type	339
8.55.1.5 Row_type	339
8.55.1.6 Rule_fun_type	339
8.55.1.7 uint	339
8.55.2 Function Documentation	339
8.55.2.1 vec_equal()	339
8.55.2.2 vec_equal_approx()	340
8.55.2.3 vec_inner_prod() [1/2]	340
8.55.2.4 vec_inner_prod() [2/2]	340
8.56 README.md File Reference	340
<b>Index</b>	<b>341</b>

# 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.

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

### Examples

#### Counting statistics in a graph

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

```
#include <iostream>
#include <ostream>
#include "../include/barry.hpp"
typedef std::vector< unsigned int > vuint;
int main() {
```

```

// Creating network of size six with five ties
netcounters::Network net(
    6, 6,
    {0, 0, 4, 4, 2, 0, 1},
    {1, 2, 0, 2, 4, 0, 1}
);

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

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

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

// Initializing the data. The program deals with 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 . . . . .	19
Phylo rules . . . . .	25



## 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	31
<a href="#">BArrayCell&lt; Cell_Type, Data_Type &gt;</a>	45
<a href="#">BArrayCell_const&lt; Cell_Type, Data_Type &gt;</a>	48
<a href="#">BArrayDense&lt; Cell_Type, Data_Type &gt;</a>	
Baseline class for binary arrays	50
<a href="#">BArrayDenseCell&lt; Cell_Type, Data_Type &gt;</a>	65
<a href="#">BArrayDenseCell_const&lt; Cell_Type, Data_Type &gt;</a>	69
<a href="#">BArrayDenseCol&lt; Cell_Type, Data_Type &gt;</a>	69
<a href="#">BArrayDenseCol_const&lt; Cell_Type, Data_Type &gt;</a>	71
<a href="#">BArrayDenseRow&lt; Cell_Type, Data_Type &gt;</a>	73
<a href="#">BArrayDenseRow_const&lt; Cell_Type, Data_Type &gt;</a>	76
<a href="#">BArrayRow&lt; Cell_Type, Data_Type &gt;</a>	78
<a href="#">BArrayRow_const&lt; Cell_Type, Data_Type &gt;</a>	80
<a href="#">BArrayVector&lt; Cell_Type, Data_Type &gt;</a>	
Row or column of a <a href="#">BArray</a>	82
<a href="#">BArrayVector_const&lt; Cell_Type, Data_Type &gt;</a>	86
<a href="#">Cell&lt; Cell_Type &gt;</a>	
Entries in <a href="#">BArray</a> . For now, it only has two members:	90
<a href="#">Cell_const&lt; Cell_Type &gt;</a>	95
<a href="#">ConstBArrayRowIter&lt; Cell_Type, Data_Type &gt;</a>	95
<a href="#">Counter&lt; Array_Type, Data_Type &gt;</a>	
A counter function based on change statistics	97
<a href="#">Counters&lt; Array_Type, Data_Type &gt;</a>	
Vector of counters	102
<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	107
<a href="#">Flock</a>	
A <a href="#">Flock</a> is a group of <a href="#">Geese</a>	109
<a href="#">FreqTable&lt; T &gt;</a>	
Database of statistics	116
<a href="#">Geese</a>	
Annotated Phylo <a href="#">Model</a>	119
<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:	131

<a href="#">NetCounterData</a>	
Data class used to store arbitrary uint or double vectors . . . . .	145
<a href="#">NetworkData</a>	
Data class for Networks . . . . .	147
<a href="#">Node</a>	
A single node for the model . . . . .	149
<a href="#">NodeData</a>	
Data definition for the <code>PhyloArray</code> class . . . . .	155
<a href="#">PhyloCounterData</a> . . . . .	156
<a href="#">PhyloRuleDynData</a> . . . . .	159
<a href="#">PowerSet&lt; Array_Type, Data_Rule_Type &gt;</a>	
Powerset of a binary array . . . . .	161
<a href="#">Progress</a>	
A simple progress bar . . . . .	168
<a href="#">Rule&lt; Array_Type, Data_Type &gt;</a>	
Rule for determining if a cell should be included in a sequence . . . . .	169
<a href="#">Rules&lt; Array_Type, Data_Type &gt;</a>	
Vector of objects of class <a href="#">Rule</a> . . . . .	171
<a href="#">StatsCounter&lt; Array_Type, Data_Type &gt;</a>	
Count stats for a single Array . . . . .	175
<a href="#">Support&lt; Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type &gt;</a>	
Compute the support of sufficient statistics . . . . .	179
<a href="#">vecHasher&lt; T &gt;</a> . . . . .	189

## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

include/barry/barray-bones.hpp . . . . .	191
include/barry/barray-iterator.hpp . . . . .	192
include/barry/barray-meat-operators.hpp . . . . .	193
include/barry/barray-meat.hpp . . . . .	198
include/barry/barraycell-bones.hpp . . . . .	216
include/barry/barraycell-meat.hpp . . . . .	216
include/barry/barraydense-bones.hpp . . . . .	217
include/barry/barraydense-meat-operators.hpp . . . . .	219
include/barry/barraydense-meat.hpp . . . . .	223
include/barry/barraydensecell-bones.hpp . . . . .	244
include/barry/barraydensecell-meat.hpp . . . . .	245
include/barry/barraydensecol-bones.hpp . . . . .	246
include/barry/barraydenserow-bones.hpp . . . . .	247
include/barry/barrayrow-bones.hpp . . . . .	249
include/barry/barrayrow-meat.hpp . . . . .	249
include/barry/barrayvector-bones.hpp . . . . .	252
include/barry/barrayvector-meat.hpp . . . . .	253
include/barry/barray-configuration.hpp . . . . .	254
include/barry/barray-debug.hpp . . . . .	256
include/barry/barray-macros.hpp . . . . .	256
include/barry/barray.hpp . . . . .	258
include/barry/cell-bones.hpp . . . . .	261
include/barry/cell-meat.hpp . . . . .	261
include/barry/col-bones.hpp . . . . .	262
include/barry/counters-bones.hpp . . . . .	262
include/barry/counters-meat.hpp . . . . .	264
include/barry/model-bones.hpp . . . . .	298
include/barry/model-meat.hpp . . . . .	300
include/barry/powerset-bones.hpp . . . . .	311
include/barry/powerset-meat.hpp . . . . .	312
include/barry/progress.hpp . . . . .	313
include/barry/rules-bones.hpp . . . . .	313
include/barry/rules-meat.hpp . . . . .	315
include/barry/statscounter-bones.hpp . . . . .	315
include/barry/statscounter-meat.hpp . . . . .	317

include/barry/statsdb.hpp . . . . .	323
include/barry/support-bones.hpp . . . . .	323
include/barry/support-meat.hpp . . . . .	325
include/barry/typedefs.hpp . . . . .	336
include/barry/counters/network-css.hpp . . . . .	276
include/barry/counters/network.hpp . . . . .	284
include/barry/counters/phylo.hpp . . . . .	291
include/barry/models/geese.hpp . . . . .	302
include/barry/models/geese/flock-bones.hpp . . . . .	303
include/barry/models/geese/flock-meat.hpp . . . . .	303
include/barry/models/geese/geese-bones.hpp . . . . .	304
include/barry/models/geese/geese-meat-constructors.hpp . . . . .	306
include/barry/models/geese/geese-meat-likelihood.hpp . . . . .	306
include/barry/models/geese/geese-meat-likelihood_exhaust.hpp . . . . .	307
include/barry/models/geese/geese-meat-predict.hpp . . . . .	308
include/barry/models/geese/geese-meat-predict_exhaust.hpp . . . . .	308
include/barry/models/geese/geese-meat-predict_sim.hpp . . . . .	309
include/barry/models/geese/geese-meat-simulate.hpp . . . . .	309
include/barry/models/geese/geese-meat.hpp . . . . .	310
include/barry/models/geese/geese-node-bones.hpp . . . . .	310

## 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<> void counter\_isolates (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter\_mutual (NetCounters< Tnet > *counters)`  
*Number of mutual ties.*
- `template<typename Tnet = Network>`  
`void counter\_istar2 (NetCounters< Tnet > *counters)`
- `template<> void counter\_istar2 (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter\_ostar2 (NetCounters< Tnet > *counters)`
- `template<> void counter\_ostar2 (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter\_ttriads (NetCounters< Tnet > *counters)`
- `template<> void counter\_ttriads (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter\_ctriads (NetCounters< Tnet > *counters)`
- `template<> void counter\_ctriads (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter\_density (NetCounters< Tnet > *counters)`
- `template<typename Tnet = Network>`  
`void counter\_idegree15 (NetCounters< Tnet > *counters)`
- `template<> void counter\_idegree15 (NetCounters< NetworkDense > *counters)`
- `template<typename Tnet = Network>`  
`void counter\_odegree15 (NetCounters< Tnet > *counters)`
- `template<> void counter\_odegree15 (NetCounters< NetworkDense > *counters)`



- `template<typename Tnet = Network>`  
`void counter_absdiff (NetCounters< Tnet > *counters, uint attr_id, double alpha=1.0)`  
*Sum of absolute attribute difference between ego and alter.*
- `template<typename Tnet = Network>`  
`void counter_diff (NetCounters< Tnet > *counters, uint attr_id, double alpha=1.0, double tail_head=true)`  
*Sum of attribute difference between ego and alter to pow(alpha)*
- `NETWORK_COUNTER` (init\_single\_attr)
- `template<typename Tnet = Network>`  
`void counter_nodeicov (NetCounters< Tnet > *counters, uint attr_id)`
- `template<typename Tnet = Network>`  
`void counter_nodeocov (NetCounters< Tnet > *counters, uint attr_id)`
- `template<typename Tnet = Network>`  
`void counter_nodecov (NetCounters< Tnet > *counters, uint attr_id)`
- `template<typename Tnet = Network>`  
`void counter_nodematch (NetCounters< Tnet > *counters, uint attr_id)`
- `template<typename Tnet = Network>`  
`void counter_idegree (NetCounters< Tnet > *counters, std::vector< uint > d)`  
*Counts number of vertices with a given in-degree.*
- `template<> void counter_idegree (NetCounters< NetworkDense > *counters, std::vector< uint > d)`
- `template<typename Tnet = Network>`  
`void counter_odegree (NetCounters< Tnet > *counters, std::vector< uint > d)`  
*Counts number of vertices with a given out-degree.*
- `template<> void counter_odegree (NetCounters< NetworkDense > *counters, std::vector< uint > d)`
- `template<typename Tnet = Network>`  
`void counter_degree (NetCounters< Tnet > *counters, std::vector< uint > d)`  
*Counts number of vertices with a given out-degree.*

### 5.3.1 Detailed Description

`Counters` for network models.

#### Parameters

<code>counters</code>	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 852 of file network.hpp.

### 5.3.2.2 counter\_ctriads() [1/2]

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

Definition at line 631 of file network.hpp.

### 5.3.2.3 counter\_ctriads() [2/2]

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

Definition at line 578 of file network.hpp.

### 5.3.2.4 counter\_degree()

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

Counts number of vertices with a given out-degree.

Definition at line 1269 of file network.hpp.

### 5.3.2.5 counter\_density()

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

Definition at line 695 of file network.hpp.

### 5.3.2.6 counter\_diff()

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

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

Definition at line 897 of file network.hpp.

### 5.3.2.7 counter\_edges()

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

Number of edges.

Definition at line 152 of file network.hpp.

### 5.3.2.8 counter\_iddegree() [1/2]

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

Definition at line 1113 of file network.hpp.

### 5.3.2.9 counter\_iddegree() [2/2]

```
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 1066 of file network.hpp.

### 5.3.2.10 counter\_iddegree15() [1/2]

```
template<>
void counter_iddegree15 (
    NetCounters< NetworkDense > * counters ) [inline]
```

Definition at line 747 of file network.hpp.

### 5.3.2.11 counter\_iddegree15() [2/2]

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

Definition at line 721 of file network.hpp.

#### 5.3.2.12 counter\_isolates() [1/2]

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

Definition at line 211 of file network.hpp.

#### 5.3.2.13 counter\_isolates() [2/2]

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

Number of isolated vertices.

Definition at line 174 of file network.hpp.

#### 5.3.2.14 counter\_istar2() [1/2]

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

Definition at line 325 of file network.hpp.

#### 5.3.2.15 counter\_istar2() [2/2]

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

Definition at line 304 of file network.hpp.

#### 5.3.2.16 counter\_mutual()

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

Number of mutual ties.

Definition at line 250 of file network.hpp.

#### 5.3.2.17 counter\_nodecov()

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

Definition at line 1010 of file network.hpp.

#### 5.3.2.18 counter\_nodeicov()

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

Definition at line 960 of file network.hpp.

#### 5.3.2.19 counter\_nodematch()

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

Definition at line 1035 of file network.hpp.

#### 5.3.2.20 counter\_nodeocov()

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

Definition at line 985 of file network.hpp.

#### 5.3.2.21 counter\_odegree() [1/2]

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

Definition at line 1214 of file network.hpp.

#### 5.3.2.22 counter\_odegree() [2/2]

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

Counts number of vertices with a given out-degree.

Definition at line 1166 of file network.hpp.

#### 5.3.2.23 counter\_odegree15() [1/2]

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

Definition at line 811 of file network.hpp.

#### 5.3.2.24 counter\_odegree15() [2/2]

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

Definition at line 785 of file network.hpp.

#### 5.3.2.25 counter\_ostar2() [1/2]

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

Definition at line 381 of file network.hpp.

#### 5.3.2.26 counter\_ostar2() [2/2]

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

Definition at line 358 of file network.hpp.

**5.3.2.27 counter\_ttriads() [1/2]**

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

Definition at line 501 of file network.hpp.

**5.3.2.28 counter\_ttriads() [2/2]**

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

Definition at line 413 of file network.hpp.

**5.3.2.29 NETWORK\_COUNTER()**

```
NETWORK_COUNTER (
    init_single_attr )
```

Definition at line 941 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.*
- void `counter_less_than_p_prop_genes_changing` (`PhyloCounters *counters`, double p, unsigned int duplication=`DEFAULT_DUPLICATION`)  
*Indicator function. Equals to one if k genes changed and zero otherwise.*
- void `counter_gains_from_0` (`PhyloCounters *counters`, `std::vector< uint > nfun`, unsigned int duplication=`DEFAULT_DUPLICATION`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `counter_pairwise_preserving` (`PhyloCounters *counters`, `uint nfunA`, `uint nfunB`, unsigned int duplication=`DEFAULT_DUPLICATION`)  
*Used when all the functions are in 0 (like the root node prob.)*

### 5.4.1 Detailed Description

`Counters` for phylogenetic modeling.

Parameters

<code>counters</code>	A pointer to a <code>PhyloCounters</code> object ( <code>Counters&lt;PhyloArray, PhyloCounterData&gt;</code> ).
-----------------------	---

### 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 1245 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 817 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 194 of file phylo.hpp.

#### 5.4.2.4 counter\_gains\_from\_0()

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

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

Needs to specify function a.

Definition at line 1582 of file phylo.hpp.

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

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

k genes gain function nfun

Definition at line 255 of file phylo.hpp.

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

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

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

Definition at line 328 of file phylo.hpp.

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

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

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

Definition at line 1344 of file phylo.hpp.

#### 5.4.2.8 counter\_less\_than\_p\_prop\_genes\_changing()

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

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

< How many genes diverge the parent

Definition at line 1465 of file phylo.hpp.

#### 5.4.2.9 counter\_longest()

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

Longest branch mutates (either by gain or by loss)

Definition at line 875 of file phylo.hpp.

#### 5.4.2.10 counter\_loss()

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

Total count of losses for an specific function.

Definition at line 617 of file phylo.hpp.

#### 5.4.2.11 counter\_maxfuns()

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

Cap the number of functions per gene.

Definition at line 546 of file phylo.hpp.

#### 5.4.2.12 counter\_neofun()

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

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 1046 of file phylo.hpp.

#### 5.4.2.13 counter\_neofun\_a2b()

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

Total number of neofunctionalization events.

Needs to specify pairs of function.

Definition at line 1126 of file phylo.hpp.

#### 5.4.2.14 counter\_overall\_changes()

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

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

Definition at line 670 of file phylo.hpp.

#### 5.4.2.15 counter\_overall\_gains()

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

Overall functional gains.

Total number of gains (irrespective of the function).

Definition at line 155 of file phylo.hpp.

#### 5.4.2.16 counter\_overall\_loss()

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

Overall functional loss.

Definition at line 503 of file phylo.hpp.

**5.4.2.17 counter\_pairwise\_preserving()**

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

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

Needs to specify function a.

Definition at line 1649 of file phylo.hpp.

**5.4.2.18 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 400 of file phylo.hpp.

**5.4.2.19 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 730 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 <code>PhyloRules</code> object ( <code>Rules&lt;PhyloArray, PhyloRuleData&gt;</code> ).
--------------	--

## 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>	<code>Support</code> 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 1813 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
- bool `is_dense` () const noexcept

#### Constructors

##### Parameters

N_	Number of rows
M_	Number of columns
source	An unsigned vector ranging from 0 to N_
target	An unsigned int vector ranging from 0 to M_
target	When <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 Cell_Type operator() (uint i, uint j, bool check_bounds=true) const`
- `void rm_cell (uint i, uint j, bool check_bounds=true, bool check_exists=true)`
- `void insert_cell (uint i, uint j, const Cell< Cell_Type > &v, bool check_bounds, bool check_exists)`
- `void insert_cell (uint i, uint j, Cell< Cell_Type > &&v, bool check_bounds, bool check_exists)`
- `void insert_cell (uint i, uint j, Cell_Type v, bool check_bounds, bool check_exists)`
- `void swap_cells (uint i0, uint j0, uint i1, uint j1, bool check_bounds=true, int check_exists=CHECK::BOTH, int *report=nullptr)`
- `void toggle_cell (uint i, uint j, bool check_bounds=true, int check_exists=EXISTS::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 63 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 66 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\_dense()**

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

Definition at line 232 of file `barray-bones.hpp`.

**7.1.3.17 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
```

**7.1.3.18 ncol()**

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

**7.1.3.19 nnozero()**

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

**7.1.3.20 nrow()**

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

**7.1.3.21 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 )
```

**7.1.3.22 operator>() [2/2]**

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

**7.1.3.23 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 )
```

**7.1.3.24 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 )
```

**7.1.3.25 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 )
```

**7.1.3.26 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 )
```

**7.1.3.27 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 )
```

**7.1.3.28 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 )
```

**7.1.3.29 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 )
```

**7.1.3.30 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 )
```

**7.1.3.31 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.

**7.1.3.32 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.

**7.1.3.33 operator==( )**

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

**7.1.3.34 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
```

**7.1.3.35 print( )**

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

**7.1.3.36 reserve()**

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

**7.1.3.37 resize()**

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

**7.1.3.38 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 )
```

**7.1.3.39 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
```

**7.1.3.40 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>	

#### 7.1.3.41 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 )
```

#### 7.1.3.42 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 )
```

#### 7.1.3.43 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 )
```

#### 7.1.3.44 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 )
```

#### 7.1.3.45 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 )
```

#### 7.1.3.46 transpose()

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

#### 7.1.3.47 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 )
```

#### 7.1.3.48 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 48 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`
- `BArrayDenseRow< Cell_Type, Data_Type > & row (uint i, bool check_bounds=true)`
- `const BArrayDenseRow_const< Cell_Type, Data_Type > row (uint i, bool check_bounds=true) const`
- `BArrayDenseCol< Cell_Type, Data_Type > & col (uint j, bool check_bounds=true)`
- `const BArrayDenseCol_const< Cell_Type, Data_Type > col (uint j, bool check_bounds=true) const`
- `Entries< Cell_Type > get_entries () const`  
*Get the edgelist.*
- `void transpose ()`
- `void clear (bool hard=true)`
- `void resize (uint N_, uint M_)`
- `void reserve ()`
- `void print (const char *fmt=nullptr,...) const`
- `bool is_dense () const noexcept`
- `const std::vector< Cell_Type > & get_data () const`
- `const Cell_Type rowsum (unsigned int i) const`
- `const Cell_Type colsum (unsigned int i) const`

### Constructors

*Parameters*

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

## 7.4.1 Detailed Description

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

Baseline class for binary arrays.

`BArrayDense` class objects are arbitrary 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 34 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 79 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 82 of file `barraydense-bones.hpp`.

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

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

Edgelist with data.

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

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

Edgelist with no data (simpler)

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

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

Copy constructor.

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

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

Move operator.

**7.4.2.7 ~BArrayDense()**

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

**7.4.3 Member Function Documentation****7.4.3.1 clear()**

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

**7.4.3.2 col()** [1/2]

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

Definition at line 482 of file barraydense-meat.hpp.

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

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

Definition at line 468 of file barraydense-meat.hpp.

**7.4.3.4 colsum()**

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

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

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

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

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

**7.4.3.7 default\_val()**

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

#### 7.4.3.8 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
```

#### 7.4.3.9 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
```

#### 7.4.3.10 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
```

#### 7.4.3.11 get\_data()

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

#### 7.4.3.12 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>`

**7.4.3.13 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
```

**7.4.3.14 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
```

**7.4.3.15 insert\_cell() [1/2]**

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

**7.4.3.16 insert\_cell() [2/2]**

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

**7.4.3.17 is\_dense()**

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

Definition at line 254 of file `barraydense-bones.hpp`.

**7.4.3.18 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
```

**7.4.3.19 ncol()**

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

**7.4.3.20 nnozero()**

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

**7.4.3.21 nrow()**

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

**7.4.3.22 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 )
```

**7.4.3.23 operator>() [2/2]**

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

**7.4.3.24 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 )
```

**7.4.3.25 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-=( ) [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 )
```

**7.4.3.29 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 )
```

**7.4.3.30 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 )
```

**7.4.3.31 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 )
```

**7.4.3.32 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.

**7.4.3.33 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.

**7.4.3.34 operator==( )**

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

**7.4.3.35 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
```



**7.4.3.36 print()**

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

**7.4.3.37 reserve()**

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

**7.4.3.38 resize()**

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

**7.4.3.39 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 )
```

**7.4.3.40 row() [1/2]**

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

**7.4.3.41 row() [2/2]**

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

#### 7.4.3.42 rowsum()

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

#### 7.4.3.43 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>	

#### 7.4.3.44 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 )
```

#### 7.4.3.45 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 )
```

#### 7.4.3.46 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 )
```

#### 7.4.3.47 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 )
```

#### 7.4.3.48 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 )
```

#### 7.4.3.49 transpose()

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

#### 7.4.3.50 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 )
```

#### 7.4.3.51 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 bararraydense-bones.hpp.

#### 7.4.4.2 BArrayDenseCol< Cell\_Type, Data\_Type >

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

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

#### 7.4.4.3 BArrayDenseCol\_const< Cell\_Type, Data\_Type >

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

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

#### 7.4.4.4 BArrayDenseRow< Cell\_Type, Data\_Type >

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

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

#### 7.4.4.5 BArrayDenseRow\_const< Cell\_Type, Data\_Type >

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

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

### 7.4.5 Member Data Documentation

#### 7.4.5.1 visited

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

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

Definition at line 64 of file `barraydense-bones.hpp`.

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

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

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

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

### Public Member Functions

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

### Friends

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

### 7.5.1 Detailed Description

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

Definition at line 15 of file `barraydensecell-bones.hpp`.

### 7.5.2 Constructor & Destructor Documentation

#### 7.5.2.1 BArrayDenseCell()

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

Definition at line 27 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 49 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 56 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 37 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 19 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 28 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 46 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 61 of file `barraydensecell-meat.hpp`.

## 7.5.4 Friends And Related Function Documentation

### 7.5.4.1 BArrayDense< Cell\_Type, Data\_Type >

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

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

### 7.5.4.2 BArrayDenseCol< Cell\_Type, Data\_Type >

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

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

### 7.5.4.3 BArrayDenseCol\_const< Cell\_Type, Data\_Type >

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

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

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

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



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

### 7.6.1 Detailed Description

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

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

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

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

## 7.7 BArrayDenseCol< Cell\_Type, Data\_Type > Class Template Reference

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

### Public Member Functions

- [BArrayDenseCol](#) ([BArrayDense](#)< Cell\_Type, Data\_Type > &array\_, unsigned int j)
- [Col\\_type](#)< Cell\_Type >::iterator & [begin](#) ()
- [Col\\_type](#)< Cell\_Type >::iterator & [end](#) ()
- [size\\_t](#) [size](#) () [const](#) [noexcept](#)
- [std::pair](#)< unsigned int, Cell\_Type \* > & [operator\(\)](#) (unsigned int i)

### Friends

- class [BArrayDense](#)< Cell\_Type, Data\_Type >
- class [BArrayDenseCell](#)< Cell\_Type, Data\_Type >
- class [BArrayDenseCell\\_const](#)< Cell\_Type, Data\_Type >

### 7.7.1 Detailed Description

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

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

### 7.7.2 Constructor & Destructor Documentation

### 7.7.2.1 BArrayDenseCol()

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

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

## 7.7.3 Member Function Documentation

### 7.7.3.1 begin()

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

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

### 7.7.3.2 end()

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

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

### 7.7.3.3 operator()()

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

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

### 7.7.3.4 size()

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

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

## 7.7.4 Friends And Related Function Documentation

### 7.7.4.1 BArrayDense< Cell\_Type, Data\_Type >

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

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

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

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

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

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

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

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

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

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

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

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

### Public Member Functions

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

## Friends

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

### 7.8.1 Detailed Description

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

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

### 7.8.2 Constructor & Destructor Documentation

#### 7.8.2.1 BArrayDenseCol\_const()

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

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

### 7.8.3 Member Function Documentation

#### 7.8.3.1 begin()

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

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

#### 7.8.3.2 end()

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

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

### 7.8.3.3 operator()

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

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

### 7.8.3.4 size()

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

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

## 7.8.4 Friends And Related Function Documentation

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

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

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

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

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

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

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

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

## 7.9 BArrayDenseRow< Cell\_Type, Data\_Type > Class Template Reference

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

## Public Member Functions

- [BArrayDenseRow](#) ([BArrayDense](#)< Cell\_Type, Data\_Type > &array\_, unsigned int i)
- [Row\\_type](#)< Cell\_Type >::iterator & [begin](#) ()
- [Row\\_type](#)< Cell\_Type >::iterator & [end](#) ()
- [size\\_t](#) [size](#) () [const noexcept](#)
- [std::pair](#)< unsigned int, [Cell](#)< Cell\_Type > > & [operator\(\)](#) (unsigned int i)

## Friends

- class [BArrayDense](#)< Cell\_Type, Data\_Type >
- class [BArrayDenseCell](#)< Cell\_Type, Data\_Type >
- class [BArrayDenseCell\\_const](#)< Cell\_Type, Data\_Type >

### 7.9.1 Detailed Description

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

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

### 7.9.2 Constructor & Destructor Documentation

#### 7.9.2.1 BArrayDenseRow()

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

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

### 7.9.3 Member Function Documentation

#### 7.9.3.1 begin()

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

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

### 7.9.3.2 end()

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

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

### 7.9.3.3 operator()()

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

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

### 7.9.3.4 size()

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

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

## 7.9.4 Friends And Related Function Documentation

### 7.9.4.1 BArrayDense< Cell\_Type, Data\_Type >

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

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

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

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

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

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

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

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

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

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

## 7.10 BArrayDenseRow\_const< Cell\_Type, Data\_Type > Class Template Reference

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

### Public Member Functions

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

### Friends

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

### 7.10.1 Detailed Description

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

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

### 7.10.2 Constructor & Destructor Documentation



### 7.10.2.1 BArrayDenseRow\_const()

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

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

## 7.10.3 Member Function Documentation

### 7.10.3.1 begin()

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

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

### 7.10.3.2 end()

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

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

### 7.10.3.3 operator()()

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

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

### 7.10.3.4 size()

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

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

## 7.10.4 Friends And Related Function Documentation

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

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

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

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

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

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

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

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

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

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

### Public Member Functions

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

### 7.11.1 Detailed Description

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

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

## 7.11.2 Constructor & Destructor Documentation

### 7.11.2.1 BArrayRow()

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

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

### 7.11.2.2 ~BArrayRow()

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

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

## 7.11.3 Member Function Documentation

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

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

### 7.11.3.2 operator\*=( )

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

### 7.11.3.3 operator+=( )

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

#### 7.11.3.4 operator-=( )

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

#### 7.11.3.5 operator/=( )

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

#### 7.11.3.6 operator=( )

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

#### 7.11.3.7 operator==( )

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

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

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

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

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

### Public Member Functions

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

### 7.12.1 Detailed Description

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

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

### 7.12.2 Constructor & Destructor Documentation

#### 7.12.2.1 BArrayRow\_const()

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

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

#### 7.12.2.2 ~BArrayRow\_const()

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

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

### 7.12.3 Member Function Documentation

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

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

#### 7.12.3.2 operator"!="()

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

### 7.12.3.3 operator<()

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

### 7.12.3.4 operator<=()

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

### 7.12.3.5 operator==()

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

### 7.12.3.6 operator>()

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

### 7.12.3.7 operator>=()

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

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

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

## 7.13 BArrayVector< Cell\_Type, Data\_Type > Class Template Reference

Row or column of a [BArray](#)

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

## Public Member Functions

- [BArrayVector](#) ([BArray](#)< [Cell\\_Type](#), [Data\\_Type](#) > \*[Array\\_](#), [uint](#) &[dim\\_](#) [uint](#) &[i\\_](#), [bool](#) [check\\_bounds](#)=true)  
Construct a new [BArrayVector](#) object.
- [~BArrayVector](#) ()
- [bool](#) [is\\_row](#) () [const](#) [noexcept](#)
- [bool](#) [is\\_col](#) () [const](#) [noexcept](#)
- [uint](#) [size](#) () [const](#) [noexcept](#)
- [std::vector](#)< [Cell\\_Type](#) >::[const\\_iterator](#) [begin](#) () [noexcept](#)
- [std::vector](#)< [Cell\\_Type](#) >::[const\\_iterator](#) [end](#) () [noexcept](#)
- [void](#) [operator=](#) ([const](#) [Cell\\_Type](#) &[val](#))
- [void](#) [operator+=](#) ([const](#) [Cell\\_Type](#) &[val](#))
- [void](#) [operator-=](#) ([const](#) [Cell\\_Type](#) &[val](#))
- [void](#) [operator\\*=](#) ([const](#) [Cell\\_Type](#) &[val](#))
- [void](#) [operator/=](#) ([const](#) [Cell\\_Type](#) &[val](#))
- [operator](#) [std::vector](#)< [Cell\\_Type](#) > () [const](#)
- [bool](#) [operator==](#) ([const](#) [Cell\\_Type](#) &[val](#)) [const](#)

### 7.13.1 Detailed Description

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

Row or column of a [BArray](#)

Template Parameters

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

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

### 7.13.2 Constructor & Destructor Documentation

#### 7.13.2.1 BArrayVector()

```
template<typename Cell_Type = bool, typename Data_Type = bool>
BArrayVector< Cell_Type, Data_Type >::BArrayVector (
    BArray< Cell_Type, Data_Type > * Array\_,
    uint &dim\_ uint & i\_,
    bool check\_bounds = true ) [inline]
```

Construct a new [BArrayVector](#) object.

Parameters

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

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

#### 7.13.2.2 `~BArrayVector()`

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

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

### 7.13.3 Member Function Documentation

#### 7.13.3.1 `begin()`

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

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

#### 7.13.3.2 `end()`

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

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

#### 7.13.3.3 `is_col()`

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

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



#### 7.13.3.4 is\_row()

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

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

#### 7.13.3.5 operator std::vector< Cell\_Type >()

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

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

#### 7.13.3.6 operator\*=( )

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

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

#### 7.13.3.7 operator+=( )

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

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

#### 7.13.3.8 operator-=( )

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

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

#### 7.13.3.9 operator/=( )

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

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

#### 7.13.3.10 operator=( )

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

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

#### 7.13.3.11 operator==( )

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

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

#### 7.13.3.12 size( )

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

Definition at line 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](#)

## 7.14 BArrayVector\_const< Cell\_Type, Data\_Type > Class Template Reference

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

## Public Member Functions

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

### 7.14.1 Detailed Description

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

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

### 7.14.2 Constructor & Destructor Documentation

#### 7.14.2.1 BArrayVector\_const()

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

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

#### 7.14.2.2 ~BArrayVector\_const()

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

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

### 7.14.3 Member Function Documentation

#### 7.14.3.1 begin()

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

#### 7.14.3.2 end()

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

#### 7.14.3.3 is\_col()

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

#### 7.14.3.4 is\_row()

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

#### 7.14.3.5 operator std::vector< Cell\_Type >()

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

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

#### 7.14.3.6 operator"!="()

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

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

#### 7.14.3.7 operator<()

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

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

#### 7.14.3.8 operator<=()

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

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

#### 7.14.3.9 operator==()

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

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

#### 7.14.3.10 operator>()

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

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

### 7.14.3.11 operator>=()

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

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

### 7.14.3.12 size()

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

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

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

## 7.15 Cell< Cell\_Type > Class Template Reference

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

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

### Public Member Functions

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

### Public Attributes

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

### 7.15.1 Detailed Description

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

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

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

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

### 7.15.2 Constructor & Destructor Documentation

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

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

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

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

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

#### 7.15.2.3 ~Cell()

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**7.15.3 Member Function Documentation**



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

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

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

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

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

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

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

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

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

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

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

**7.15.3.5 operator Cell\_Type()**

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

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

**7.15.3.6 operator"!=( )**

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

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

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

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

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

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

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

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

#### 7.15.3.9 operator==()

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

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

### 7.15.4 Member Data Documentation

#### 7.15.4.1 active

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

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

#### 7.15.4.2 value

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

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

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

## 7.16 Cell\_const< Cell\_Type > Class Template Reference

### 7.16.1 Detailed Description

```
template<typename Cell_Type>
class Cell_const< Cell_Type >
```

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

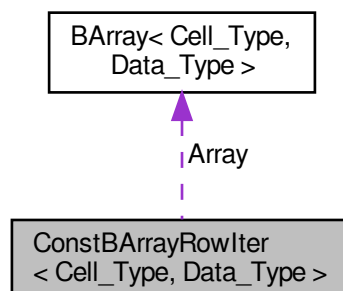
The documentation for this class was generated from the following file:

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

## 7.17 ConstBArrayRowIter< Cell\_Type, Data\_Type > Class Template Reference

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

Collaboration diagram for ConstBArrayRowIter< Cell\_Type, Data\_Type >:



## Public Member Functions

- [ConstBArrayRowIter](#) ([const BArray](#)< Cell\_Type, Data\_Type > \*[Array\\_](#))
- [~ConstBArrayRowIter](#) ()

## Public Attributes

- [uint](#) [current\\_row](#)
- [uint](#) [current\\_col](#)
- [Row\\_type](#)< Cell\_Type >::const\_iterator [iter](#)
- [const BArray](#)< Cell\_Type, Data\_Type > \* [Array](#)

### 7.17.1 Detailed Description

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

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

### 7.17.2 Constructor & Destructor Documentation

#### 7.17.2.1 ConstBArrayRowIter()

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

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

#### 7.17.2.2 ~ConstBArrayRowIter()

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

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

### 7.17.3 Member Data Documentation

### 7.17.3.1 Array

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

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

### 7.17.3.2 current\_col

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

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

### 7.17.3.3 current\_row

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

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

### 7.17.3.4 iter

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

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

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

- `include/barry/barray-iterator.hpp`

## 7.18 Counter< Array\_Type, Data\_Type > Class Template Reference

A counter function based on change statistics.

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

### Public Member Functions

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

**Creator passing a counter and an initializer**

*Parameters*

count_fun↔ _	<i>The main counter function.</i>
init_fun_ _	<i>The initializer function can also be used to check if the <a href="#">BArray</a> as the needed variables (see <a href="#">BArray::data</a>).</i>
data_ _	<i>Data to be used with the counter.</i>
delete_↔ data_ _	<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](#) = ""

**7.18.1 Detailed Description**

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

A counter function based on change statistics.

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

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

**7.18.2 Constructor & Destructor Documentation**

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

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

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

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

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
Counter< Array_Type, Data_Type >::Counter (
    Counter_fun_type< Array_Type, Data_Type > count_fun_,
    Counter_fun_type< Array_Type, Data_Type > init_fun_ = 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.

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

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

Copy constructor.

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

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

Move constructor.

**7.18.2.5 ~Counter()**

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

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

## 7.18.3 Member Function Documentation

### 7.18.3.1 count()

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

### 7.18.3.2 get\_description()

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

### 7.18.3.3 get\_name()

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

### 7.18.3.4 init()

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

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

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

Copy assignment.



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

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

Move assignment.

## 7.18.4 Member Data Documentation

### 7.18.4.1 count\_fun

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

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

### 7.18.4.2 data

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

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

### 7.18.4.3 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.

### 7.18.4.4 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.

#### 7.18.4.5 init\_fun

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

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

#### 7.18.4.6 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

## 7.19 Counters< Array\_Type, Data\_Type > Class Template Reference

Vector of counters.

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

### Public Member Functions

- [Counters](#) ()
- [~Counters](#) ()
- [Counters](#) (const [Counters](#)< Array\_Type, Data\_Type > &counter\_)  
*Copy constructor.*
- [Counters](#) ([Counters](#)< Array\_Type, Data\_Type > &&counters\_) noexcept  
*Move constructor.*
- [Counters](#)< Array\_Type, Data\_Type > operator= (const [Counters](#)< Array\_Type, Data\_Type > &counter\_)  
*Copy assignment constructor.*
- [Counters](#)< Array\_Type, Data\_Type > & operator= ([Counters](#)< Array\_Type, Data\_Type > &&counter\_) noexcept  
*Move assignment constructor.*
- [Counter](#)< Array\_Type, Data\_Type > & operator[] (uint idx)  
*Returns a pointer to a particular counter.*
- std::size\_t [size](#) () const noexcept  
*Number of counters in the set.*
- void [add\\_counter](#) ([Counter](#)< Array\_Type, Data\_Type > &counter)
- void [add\\_counter](#) ([Counter](#)< 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

### 7.19.1 Detailed Description

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

Vector of counters.

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

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

### 7.19.2 Constructor & Destructor Documentation

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

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

#### 7.19.2.2 ~Counters()

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

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

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

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

Copy constructor.

#### Parameters

<i>counter_</i> ↔	
—	

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

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

Move constructor.

#### Parameters

<i>counters_↔</i>	
—	

## 7.19.3 Member Function Documentation

### 7.19.3.1 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 )
```

### 7.19.3.2 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 )
```

### 7.19.3.3 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_ = "" )
```

**7.19.3.4 clear()**

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

**7.19.3.5 get\_descriptions()**

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

**7.19.3.6 get\_names()**

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

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

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

Copy assignment constructor.

**Parameters**

<i>counter_↔</i>	
_	

**Returns**

Counters<Array\_Type,Data\_Type>

**7.19.3.8 operator=() [2/2]**

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

Move assignment constructor.

## Parameters

<i>counter</i> ↔	
—	

## Returns

Counters<Array\_Type,Data\_Type>&

## 7.19.3.9 operator[]()

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

Returns a pointer to a particular counter.

## Parameters

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

## Returns

Counter<Array\_Type,Data\_Type>\*

## 7.19.3.10 size()

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

Number of counters in the set.

## Returns

uint

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

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

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

## 7.20 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`

### 7.20.1 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 79 of file `typedefs.hpp`.

### 7.20.2 Constructor & Destructor Documentation

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

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

Definition at line 85 of file `typedefs.hpp`.

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

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

Definition at line 86 of file typedefs.hpp.

### 7.20.2.3 ~Entries()

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

Definition at line 93 of file typedefs.hpp.

## 7.20.3 Member Function Documentation

### 7.20.3.1 resize()

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

Definition at line 95 of file typedefs.hpp.

## 7.20.4 Member Data Documentation

### 7.20.4.1 source

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

Definition at line 81 of file typedefs.hpp.

### 7.20.4.2 target

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

Definition at line 82 of file typedefs.hpp.



## 7.20.4.3 val

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

Definition at line 83 of file typedefs.hpp.

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

- include/barry/typedefs.hpp

## 7.21 Flock Class Reference

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

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

### Public Member Functions

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

### Information about the model

- unsigned int [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](#), std::vector< size\_t > \*dist=nullptr) const noexcept  
*Check polytomies and return the largest.*
- void [print](#) () const

## Public Attributes

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

### 7.21.1 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`.

### 7.21.2 Constructor & Destructor Documentation

#### 7.21.2.1 `Flock()`

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

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

#### 7.21.2.2 `~Flock()`

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

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

### 7.21.3 Member Function Documentation

#### 7.21.3.1 `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.

**7.21.3.2 colnames()**

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

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

**7.21.3.3 get\_counters()**

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

Definition at line 100 of file flock-meat.hpp.

**7.21.3.4 get\_model()**

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

Definition at line 131 of file flock-meat.hpp.

**7.21.3.5 get\_stats\_support()**

```
std::vector< std::vector< double > > * Flock::get_stats_support ( ) [inline]
```

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

### 7.21.3.6 get\_stats\_target()

```
std::vector< std::vector< double > > * Flock::get_stats_target ( ) [inline]
```

Definition at line 124 of file flock-meat.hpp.

### 7.21.3.7 get\_support\_fun()

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

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

### 7.21.3.8 init()

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

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

### 7.21.3.9 likelihood\_joint()

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

Returns the joint likelihood of the model.

#### Parameters

<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 138 of file flock-meat.hpp.

#### 7.21.3.10 nfuncs()

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

Definition at line 167 of file flock-meat.hpp.

#### 7.21.3.11 nleafs()

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

Definition at line 195 of file flock-meat.hpp.

#### 7.21.3.12 nnodes()

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

Definition at line 181 of file flock-meat.hpp.

#### 7.21.3.13 nterms()

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

Definition at line 209 of file flock-meat.hpp.

#### 7.21.3.14 ntrees()

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

Definition at line 174 of file flock-meat.hpp.

#### 7.21.3.15 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 302 of file flock-meat.hpp.

**7.21.3.16 parse\_polytomies()**

```
unsigned int Flock::parse_polytomies (
    bool verb = true,
    std::vector< size_t > * dist = nullptr ) const [inline], [noexcept]
```

Check polytomies and return the largest.

Definition at line 231 of file flock-meat.hpp.

**7.21.3.17 print()**

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

Definition at line 258 of file flock-meat.hpp.

**7.21.3.18 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>rengine.seed()</code> member object.
----------	--

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

### 7.21.3.19 support\_size()

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

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

## 7.21.4 Member Data Documentation

### 7.21.4.1 dat

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

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

### 7.21.4.2 initialized

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

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

### 7.21.4.3 model

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

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

### 7.21.4.4 nfunctions

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

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

### 7.21.4.5 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

## 7.22 FreqTable< T > Class Template Reference

Database of statistics.

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

### Public Member Functions

- [FreqTable](#) ()
- [~FreqTable](#) ()
- [size\\_t add](#) ([const](#) std::vector< T > &x, [size\\_t](#) \*h\_precomp)
- [Counts\\_type as\\_vector](#) () [const](#)
- [const](#) std::vector< double > & [get\\_data](#) () [const](#)
- [const](#) std::unordered\_map< [size\\_t](#), [size\\_t](#) > & [get\\_index](#) () [const](#)
- void [clear](#) ()
- void [reserve](#) ([size\\_t](#) n, [size\\_t](#) k)
- void [print](#) () [const](#)
- [size\\_t](#) [size](#) () [const](#) [noexcept](#)
- *Number of unique elements in the table. (.*
- [size\\_t](#) [make\\_hash](#) ([const](#) std::vector< double > &x) [const](#)

### 7.22.1 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.

### 7.22.2 Constructor & Destructor Documentation



### 7.22.2.1 FreqTable()

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

Definition at line 28 of file statsdb.hpp.

### 7.22.2.2 ~FreqTable()

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

Definition at line 29 of file statsdb.hpp.

## 7.22.3 Member Function Documentation

### 7.22.3.1 add()

```
template<typename T >
size_t FreqTable< T >::add (
    const std::vector< T > & x,
    size_t * h_precomp ) [inline]
```

Definition at line 53 of file statsdb.hpp.

### 7.22.3.2 as\_vector()

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

Definition at line 133 of file statsdb.hpp.

### 7.22.3.3 clear()

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

Definition at line 162 of file statsdb.hpp.

#### 7.22.3.4 get\_data()

```
template<typename T = double>
const std::vector< double >& FreqTable< T >::get_data ( ) const [inline]
```

Definition at line 34 of file statsdb.hpp.

#### 7.22.3.5 get\_index()

```
template<typename T = double>
const std::unordered_map<size_t,size_t>& FreqTable< T >::get_index ( ) const [inline]
```

Definition at line 35 of file statsdb.hpp.

#### 7.22.3.6 make\_hash()

```
template<typename T >
size_t FreqTable< T >::make_hash (
    const std::vector< double > & x ) const [inline]
```

Definition at line 233 of file statsdb.hpp.

#### 7.22.3.7 print()

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

Definition at line 198 of file statsdb.hpp.

#### 7.22.3.8 reserve()

```
template<typename T >
void FreqTable< T >::reserve (
    size_t n,
    size_t k ) [inline]
```

Definition at line 176 of file statsdb.hpp.

### 7.22.3.9 size()

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

Number of unique elements in the table. (.

#### Returns

size\_t

Definition at line 225 of file statsdb.hpp.

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

- include/barry/statsdb.hpp

## 7.23 Geese Class Reference

Annotated Phylo [Model](#).

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

### Public Member Functions

- [~Geese](#) ()
- void [init](#) (unsigned int bar\_width=[BARRY\\_PROGRESS\\_BAR\\_WIDTH](#))
- void [inherit\\_support](#) (const [Geese](#) &model\_, bool delete\_support\_=false)
- void [calc\\_sequence](#) ([Node](#) \*n=nullptr)
- void [calc\\_reduced\\_sequence](#) ()
- double [likelihood](#) (const std::vector< double > &par, bool as\_log=false, bool use\_reduced\_sequence=true)
- double [likelihood\\_exhaust](#) (const std::vector< double > &par)
- std::vector< double > [get\\_probabilities](#) () const
- void [set\\_seed](#) (const unsigned int &s)
- std::vector< std::vector< unsigned int > > [simulate](#) (const std::vector< double > &par)
- std::vector< std::vector< double > > [observed\\_counts](#) ()
- void [print\\_observed\\_counts](#) ()
- void [print](#) () const
 

*Prints information about the GEESE.*
- void [init\\_node](#) ([Node](#) &n)
- void [update\\_annotations](#) (unsigned int nodeid, std::vector< unsigned int > newann)
- std::vector< std::vector< bool > > [get\\_states](#) () const
 

*Powerset of a gene's possible states.*
- std::vector< unsigned int > [get\\_annotated\\_nodes](#) () const
 

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

#### Construct a new Geese object

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

**Parameters**

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

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

- 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, std::vector< size\_t > \*dist=nullptr) const noexcept  
*Check polytomies and return the largest.*

**Geese prediction**

Calculate the conditional probability

**Parameters**

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

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

#### Returns

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

- `std::vector< std::vector< double > >` [predict](#) (`const std::vector< double > &par`, `std::vector< std::vector< double > > *res_prob=nullptr`, `bool leave_one_out=false`, `bool only_annotated=false`, `bool use_reduced_sequence=true`)
- `std::vector< std::vector< double > >` [predict\\_backend](#) (`const std::vector< double > &par`, `bool use_reduced_sequence`, `const std::vector< uint > &preorder`)
- `std::vector< std::vector< double > >` [predict\\_exhaust\\_backend](#) (`const std::vector< double > &par`, `const std::vector< uint > &preorder`)
- `std::vector< std::vector< double > >` [predict\\_exhaust](#) (`const std::vector< double > &par`)
- `std::vector< std::vector< double > >` [predict\\_sim](#) (`const std::vector< double > &par`, `bool only_annotated=false`, `unsigned int nsims=10000u`)

#### Non-const pointers to shared objects in `<tt>Geese</tt>`

These functions provide direct access to some member objects that are shared by the nodes within [Geese](#).

#### Returns

[get\\_engine\(\)](#) returns the Pseudo-RNG engine used.

[get\\_counters\(\)](#) returns the vector of counters used.

[get\\_model\(\)](#) returns the [Model](#) object used.

[get\\_support\\_fun\(\)](#) returns the computed support of the model.

- `std::mt19937 * get\_engine ()`
- `phylocounters::PhyloCounters * get\_counters ()`
- `phylocounters::PhyloModel * get\_model ()`
- `phylocounters::PhyloSupport * get\_support\_fun ()`

## Public Attributes

- unsigned int [nfunctions](#)
- `std::map< unsigned int, Node >` [nodes](#)
- `barry::MapVec_type< unsigned int >` [map\\_to\\_nodes](#)
- `std::vector< std::vector< std::vector< size_t > > >` [pset\\_loc](#)  
Locations of columns.
- `std::vector< unsigned int >` [sequence](#)
- `std::vector< unsigned int >` [reduced\\_sequence](#)
- bool [initialized](#) = `false`
- bool [delete\\_engine](#) = `false`
- bool [delete\\_support](#) = `false`

### 7.23.1 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`.

## 7.23.2 Constructor & Destructor Documentation

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

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

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

### 7.23.2.2 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.

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

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

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

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

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

Definition at line 293 of file geese-meat-constructors.hpp.

### 7.23.2.5 ~Geese()

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

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

### 7.23.3 Member Function Documentation

#### 7.23.3.1 `calc_reduced_sequence()`

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

Definition at line 324 of file `geese-meat.hpp`.

#### 7.23.3.2 `calc_sequence()`

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

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

#### 7.23.3.3 `colnames()`

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

Names of the terms in the model.

Definition at line 446 of file `geese-meat.hpp`.

#### 7.23.3.4 `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 662 of file `geese-meat.hpp`.

#### 7.23.3.5 `get_counters()`

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

Definition at line 645 of file `geese-meat.hpp`.

### 7.23.3.6 get\_model()

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

Definition at line 650 of file geese-meat.hpp.

### 7.23.3.7 get\_probabilities()

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

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

### 7.23.3.8 get\_rengine()

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

Definition at line 640 of file geese-meat.hpp.

### 7.23.3.9 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 658 of file geese-meat.hpp.

### 7.23.3.10 get\_support\_fun()

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

Definition at line 654 of file geese-meat.hpp.



#### 7.23.3.11 inherit\_support()

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

Definition at line 223 of file geese-meat.hpp.

#### 7.23.3.12 init()

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

Definition at line 96 of file geese-meat.hpp.

#### 7.23.3.13 init\_node()

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

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

#### 7.23.3.14 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.

#### 7.23.3.15 likelihood\_exhaust()

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

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

#### 7.23.3.16 nannotations()

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

Number of annotations.

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

#### 7.23.3.17 nfuncs()

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

Number of functions analyzed.

Definition at line 393 of file geese-meat.hpp.

#### 7.23.3.18 nleafs()

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

Number of leaf.

Definition at line 407 of file geese-meat.hpp.

#### 7.23.3.19 nnodes()

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

Number of nodes (interior + leaf)

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

#### 7.23.3.20 nterms()

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

Number of terms included.

Definition at line 419 of file geese-meat.hpp.

### 7.23.3.21 observed\_counts()

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

Definition at line 488 of file geese-meat.hpp.

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

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

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

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

### 7.23.3.24 parse\_polytomies()

```
unsigned int Geese::parse_polytomies (
    bool verb = true,
    std::vector< size_t > * dist = nullptr ) const [inline], [noexcept]
```

Check polytomies and return the largest.

Definition at line 453 of file geese-meat.hpp.

### 7.23.3.25 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.

### 7.23.3.26 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.

### 7.23.3.27 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.

### 7.23.3.28 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.

### 7.23.3.29 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.

### 7.23.3.30 print()

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

Prints information about the GEESE.

Definition at line 622 of file geese-meat.hpp.

#### 7.23.3.31 print\_observed\_counts()

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

Definition at line 559 of file geese-meat.hpp.

#### 7.23.3.32 set\_seed()

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

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

#### 7.23.3.33 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.

#### 7.23.3.34 support\_size()

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

Number of unique sets of sufficient stats.

Definition at line 427 of file geese-meat.hpp.

#### 7.23.3.35 update\_annotations()

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

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

### 7.23.4 Member Data Documentation

#### 7.23.4.1 delete\_rengine

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

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

#### 7.23.4.2 delete\_support

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

Definition at line 119 of file geese-bones.hpp.

#### 7.23.4.3 initialized

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

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

#### 7.23.4.4 map\_to\_nodes

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

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

#### 7.23.4.5 nfunctions

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

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

#### 7.23.4.6 nodes

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

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

#### 7.23.4.7 pset\_loc

```
std::vector< std::vector< std::vector< size_t > > > Geese::pset_loc
```

Locations of columns.

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

#### 7.23.4.8 reduced\_sequence

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

Definition at line 114 of file geese-bones.hpp.

#### 7.23.4.9 sequence

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

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

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

- [include/barry/models/geese/geese-bones.hpp](#)
- [include/barry/models/geese/geese-meat-constructors.hpp](#)
- [include/barry/models/geese/geese-meat-likelihood.hpp](#)
- [include/barry/models/geese/geese-meat-likelihood\\_exhaust.hpp](#)
- [include/barry/models/geese/geese-meat-predict.hpp](#)
- [include/barry/models/geese/geese-meat-predict\\_exhaust.hpp](#)
- [include/barry/models/geese/geese-meat-predict\\_sim.hpp](#)
- [include/barry/models/geese/geese-meat-simulate.hpp](#)
- [include/barry/models/geese/geese-meat.hpp](#)

## 7.24 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_fun` ()

### 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_target`.

#### Returns

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

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

- `std::vector< std::vector< double > >` \* `get_stats_target()`  
Raw pointers to the support and target statistics.
- `std::vector< std::vector< double > >` \* `get_stats_support()`
- `std::vector< unsigned int >` \* `get_arrays2support()`
- `std::vector< std::vector< Array_Type > >` \* `get_pset_arrays()`
- `std::vector< std::vector< std::vector< double > > >` \* `get_pset_stats()`  
Statistics of the support(s)
- `std::vector< std::vector< double > >` \* `get_pset_probs()`

### 7.24.1 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 46 of file model-bones.hpp.

### 7.24.2 Constructor & Destructor Documentation

#### 7.24.2.1 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 ( )
```

#### 7.24.2.2 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_ )
```

### 7.24.2.3 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_ )
```

### 7.24.2.4 ~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 147 of file model-bones.hpp.

## 7.24.3 Member Function Documentation

### 7.24.3.1 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 keygen to obtain a double vector and create a hash of it. If the hash has been computed earlier, the support is recycled.

#### Returns

The number of the array.

### 7.24.3.2 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 )
```

#### 7.24.3.3 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 )
```

#### 7.24.3.4 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 )
```

#### 7.24.3.5 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 )
```

#### 7.24.3.6 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 )
```

#### 7.24.3.7 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 )
```

### 7.24.3.8 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 )
```

### 7.24.3.9 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 )
```

### 7.24.3.10 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 )
```

### 7.24.3.11 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
```

### 7.24.3.12 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  $P\{Y(i,j) = 1 \mid Y^{\setminus C}, \theta\}$ , 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

## 7.24.3.13 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_ )
```

## 7.24.3.14 get\_arrays2support()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↔
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< unsigned int >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_↔
Rule_Dyn_Type >::get_arrays2support ( )
```

## 7.24.3.15 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 ( )
```

## 7.24.3.16 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 )
```

### 7.24.3.17 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 )
```

### 7.24.3.18 get\_pset\_arrays()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector< Array_Type > >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_pset_arrays ( )
```

### 7.24.3.19 get\_pset\_probs()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector<double> > >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_pset_probs ( )
```

### 7.24.3.20 get\_pset\_stats() [1/2]

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector< std::vector<double> > > >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_pset_stats ( )
```

Statistics of the support(s)

### 7.24.3.21 get\_pset\_stats() [2/2]

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

**7.24.3.22 get\_engine()**

```
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_engine ( ) const
```

**7.24.3.23 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 ( )
```

**7.24.3.24 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 ( )
```

**7.24.3.25 get\_stats\_support()**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector< double > >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_stats_support ( )
```

**7.24.3.26 get\_stats\_target()**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< std::vector< double > >* Model< Array_Type, Data_Counter_Type, Data_Rule_Type,
Data_Rule_Dyn_Type >::get_stats_target ( )
```

Raw pointers to the support and target statistics.



### 7.24.3.27 get\_support\_fun()

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
Support<Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type>* Model< Array_Type,
Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_support_fun ( )
```

### 7.24.3.28 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 )
```

### 7.24.3.29 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 )
```

### 7.24.3.30 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 )
```

**7.24.3.31 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 )
```

**7.24.3.32 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]
```

**7.24.3.33 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_ )
```

**7.24.3.34 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.

**7.24.3.35 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
```

### 7.24.3.36 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 = {} )
```

### 7.24.3.37 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 )
```

### 7.24.3.38 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_ )
```

### 7.24.3.39 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_ )
```

### 7.24.3.40 set\_engine()

```
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_engine (
    std::mt19937 * engine_,
    bool delete_ = false ) [inline]
```

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

**7.24.3.41 set\_rules()**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_rules (
    Rules< Array_Type, Data_Rule_Type > * rules_ )
```

**7.24.3.42 set\_rules\_dyn()**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_rules_dyn
(
    Rules< Array_Type, Data_Rule_Dyn_Type > * rules_ )
```

**7.24.3.43 set\_seed()**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
void Model< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::set_seed (
    unsigned int s ) [inline]
```

Definition at line 127 of file model-bones.hpp.

**7.24.3.44 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]
```

**7.24.3.45 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]
```

#### 7.24.3.46 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]
```

#### 7.24.3.47 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](#)

## 7.25 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](#)

### 7.25.1 Detailed Description

Data class used to store arbitrary uint or double vectors.

Definition at line 56 of file network.hpp.

### 7.25.2 Constructor & Destructor Documentation

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

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

Definition at line 62 of file network.hpp.

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

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

Definition at line 63 of file network.hpp.

### 7.25.2.3 ~NetCounterData()

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

Definition at line 68 of file network.hpp.

## 7.25.3 Member Data Documentation

### 7.25.3.1 indices

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

Definition at line 59 of file network.hpp.

### 7.25.3.2 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](#)

## 7.26 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](#)

#### 7.26.1 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](#).

#### 7.26.2 Constructor & Destructor Documentation

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

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

Definition at line 25 of file [network.hpp](#).

##### 7.26.2.2 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.

### 7.26.2.3 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.

### 7.26.2.4 ~NetworkData()

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

Definition at line 51 of file network.hpp.

## 7.26.3 Member Data Documentation

### 7.26.3.1 directed

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

Definition at line 22 of file network.hpp.



### 7.26.3.2 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](#)

## 7.27 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.*

### 7.27.1 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`.

### 7.27.2 Constructor & Destructor Documentation

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

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

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

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

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

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

### 7.27.2.3 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.

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

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

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

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

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

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

### 7.27.2.6 ~Node()

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

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

## 7.27.3 Member Function Documentation

### 7.27.3.1 `get_parent()`

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

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

### 7.27.3.2 `is_leaf()`

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

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

### 7.27.3.3 `noffspring()`

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

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

## 7.27.4 Member Data Documentation

### 7.27.4.1 `annotations`

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

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

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

### 7.27.4.2 `array`

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

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

### 7.27.4.3 arrays

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

Arrays given all possible states.

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

### 7.27.4.4 duplication

```
bool Node::duplication
```

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

### 7.27.4.5 id

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

Id of the node (as specified in the input)

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

### 7.27.4.6 narray

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

ID of the array in the model.

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

### 7.27.4.7 offspring

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

Offspring nodes.

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

#### 7.27.4.8 ord

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

Order in which the node was created.

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

#### 7.27.4.9 parent

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

Parent node.

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

#### 7.27.4.10 probability

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

The probability of observing each state.

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

#### 7.27.4.11 subtree\_prob

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

Induced subtree probabilities.

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

#### 7.27.4.12 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](#)

## 7.28 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

### 7.28.1 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`.

### 7.28.2 Constructor & Destructor Documentation

#### 7.28.2.1 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`.

### 7.28.3 Member Data Documentation

### 7.28.3.1 blengths

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

Branch length.

Definition at line 44 of file phylo.hpp.

### 7.28.3.2 duplication

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

Definition at line 54 of file phylo.hpp.

### 7.28.3.3 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](#)

## 7.29 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)
- [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](#) ()



## 7.29.1 Detailed Description

Definition at line 69 of file phylo.hpp.

## 7.29.2 Constructor & Destructor Documentation

### 7.29.2.1 PhyloCounterData()

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

Definition at line 75 of file phylo.hpp.

## 7.29.3 Member Function Documentation

### 7.29.3.1 at()

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

Definition at line 80 of file phylo.hpp.

### 7.29.3.2 begin()

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

Definition at line 88 of file phylo.hpp.

### 7.29.3.3 empty()

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

Definition at line 91 of file phylo.hpp.

#### 7.29.3.4 end()

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

Definition at line 89 of file phylo.hpp.

#### 7.29.3.5 get\_counters()

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

Definition at line 92 of file phylo.hpp.

#### 7.29.3.6 operator()()

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

Definition at line 81 of file phylo.hpp.

#### 7.29.3.7 operator[]()

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

Definition at line 82 of file phylo.hpp.

#### 7.29.3.8 push\_back()

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

Definition at line 84 of file phylo.hpp.

#### 7.29.3.9 reserve()

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

Definition at line 83 of file phylo.hpp.

### 7.29.3.10 shrink\_to\_fit()

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

Definition at line 85 of file phylo.hpp.

### 7.29.3.11 size()

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

Definition at line 86 of file phylo.hpp.

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

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

## 7.30 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](#)

### 7.30.1 Detailed Description

Definition at line 1785 of file phylo.hpp.

### 7.30.2 Constructor & Destructor Documentation

### 7.30.2.1 PhyloRuleDynData()

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

Definition at line 1792 of file phylo.hpp.

### 7.30.2.2 ~PhyloRuleDynData()

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

Definition at line 1801 of file phylo.hpp.

## 7.30.3 Member Data Documentation

### 7.30.3.1 counts

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

Definition at line 1787 of file phylo.hpp.

### 7.30.3.2 duplication

```
uint PhyloRuleDynData::duplication
```

Definition at line 1791 of file phylo.hpp.

### 7.30.3.3 lb

```
uint PhyloRuleDynData::lb
```

Definition at line 1789 of file phylo.hpp.

#### 7.30.3.4 pos

```
uint PhyloRuleDynData::pos
```

Definition at line 1788 of file phylo.hpp.

#### 7.30.3.5 ub

```
uint PhyloRuleDynData::ub
```

Definition at line 1790 of file phylo.hpp.

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

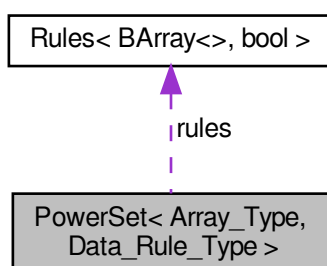
- include/barry/counters/[phylo.hpp](#)

## 7.31 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< size\_t > `coordinates_free`
- std::vector< size\_t > `coordinates_locked`
- size\_t `n_free`
- size\_t `n_locked`

### 7.31.1 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.

## 7.31.2 Constructor & Destructor Documentation

### 7.31.2.1 PowerSet() [1/3]

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

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

### 7.31.2.2 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 44 of file powerset-bones.hpp.

### 7.31.2.3 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.

### 7.31.2.4 ~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.

## 7.31.3 Member Function Documentation

### 7.31.3.1 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 175 of file powerset-meat.hpp.

### 7.31.3.2 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 184 of file powerset-meat.hpp.

### 7.31.3.3 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 194 of file powerset-meat.hpp.

### 7.31.3.4 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 76 of file powerset-bones.hpp.

### 7.31.3.5 calc()

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

Definition at line 147 of file powerset-meat.hpp.



#### 7.31.3.6 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 77 of file powerset-bones.hpp.

#### 7.31.3.7 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 75 of file powerset-bones.hpp.

#### 7.31.3.8 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 74 of file powerset-bones.hpp.

#### 7.31.3.9 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.

#### 7.31.3.10 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 79 of file powerset-bones.hpp.

#### 7.31.3.11 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 163 of file powerset-meat.hpp.

#### 7.31.3.12 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 78 of file powerset-bones.hpp.

### 7.31.4 Member Data Documentation

#### 7.31.4.1 coordinates\_free

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< size_t > PowerSet< Array_Type, Data_Rule_Type >::coordinates_free
```

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

#### 7.31.4.2 coordinates\_locked

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
std::vector< size_t > PowerSet< Array_Type, Data_Rule_Type >::coordinates_locked
```

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

#### 7.31.4.3 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 25 of file powerset-bones.hpp.

#### 7.31.4.4 EmptyArray

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
Array_Type PowerSet< Array_Type, Data_Rule_Type >::EmptyArray
```

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

#### 7.31.4.5 M

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
uint PowerSet< Array_Type, Data_Rule_Type >::M
```

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

#### 7.31.4.6 N

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
uint PowerSet< Array_Type, Data_Rule_Type >::N
```

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

#### 7.31.4.7 n\_free

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
size_t PowerSet< Array_Type, Data_Rule_Type >::n_free
```

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

#### 7.31.4.8 n\_locked

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
size_t PowerSet< Array_Type, Data_Rule_Type >::n_locked
```

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

#### 7.31.4.9 rules

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
Rules<Array_Type,Data_Rule_Type>* PowerSet< Array_Type, Data_Rule_Type >::rules
```

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

#### 7.31.4.10 rules\_deleted

```
template<typename Array_Type = BArray<>, typename Data_Rule_Type = bool>
bool PowerSet< Array_Type, Data_Rule_Type >::rules_deleted = false
```

Definition at line 29 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

## 7.32 Progress Class Reference

A simple progress bar.

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

### Public Member Functions

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

#### 7.32.1 Detailed Description

A simple progress bar.

Definition at line 11 of file progress.hpp.

#### 7.32.2 Constructor & Destructor Documentation

##### 7.32.2.1 Progress()

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

Definition at line 30 of file progress.hpp.

### 7.32.2.2 ~Progress()

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

Definition at line 23 of file progress.hpp.

## 7.32.3 Member Function Documentation

### 7.32.3.1 end()

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

Definition at line 52 of file progress.hpp.

### 7.32.3.2 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](#)

## 7.33 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)`

**7.33.1 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`.

**7.33.2 Constructor & Destructor Documentation****7.33.2.1 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`.

**7.33.2.2 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`.

### 7.33.2.3 ~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.

## 7.33.3 Member Function Documentation

### 7.33.3.1 D()

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

Read/Write access to the data.

### 7.33.3.2 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

## 7.34 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< size\_t > \*free, std::vector< size\_t > \*locked=nullptr)  
*Computes the sequence of free and locked cells in an [BArray](#).*

### Rule adding

*Parameters*

rule	
------	--

- void [add\\_rule](#) ([Rule](#)< Array\_Type, Data\_Type > &rule)
- void [add\\_rule](#) ([Rule](#)< Array\_Type, Data\_Type > \*rule)
- void [add\\_rule](#) ([Rule\\_fun\\_type](#)< Array\_Type, Data\_Type > rule\_, Data\_Type \*[data\\_](#)=nullptr, bool [delete\\_data\\_](#)=false)

**7.34.1 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.

**7.34.2 Constructor & Destructor Documentation****7.34.2.1 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.

**7.34.2.2 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.



**7.34.2.3 ~Rules()**

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

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

**7.34.3 Member Function Documentation****7.34.3.1 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.

**7.34.3.2 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.

**7.34.3.3 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.

**7.34.3.4 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.

**7.34.3.5 get\_seq()**

```
template<typename Array_Type , typename Data_Type >
void Rules< Array_Type, Data_Type >::get_seq (
    const Array_Type & a,
    std::vector< size_t > * free,
    std::vector< size_t > * locked = nullptr ) [inline]
```

Computes the sequence of free and locked cells in an [BArray](#).

## Parameters

<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 140 of file rules-meat.hpp.

**7.34.3.6 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.

**7.34.3.7 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.

## 7.34.3.8 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

## 7.35 StatsCounter&lt; Array\_Type, Data\_Type &gt; Class Template Reference

Count stats for a single Array.

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

## Public Member Functions

- [StatsCounter](#) (const Array\_Type \*Array\_)  
*Creator of a [StatsCounter](#)*
- [StatsCounter](#) (const StatsCounter< Array\_Type, Data\_Type > &counter)  
*Copy constructor.*
- [StatsCounter](#) ()  
*Can be created without setting the array.*
- [~StatsCounter](#) ()
- void [reset\\_array](#) (const Array\_Type \*Array\_)  
*Changes the reference array for the counting.*
- void [add\\_counter](#) (Counter< Array\_Type, Data\_Type > \*f\_)
- void [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

## 7.35.1 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 19 of file statscounter-bones.hpp.

## 7.35.2 Constructor & Destructor Documentation

### 7.35.2.1 StatsCounter() [1/3]

```
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 42 of file statscounter-bones.hpp.

### 7.35.2.2 StatsCounter() [2/3]

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

Copy constructor.

#### Parameters

<a href="#">counter</a>	
-------------------------	--

### 7.35.2.3 StatsCounter() [3/3]

```
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 64 of file statscounter-bones.hpp.

### 7.35.2.4 ~StatsCounter()

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

## 7.35.3 Member Function Documentation

### 7.35.3.1 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_ )
```

### 7.35.3.2 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_ )
```

### 7.35.3.3 count\_all()

```
template<typename Array_Type , typename Data_Type >
std::vector< double > StatsCounter< Array_Type, Data_Type >::count_all [inline]
```

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

### 7.35.3.4 count\_current()

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

### 7.35.3.5 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.

**7.35.3.6 get\_counters()**

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

**7.35.3.7 get\_descriptions()**

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

**7.35.3.8 get\_names()**

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

**7.35.3.9 reset\_array()**

```
template<typename Array_Type = BArray<>, typename Data_Type = bool>
void StatsCounter< Array_Type, Data_Type >::reset_array (
    const Array_Type * Array_ )
```

Changes the reference array for the counting.

**Parameters**

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

**7.35.3.10 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 files:

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

## 7.36 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.*
- std::vector< double > [get\\_counts](#) () [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

<a href="#">f_</a>	<i>A rule to be added.</i>
<a href="#">counters_</a> ↔	<i>A vector of rules to be added.</i>
—	

- void [add\\_rule](#) ([Rule](#)< [Array\\_Type](#), [Data\\_Rule\\_Type](#) > \*[f\\_](#))
- void [add\\_rule](#) ([Rule](#)< [Array\\_Type](#), [Data\\_Rule\\_Type](#) > [f\\_](#))
- void [set\\_rules](#) ([Rules](#)< [Array\\_Type](#), [Data\\_Rule\\_Type](#) > \*[rules\\_](#))
- void [add\\_rule\\_dyn](#) ([Rule](#)< [Array\\_Type](#), [Data\\_Rule\\_Dyn\\_Type](#) > \*[f\\_](#))
- void [add\\_rule\\_dyn](#) ([Rule](#)< [Array\\_Type](#), [Data\\_Rule\\_Dyn\\_Type](#) > [f\\_](#))
- void [set\\_rules\\_dyn](#) ([Rules](#)< [Array\\_Type](#), [Data\\_Rule\\_Dyn\\_Type](#) > \*[rules\\_](#))
- bool [eval\\_rules\\_dyn](#) ([const](#) [std::vector](#)< [double](#) > &[counts](#), [const](#) [uint](#) &[i](#), [const](#) [uint](#) &[j](#))

## Public Attributes

- [uint](#) [N](#)
- [uint](#) [M](#)
- bool [delete\\_counters](#) = true
- bool [delete\\_rules](#) = true
- bool [delete\\_rules\\_dyn](#) = true
- [uint](#) [max\\_num\\_elements](#) = [BARRY\\_MAX\\_NUM\\_ELEMENTS](#)
- [std::vector](#)< [double](#) > [current\\_stats](#)
- [std::vector](#)< [size\\_t](#) > [coordinates\\_free](#)
- [std::vector](#)< [size\\_t](#) > [coordinates\\_locked](#)
- [size\\_t](#) [cooriantes\\_n\\_free](#)
- [size\\_t](#) [cooriantes\\_n\\_locked](#)
- [std::vector](#)< [double](#) > [change\\_stats](#)
- [std::vector](#)< [size\\_t](#) > [hashes](#)
- [std::vector](#)< bool > [hashes\\_initialized](#)
- [size\\_t](#) [n\\_counters](#)

### 7.36.1 Detailed Description

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

Compute the support of sufficient statistics.

Given an array and a set of counters, this object iterates throughout the support set of the Array while at the same time computing the support of the sufficient statitics.

The members [rule](#) and [rule\\_dyn](#) allow constraining the support. The first will establish which cells of the array will be used to iterate, for example, in the case of social networks, self-loops are not allowed, so the entire diagonal would be fixed to zero, reducing the size of the support.

In the case of [rule\\_dyn](#), the function will stablish dynamically whether the current state will be included in the counts or not. For example, this set of rules can be used to constrain the support to networks that have a prescribed degree sequence.

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



## 7.36.2 Constructor & Destructor Documentation

### 7.36.2.1 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 80 of file support-bones.hpp.

### 7.36.2.2 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 89 of file support-bones.hpp.

### 7.36.2.3 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 96 of file support-bones.hpp.

### 7.36.2.4 ~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 103 of file support-bones.hpp.

## 7.36.3 Member Function Documentation

### 7.36.3.1 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_ )
```

### 7.36.3.2 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_ )
```

### 7.36.3.3 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_ )
```

### 7.36.3.4 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_ )
```

### 7.36.3.5 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_ )
```

### 7.36.3.6 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_ )
```

### 7.36.3.7 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.

### 7.36.3.8 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 )
```

### 7.36.3.9 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.

**7.36.3.10 get\_counts()**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
std::vector< double > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::get_counts ( ) const
```

**7.36.3.11 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.

**7.36.3.12 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
```

**7.36.3.13 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).

**7.36.3.14 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).

#### 7.36.3.15 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 )
```

#### 7.36.3.16 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
```

#### 7.36.3.17 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
( )
```

#### 7.36.3.18 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_ )
```

#### 7.36.3.19 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_ )
```

### 7.36.3.20 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_ )
```

### 7.36.3.21 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_ )
```

## 7.36.4 Member Data Documentation

### 7.36.4.1 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< double > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_↵
_Type >::change_stats
```

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

### 7.36.4.2 coordiantes\_n\_free

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::coordiantes↵
_n_free
```

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

### 7.36.4.3 coordiantes\_n\_locked

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::coordiantes↵
_n_locked
```

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

#### 7.36.4.4 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< size_t > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn↵
_Type >::coordinates_free
```

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

#### 7.36.4.5 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< size_t > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn↵
_Type >::coordinates_locked
```

Definition at line 70 of file support-bones.hpp.

#### 7.36.4.6 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 68 of file support-bones.hpp.

#### 7.36.4.7 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 62 of file support-bones.hpp.

#### 7.36.4.8 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 63 of file support-bones.hpp.

#### 7.36.4.9 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 64 of file support-bones.hpp.

#### 7.36.4.10 hashes

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵  
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>  
std::vector< size_t > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_↵  
_Type >::hashes
```

Definition at line 74 of file support-bones.hpp.

#### 7.36.4.11 hashes\_initialized

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_↵  
Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>  
std::vector< bool > Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_↵  
Type >::hashes_initialized
```

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

#### 7.36.4.12 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 61 of file support-bones.hpp.

#### 7.36.4.13 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 65 of file support-bones.hpp.



**7.36.4.14 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 61 of file support-bones.hpp.

**7.36.4.15 n\_counters**

```
template<typename Array_Type = BArray<>, typename Data_Counter_Type = bool, typename Data_Rule_Type = bool, typename Data_Rule_Dyn_Type = bool>
size_t Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >::n_counters
```

Definition at line 76 of file support-bones.hpp.

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

- include/barry/support-bones.hpp

**7.37 vecHasher< T > Struct Template Reference**

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

**Public Member Functions**

- std::size\_t operator() (std::vector< T > const &dat) const noexcept

**7.37.1 Detailed Description**

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

Definition at line 106 of file typedefs.hpp.

**7.37.2 Member Function Documentation****7.37.2.1 operator()()**

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

Definition at line 109 of file typedefs.hpp.

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

- include/barry/typedefs.hpp



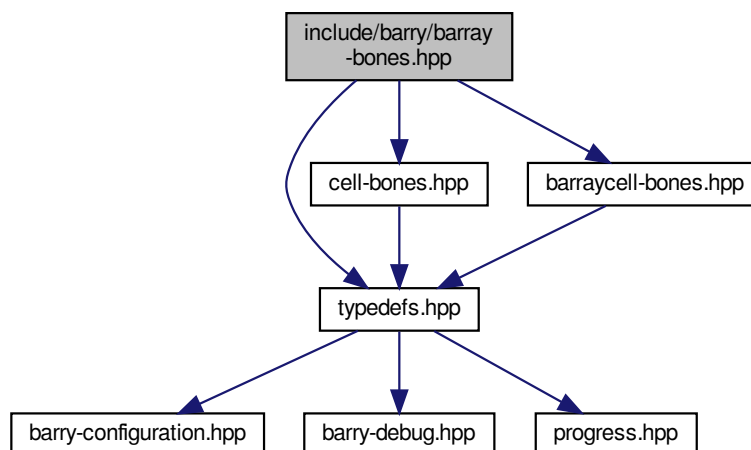
## Chapter 8

# File Documentation

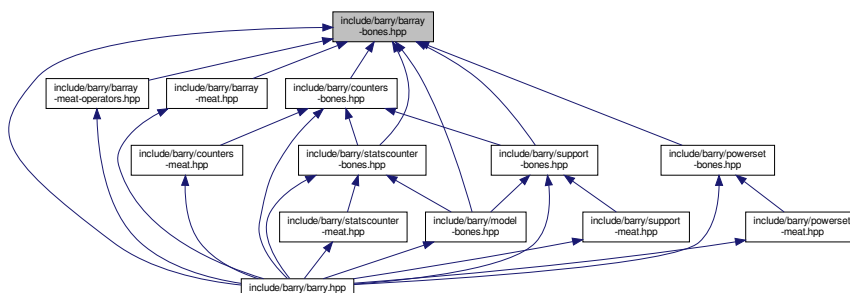
### 8.1 include/barry/barray-bones.hpp File Reference

```
#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

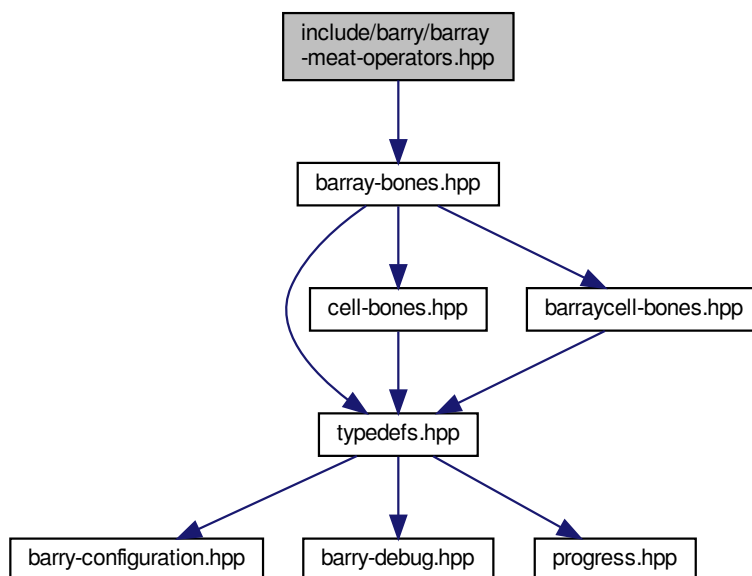
### Classes

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

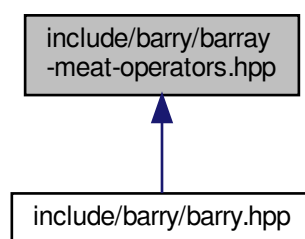
## 8.3 include/barry/barray-meat-operators.hpp File Reference

```
#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

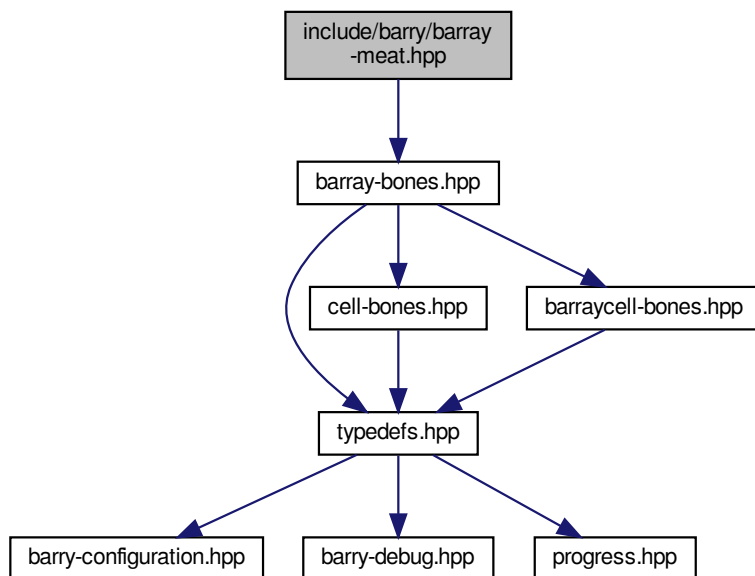
```
return * this
```

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

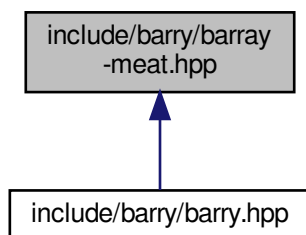
## 8.4 include/barry/barray-meat.hpp File Reference

```
#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 17 of file `barray-meat.hpp`.

### 8.4.1.2 BARRAY\_TEMPLATE\_ARGS

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

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

### 8.4.1.3 BARRAY\_TYPE

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

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

### 8.4.1.4 COL

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

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

### 8.4.1.5 ROW

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

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

## 8.4.2 Function Documentation

### 8.4.2.1 ans()

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

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

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

Definition at line 230 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 339 of file barray-meat.hpp.

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

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

Definition at line 586 of file barray-meat.hpp.

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

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

Definition at line 272 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 321 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 361 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 1119 of file barray-meat.hpp.

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

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

Definition at line 441 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 1058 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 51 of file barray-meat.hpp.

**8.4.2.29 if()** [1/17]

```
else if (
    !check0 && check1 )
```

Definition at line 997 of file barray-meat.hpp.

**8.4.2.30 if()** [2/17]

```
else if (
    !check0 & check1 )
```

Definition at line 845 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 853 of file barray-meat.hpp.

**8.4.2.33 if()** [5/17]

```
else if (
    check0 &&! check1 )
```

Definition at line 988 of file barray-meat.hpp.

**8.4.2.34 if()** [6/17]

```
if (
    check0 && check1 )
```

Definition at line 961 of file barray-meat.hpp.

**8.4.2.35 if()** [7/17]

```
if (
    check0 & check1 )
```

Definition at line 827 of file barray-meat.hpp.

**8.4.2.36 if()** [8/17]

```
else if (
    check_exists == CHECK::BOTH )
```

Definition at line 668 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 86 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 136 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 34 of file barray-meat.hpp.

#### 8.4.3.2 ans

```
return ans
```

Definition at line 438 of file barray-meat.hpp.

#### 8.4.3.3 Array\_

```
Data_Type & Array_
```

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

#### 8.4.3.4 check\_bounds

```
bool check_bounds
```

**Initial value:**

```
{  
    if (check_bounds) {  
        out_of_range(i0,0u);  
        out_of_range(i1,0u);  
    }  
  
    bool move0=true, move1=true
```

Definition at line 661 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 662 of file barray-meat.hpp.

#### 8.4.3.6 col0

```
auto col0 = COL(j)
```

Definition at line 1050 of file barray-meat.hpp.

#### 8.4.3.7 const

```
uint bool check_bounds const
```

##### Initial value:

```
{  
    if (i >= N)  
        throw std::range_error("The row is out of range.")
```

Definition at line 391 of file barray-meat.hpp.

#### 8.4.3.8 copy\_data

```
Data_Type bool copy_data
```

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

#### 8.4.3.9 data

```
data = data_
```

Definition at line 354 of file barray-meat.hpp.

#### 8.4.3.10 delete\_data

```
delete_data = delete_data_
```

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

#### 8.4.3.11 delete\_data\_

```
bool delete_data_
```

##### Initial value:

```
{  
    if ((data != nullptr) && delete_data)  
        delete data
```

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

#### 8.4.3.12 else

```
else (  
    void )
```

##### Initial value:

```
{  
    ROW(i).insert(std::pair< uint, Cell<Cell_Type>>(j, v))
```

Definition at line 692 of file barray-meat.hpp.

#### 8.4.3.13 false

```
row false
```

Definition at line 1031 of file barray-meat.hpp.

#### 8.4.3.14 first

```
row first
```

Definition at line 1031 of file barray-meat.hpp.



**8.4.3.15 i1**

```
uint i1
```

Definition at line 765 of file barray-meat.hpp.

**8.4.3.16 j**

```
uint j
```

**Initial value:**

```
{  
    if (init_fun == nullptr)  
        return 0.0
```

Definition at line 403 of file barray-meat.hpp.

**8.4.3.17 j0**

```
uint j0
```

Definition at line 764 of file barray-meat.hpp.

**8.4.3.18 j1**

```
uint j1
```

Definition at line 765 of file barray-meat.hpp.

**8.4.3.19 M**

```
M = M_
```

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

### 8.4.3.20 M\_

```
uint M_
```

#### Initial value:

```
{  
  
    if (N_ < N)  
        for (uint i = N_; i < N; ++i)  
            zero_row(i, false)
```

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

### 8.4.3.21 N

```
if (source.size() != target.size()) throw std::if (source.size() != value.size()) throw std::N =  
N_
```

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

### 8.4.3.22 NCells

```
NCells
```

Definition at line 696 of file barray-meat.hpp.

### 8.4.3.23 report

```
uint uint uint bool int int* report
```

Definition at line 768 of file barray-meat.hpp.

### 8.4.3.24 return

```
return
```

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

#### 8.4.3.25 row0

```
auto row0 = ROW(i)
```

Definition at line 1029 of file barray-meat.hpp.

#### 8.4.3.26 search

```
auto search = ROW(i).find(j)
```

Definition at line 415 of file barray-meat.hpp.

#### 8.4.3.27 source

```
uint const std::vector< uint > & source
```

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

#### 8.4.3.28 target

```
uint const std::vector< uint > const std::vector< uint > & target
```

Definition at line 32 of file barray-meat.hpp.

#### 8.4.3.29 v

```
uint Cell_Type v
```

Definition at line 660 of file barray-meat.hpp.

#### 8.4.3.30 value

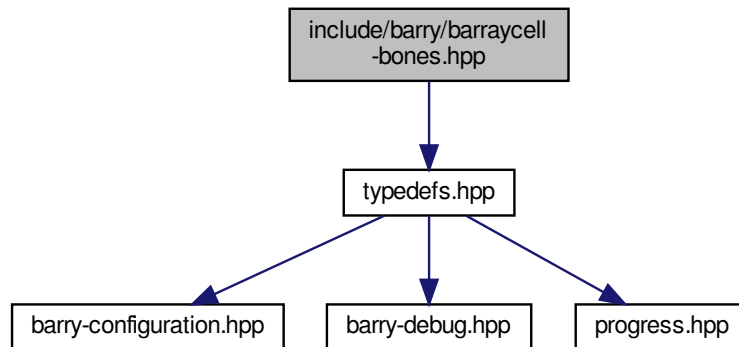
```
uint const std::vector< uint > const std::vector< uint > const std::vector< Cell_Type >&  
value
```

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

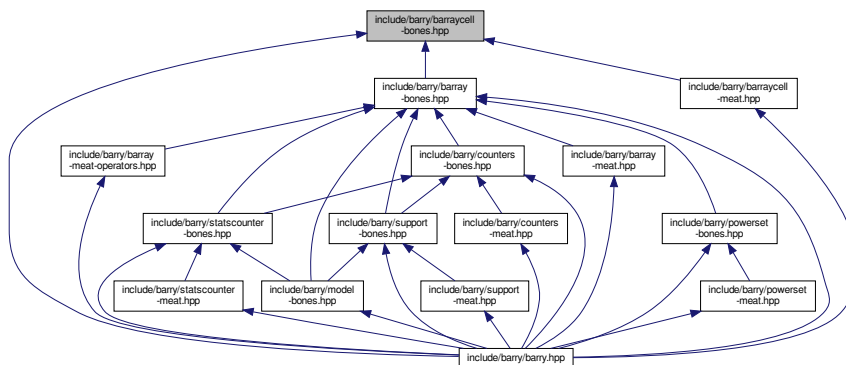
## 8.5 include/barry/barraycell-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barraycell-bones.hpp:



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



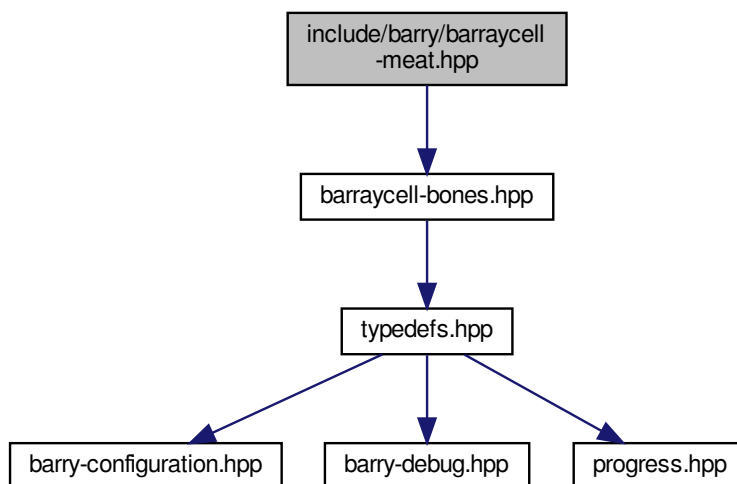
### Classes

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

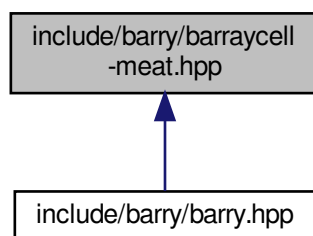
## 8.6 include/barry/barraycell-meat.hpp File Reference

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

Include dependency graph for barraycell-meat.hpp:



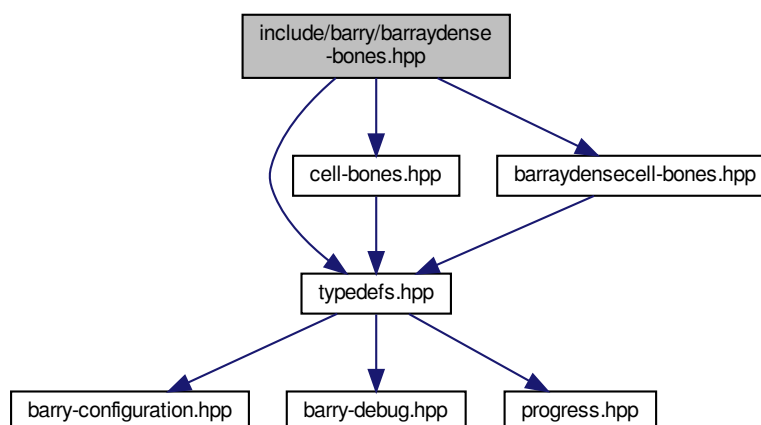
This graph shows which files directly or indirectly include this file:



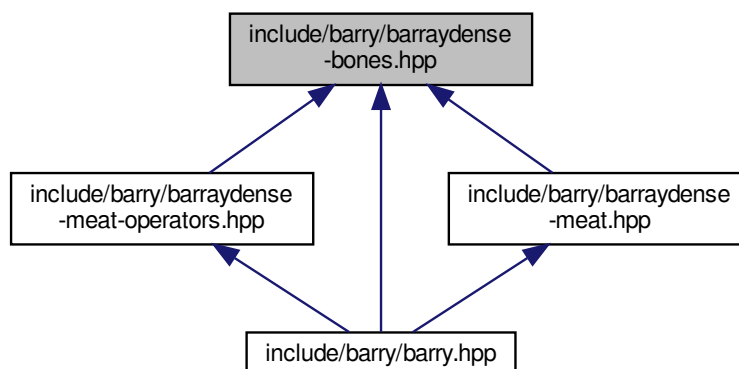
## 8.7 include/barry/barraydense-bones.hpp File Reference

```
#include "typedefs.hpp"
#include "cell-bones.hpp"
#include "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

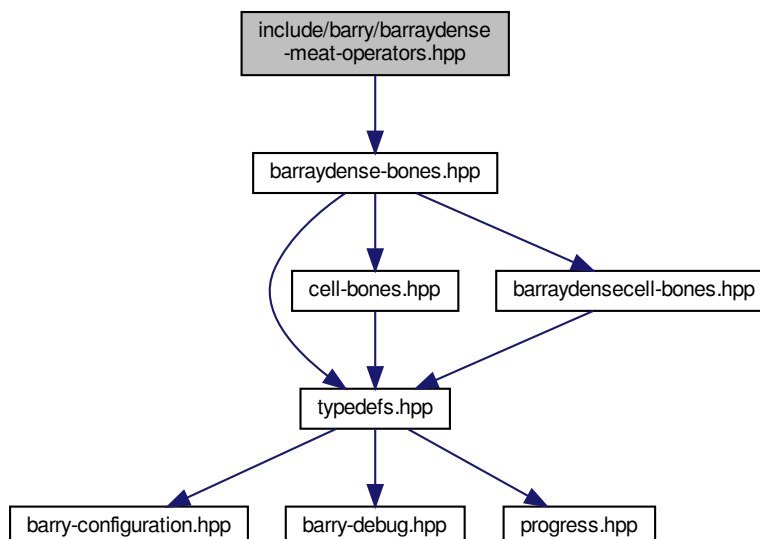
```
#define BARRY_BARRAYDENSE_BONES_HPP 1
```

Definition at line 8 of file barraydense-bones.hpp.

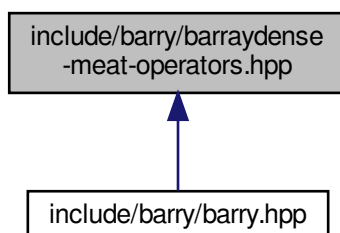
## 8.8 include/barry/barraydense-meat-operators.hpp File Reference

```
#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_BARRAYDENSE_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\_BARRAYDENSE\_MEAT\_OPERATORS\_HPP

```
#define BARRY_BARRAYDENSE_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()

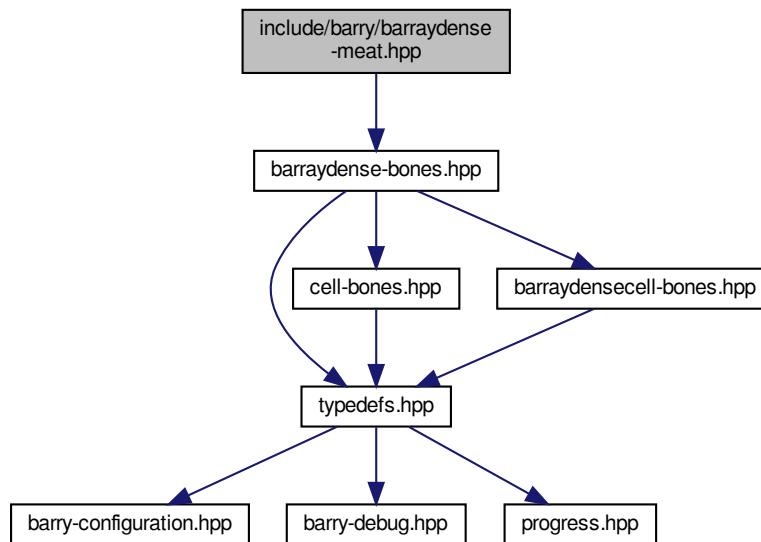
```
template const BDENSE_TYPE ( ) &
```

Definition at line 22 of file barrydense-meat-operators.hpp.

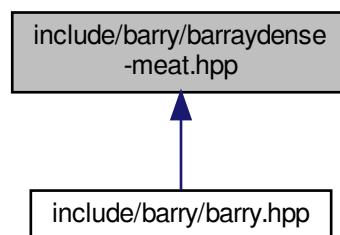
## 8.9 include/barry/barraydense-meat.hpp File Reference

```
#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 static_cast<Cell_Type>(0.0)`

## Functions

- [BDENSE\\_TEMPLATE](#) ([BArrayDense](#))([uint](#) N\_
- [el](#) [resize](#) (N \*M, [ZERO\\_CELL](#))
- [el\\_rowsums](#) [resize](#) (N, [ZERO\\_CELL](#))
- [el\\_colsums](#) [resize](#) (M, [ZERO\\_CELL](#))
- [for](#) ([uint](#) i=0u;i< [source.size](#)();++i)
- [BDENSE\\_TEMPLATE](#) ([BArrayDense](#))([const](#) [BDENSE\\_TYPE](#)() &[Array\\_](#)
- [bool](#) M ([Array\\_](#).M)
- [BDENSE\\_TEMPLATE](#) ([BDENSE\\_TYPE](#)() &, [operator=](#))([const](#) [BDENSE\\_TYPE](#)() &[Array\\_](#))
- [BDENSE\\_TEMPLATE](#) ([BArrayDense](#))([BDENSE\\_TYPE](#)() &&x) [noexcept](#)
- [BDENSE\\_TEMPLATE](#) ([BDENSE\\_TYPE](#)() &, [operator=](#))([BDENSE\\_TYPE](#)() &&x) [noexcept](#)
- [BDENSE\\_TEMPLATE](#) ([bool](#), [operator=](#))([const](#) [BDENSE\\_TYPE](#)() &[Array\\_](#))
- [BDENSE\\_TEMPLATE](#) ([BArrayDense](#))()
- [BDENSE\\_TEMPLATE](#) ([void](#), [set\\_data](#))([Data\\_Type](#) \*[data\\_](#)
- [BDENSE\\_TEMPLATE](#) ([Data\\_Type](#) \*, D)()
- [BDENSE\\_TEMPLATE](#) ([const](#) [Data\\_Type](#) \*, D)() [const](#)
- [BDENSE\\_TEMPLATE](#) ([void](#), [out\\_of\\_range](#))([uint](#) i
- [BDENSE\\_TEMPLATE](#) ([Cell\\_Type](#), [get\\_cell](#))([uint](#) i
- [BDENSE\\_TEMPLATE](#) ([std::vector](#)< [Cell\\_Type](#) >, [get\\_row\\_vec](#))([uint](#) i
- [std::vector](#)< [Cell\\_Type](#) > [ans](#) ([ncol](#)(), [static\\_cast](#)< [Cell\\_Type](#) >(false))
- [BDENSE\\_TEMPLATE](#) ([void](#), [get\\_row\\_vec](#))([std](#)
- [BDENSE\\_TEMPLATE](#) ([Entries](#)< [Cell\\_Type](#) >, [get\\_entries](#)()) [const](#)
- [BDENSE\\_TEMPLATE](#) ([bool](#), [is\\_empty](#))([uint](#) i
- [BDENSE\\_TEMPLATE](#) ([unsigned](#) [int](#), [nrow](#)()) [const](#) [noexcept](#)
- [BDENSE\\_TEMPLATE](#) ([unsigned](#) [int](#), [ncol](#)()) [const](#) [noexcept](#)
- [BDENSE\\_TEMPLATE](#) ([unsigned](#) [int](#), [nnozero](#)()) [const](#) [noexcept](#)
- [BDENSE\\_TEMPLATE](#) ([Cell](#)< [Cell\\_Type](#) >, [default\\_val](#)()) [const](#)
- [BDENSE\\_TEMPLATE](#) ([BDENSE\\_TYPE](#)() &, [operator+=](#))([const](#) [std](#)
- [BDENSE\\_TEMPLATE](#) ([BDENSE\\_TYPE](#)() &, [operator-=](#))([const](#) [std](#)
- [BDENSE\\_TEMPLATE](#) ([void](#), [insert\\_cell](#))([uint](#) i
- [if](#) ([el](#)[[POS](#)(i, j)]==[BARRY\\_ZERO\\_DENSE](#))
- [BDENSE\\_TEMPLATE](#) ([void](#), [swap\\_cells](#))([uint](#) i0
- [if](#) ((i0==i1) &&(j0==j1)) [return](#)
- [rm\\_cell](#) (i0, j0, false, false)
- [rm\\_cell](#) (i1, j1, false, false)
- [insert\\_cell](#) (i0, j0, val1, false, false)
- [insert\\_cell](#) (i1, j1, val0, false, false)
- [BDENSE\\_TEMPLATE](#) ([void](#), [toggle\\_cell](#))([uint](#) i
- [else](#) [rm\\_cell](#) (i, j, false, false)
- [BDENSE\\_TEMPLATE](#) ([void](#), [swap\\_rows](#))([uint](#) i0
- [BDENSE\\_TEMPLATE](#) ([void](#), [swap\\_cols](#))([uint](#) j0
- [BDENSE\\_TEMPLATE](#) ([void](#), [zero\\_row](#))([uint](#) i
- [if](#) ([el\\_rowsums](#)[i]==[ZERO\\_CELL](#)) [return](#)
- [BDENSE\\_TEMPLATE](#) ([void](#), [zero\\_col](#))([uint](#) j
- [if](#) ([el\\_colsums](#)[j]==[ZERO\\_CELL](#)) [return](#)
- [BDENSE\\_TEMPLATE](#) ([void](#), [transpose](#)())
- [BDENSE\\_TEMPLATE](#) ([void](#), [clear](#))([bool](#) hard)
- [BDENSE\\_TEMPLATE](#) ([void](#), [resize](#))([uint](#) N\_
- [el](#) [resize](#) (N\_ \*M\_, [ZERO\\_CELL](#))
- [el\\_rowsums](#) [resize](#) (N\_, [ZERO\\_CELL](#))
- [el\\_colsums](#) [resize](#) (M\_, [ZERO\\_CELL](#))
- [BDENSE\\_TEMPLATE](#) ([void](#), [reserve](#)())
- [BDENSE\\_TEMPLATE](#) ([void](#), [print](#))([const](#) [char](#) \*fmt
- [va\\_start](#) (args, fmt)

- `vprintf` (fmt, args)
- `va_end` (args)
- `BDENSE_TEMPLATE` (const std::vector< Cell\_Type > &, get\_data()) const
- `BDENSE_TEMPLATE` (const Cell\_Type, rowsum)(unsigned int i) const
- `BDENSE_TEMPLATE` (const Cell\_Type, colsum)(unsigned int j) const

## Variables

- `uint M_`
- `uint const` std::vector< `uint` > & `source`
- `uint const` std::vector< `uint` > `const` std::vector< `uint` > & `target`
- `uint const` std::vector< `uint` > `const` std::vector< `uint` > `const` std::vector< Cell\_Type > & `value`
- `uint const` std::vector< `uint` > `const` std::vector< `uint` > `const` std::vector< Cell\_Type > bool `add`
- `if`(source.size() !=value.size()) throw std `N` = `N_`
- `M` = `M_`
- `return`
- bool `copy_data`
- bool `delete_data_`
- `data` = `data_`
- `delete_data` = `delete_data_`
- `uint j` const
- `uint j`
- `return` el [POS(i, j)] == `ZERO_CELL`
- `return` `ans`
- `uint const` Cell< Cell\_Type > & `v`
- `uint const` Cell< Cell\_Type > bool `check_bounds`
- `uint const` Cell< Cell\_Type > bool bool `check_exists`
- `else`
- `el_rowsums` [i] = (`v.value` - `old`)
- `el_colsums` [j] = (`v.value` - `old`)
- `uint j0`
- `uint` `uint i1`
- `uint` `uint` `uint j1`
- `uint` `uint` `uint` bool int int \* `report`
- Cell\_Type `val0` = `el`[POS(i0,j0)]
- Cell\_Type `val1` = `el`[POS(i1,j1)]
- `false`
- `col`

## 8.9.1 Macro Definition Documentation

### 8.9.1.1 BDENSE\_TEMPLATE

```
#define BDENSE_TEMPLATE(
    a,
    b )  template BDENSE_TEMPLATE_ARGS() inline a BDENSE_TYPE():b
```

Definition at line 27 of file barraydense-meat.hpp.

### 8.9.1.2 BDENSE\_TEMPLATE\_ARGS

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

Definition at line 25 of file `barraydense-meat.hpp`.

### 8.9.1.3 BDENSE\_TYPE

```
#define BDENSE_TYPE( ) BArrayDense<Cell_Type, Data_Type>
```

Definition at line 23 of file `barraydense-meat.hpp`.

### 8.9.1.4 COL

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

Definition at line 31 of file `barraydense-meat.hpp`.

### 8.9.1.5 POS

```
#define POS(  
    a,  
    b ) (b)*N + (a)
```

Definition at line 32 of file `barraydense-meat.hpp`.

### 8.9.1.6 POS\_N

```
#define POS_N(  
    a,  
    b,  
    c ) (b)*(c) + (a)
```

Definition at line 33 of file `barraydense-meat.hpp`.

### 8.9.1.7 ROW

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

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

### 8.9.1.8 ZERO\_CELL

```
#define ZERO_CELL static_cast<Cell_Type>(0.0)
```

Definition at line 38 of file barraydense-meat.hpp.

## 8.9.2 Function Documentation

### 8.9.2.1 ans()

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

### 8.9.2.2 BDENSE\_TEMPLATE() [1/37]

```
BDENSE_TEMPLATE (  
    BArrayDense ) && [noexcept]
```

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

### 8.9.2.3 BDENSE\_TEMPLATE() [2/37]

```
BDENSE_TEMPLATE (  
    BArrayDense ) const &
```

### 8.9.2.4 BDENSE\_TEMPLATE() [3/37]

```
BDENSE_TEMPLATE (  
    BArrayDense )
```

#### 8.9.2.5 BDENSE\_TEMPLATE() [4/37]

```
BDENSE_TEMPLATE (
    ~ BArrayDense )
```

Definition at line 318 of file `barraydense-meat.hpp`.

#### 8.9.2.6 BDENSE\_TEMPLATE() [5/37]

```
BDENSE_TEMPLATE (
    BDENSE_TYPE() & ,
    operator+ ) const
```

Definition at line 558 of file `barraydense-meat.hpp`.

#### 8.9.2.7 BDENSE\_TEMPLATE() [6/37]

```
BDENSE_TEMPLATE (
    BDENSE_TYPE() & ,
    operator- ) const
```

Definition at line 576 of file `barraydense-meat.hpp`.

#### 8.9.2.8 BDENSE\_TEMPLATE() [7/37]

```
BDENSE_TEMPLATE (
    BDENSE_TYPE() & ,
    operator ) && [noexcept]
```

Definition at line 257 of file `barraydense-meat.hpp`.

#### 8.9.2.9 BDENSE\_TEMPLATE() [8/37]

```
BDENSE_TEMPLATE (
    BDENSE_TYPE() & ,
    operator ) const &
```

Definition at line 194 of file `barraydense-meat.hpp`.



**8.9.2.10 BDENSE\_TEMPLATE()** [9/37]

```
BDENSE_TEMPLATE (
    bool ,
    is_empty )
```

**8.9.2.11 BDENSE\_TEMPLATE()** [10/37]

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

Definition at line 300 of file barraydense-meat.hpp.

**8.9.2.12 BDENSE\_TEMPLATE()** [11/37]

```
BDENSE_TEMPLATE (
    Cell< Cell_Type > ,
    default_val ) const
```

Definition at line 554 of file barraydense-meat.hpp.

**8.9.2.13 BDENSE\_TEMPLATE()** [12/37]

```
BDENSE_TEMPLATE (
    Cell_Type ,
    get_cell )
```

**8.9.2.14 BDENSE\_TEMPLATE()** [13/37]

```
BDENSE_TEMPLATE (
    const Cell_Type,
    colsum ) const
```

Definition at line 991 of file barraydense-meat.hpp.

**8.9.2.15 BDENSE\_TEMPLATE()** [14/37]

```
BDENSE_TEMPLATE (
    const Cell_Type,
    rowsum ) const
```

Definition at line 986 of file bararraydense-meat.hpp.

**8.9.2.16 BDENSE\_TEMPLATE()** [15/37]

```
BDENSE_TEMPLATE (
    const Data_Type * ,
    D ) const
```

Definition at line 345 of file bararraydense-meat.hpp.

**8.9.2.17 BDENSE\_TEMPLATE()** [16/37]

```
BDENSE_TEMPLATE (
    const std::vector< Cell_Type > & ,
    get_data ) const
```

Definition at line 981 of file bararraydense-meat.hpp.

**8.9.2.18 BDENSE\_TEMPLATE()** [17/37]

```
BDENSE_TEMPLATE (
    Data_Type * ,
    D )
```

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

**8.9.2.19 BDENSE\_TEMPLATE()** [18/37]

```
BDENSE_TEMPLATE (
    Entries< Cell_Type > ,
    get_entries ) const
```

Definition at line 494 of file bararraydense-meat.hpp.

**8.9.2.20 BDENSE\_TEMPLATE()** [19/37]

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

**8.9.2.21 BDENSE\_TEMPLATE()** [20/37]

```
BDENSE_TEMPLATE (
    unsigned int ,
    ncol ) const [noexcept]
```

Definition at line 540 of file barraydense-meat.hpp.

**8.9.2.22 BDENSE\_TEMPLATE()** [21/37]

```
BDENSE_TEMPLATE (
    unsigned int ,
    nnozero ) const [noexcept]
```

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

**8.9.2.23 BDENSE\_TEMPLATE()** [22/37]

```
BDENSE_TEMPLATE (
    unsigned int ,
    nrow ) const [noexcept]
```

Definition at line 536 of file barraydense-meat.hpp.

**8.9.2.24 BDENSE\_TEMPLATE()** [23/37]

```
BDENSE_TEMPLATE (
    void ,
    clear )
```

Definition at line 888 of file barraydense-meat.hpp.

**8.9.2.25 BDENSE\_TEMPLATE()** [24/37]

```
BDENSE_TEMPLATE (
    void ,
    get_row_vec )
```

Definition at line 394 of file baraydense-meat.hpp.

**8.9.2.26 BDENSE\_TEMPLATE()** [25/37]

```
BDENSE_TEMPLATE (
    void ,
    insert_cell )
```

**8.9.2.27 BDENSE\_TEMPLATE()** [26/37]

```
BDENSE_TEMPLATE (
    void ,
    out_of_range )
```

**8.9.2.28 BDENSE\_TEMPLATE()** [27/37]

```
BDENSE_TEMPLATE (
    void ,
    print ) const
```

**8.9.2.29 BDENSE\_TEMPLATE()** [28/37]

```
BDENSE_TEMPLATE (
    void ,
    reserve )
```

Definition at line 938 of file baraydense-meat.hpp.

**8.9.2.30 BDENSE\_TEMPLATE()** [29/37]

```
BDENSE_TEMPLATE (
    void ,
    resize )
```

**8.9.2.31 BDENSE\_TEMPLATE()** [30/37]

```
BDENSE_TEMPLATE (
    void ,
    set_data )
```

**8.9.2.32 BDENSE\_TEMPLATE()** [31/37]

```
BDENSE_TEMPLATE (
    void ,
    swap_cells )
```

**8.9.2.33 BDENSE\_TEMPLATE()** [32/37]

```
BDENSE_TEMPLATE (
    void ,
    swap_cols )
```

**8.9.2.34 BDENSE\_TEMPLATE()** [33/37]

```
BDENSE_TEMPLATE (
    void ,
    swap_rows )
```

**8.9.2.35 BDENSE\_TEMPLATE()** [34/37]

```
BDENSE_TEMPLATE (
    void ,
    toggle_cell )
```

**8.9.2.36 BDENSE\_TEMPLATE()** [35/37]

```
BDENSE_TEMPLATE (
    void ,
    transpose )
```

Definition at line 860 of file barraydense-meat.hpp.

**8.9.2.37 BDENSE\_TEMPLATE() [36/37]**

```
BDENSE_TEMPLATE (
    void ,
    zero_col )
```

**8.9.2.38 BDENSE\_TEMPLATE() [37/37]**

```
BDENSE_TEMPLATE (
    void ,
    zero_row )
```

**8.9.2.39 for()**

```
for ( )
```

Definition at line 64 of file `barraydense-meat.hpp`.

**8.9.2.40 if() [1/4]**

```
if (
    (i0==i1) && (j0==j1) )
```

**8.9.2.41 if() [2/4]**

```
if (
    el [POS(i, j)] == BARRY_ZERO_DENSE )
```

Definition at line 655 of file `barraydense-meat.hpp`.

**8.9.2.42 if() [3/4]**

```
if (
    el_colsums [j] == ZERO_CELL )
```

**8.9.2.43 if()** [4/4]

```
if (
    el_rowsums [i] == ZERO_CELL )
```

**8.9.2.44 insert\_cell()** [1/2]

```
insert_cell (
    i0 ,
    j0 ,
    val1 ,
    false ,
    false )
```

**8.9.2.45 insert\_cell()** [2/2]

```
insert_cell (
    i1 ,
    j1 ,
    val0 ,
    false ,
    false )
```

**8.9.2.46 M()**

```
bool M (
    Array_ M )
```

Definition at line 157 of file barraydense-meat.hpp.

**8.9.2.47 resize()** [1/6]

```
el_colsums resize (
    M ,
    ZERO_CELL )
```

**8.9.2.48** `resize()` [2/6]

```
el_colsums resize (
    M_ ,
    ZERO_CELL )
```

**8.9.2.49** `resize()` [3/6]

```
el resize (
    N * M,
    ZERO_CELL )
```

**8.9.2.50** `resize()` [4/6]

```
el_rowsums resize (
    N ,
    ZERO_CELL )
```

**8.9.2.51** `resize()` [5/6]

```
el resize (
    N_ * M_,
    ZERO_CELL )
```

**8.9.2.52** `resize()` [6/6]

```
el_rowsums resize (
    N_ ,
    ZERO_CELL )
```

**8.9.2.53** `rm_cell()` [1/3]

```
else rm_cell (
    i ,
    j ,
    false ,
    false )
```



**8.9.2.54 rm\_cell()** [2/3]

```
rm_cell (
    i0 ,
    j0 ,
    false ,
    false )
```

**8.9.2.55 rm\_cell()** [3/3]

```
rm_cell (
    i1 ,
    j1 ,
    false ,
    false )
```

**8.9.2.56 va\_end()**

```
va_end (
    args )
```

**8.9.2.57 va\_start()**

```
va_start (
    args ,
    fmt )
```

**8.9.2.58 vprintf()**

```
vprintf (
    fmt ,
    args )
```

**8.9.3 Variable Documentation**

### 8.9.3.1 add

```
uint const std::vector< uint > const std::vector< uint > bool add
```

#### Initial value:

```
{  
    if (source.size() != target.size())  
        throw std::length_error("-source- and -target- don't match on length.")  
}
```

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

### 8.9.3.2 ans

```
return ans
```

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

### 8.9.3.3 check\_bounds

```
bool check_bounds
```

#### Initial value:

```
{  
    if (check_bounds)  
    {  
        out_of_range(i0,0u);  
        out_of_range(i1,0u);  
    }  
  
    for (uint j = 0u; j < M; ++j)  
        std::swap(e1[POS(i0, j)], e1[POS(i1, j)])  
}
```

Definition at line 646 of file baraydense-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 647 of file baraydense-meat.hpp.

### 8.9.3.5 col

col

Definition at line 835 of file barraydense-meat.hpp.

### 8.9.3.6 const

const

#### Initial value:

```
{  
    if (i >= N)  
        throw std::range_error("The row is out of range.")
```

Definition at line 352 of file barraydense-meat.hpp.

### 8.9.3.7 copy\_data

bool copy\_data

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

### 8.9.3.8 data

data = data\_

Definition at line 334 of file barraydense-meat.hpp.

### 8.9.3.9 delete\_data

delete\_data = delete\_data\_

Definition at line 335 of file barraydense-meat.hpp.

### 8.9.3.10 delete\_data\_

```
bool delete_data_
```

#### Initial value:

```
{  
    if ((data != nullptr) && delete_data)  
        delete data
```

Definition at line 328 of file baraydense-meat.hpp.

### 8.9.3.11 el

```
return el == ZERO_CELL
```

Definition at line 373 of file baraydense-meat.hpp.

### 8.9.3.12 el\_colsums

```
el_colsums[j] = (v.value - old)
```

Definition at line 667 of file baraydense-meat.hpp.

### 8.9.3.13 el\_rowsums

```
el_rowsums[i] = (v.value - old)
```

Definition at line 666 of file baraydense-meat.hpp.

### 8.9.3.14 else

```
else (  
    void )
```

#### Initial value:

```
{  
    Cell_Type old = el[POS(i,j)]
```

Definition at line 662 of file baraydense-meat.hpp.

### 8.9.3.15 false

false

Definition at line 759 of file barraydense-meat.hpp.

### 8.9.3.16 i1

uint i1

Definition at line 713 of file barraydense-meat.hpp.

### 8.9.3.17 j

j

Definition at line 365 of file barraydense-meat.hpp.

### 8.9.3.18 j0

uint j0

Definition at line 712 of file barraydense-meat.hpp.

### 8.9.3.19 j1

uint j1

Definition at line 713 of file barraydense-meat.hpp.

### 8.9.3.20 M

M = M\_

Definition at line 57 of file barraydense-meat.hpp.

### 8.9.3.21 M\_

```
uint M_
```

#### Initial value:

```
{  
    std::vector< Cell_Type > el_tmp(el)
```

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

### 8.9.3.22 N

```
N = N_
```

Definition at line 56 of file baraydense-meat.hpp.

### 8.9.3.23 report

```
uint uint uint bool int int* report
```

#### Initial value:

```
{  
  
    if (check_bounds) {  
        out_of_range(i0, j0);  
        out_of_range(i1, j1);  
    }  
  
    if (report != nullptr)  
        (*report) = EXISTS::BOTH
```

Definition at line 716 of file baraydense-meat.hpp.

### 8.9.3.24 return

```
return
```

Definition at line 94 of file baraydense-meat.hpp.

### 8.9.3.25 source

```
uint const std::vector< uint >& source
```

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

#### 8.9.3.26 target

```
uint const std::vector< uint > const std::vector< uint >& target
```

Definition at line 45 of file barraydense-meat.hpp.

#### 8.9.3.27 v

```
uint Cell_Type v
```

Definition at line 645 of file barraydense-meat.hpp.

#### 8.9.3.28 val0

```
Cell_Type val0 = el[POS(i0,j0)]
```

Definition at line 734 of file barraydense-meat.hpp.

#### 8.9.3.29 val1

```
Cell_Type val1 = el[POS(i1,j1)]
```

Definition at line 735 of file barraydense-meat.hpp.

#### 8.9.3.30 value

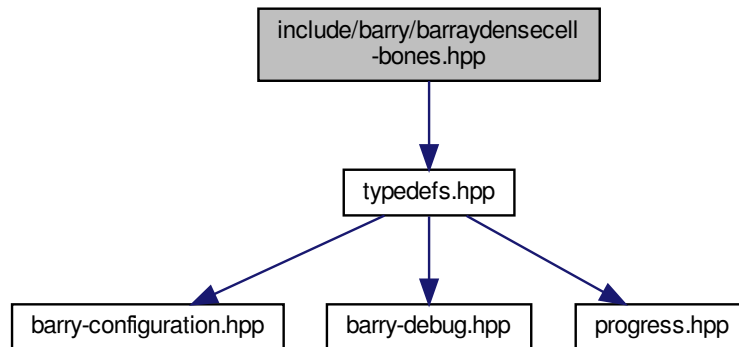
```
uint const std::vector< uint > const std::vector< uint > const std::vector< Cell_Type >&  
value
```

Definition at line 46 of file barraydense-meat.hpp.

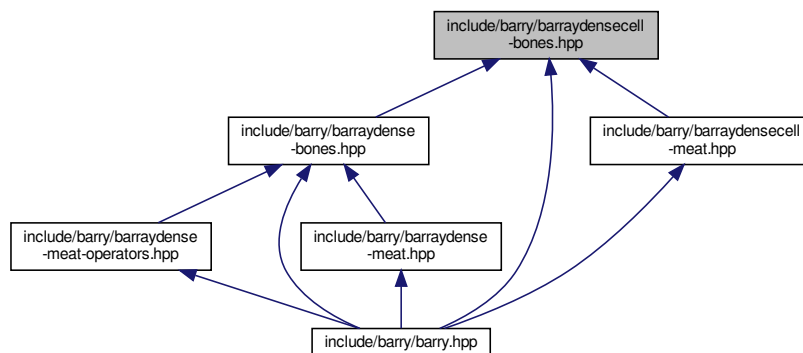
## 8.10 include/barry/barraydensecell-bones.hpp File Reference

```
#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 >`

### Macros

- `#define POS(a, b) (a) + (b) * N`

#### 8.10.1 Macro Definition Documentation



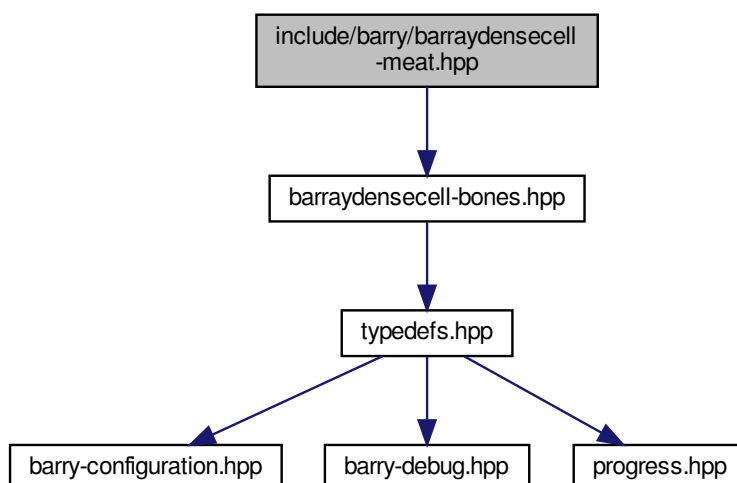
### 8.10.1.1 POS

```
#define POS(  
    a,  
    b ) (a) + (b) * N
```

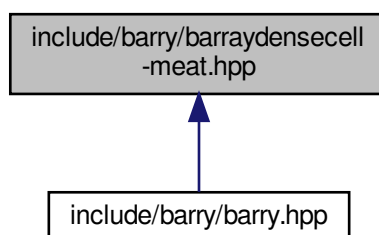
Definition at line 6 of file barraydensecell-bones.hpp.

## 8.11 include/barry/barraydensecell-meat.hpp File Reference

```
#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) * dat->N`

### 8.11.1 Macro Definition Documentation

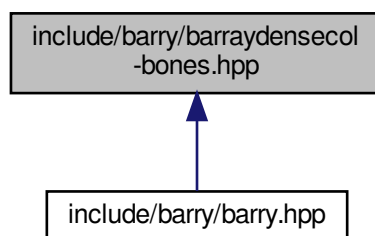
#### 8.11.1.1 POS

```
#define POS(  
    a,  
    b ) (a) + (b) * dat->N
```

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

## 8.12 `include/barry/barraydensecol-bones.hpp` File Reference

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



## Classes

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

## Macros

- `#define POS(a, b) (b)*N + (a)`
- `#define POS_N(a, b, c) (b)*(c) + (a)`
- `#define ZERO_CELL static_cast<Cell_Type>(0.0)`

## 8.12.1 Macro Definition Documentation

### 8.12.1.1 POS

```
#define POS(  
    a,  
    b ) (b)*N + (a)
```

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

### 8.12.1.2 POS\_N

```
#define POS_N(  
    a,  
    b,  
    c ) (b)*(c) + (a)
```

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

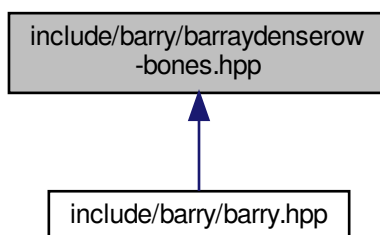
### 8.12.1.3 ZERO\_CELL

```
#define ZERO_CELL static_cast<Cell_Type>(0.0)
```

Definition at line 6 of file barraydenserow-bones.hpp.

## 8.13 include/barry/barraydenserow-bones.hpp File Reference

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



## Classes

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

## Macros

- #define [POS](#)(a, b) (b) \* [N](#) + (a)
- #define [POS\\_N](#)(a, b, c) (b)\*(c) + (a)
- #define [ZERO\\_CELL](#) static\_cast< Cell\_Type >(0.0)

### 8.13.1 Macro Definition Documentation

#### 8.13.1.1 POS

```
#define POS(  
    a,  
    b ) (b) * N + (a)
```

Definition at line 4 of file `barraydenserow-bones.hpp`.

#### 8.13.1.2 POS\_N

```
#define POS_N(  
    a,  
    b,  
    c ) (b)*(c) + (a)
```

Definition at line 5 of file `barraydenserow-bones.hpp`.

#### 8.13.1.3 ZERO\_CELL

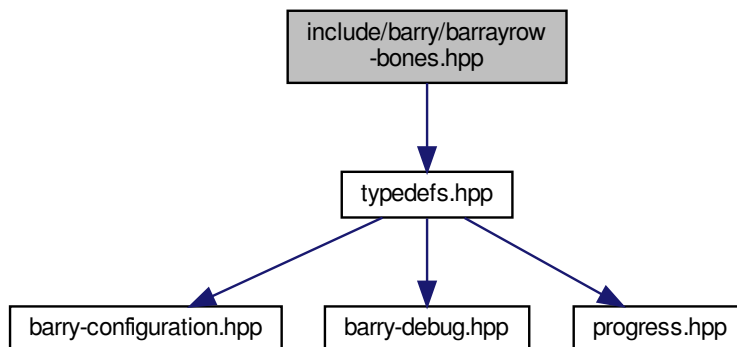
```
#define ZERO_CELL static_cast< Cell_Type >(0.0)
```

Definition at line 6 of file `barraydenserow-bones.hpp`.

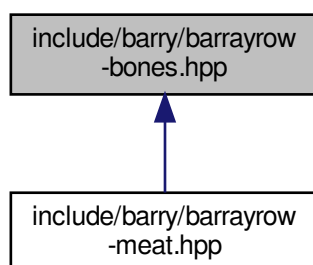
## 8.14 include/barry/barrayrow-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for barrayrow-bones.hpp:



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



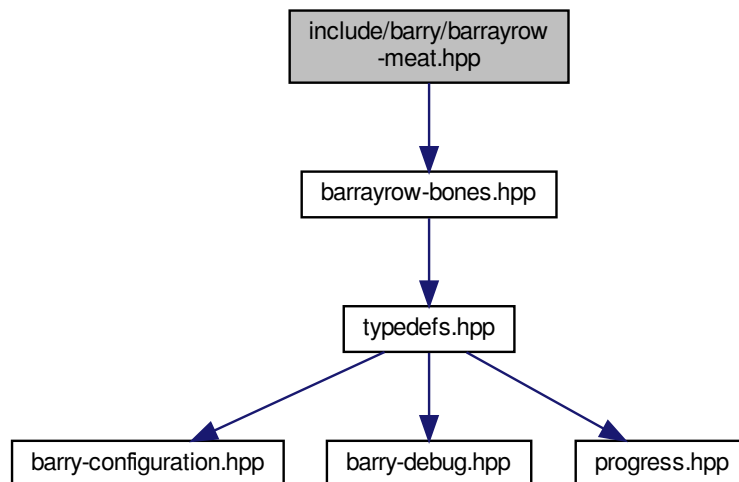
### Classes

- class `BArrayRow< Cell_Type, Data_Type >`
- class `BArrayRow_const< Cell_Type, Data_Type >`

## 8.15 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)`

### 8.15.1 Macro Definition Documentation

#### 8.15.1.1 BARRY\_BARRAYROW\_MEAT\_HPP

```
#define BARRY_BARRAYROW_MEAT_HPP 1
```

Definition at line 4 of file `barrayrow-meat.hpp`.

### 8.15.1.2 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.

### 8.15.1.3 BROW\_TEMPLATE\_ARGS

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

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

### 8.15.1.4 BROW\_TYPE

```
#define BROW_TYPE( ) BArrayRow<Cell_Type, Data_Type>
```

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

## 8.15.2 Function Documentation

### 8.15.2.1 BROW\_TEMPLATE() [1/5]

```
BROW_TEMPLATE (  
    void ,  
    operator* ) const &
```

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

### 8.15.2.2 BROW\_TEMPLATE() [2/5]

```
BROW_TEMPLATE (  
    void ,  
    operator+ ) const &
```

Definition at line 27 of file barrayrow-meat.hpp.

### 8.15.2.3 BROW\_TEMPLATE() [3/5]

```
BROW_TEMPLATE (
    void ,
    operator- ) const &
```

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

### 8.15.2.4 BROW\_TEMPLATE() [4/5]

```
BROW_TEMPLATE (
    void ,
    operator/ ) const &
```

Definition at line 57 of file bararrayrow-meat.hpp.

### 8.15.2.5 BROW\_TEMPLATE() [5/5]

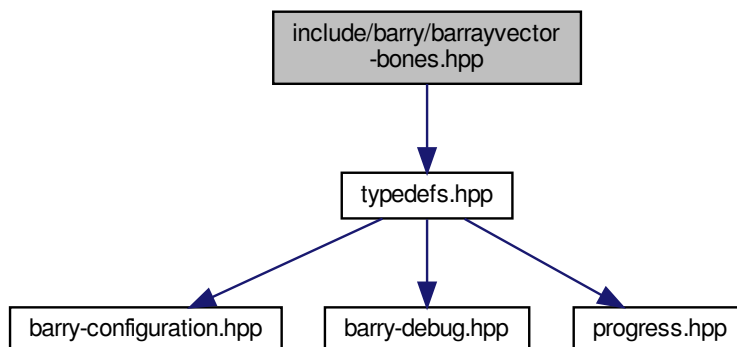
```
BROW_TEMPLATE (
    void ,
    operator ) const &
```

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

## 8.16 include/barry/barrayvector-bones.hpp File Reference

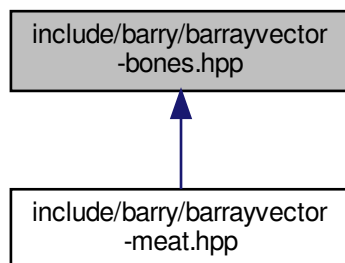
```
#include "typedefs.hpp"
```

Include dependency graph for barrayvector-bones.hpp:





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



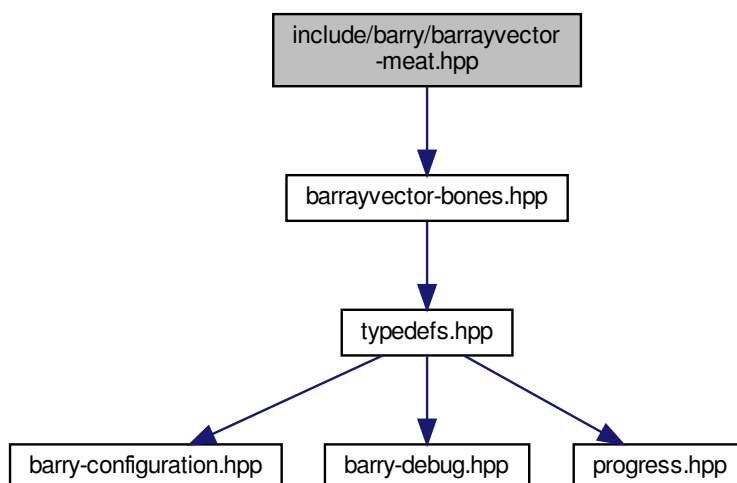
## Classes

- class [BArrayVector< Cell\\_Type, Data\\_Type >](#)  
*Row or column of a [BArray](#)*
- class [BArrayVector\\_const< Cell\\_Type, Data\\_Type >](#)

## 8.17 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`

### 8.17.1 Macro Definition Documentation

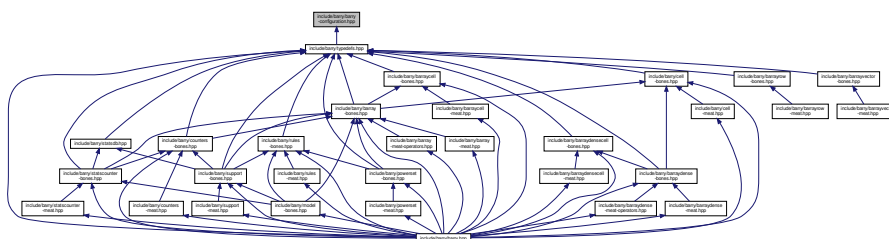
#### 8.17.1.1 BARRY\_BARRAYVECTOR\_MEAT\_HPP

```
#define BARRY_BARRAYVECTOR_MEAT_HPP 1
```

Definition at line 4 of file `barrayvector-meat.hpp`.

## 8.18 include/barry/barry-configuration.hpp File Reference

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



## Configuration MACROS

These are mostly related to performance. The definitions follow:

- `BARRY_USE_UNORDERED_MAP` If specified, then barry is compiled using `std::unordered_map`. Otherwise it will use `std::map` for the arrays.
- `BARRY_USE_SAFE_EXP` When specified, it will multiply all likelihoods in `Model` by  $(1/-100)/(-100)$  so that numerical overflows are avoided.
- `BARRY_USE_ISFINITE` When specified, it will introduce a macro that checks whether the likelihood is finite or not.
- `printf_barry` If not specified, will be defined as `printf`.
- `BARRY_DEBUG_LEVEL`, when defined, will make things verbose.
- `#define BARRY_SAFE_EXP -100.0`
- `#define BARRY_ISFINITE(a)`
- `#define BARRY_CHECK_SUPPORT(x, maxs)`
- `#define printf_barry printf`
- `#define BARRY_MAX_NUM_ELEMENTS static_cast< size_t >(UINT_MAX/2u)`
- `template<typename Ta, typename Tb >`  
`using Map = std::map< Ta, Tb >`

## 8.18.1 Macro Definition Documentation

### 8.18.1.1 BARRY\_CHECK\_SUPPORT

```
#define BARRY_CHECK_SUPPORT(  
    x,  
    maxs )
```

Definition at line 47 of file barry-configuration.hpp.

### 8.18.1.2 BARRY\_ISFINITE

```
#define BARRY_ISFINITE(  
    a )
```

Definition at line 40 of file barry-configuration.hpp.

### 8.18.1.3 BARRY\_MAX\_NUM\_ELEMENTS

```
#define BARRY_MAX_NUM_ELEMENTS static_cast< size_t > (UINT_MAX/2u)
```

Definition at line 55 of file barry-configuration.hpp.

### 8.18.1.4 BARRY\_SAFE\_EXP

```
#define BARRY_SAFE_EXP -100.0
```

Definition at line 33 of file barry-configuration.hpp.

### 8.18.1.5 printf\_barry

```
#define printf_barry printf
```

Definition at line 51 of file barry-configuration.hpp.

## 8.18.2 Typedef Documentation

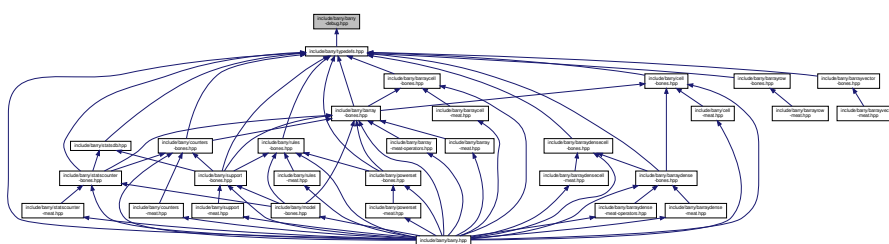
### 8.18.2.1 Map

```
template<typename Ta , typename Tb >
using Map = std::map<Ta,Tb>
```

Definition at line 27 of file barry-configuration.hpp.

## 8.19 include/barry/barry-debug.hpp File Reference

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



### Macros

- #define `BARRY_DEBUG_LEVEL` 0

### 8.19.1 Macro Definition Documentation

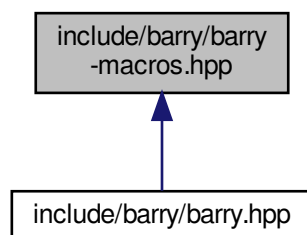
#### 8.19.1.1 BARRY\_DEBUG\_LEVEL

```
#define BARRY_DEBUG_LEVEL 0
```

Definition at line 5 of file barry-debug.hpp.

## 8.20 include/barry/barry-macros.hpp File Reference

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



## Macros

- `#define BARRY_ZERO Cell<Cell_Type>(0.0)`
- `#define BARRY_ZERO_DENSE static_cast<Cell_Type>(0.0)`
- `#define BARRY_ONE Cell<Cell_Type>(1.0)`
- `#define BARRY_ONE_DENSE static_cast<Cell_Type>(1.0)`
- `#define BARRY_UNUSED(expr) do { (void)(expr); } while (0);`

### 8.20.1 Macro Definition Documentation

#### 8.20.1.1 BARRY\_ONE

```
#define BARRY_ONE Cell<Cell_Type>(1.0)
```

Definition at line 7 of file barry-macros.hpp.

#### 8.20.1.2 BARRY\_ONE\_DENSE

```
#define BARRY_ONE_DENSE static_cast<Cell_Type>(1.0)
```

Definition at line 8 of file barry-macros.hpp.

#### 8.20.1.3 BARRY\_UNUSED

```
#define BARRY_UNUSED(  
    expr ) do { (void)(expr); } while (0);
```

Definition at line 10 of file barry-macros.hpp.

#### 8.20.1.4 BARRY\_ZERO

```
#define BARRY_ZERO Cell<Cell_Type>(0.0)
```

Definition at line 4 of file barry-macros.hpp.

### 8.20.1.5 BARRY\_ZERO\_DENSE

```
#define BARRY_ZERO_DENSE static_cast<Cell_Type>(0.0)
```

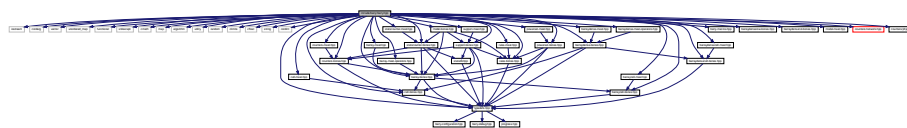
Definition at line 5 of file barry-macros.hpp.

## 8.21 include/barry/barry.hpp File Reference

```
#include <iostream>
#include <cstdint>
#include <vector>
#include <unordered_map>
#include <functional>
#include <stdexcept>
#include <cmath>
#include <map>
#include <algorithm>
#include <utility>
#include <random>
#include <climits>
#include <cfloat>
#include <string>
#include <cstdint>
#include "typedefs.hpp"
#include "barry-macros.hpp"
#include "cell-bones.hpp"
#include "cell-meat.hpp"
#include "barray-bones.hpp"
#include "barraycell-bones.hpp"
#include "barray-meat.hpp"
#include "barraycell-meat.hpp"
#include "barray-meat-operators.hpp"
#include "barraydense-bones.hpp"
#include "barraydensecell-bones.hpp"
#include "barraydenserow-bones.hpp"
#include "barraydensecol-bones.hpp"
#include "barraydense-meat.hpp"
#include "barraydensecell-meat.hpp"
#include "barraydense-meat-operators.hpp"
#include "counters-bones.hpp"
#include "counters-meat.hpp"
#include "statscounter-bones.hpp"
#include "statscounter-meat.hpp"
#include "support-bones.hpp"
#include "support-meat.hpp"
#include "powerset-bones.hpp"
#include "powerset-meat.hpp"
#include "model-bones.hpp"
#include "model-meat.hpp"
#include "rules-bones.hpp"
#include "rules-meat.hpp"
#include "counters/network.hpp"
```

```
#include "counters/phylo.hpp"
```

Include 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)`

### 8.21.1 Macro Definition Documentation

#### 8.21.1.1 BARRY\_HPP

```
#define BARRY_HPP
```

Definition at line 22 of file barry.hpp.

#### 8.21.1.2 BARRY\_VERSION

```
#define BARRY_VERSION 0.1
```

Definition at line 24 of file barry.hpp.

### 8.21.1.3 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 86 of file barry.hpp.

### 8.21.1.4 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 89 of file barry.hpp.

### 8.21.1.5 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 93 of file barry.hpp.

### 8.21.1.6 RULE\_LAMBDA

```
#define RULE_LAMBDA(  
    a )
```

**Value:**

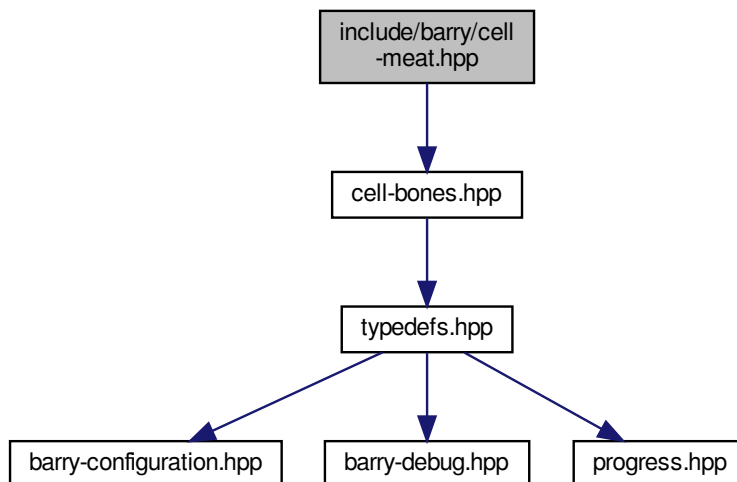
```
template <typename Array_Type = barry::BArray<>, typename Data_Type = bool> \  
Rule_fun_type<Array_Type, Data_Type> a = \  
[] (const Array_Type & Array, uint i, uint j, Data_Type * data) \  
{
```

Definition at line 96 of file barry.hpp.

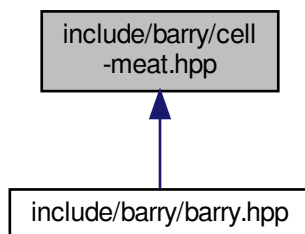




Include dependency graph for cell-meat.hpp:



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

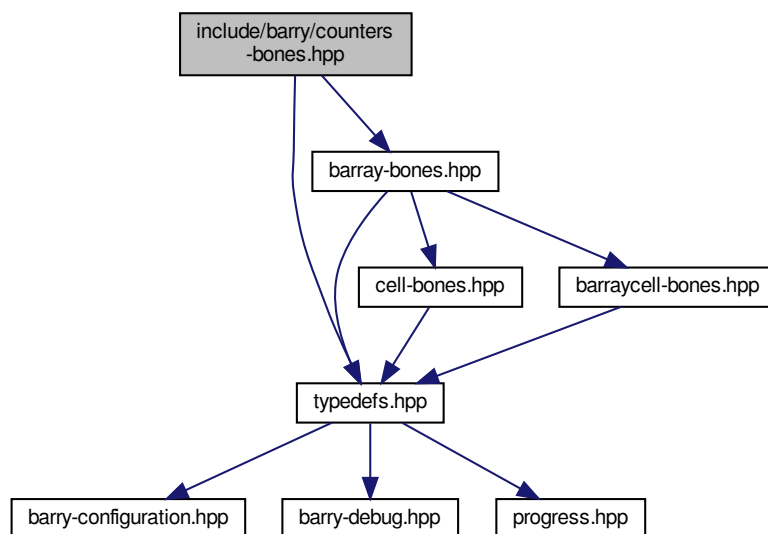


## 8.24 include/barry/col-bones.hpp File Reference

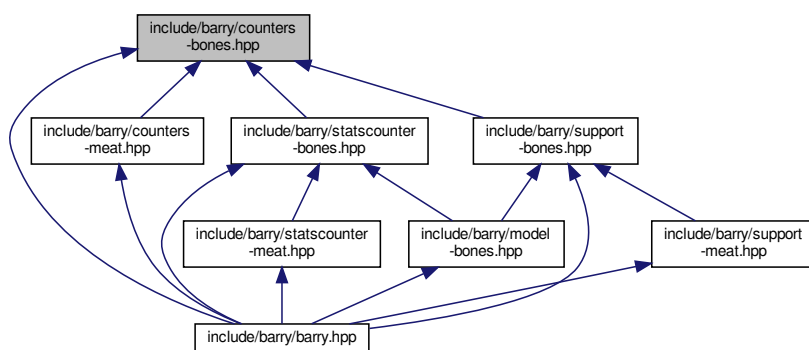
## 8.25 include/barry/counters-bones.hpp File Reference

```
#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:



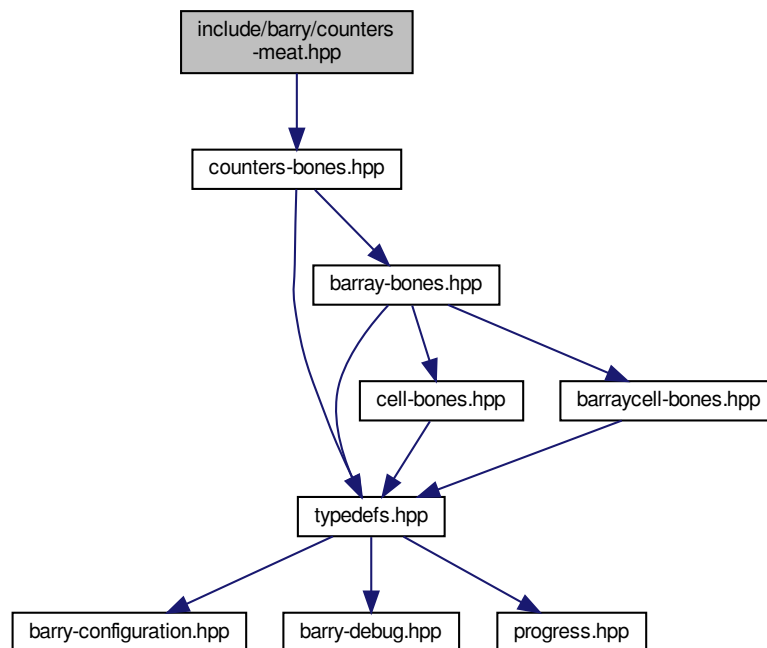
## Classes

- class `Counter< Array_Type, Data_Type >`  
A counter function based on change statistics.
- class `Counters< Array_Type, Data_Type >`  
Vector of counters.

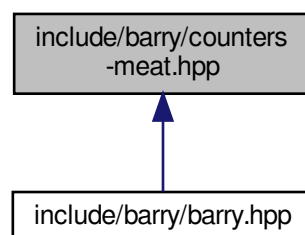
## 8.26 include/barry/counters-meat.hpp File Reference

```
#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\_

## 8.26.1 Macro Definition Documentation

### 8.26.1.1 COUNTER\_TEMPLATE

```
#define COUNTER_TEMPLATE(  
    a,  
    b )  template COUNTER_TEMPLATE_ARGS() inline a COUNTER_TYPE()::b
```

Definition at line 10 of file counters-meat.hpp.

### 8.26.1.2 COUNTER\_TEMPLATE\_ARGS

```
#define COUNTER_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

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

### 8.26.1.3 COUNTER\_TYPE

```
#define COUNTER_TYPE( ) Counter<Array_Type,Data_Type>
```

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

### 8.26.1.4 COUNTERS\_TEMPLATE

```
#define COUNTERS_TEMPLATE(  
    a,  
    b )  template COUNTERS_TEMPLATE_ARGS() inline a COUNTERS_TYPE()::b
```

Definition at line 155 of file counters-meat.hpp.

### 8.26.1.5 COUNTERS\_TEMPLATE\_ARGS

```
#define COUNTERS_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

Definition at line 153 of file counters-meat.hpp.

### 8.26.1.6 COUNTERS\_TYPE

```
#define COUNTERS_TYPE( ) Counters<Array_Type,Data_Type>
```

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

## 8.26.2 Function Documentation

### 8.26.2.1 count\_fun()

```
return count_fun (
    Array ,
    i ,
    j ,
    data )
```

### 8.26.2.2 COUNTER\_TEMPLATE() [1/7]

```
COUNTER_TEMPLATE (
    Counter ) const
```

### 8.26.2.3 COUNTER\_TEMPLATE() [2/7]

```
COUNTER_TEMPLATE (
    COUNTER_TYPE() & ,
    operator )
```

### 8.26.2.4 COUNTER\_TEMPLATE() [3/7]

```
COUNTER_TEMPLATE (
    COUNTER_TYPE() ,
    operator ) const
```

#### 8.26.2.5 COUNTER\_TEMPLATE() [4/7]

```
COUNTER_TEMPLATE (
    double ,
    count ) &
```

[< Move assignment](#)

#### 8.26.2.6 COUNTER\_TEMPLATE() [5/7]

```
COUNTER_TEMPLATE (
    double ,
    init ) &
```

#### 8.26.2.7 COUNTER\_TEMPLATE() [6/7]

```
COUNTER_TEMPLATE (
    std::string ,
    get_description ) const
```

Definition at line 143 of file counters-meat.hpp.

#### 8.26.2.8 COUNTER\_TEMPLATE() [7/7]

```
COUNTER_TEMPLATE (
    std::string ,
    get_name ) const
```

Definition at line 139 of file counters-meat.hpp.

#### 8.26.2.9 COUNTERS\_TEMPLATE() [1/8]

```
COUNTERS_TEMPLATE (
    Counters )
```

Definition at line 158 of file counters-meat.hpp.



**8.26.2.10 COUNTERS\_TEMPLATE()** [2/8]

```
COUNTERS_TEMPLATE (
    COUNTER_TYPE() & ,
    operator [] )
```

Definition at line 165 of file counters-meat.hpp.

**8.26.2.11 COUNTERS\_TEMPLATE()** [3/8]

```
COUNTERS_TEMPLATE (
    COUNTERS_TYPE() & ,
    operator )
```

**8.26.2.12 COUNTERS\_TEMPLATE()** [4/8]

```
COUNTERS_TEMPLATE (
    COUNTERS_TYPE() ,
    operator ) const
```

**8.26.2.13 COUNTERS\_TEMPLATE()** [5/8]

```
COUNTERS_TEMPLATE (
    std::vector< std::string > ,
    get_descriptions ) const
```

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

**8.26.2.14 COUNTERS\_TEMPLATE()** [6/8]

```
COUNTERS_TEMPLATE (
    std::vector< std::string > ,
    get_names ) const
```

Definition at line 344 of file counters-meat.hpp.

**8.26.2.15 COUNTERS\_TEMPLATE()** [7/8]

```
COUNTERS_TEMPLATE (
    void ,
    add_counter )
```

**8.26.2.16 COUNTERS\_TEMPLATE()** [8/8]

```
COUNTERS_TEMPLATE (
    void ,
    clear )
```

Definition at line 320 of file counters-meat.hpp.

**8.26.2.17 data()**

```
Data_Type&& counter_ data (
    std::move(counter_.data) )
```

**8.26.2.18 delete\_data()** [1/3]

```
Data_Type&& counter_ delete_data (
    std::move(counter_.delete_data) )
```

**8.26.2.19 delete\_data()** [2/3]

```
Data_Type&& counters_ delete_data (
    std::move(counters_.delete_data) )
```

**8.26.2.20 delete\_data()** [3/3]

```
Data_Type Data_Type delete_data (
    true )
```

**8.26.2.21 delete\_to\_be\_deleted() [1/2]**

```
Data_Type&& counters_ delete_to_be_deleted (
    std::move(counters_.delete_to_be_deleted) )
```

Definition at line 203 of file counters-meat.hpp.

**8.26.2.22 delete\_to\_be\_deleted() [2/2]**

```
Data_Type Data_Type delete_to_be_deleted (
    true )
```

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

**8.26.2.23 desc()**

```
Data_Type&& counter_ desc (
    std::move(counter_.desc) )
```

Move constructor.

Definition at line 48 of file counters-meat.hpp.

**8.26.2.24 init\_fun() [1/3]**

```
return init_fun (
    Array ,
    i ,
    j ,
    data )
```

**8.26.2.25 init\_fun() [2/3]**

```
Data_Type init_fun (
    counter_. init_fun )
```

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

**8.26.2.26 init\_fun() [3/3]**

```
Data_Type&& counter_ init_fun (
    std::move(counter_.init_fun) )
```

**8.26.2.27 name()**

```
Data_Type&& counter_ name (
    std::move(counter_.name) )
```

**8.26.2.28 push\_back() [1/2]**

```
data push_back (
    new Counter< Array_Type, Data_Type > count_fun_, init_fun_, data_, delete_data_,
    name_, desc_ )
```

**8.26.2.29 push\_back() [2/2]**

```
data push_back (
    new Counter< Array_Type, Data_Type > counter )
```

**8.26.2.30 to\_be\_deleted() [1/2]**

```
Data_Type Data_Type to_be_deleted (
    new std::vector< uint > 0u )
```

**8.26.2.31 to\_be\_deleted() [2/2]**

```
Data_Type&& counters_ to_be_deleted (
    std::move(counters_.to_be_deleted) )
```

**8.26.3 Variable Documentation**

### 8.26.3.1 count\_fun\_

Data\_Type count\_fun\_

Definition at line 293 of file counters-meat.hpp.

### 8.26.3.2 counter

Data\_Type \* counter

#### Initial value:

```
{  
    to_be_deleted->push_back(data->size())
```

Definition at line 275 of file counters-meat.hpp.

### 8.26.3.3 counter\_

Data\_Type & counter\_

#### Initial value:

```
{  
    if (this != &counter_) {  
        this->count_fun = counter_.count_fun;  
        this->init_fun = counter_.init_fun;  
        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.

### 8.26.3.4 data\_

Data\_Type Counter\_fun\_type<Array\_Type,Data\_Type> Data\_Type\* data\_

Definition at line 295 of file counters-meat.hpp.

### 8.26.3.5 delete\_data\_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> Data_Type bool delete_data_
```

Definition at line 296 of file counters-meat.hpp.

### 8.26.3.6 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 298 of file counters-meat.hpp.

### 8.26.3.7 i

```
uint i
```

Definition at line 119 of file counters-meat.hpp.

### 8.26.3.8 init\_fun\_

```
Data_Type Counter_fun_type<Array_Type,Data_Type> init_fun_
```

Definition at line 294 of file counters-meat.hpp.

### 8.26.3.9 j

```
uint uint j
```

#### Initial value:

```
{  
    if (count_fun == nullptr)  
        return 0.0
```

Definition at line 119 of file counters-meat.hpp.

### 8.26.3.10 name\_

Data\_Type Counter\_fun\_type<Array\_Type,Data\_Type> Data\_Type bool std::string name\_

Definition at line 297 of file counters-meat.hpp.

### 8.26.3.11 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 42 of file counters-meat.hpp.

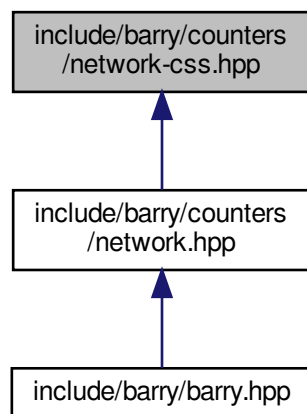
### 8.26.3.12 return

return

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

## 8.27 include/barry/counters/network-css.hpp File Reference

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



### Macros

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

### Functions

- `template<typename Tnet = Network>`  
`void counter_css_partially_false_recip_commi (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)`  
*Counts errors of commission.*
- `template<typename Tnet = Network>`  
`void counter_css_partially_false_recip_omiss (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)`  
*Counts errors of omission.*
- `template<typename Tnet = Network>`  
`void counter_css_completely_false_recip_comiss (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)`  
*Counts completely false reciprocity (comission)*



- `template<typename Tnet = Network>`  
`void counter_css_completely_false_recip_omiss (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)`  
*Counts completely false reciprocity (omission)*
- `template<typename Tnet = Network>`  
`void counter_css_mixed_recip (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)`  
*Counts mixed reciprocity errors.*
- `template<typename Tnet = Network>`  
`void counter_css_census01 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)`
- `template<typename Tnet = Network>`  
`void counter_css_census02 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)`
- `template<typename Tnet = Network>`  
`void counter_css_census03 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)`
- `template<typename Tnet = Network>`  
`void counter_css_census04 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)`
- `template<typename Tnet = Network>`  
`void counter_css_census05 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)`
- `template<typename Tnet = Network>`  
`void counter_css_census06 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)`
- `template<typename Tnet = Network>`  
`void counter_css_census07 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)`
- `template<typename Tnet = Network>`  
`void counter_css_census08 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)`
- `template<typename Tnet = Network>`  
`void counter_css_census09 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)`
- `template<typename Tnet = Network>`  
`void counter_css_census10 (NetCounters< Tnet > *counters, uint netsize, const std::vector< uint > &end_)`

## 8.27.1 Macro Definition Documentation

### 8.27.1.1 CSS\_APPEND

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

#### Value:

```
std::string name_ = (name);\nfor (uint i = 0u; i < end_.size(); ++i) { \n\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.

### 8.27.1.2 CSS\_CASE\_ELSE

```
#define CSS_CASE_ELSE( )
```

Definition at line 27 of file network-css.hpp.

### 8.27.1.3 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.

### 8.27.1.4 CSS\_CASE\_TRUTH

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

Definition at line 13 of file network-css.hpp.

### 8.27.1.5 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.

### 8.27.1.6 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.

### 8.27.1.7 CSS\_NET\_COUNTER\_LAMBDA\_INIT

```
#define CSS_NET_COUNTER_LAMBDA_INIT( )
```

**Value:**

```
NETWORK_COUNTER_LAMBDA(tmp_init) {\n    CSS_CHECK_SIZE_INIT() \n    return 0.0; \n};
```

Definition at line 49 of file network-css.hpp.

### 8.27.1.8 CSS\_PERCEIVED\_CELLS

```
#define CSS_PERCEIVED_CELLS( )
```

**Value:**

```
double tji = static_cast<double>(Array(j - s, i - s, false)); \ndouble pji = static_cast<double>(Array(j, i, false)); \ndouble tij = static_cast<double>(Array(i - s, j - s, false));
```

Definition at line 21 of file network-css.hpp.

### 8.27.1.9 CSS\_SIZE

```
#define CSS_SIZE( )
```

**Value:**

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

Definition at line 7 of file network-css.hpp.

### 8.27.1.10 CSS\_TRUE\_CELLS

```
#define CSS_TRUE_CELLS( )
```

**Value:**

```
double tji = static_cast<double>(Array(j, i, false)); \ndouble pij = static_cast<double>(Array(i + s, j + s, false)); \ndouble pji = static_cast<double>(Array(j + s, i + s, false));
```

Definition at line 14 of file network-css.hpp.

## 8.27.2 Function Documentation

### 8.27.2.1 counter\_css\_census01()

```
template<typename Tnet = Network>
void counter_css_census01 (
    NetCounters< Tnet > * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 275 of file network-css.hpp.

### 8.27.2.2 counter\_css\_census02()

```
template<typename Tnet = Network>
void counter_css_census02 (
    NetCounters< Tnet > * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 325 of file network-css.hpp.

### 8.27.2.3 counter\_css\_census03()

```
template<typename Tnet = Network>
void counter_css_census03 (
    NetCounters< Tnet > * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 364 of file network-css.hpp.

### 8.27.2.4 counter\_css\_census04()

```
template<typename Tnet = Network>
void counter_css_census04 (
    NetCounters< Tnet > * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 403 of file network-css.hpp.

### 8.27.2.5 counter\_css\_census05()

```
template<typename Tnet = Network>
void counter_css_census05 (
    NetCounters< Tnet > * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 442 of file network-css.hpp.

### 8.27.2.6 counter\_css\_census06()

```
template<typename Tnet = Network>
void counter_css_census06 (
    NetCounters< Tnet > * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 481 of file network-css.hpp.

### 8.27.2.7 counter\_css\_census07()

```
template<typename Tnet = Network>
void counter_css_census07 (
    NetCounters< Tnet > * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 520 of file network-css.hpp.

### 8.27.2.8 counter\_css\_census08()

```
template<typename Tnet = Network>
void counter_css_census08 (
    NetCounters< Tnet > * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 559 of file network-css.hpp.

### 8.27.2.9 counter\_css\_census09()

```
template<typename Tnet = Network>
void counter_css_census09 (
    NetCounters< Tnet > * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 598 of file network-css.hpp.

### 8.27.2.10 counter\_css\_census10()

```
template<typename Tnet = Network>
void counter_css_census10 (
    NetCounters< Tnet > * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Definition at line 637 of file network-css.hpp.

### 8.27.2.11 counter\_css\_completely\_false\_recip\_comiss()

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

Counts completely false reciprocity (comission)

Definition at line 154 of file network-css.hpp.

### 8.27.2.12 counter\_css\_completely\_false\_recip\_omiss()

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

Counts completely false reciprocity (omission)

Definition at line 194 of file network-css.hpp.

**8.27.2.13 counter\_css\_mixed\_recip()**

```
template<typename Tnet = Network>
void counter_css_mixed_recip (
    NetCounters< Tnet > * counters,
    uint netsize,
    const std::vector< uint > & end_ ) [inline]
```

Counts mixed reciprocity errors.

Definition at line 234 of file network-css.hpp.

**8.27.2.14 counter\_css\_partially\_false\_recip\_commi()**

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

Counts errors of commission.

**Parameters**

<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.

**8.27.2.15 counter\_css\_partially\_false\_recip\_omiss()**

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

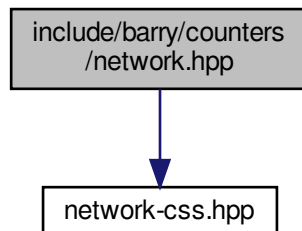
Counts errors of omission.

Definition at line 110 of file network-css.hpp.

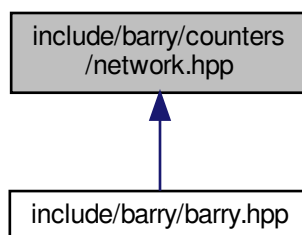
## 8.28 include/barry/counters/network.hpp File Reference

```
#include "network-css.hpp"
```

Include dependency graph for network.hpp:



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



### Classes

- class [NetworkData](#)  
*Data class for Networks.*
- class [NetCounterData](#)  
*Data class used to store arbitrary uint or double vectors.*

### Macros

- #define [NET\\_C\\_DATA\\_IDX\(i\)](#) ([data](#)->indices[i])
- #define [NET\\_C\\_DATA\\_NUM\(i\)](#) ([data](#)->numbers[i])



**Macros for defining counters**

- #define `NETWORK_COUNTER(a)`
- #define `NETWORK_COUNTER_LAMBDA(a)`
- #define `NETWORKDENSE_COUNTER_LAMBDA(a)`

**Macros for defining rules**

- #define `NETWORK_RULE(a)`
- #define `NETWORK_RULE_LAMBDA(a)`

**Functions**

- template<typename Tnet = Network>  
void `counter_edges` (NetCounters< Tnet > \*counters)  
*Number of edges.*
- template<typename Tnet = Network>  
void `counter_isolates` (NetCounters< Tnet > \*counters)  
*Number of isolated vertices.*
- template<> void `counter_isolates` (NetCounters< NetworkDense > \*counters)
- template<typename Tnet = Network>  
void `counter_mutual` (NetCounters< Tnet > \*counters)  
*Number of mutual ties.*
- template<typename Tnet = Network>  
void `counter_istar2` (NetCounters< Tnet > \*counters)
- template<> void `counter_istar2` (NetCounters< NetworkDense > \*counters)
- template<typename Tnet = Network>  
void `counter_ostar2` (NetCounters< Tnet > \*counters)
- template<> void `counter_ostar2` (NetCounters< NetworkDense > \*counters)
- template<typename Tnet = Network>  
void `counter_ttriads` (NetCounters< Tnet > \*counters)
- template<> void `counter_ttriads` (NetCounters< NetworkDense > \*counters)
- template<typename Tnet = Network>  
void `counter_ctriads` (NetCounters< Tnet > \*counters)
- template<> void `counter_ctriads` (NetCounters< NetworkDense > \*counters)
- template<typename Tnet = Network>  
void `counter_density` (NetCounters< Tnet > \*counters)
- template<typename Tnet = Network>  
void `counter_idegree15` (NetCounters< Tnet > \*counters)
- template<> void `counter_idegree15` (NetCounters< NetworkDense > \*counters)
- template<typename Tnet = Network>  
void `counter_odegree15` (NetCounters< Tnet > \*counters)
- template<> void `counter_odegree15` (NetCounters< NetworkDense > \*counters)
- template<typename Tnet = Network>  
void `counter_absdiff` (NetCounters< Tnet > \*counters, uint attr\_id, double alpha=1.0)  
*Sum of absolute attribute difference between ego and alter.*
- template<typename Tnet = Network>  
void `counter_diff` (NetCounters< Tnet > \*counters, uint attr\_id, double alpha=1.0, double tail\_head=true)  
*Sum of attribute difference between ego and alter to pow(alpha)*
- `NETWORK_COUNTER` (init\_single\_attr)
- template<typename Tnet = Network>  
void `counter_nodeicov` (NetCounters< Tnet > \*counters, uint attr\_id)
- template<typename Tnet = Network>  
void `counter_nodeocov` (NetCounters< Tnet > \*counters, uint attr\_id)

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

### Rules for network models

#### Parameters

rules	A pointer to a <i>NetRules</i> object ( <i>Rules&lt;Network, bool&gt;</i> ).
-------	--

- `template<typename Tnet = Network>`  
`void rules_zerodiag (NetRules< Tnet > *rules)`  
*Number of edges.*

### Convenient typedefs for network objects.

- `#define BARRY_ZERO_NETWORK 0.0`
- `#define BARRY_ZERO_NETWORK_DENSE 0`
- `typedef BArray< double, NetworkData > Network`
- `typedef BArrayDense< int, NetworkData > NetworkDense`
- `template<typename Tnet = Network>`  
`using NetCounter = Counter< Tnet, NetCounterData >`
- `template<typename Tnet = Network>`  
`using NetCounters = Counters< Tnet, NetCounterData >`
- `template<typename Tnet = Network>`  
`using NetSupport = Support< Tnet, NetCounterData >`
- `template<typename Tnet = Network>`  
`using NetStatsCounter = StatsCounter< Tnet, NetCounterData >`
- `template<typename Tnet >`  
`using NetModel = Model< Tnet, NetCounterData >`
- `template<typename Tnet = Network>`  
`using NetRule = Rule< Tnet, bool >`
- `template<typename Tnet = Network>`  
`using NetRules = Rules< Tnet, bool >`

## 8.28.1 Macro Definition Documentation

### 8.28.1.1 BARRY\_ZERO\_NETWORK

```
#define BARRY_ZERO_NETWORK 0.0
```

Definition at line 85 of file network.hpp.

### 8.28.1.2 BARRY\_ZERO\_NETWORK\_DENSE

```
#define BARRY_ZERO_NETWORK_DENSE 0
```

Definition at line 86 of file network.hpp.

### 8.28.1.3 NET\_C\_DATA\_IDX

```
#define NET_C_DATA_IDX(  
    i ) (data->indices[i])
```

Definition at line 74 of file network.hpp.

### 8.28.1.4 NET\_C\_DATA\_NUM

```
#define NET_C_DATA_NUM(  
    i ) (data->numbers[i])
```

Definition at line 75 of file network.hpp.

### 8.28.1.5 NETWORK\_COUNTER

```
#define NETWORK_COUNTER(  
    a )
```

#### Value:

```
template<typename Tnet = Network>\ninline double (a) (const Tnet & Array, uint i, uint j, NetCounterData * data)
```

Function for definition of a network counter function

Definition at line 114 of file network.hpp.

### 8.28.1.6 NETWORK\_COUNTER\_LAMBDA

```
#define NETWORK_COUNTER_LAMBDA(  
    a )
```

**Value:**

```
Counter_fun_type<Tnet, NetCounterData> a = \  
    [] (const Tnet & Array, uint i, uint j, NetCounterData * data)
```

Lambda function for definition of a network counter function

Definition at line 119 of file network.hpp.

### 8.28.1.7 NETWORK\_RULE

```
#define NETWORK_RULE(  
    a )
```

**Value:**

```
template<typename Tnet = Network>\  
inline bool (a) (const Tnet & Array, uint i, uint j, bool * data)
```

Function for definition of a network counter function

Definition at line 133 of file network.hpp.

### 8.28.1.8 NETWORK\_RULE\_LAMBDA

```
#define NETWORK_RULE_LAMBDA(  
    a )
```

**Value:**

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

Lambda function for definition of a network counter function

Definition at line 138 of file network.hpp.

### 8.28.1.9 NETWORKDENSE\_COUNTER\_LAMBDA

```
#define NETWORKDENSE_COUNTER_LAMBDA(  
    a )
```

**Value:**

```
Counter_fun_type<NetworkDense, NetCounterData> a = \  
    [] (const NetworkDense & Array, uint i, uint j, NetCounterData * data)
```

Definition at line 123 of file network.hpp.

## 8.28.2 Typedef Documentation

### 8.28.2.1 NetCounter

```
template<typename Tnet = Network>  
using NetCounter = Counter<Tnet, NetCounterData >
```

Definition at line 89 of file network.hpp.

### 8.28.2.2 NetCounters

```
template<typename Tnet = Network>  
using NetCounters = Counters<Tnet, NetCounterData>
```

Definition at line 92 of file network.hpp.

### 8.28.2.3 NetModel

```
template<typename Tnet >  
using NetModel = Model<Tnet, NetCounterData>
```

Definition at line 101 of file network.hpp.

### 8.28.2.4 NetRule

```
template<typename Tnet = Network>  
using NetRule = Rule<Tnet, bool>
```

Definition at line 104 of file network.hpp.

### 8.28.2.5 NetRules

```
template<typename Tnet = Network>  
using NetRules = Rules<Tnet, bool>
```

Definition at line 107 of file network.hpp.

### 8.28.2.6 NetStatsCounter

```
template<typename Tnet = Network>
using NetStatsCounter = StatsCounter<Tnet, NetCounterData>
```

Definition at line 98 of file network.hpp.

### 8.28.2.7 NetSupport

```
template<typename Tnet = Network>
using NetSupport = Support<Tnet, NetCounterData >
```

Definition at line 95 of file network.hpp.

### 8.28.2.8 Network

```
typedef BArray<double, NetworkData> Network
```

Definition at line 82 of file network.hpp.

### 8.28.2.9 NetworkDense

```
typedef BArrayDense<int, NetworkData> NetworkDense
```

Definition at line 83 of file network.hpp.

## 8.28.3 Function Documentation

### 8.28.3.1 rules\_zerodiag()

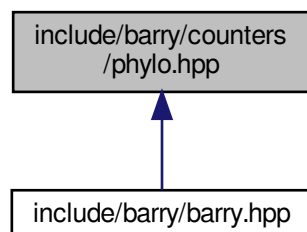
```
template<typename Tnet = Network>
void rules_zerodiag (
    NetRules< Tnet > * rules ) [inline]
```

Number of edges.

Definition at line 1325 of file network.hpp.

## 8.29 include/barry/counters/phylo.hpp File Reference

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



### Classes

- class [NodeData](#)  
*Data definition for the `PhyloArray` class.*
- class [PhyloCounterData](#)
- class [PhyloRuleDynData](#)

### Macros

- `#define` [DEFAULT\\_DUPLICATION](#) 1u
- `#define` [DUPL\\_SPEC](#) 0u
- `#define` [DUPL\\_DUPL](#) 1u
- `#define` [DUPL\\_EITH](#) 2u
- `#define` [MAKE\\_DUPL\\_VARS](#)()
- `#define` [IS\\_EITHER](#)() (DATA\_AT == [DUPL\\_EITH](#))
- `#define` [IS\\_DUPLICATION](#)() ((DATA\_AT == [DUPL\\_DUPL](#)) & (DPL))
- `#define` [IS\\_SPECIATION](#)() ((DATA\_AT == [DUPL\\_SPEC](#)) & (!DPL))
- `#define` [IF\\_MATCHES](#)()
- `#define` [IF\\_NOTMATCHES](#)()
- `#define` [PHYLO\\_COUNTER\\_LAMBDA](#)(a)  
*Extension of a simple counter.*
- `#define` [PHYLO\\_RULE\\_DYN\\_LAMBDA](#)(a)
- `#define` [PHYLO\\_CHECK\\_MISSING](#)()

### Typedefs

- `typedef` `std::vector< std::pair< uint, uint > >` [PhyloRuleData](#)

### Convenient typedefs for Node objects.

- typedef `BArrayDense< uint, NodeData >` `PhyloArray`
- typedef `Counter< PhyloArray, PhyloCounterData >` `PhyloCounter`
- typedef `Counters< PhyloArray, PhyloCounterData >` `PhyloCounters`
- typedef `Rule< PhyloArray, PhyloRuleData >` `PhyloRule`
- typedef `Rules< PhyloArray, PhyloRuleData >` `PhyloRules`
- typedef `Rule< PhyloArray, PhyloRuleDynData >` `PhyloRuleDyn`
- typedef `Rules< PhyloArray, PhyloRuleDynData >` `PhyloRulesDyn`
- typedef `Support< PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData >` `PhyloSupport`
- typedef `StatsCounter< PhyloArray, PhyloCounterData >` `PhyloStatsCounter`
- typedef `Model< PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData >` `PhyloModel`
- typedef `PowerSet< PhyloArray, PhyloRuleData >` `PhyloPowerSet`

### Functions

- std::string `get_last_name` (unsigned int d)
- void `counter_overall_gains` (`PhyloCounters *counters`, unsigned int duplication=`DEFAULT_DUPLICATION`)  
*Overall functional gains.*
- void `counter_gains` (`PhyloCounters *counters`, std::vector< uint > nfun, unsigned int duplication=`DEFAULT_DUPLICATION`)  
*Functional gains for a specific function (nfun).*
- void `counter_gains_k_offspring` (`PhyloCounters *counters`, std::vector< uint > nfun, uint k=1u, unsigned int duplication=`DEFAULT_DUPLICATION`)  
*k genes gain function nfun*
- void `counter_genes_changing` (`PhyloCounters *counters`, unsigned int duplication=`DEFAULT_DUPLICATION`)  
*Keeps track of how many genes are changing (either 0, 1, or 2 if dealing with regular trees.)*
- void `counter_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 `counter_less_than_p_prop_genes_changing` (`PhyloCounters *counters`, double p, unsigned int duplication=`DEFAULT_DUPLICATION`)  
*Indicator function. Equals to one if k genes changed and zero otherwise.*



- void `counter_gains_from_0` (`PhyloCounters` \*counters, `std::vector< uint >` nfun, unsigned int duplication=`DEFAULT_DUPLICATION`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `counter_pairwise_preserving` (`PhyloCounters` \*counters, `uint` nfunA, `uint` nfunB, unsigned int duplication=`DEFAULT_DUPLICATION`)  
*Used when all the functions are in 0 (like the root node prob.)*
- void `rule_dyn_limit_changes` (`PhyloSupport` \*support, `uint` pos, `uint` lb, `uint` ub, unsigned int duplication=`DEFAULT_DUPLICATION`)  
*Overall functional gains.*

## 8.29.1 Macro Definition Documentation

### 8.29.1.1 DEFAULT\_DUPLICATION

```
#define DEFAULT_DUPLICATION 1u
```

Definition at line 5 of file phylo.hpp.

### 8.29.1.2 DUPL\_DUPL

```
#define DUPL_DUPL 1u
```

Definition at line 7 of file phylo.hpp.

### 8.29.1.3 DUPL\_EITH

```
#define DUPL_EITH 2u
```

Definition at line 8 of file phylo.hpp.

### 8.29.1.4 DUPL\_SPEC

```
#define DUPL_SPEC 0u
```

Definition at line 6 of file phylo.hpp.

### 8.29.1.5 IF\_MATCHES

```
#define IF_MATCHES( )
```

**Value:**

```
MAKE_DUPL_VARS() \
if (IS_EITHER() | IS_DUPLICATION() | IS_SPECIATION())
```

Definition at line 19 of file phylo.hpp.

### 8.29.1.6 IF\_NOTMATCHES

```
#define IF_NOTMATCHES( )
```

**Value:**

```
MAKE_DUPL_VARS() \
if (!IS_EITHER() & !IS_DUPLICATION() & !IS_SPECIATION())
```

Definition at line 21 of file phylo.hpp.

### 8.29.1.7 IS\_DUPLICATION

```
#define IS_DUPLICATION( ) ((DATA_AT == DUPL_DUPL) & (DPL))
```

Definition at line 16 of file phylo.hpp.

### 8.29.1.8 IS\_EITHER

```
#define IS_EITHER( ) (DATA_AT == DUPL_EITH)
```

Definition at line 15 of file phylo.hpp.

### 8.29.1.9 IS\_SPECIATION

```
#define IS_SPECIATION( ) ((DATA_AT == DUPL_SPEC) & (!DPL))
```

Definition at line 17 of file phylo.hpp.

### 8.29.1.10 MAKE\_DUPL\_VARS

```
#define MAKE_DUPL_VARS( )
```

**Value:**

```
bool DPL = Array.D()->duplication; \
unsigned int DATA_AT = data->operator[](0u);
```

Definition at line 11 of file phylo.hpp.

### 8.29.1.11 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 137 of file phylo.hpp.

### 8.29.1.12 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 131 of file phylo.hpp.

### 8.29.1.13 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 134 of file phylo.hpp.

## 8.29.2 Typedef Documentation

### 8.29.2.1 PhyloArray

```
typedef BArrayDense<uint, NodeData> PhyloArray
```

Definition at line 104 of file phylo.hpp.

### 8.29.2.2 PhyloCounter

```
typedef Counter<PhyloArray, PhyloCounterData > PhyloCounter
```

Definition at line 105 of file phylo.hpp.

### 8.29.2.3 PhyloCounters

```
typedef Counters< PhyloArray, PhyloCounterData> PhyloCounters
```

Definition at line 106 of file phylo.hpp.

### 8.29.2.4 PhyloModel

```
typedef Model<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloModel
```

Definition at line 116 of file phylo.hpp.

### 8.29.2.5 PhyloPowerSet

```
typedef PowerSet<PhyloArray, PhyloRuleData> PhyloPowerSet
```

Definition at line 117 of file phylo.hpp.

### 8.29.2.6 PhyloRule

```
typedef Rule<PhyloArray,PhyloRuleData> PhyloRule
```

Definition at line 108 of file phylo.hpp.

### 8.29.2.7 PhyloRuleData

```
typedef std::vector< std::pair< uint, uint > > PhyloRuleData
```

Definition at line 97 of file phylo.hpp.

### 8.29.2.8 PhyloRuleDyn

```
typedef Rule<PhyloArray,PhyloRuleDynData> PhyloRuleDyn
```

Definition at line 111 of file phylo.hpp.

### 8.29.2.9 PhyloRules

```
typedef Rules<PhyloArray,PhyloRuleData> PhyloRules
```

Definition at line 109 of file phylo.hpp.

### 8.29.2.10 PhyloRulesDyn

```
typedef Rules<PhyloArray,PhyloRuleDynData> PhyloRulesDyn
```

Definition at line 112 of file phylo.hpp.

### 8.29.2.11 PhyloStatsCounter

```
typedef StatsCounter<PhyloArray, PhyloCounterData> PhyloStatsCounter
```

Definition at line 115 of file phylo.hpp.

### 8.29.2.12 PhyloSupport

```
typedef Support<PhyloArray, PhyloCounterData, PhyloRuleData, PhyloRuleDynData > PhyloSupport
```

Definition at line 114 of file phylo.hpp.

## 8.29.3 Function Documentation

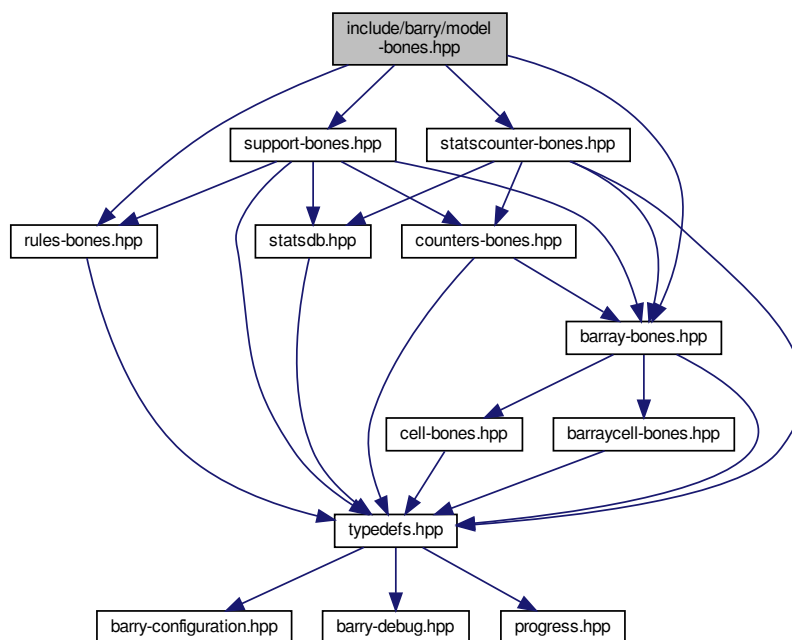
### 8.29.3.1 get\_last\_name()

```
std::string get_last_name (
    unsigned int d ) [inline]
```

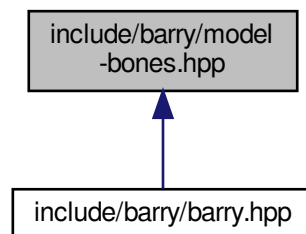
Definition at line 142 of file phylo.hpp.

## 8.30 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)*

### 8.30.1 Function Documentation

#### 8.30.1.1 [keygen\\_default\(\)](#)

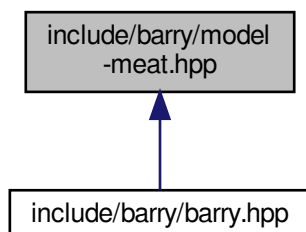
```
template<typename Array_Type >
std::vector< double > keygen_default (
    const Array_Type & Array_ ) [inline]
```

Array Hasher class (used for computing support)

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

## 8.31 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 std::vector< double > &support)`
- `double likelihood_ (const std::vector< double > &stats_target, const std::vector< double > &params, const double normalizing_constant, bool log_=false)`
- `MODEL_TEMPLATE (, Model)()`
- `MODEL_TEMPLATE (, Model)(const MODEL_TYPE() &Model_)`

### 8.31.1 Macro Definition Documentation

#### 8.31.1.1 MODEL\_TEMPLATE

```

#define MODEL_TEMPLATE(
    a,
    b )  template MODEL_TEMPLATE_ARGS() inline a MODEL_TYPE()::b
  
```

Definition at line 97 of file model-meat.hpp.



### 8.31.1.2 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 94 of file model-meat.hpp.

### 8.31.1.3 MODEL\_TYPE

```
#define MODEL_TYPE( )
```

**Value:**

```
Model<Array_Type, Data_Counter_Type, Data_Rule_Type,\  
  Data_Rule_Dyn_Type>
```

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

## 8.31.2 Function Documentation

### 8.31.2.1 likelihood\_()

```
double likelihood_ (  
    const std::vector< double > & stats_target,  
    const std::vector< double > & params,  
    const double normalizing_constant,  
    bool log_ = false ) [inline]
```

Definition at line 56 of file model-meat.hpp.

### 8.31.2.2 MODEL\_TEMPLATE() [1/2]

```
MODEL_TEMPLATE (  
    Model )
```

Definition at line 101 of file model-meat.hpp.

### 8.31.2.3 MODEL\_TEMPLATE() [2/2]

```
MODEL_TEMPLATE (
    Model ) const &
```

Definition at line 159 of file model-meat.hpp.

### 8.31.2.4 update\_normalizing\_constant()

```
double update_normalizing_constant (
    const std::vector< double > & params,
    const std::vector< double > & support ) [inline]
```

Definition at line 11 of file model-meat.hpp.

## 8.32 include/barry/models/geese.hpp File Reference

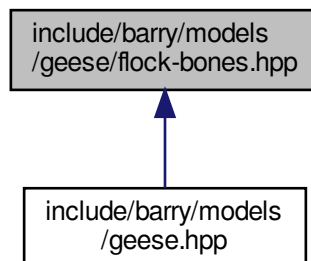
```
#include "geese/geese-node-bones.hpp"
#include "geese/geese-bones.hpp"
#include "geese/geese-meat.hpp"
#include "geese/geese-meat-constructors.hpp"
#include "geese/geese-meat-likelihood.hpp"
#include "geese/geese-meat-likelihood_exhaust.hpp"
#include "geese/geese-meat-simulate.hpp"
#include "geese/geese-meat-predict.hpp"
#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:



## 8.33 include/barry/models/geese/flock-bones.hpp File Reference

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



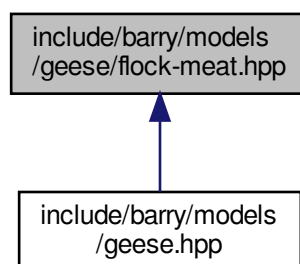
### Classes

- class [Flock](#)

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

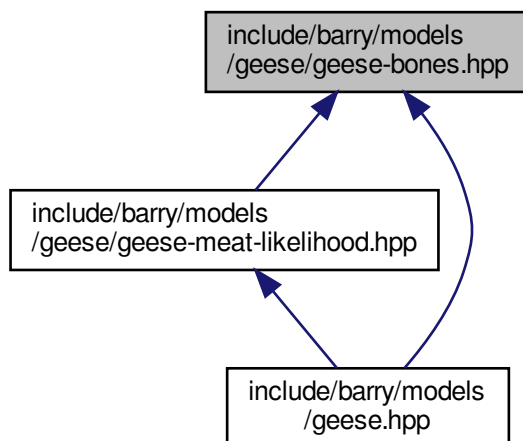
## 8.34 include/barry/models/geese/flock-meat.hpp File Reference

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



## 8.35 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)`

#### 8.35.1 Macro Definition Documentation

### 8.35.1.1 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.

## 8.35.2 Function Documentation

### 8.35.2.1 keygen\_full()

```
std::vector< double > keygen_full (   
    const phylocounters::PhyloArray & array ) [inline]
```

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

### 8.35.2.2 RULE\_FUNCTION()

```
RULE_FUNCTION (   
    rule_empty_free )
```

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

### 8.35.2.3 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.

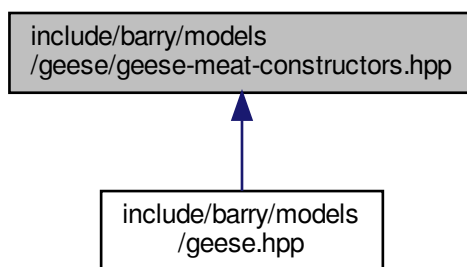
### 8.35.2.4 vector\_caster()

```
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.

### 8.36 include/barry/models/geese/geese-meat-constructors.hpp File Reference

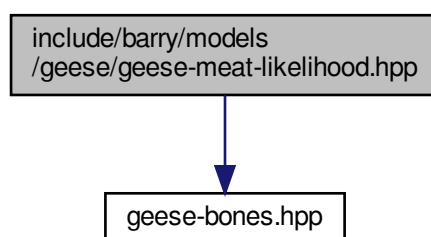
This graph shows which files directly or indirectly include this file:



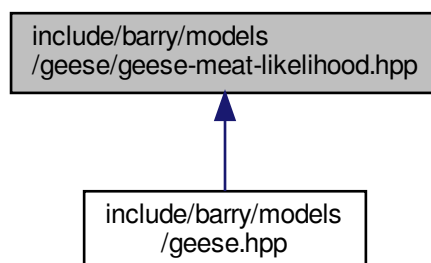
### 8.37 include/barry/models/geese/geese-meat-likelihood.hpp File Reference

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

Include dependency graph for geese-meat-likelihood.hpp:

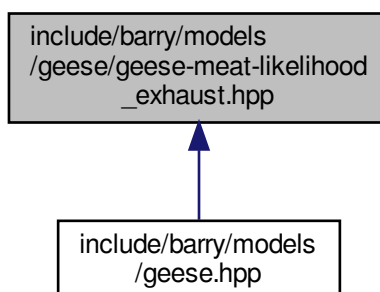


This graph shows which files directly or indirectly include this file:



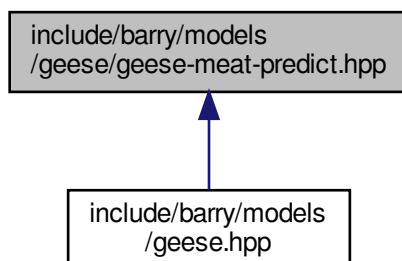
### 8.38 include/barry/models/geese/geese-meat-likelihood\_exhaust.hpp File Reference

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



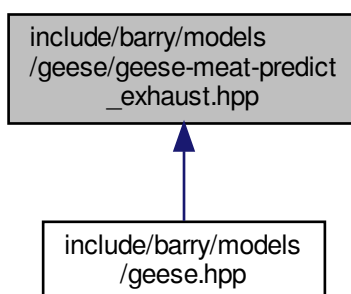
### 8.39 include/barry/models/geese/geese-meat-predict.hpp File Reference

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



### 8.40 include/barry/models/geese/geese-meat-predict\_exhaust.hpp File Reference

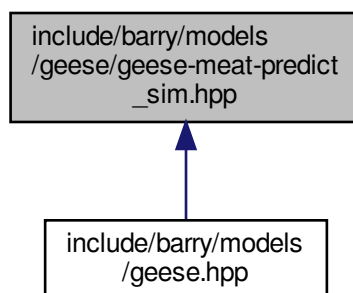
This graph shows which files directly or indirectly include this file:





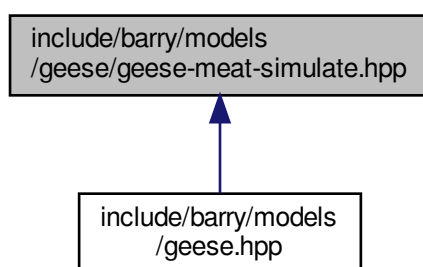
## 8.41 include/barry/models/geese/geese-meat-predict\_sim.hpp File Reference

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



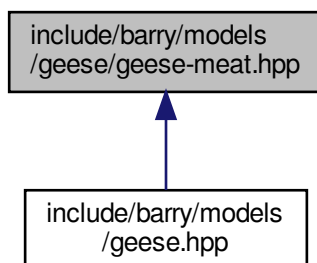
## 8.42 include/barry/models/geese/geese-meat-simulate.hpp File Reference

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



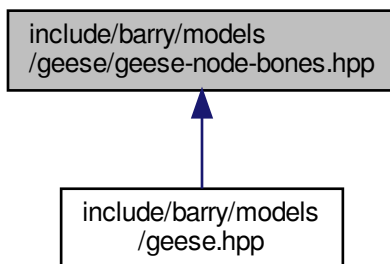
### 8.43 include/barry/models/geese/geese-meat.hpp File Reference

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



### 8.44 include/barry/models/geese/geese-node-bones.hpp File Reference

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



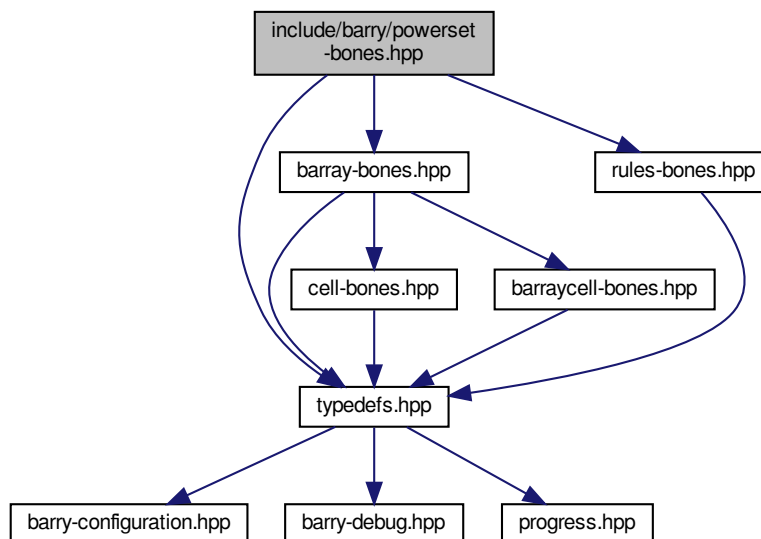
#### Classes

- class [Node](#)  
*A single node for the model.*

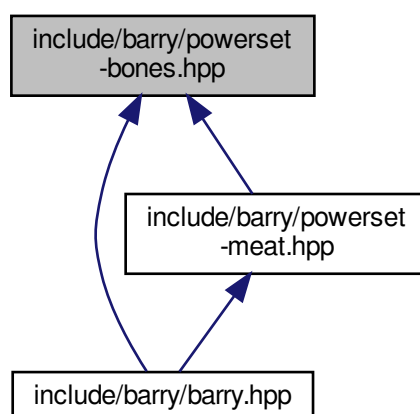
## 8.45 include/barry/powerset-bones.hpp File Reference

```
#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

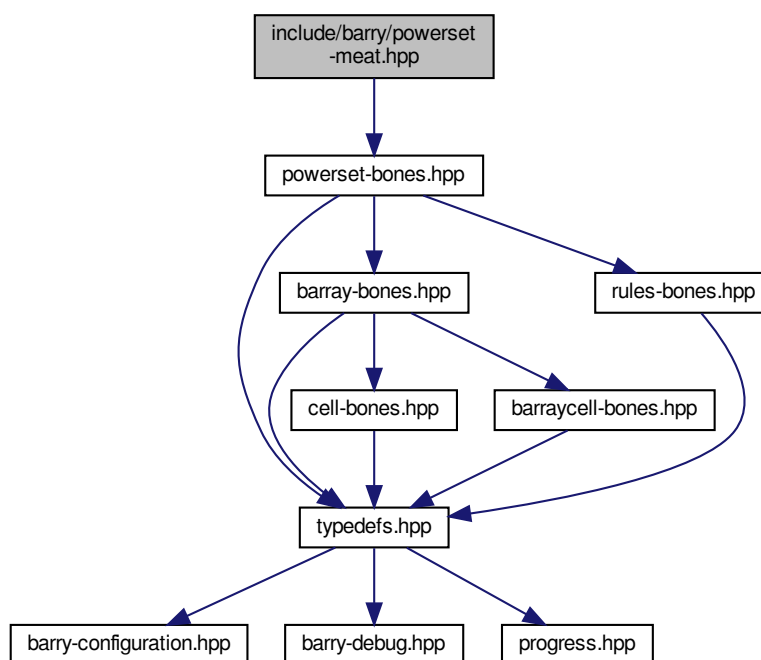
- class [PowerSet< Array\\_Type, Data\\_Rule\\_Type >](#)

*Powerset of a binary array.*

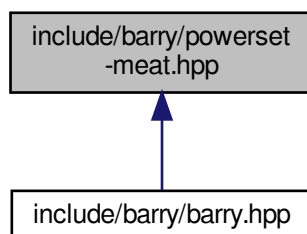
## 8.46 include/barry/powerset-meat.hpp File Reference

```
#include "powerset-bones.hpp"
```

Include dependency graph for powerset-meat.hpp:

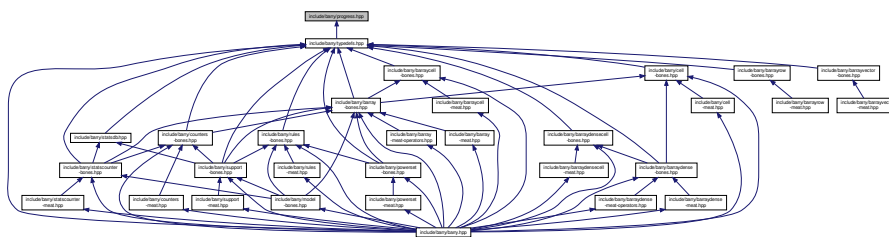


This graph shows which files directly or indirectly include this file:



#### 8.47 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

### 8.47.1 Macro Definition Documentation

#### 8.47.1.1 BARRY\_PROGRESS\_BAR\_WIDTH

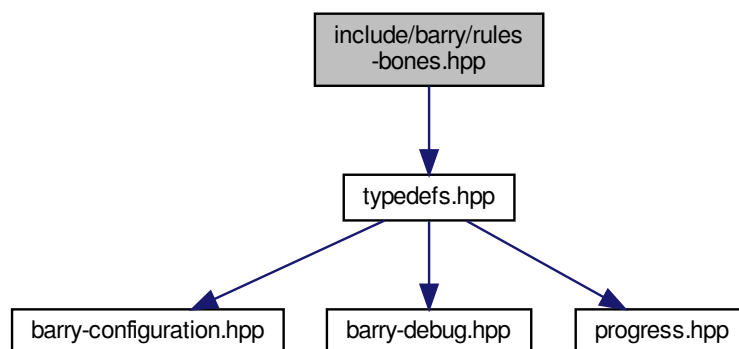
```
#define BARRY_PROGRESS_BAR_WIDTH 80
```

Definition at line 5 of file progress.hpp.

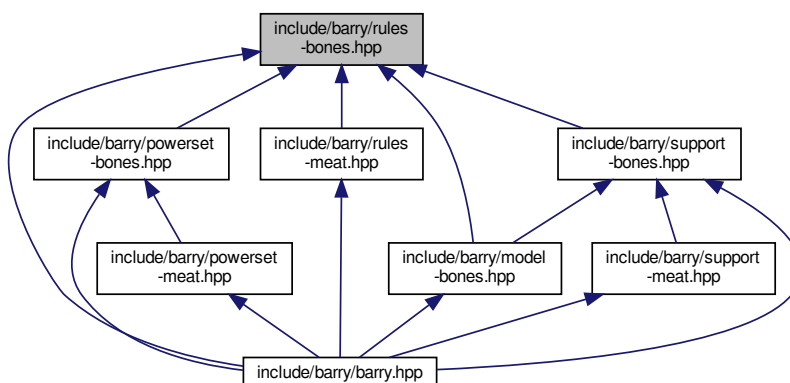
#### 8.48 include/barry/rules-bones.hpp File Reference

```
#include "typedefs.hpp"
```

Include dependency graph for rules-bones.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [Rule< Array\\_Type, Data\\_Type >](#)  
*Rule for determining if a cell should be included in a sequence.*
- class [Rules< Array\\_Type, Data\\_Type >](#)  
*Vector of objects of class Rule.*

## Functions

- `template<typename Array_Type , typename Data_Type >`  
`bool rule_fun_default (const Array_Type *array, uint i, uint j, Data_Type *dat)`

### 8.48.1 Function Documentation

#### 8.48.1.1 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 )

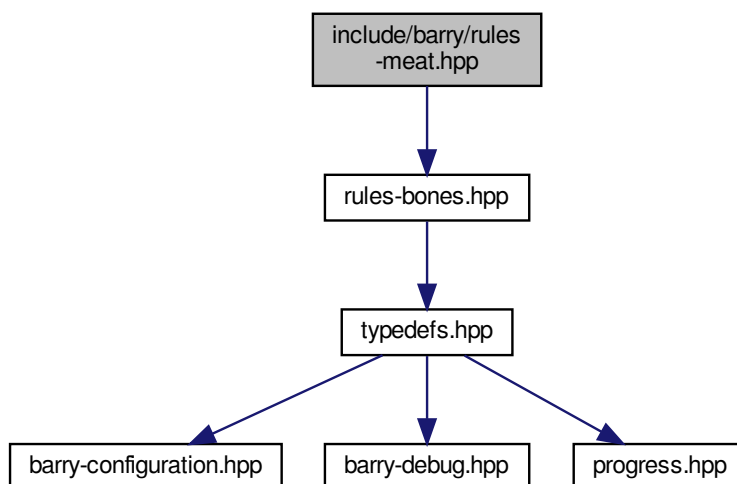
```

Definition at line 7 of file rules-bones.hpp.

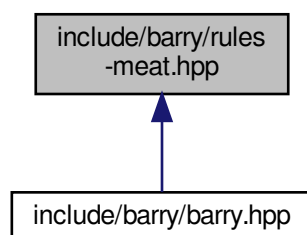
## 8.49 include/barry/rules-meat.hpp File Reference

```
#include "rules-bones.hpp"
```

Include dependency graph for rules-meat.hpp:



This graph shows which files directly or indirectly include this file:

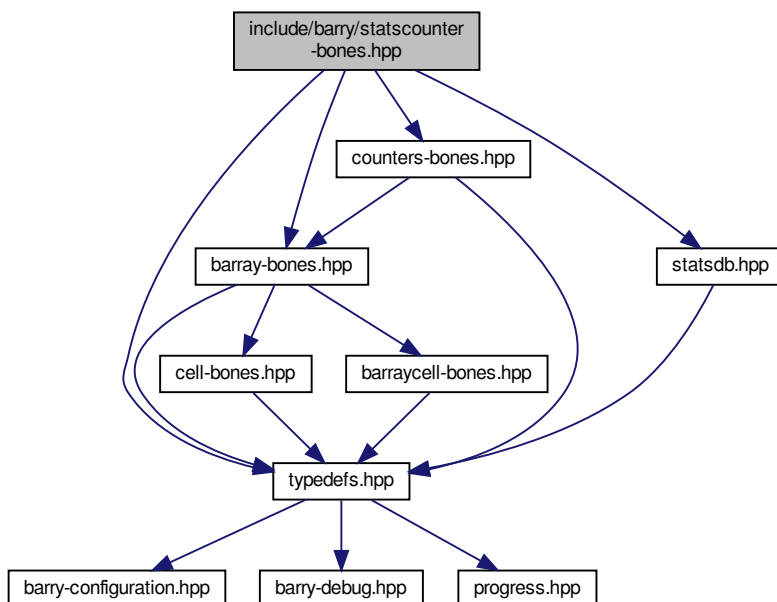


## 8.50 include/barry/statscounter-bones.hpp File Reference

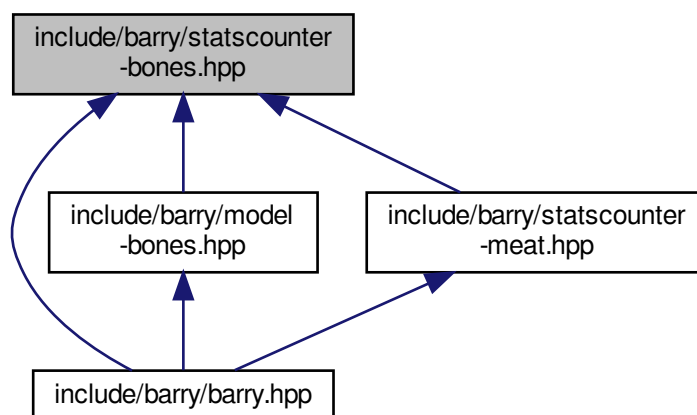
```
#include "typedefs.hpp"  
#include "barray-bones.hpp"  
#include "statsdb.hpp"
```

```
#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 >](#)

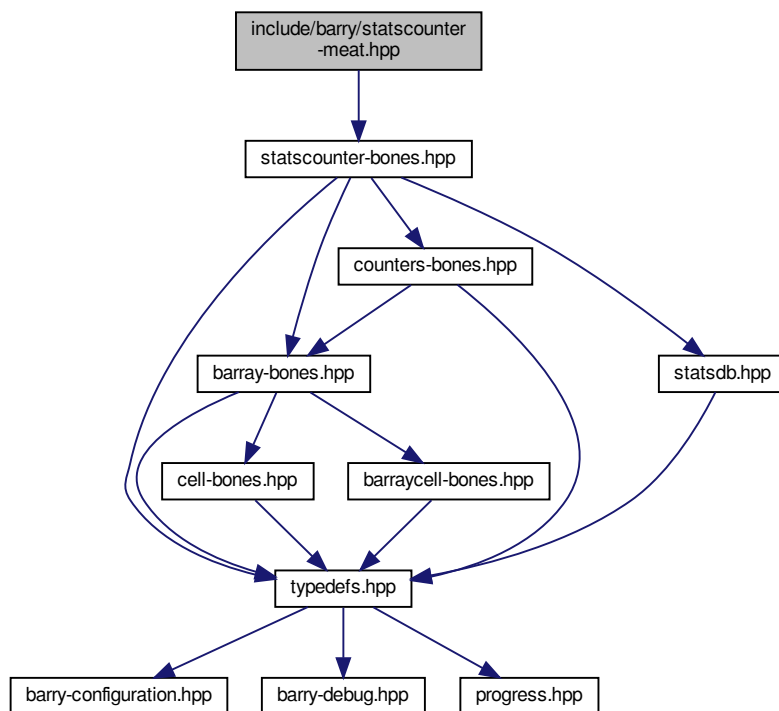
*Count stats for a single Array.*



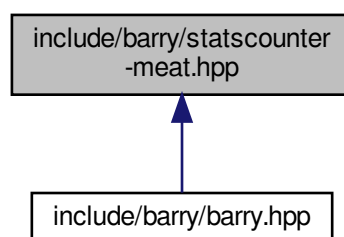
## 8.51 include/barry/statscounter-meat.hpp File Reference

```
#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`)(`const StatsCounter`< `Array_Type`
- `EmptyArray` `clear` ()
- `STATSCOUNTER_TEMPLATE` (,~`StatsCounter`())
- `STATSCOUNTER_TEMPLATE` (void, `reset_array`)(`const Array_Type` \*`Array_`)
- `STATSCOUNTER_TEMPLATE` (void, `add_counter`)(`Counter`< `Array_Type`
- `STATSCOUNTER_TEMPLATE` (void, `set_counters`)(`Counters`< `Array_Type`
- `STATSCOUNTER_TEMPLATE` (void, `count_init`)(`uint i`
- `current_stats` `resize` (`counters`->`size`(), 0.0)
- `for` (`uint n`=0;`n`< `counters`->`size`();++`n`) `current_stats`[`n`]
- `STATSCOUNTER_TEMPLATE` (void, `count_current`)(`uint i`
- `STATSCOUNTER_TEMPLATE` (`std::vector`< `std::string` >, `get_names`()) `const`
- `STATSCOUNTER_TEMPLATE` (`std::vector`< `std::string` >, `get_descriptions`()) `const`

## Variables

- `Data_Type` & `counter`
- `EmptyArray` = \*`Array`
- `current_stats` = `counter.current_stats`
- `counters` = new `Counters`<`Array_Type`,`Data_Type`>((\*`counter.counters`))
- `counter_deleted` = `false`
- `Data_Type` \* `f_`
- `return`
- `Data_Type` \* `counters_`
- `uint j`

## 8.51.1 Macro Definition Documentation

### 8.51.1.1 STATSCOUNTER\_TEMPLATE

```
#define STATSCOUNTER_TEMPLATE(
    a,
    b )  template STATSCOUNTER_TEMPLATE_ARGS() inline a STATSCOUNTER_TYPE()::b
```

Definition at line 11 of file statscounter-meat.hpp.

### 8.51.1.2 STATSCOUNTER\_TEMPLATE\_ARGS

```
template STATSCOUNTER_TEMPLATE_ARGS( ) <typename Array_Type, typename Data_Type>
```

Definition at line 9 of file statscounter-meat.hpp.

### 8.51.1.3 STATSCOUNTER\_TYPE

```
template Data_Type * STATSCOUNTER_TYPE( ) StatsCounter<Array_Type,Data_Type>
```

Definition at line 7 of file statscounter-meat.hpp.

## 8.51.2 Function Documentation

### 8.51.2.1 clear()

```
EmptyArray clear ( )
```

### 8.51.2.2 for()

```
for (
    uint n = 0u;n< counters->size();++n )
```

### 8.51.2.3 resize()

```
current_stats resize (
    counters-> size(),
    0. 0 )
```

### 8.51.2.4 STATSCOUNTER\_TEMPLATE() [1/9]

```
STATSCOUNTER_TEMPLATE (
    StatsCounter ) const
```

### 8.51.2.5 STATSCOUNTER\_TEMPLATE() [2/9]

```
STATSCOUNTER_TEMPLATE (
    ~ StatsCounter )
```

Definition at line 30 of file statscounter-meat.hpp.

**8.51.2.6 STATSCOUNTER\_TEMPLATE() [3/9]**

```
STATSCOUNTER_TEMPLATE (
    std::vector< std::string > ,
    get_descriptions ) const
```

Definition at line 267 of file statscounter-meat.hpp.

**8.51.2.7 STATSCOUNTER\_TEMPLATE() [4/9]**

```
STATSCOUNTER_TEMPLATE (
    std::vector< std::string > ,
    get_names ) const
```

Definition at line 262 of file statscounter-meat.hpp.

**8.51.2.8 STATSCOUNTER\_TEMPLATE() [5/9]**

```
STATSCOUNTER_TEMPLATE (
    void ,
    add_counter )
```

**8.51.2.9 STATSCOUNTER\_TEMPLATE() [6/9]**

```
STATSCOUNTER_TEMPLATE (
    void ,
    count_current )
```

**8.51.2.10 STATSCOUNTER\_TEMPLATE() [7/9]**

```
STATSCOUNTER_TEMPLATE (
    void ,
    count_init )
```

**8.51.2.11 STATSCOUNTER\_TEMPLATE() [8/9]**

```
STATSCOUNTER_TEMPLATE (
    void ,
    reset_array ) const
```

Definition at line 37 of file statscounter-meat.hpp.

### 8.51.2.12 STATSCOUNTER\_TEMPLATE() [9/9]

```
STATSCOUNTER_TEMPLATE (
    void ,
    set_counters )
```

## 8.51.3 Variable Documentation

### 8.51.3.1 counter

```
Data_Type& counter
```

#### Initial value:

```
{
    Array      = counter.Array
```

Definition at line 15 of file statscounter-meat.hpp.

### 8.51.3.2 counter\_deleted

```
counter_deleted = false
```

Definition at line 26 of file statscounter-meat.hpp.

### 8.51.3.3 counters

```
counters = new Counters<Array_Type,Data_Type>((*counter.counters))
```

Definition at line 25 of file statscounter-meat.hpp.

### 8.51.3.4 counters\_

```
Data_Type* counters_
```

#### Initial value:

```
{
    if (!counter_deleted)
        delete counters
```

Definition at line 64 of file statscounter-meat.hpp.

#### 8.51.3.5 current\_stats

```
current_stats = counter.current_stats
```

Definition at line 22 of file statscounter-meat.hpp.

#### 8.51.3.6 EmptyArray

```
EmptyArray = *Array
```

Definition at line 20 of file statscounter-meat.hpp.

#### 8.51.3.7 f\_

```
Data_Rule_Dyn_Type f_
```

##### Initial value:

```
{  
    counters->add_counter(f_)
```

Definition at line 47 of file statscounter-meat.hpp.

#### 8.51.3.8 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 77 of file statscounter-meat.hpp.

#### 8.51.3.9 return

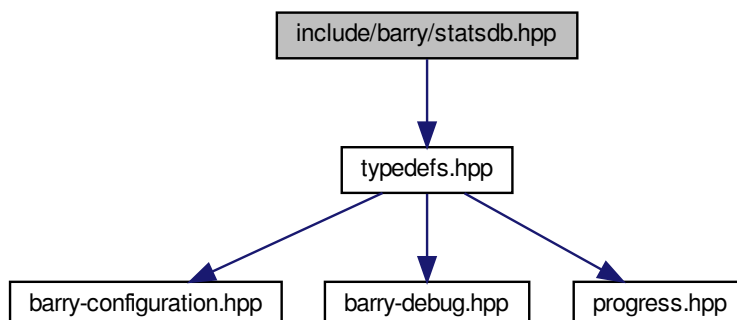
```
return
```

Definition at line 51 of file statscounter-meat.hpp.

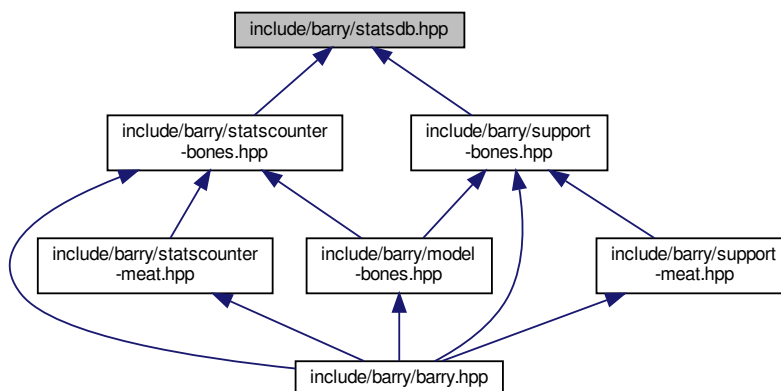
## 8.52 include/barry/statsdb.hpp File Reference

```
#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.*

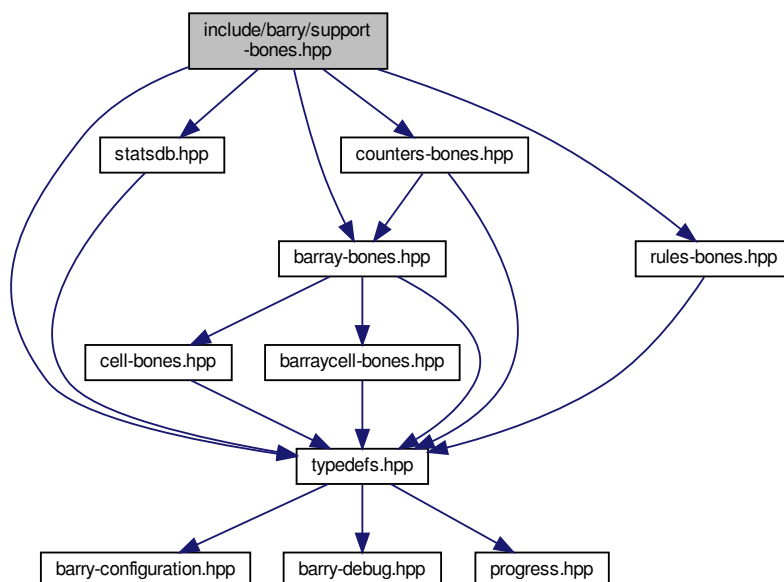
## 8.53 include/barry/support-bones.hpp File Reference

```
#include "typedefs.hpp"
```

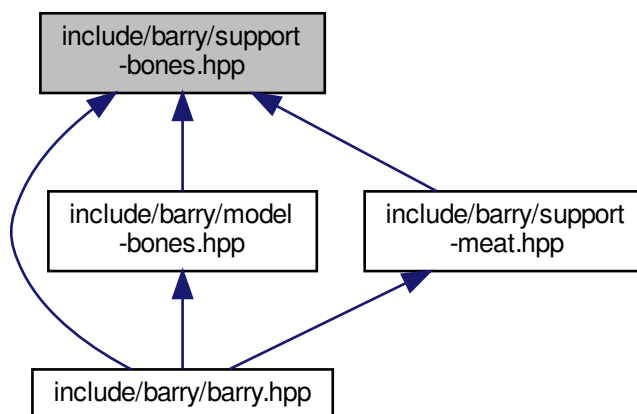
```
#include "barray-bones.hpp"
```

```
#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

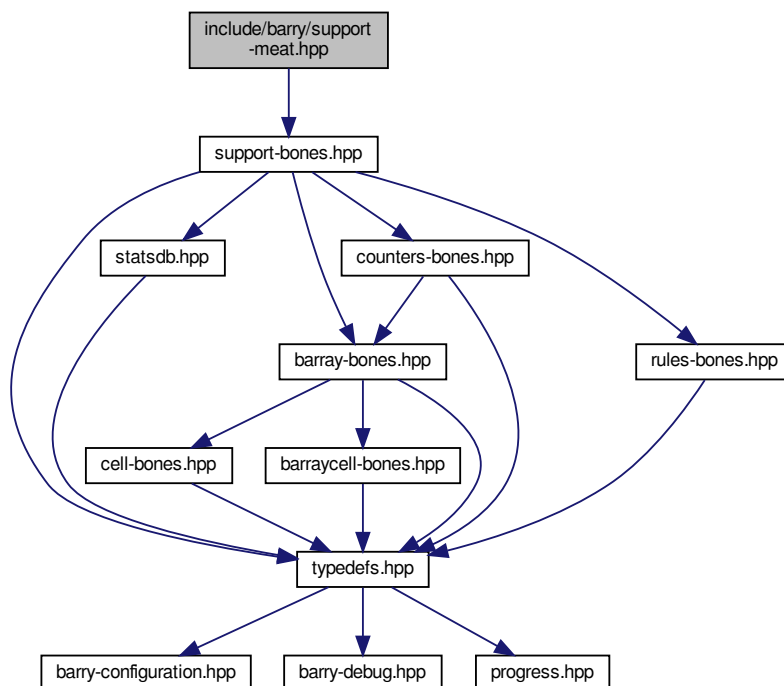
- class `Support< Array_Type, Data_Counter_Type, Data_Rule_Type, Data_Rule_Dyn_Type >`  
Compute the support of sufficient statistics.



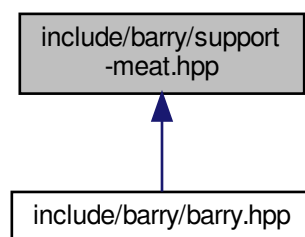
## 8.54 include/barry/support-meat.hpp File Reference

```
#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\_sparse)(uint pos
- [calc\\_backend\\_sparse](#) (pos+1u, array\_bank, stats\_bank)
- [EmptyArray](#) insert\_cell (coord\_i, coord\_j, EmptyArray.default\_val().value, false, false)
- [for](#) (uint n=0u;n< n\_counters;++n)
- [if](#) (rules\_dyn->size() > 0u)
- [if](#) (array\_bank !=nullptr) array\_bank -> push\_back([EmptyArray](#))
- [if](#) (stats\_bank !=nullptr) stats\_bank -> push\_back(current\_stats)
- [EmptyArray](#) rm\_cell (coord\_i, coord\_j, false, false)
- [if](#) (change\_stats\_different > 0u)
- [SUPPORT\\_TEMPLATE](#) (void, calc\_backend\_dense)(uint pos
- [calc\\_backend\\_dense](#) (pos+1u, array\_bank, stats\_bank)
- [EmptyArray](#) insert\_cell (coord\_i, coord\_j, 1, false, false)
- [SUPPORT\\_TEMPLATE](#) (void, calc)(std
- [SUPPORT\\_TEMPLATE](#) (void, add\_counter)(Counter< Array\_Type
- [SUPPORT\\_TEMPLATE](#) (void, set\_counters)(Counters< Array\_Type
- [SUPPORT\\_TEMPLATE](#) (void, add\_rule)(Rule< Array\_Type
- [SUPPORT\\_TEMPLATE](#) (void, set\_rules)(Rules< Array\_Type
- [SUPPORT\\_TEMPLATE](#) (void, add\_rule\_dyn)(Rule< Array\_Type
- [SUPPORT\\_TEMPLATE](#) (void, set\_rules\_dyn)(Rules< Array\_Type
- [SUPPORT\\_TEMPLATE](#) (bool, eval\_rules\_dyn)(const std
- [SUPPORT\\_TEMPLATE](#) (std::vector< double >, get\_counts)() const
- [SUPPORT\\_TEMPLATE](#) (std::vector< double > \*, get\_current\_stats)()
- [SUPPORT\\_TEMPLATE](#) (void, print)() const
- [SUPPORT\\_TEMPLATE](#) (const FreqTable<> &, get\_data)() const

## Variables

- std::vector< Array\_Type > \* [array\\_bank](#)
- std::vector< Array\_Type > std::vector< std::vector< double > > \* [stats\\_bank](#)
- const size\_t & [coord\\_i](#) = coordinates\_free[pos \* 2u]
- const size\_t & [coord\\_j](#) = coordinates\_free[pos \* 2u + 1u]
- double [tmp\\_chng](#)
- unsigned int [change\\_stats\\_different](#) = hashes\_initialized[pos] ? 0u : 1u
- [else](#)
- & [hashes](#) [pos]
- [return](#)
- Data\_Counter\_Type \* [f\\_](#)
- Data\_Counter\_Type \* [counters\\_](#)
- [delete\\_counters](#) = false
- [counters](#) = [counters\\_](#)
- Data\_Rule\_Type \* [rules\\_](#)
- [delete\\_rules](#) = false
- [rules](#) = [rules\\_](#)
- [delete\\_rules\\_dyn](#) = false
- [rules\\_dyn](#) = [rules\\_](#)

### 8.54.1 Macro Definition Documentation

#### 8.54.1.1 BARRY\_SUPPORT\_MEAT\_HPP

```
#define BARRY_SUPPORT_MEAT_HPP 1
```

Definition at line 4 of file support-meat.hpp.

#### 8.54.1.2 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.

#### 8.54.1.3 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.

#### 8.54.1.4 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.

### 8.54.2 Function Documentation

#### 8.54.2.1 calc\_backend\_dense()

```
calc_backend_dense (
    pos+ 1u,
    array_bank ,
    stats_bank )
```

#### 8.54.2.2 calc\_backend\_sparse()

```
calc_backend_sparse (
    pos+ 1u,
    array_bank ,
    stats_bank )
```

#### 8.54.2.3 for()

```
for ( )
```

Definition at line 162 of file support-meat.hpp.

#### 8.54.2.4 if() [1/4]

```
if (
    array_bank !    = nullptr ) -> push_back(EmptyArray)
```

#### 8.54.2.5 if() [2/4]

```
if (
    change_stats_different ,
    0u )
```

Definition at line 243 of file support-meat.hpp.

#### 8.54.2.6 if() [3/4]

```
if (
    rules_dyn-> size(),
    0u )
```

Definition at line 191 of file support-meat.hpp.

#### 8.54.2.7 if() [4/4]

```
if (
    stats_bank !    = nullptr ) -> push_back(current_stats)
```

#### 8.54.2.8 insert\_cell() [1/2]

```
EmptyArray insert_cell (
    coord_i ,
    coord_j ,
    1 ,
    false ,
    false )
```

#### 8.54.2.9 insert\_cell() [2/2]

```
EmptyArray insert_cell (
    coord_i ,
    coord_j ,
    EmptyArray.default_val().value,
    false ,
    false )
```

#### 8.54.2.10 rm\_cell()

```
EmptyArray rm_cell (
    coord_i ,
    coord_j ,
    false ,
    false )
```

#### 8.54.2.11 SUPPORT\_TEMPLATE() [1/17]

```
SUPPORT_TEMPLATE (
    bool ,
    eval_rules_dyn ) const
```

Definition at line 499 of file support-meat.hpp.

**8.54.2.12 SUPPORT\_TEMPLATE() [2/17]**

```
SUPPORT_TEMPLATE (
    const FreqTable<> & ,
    get_data ) const
```

Definition at line 548 of file support-meat.hpp.

**8.54.2.13 SUPPORT\_TEMPLATE() [3/17]**

```
SUPPORT_TEMPLATE (
    std::vector< double > * ,
    get_current_stats )
```

Definition at line 533 of file support-meat.hpp.

**8.54.2.14 SUPPORT\_TEMPLATE() [4/17]**

```
SUPPORT_TEMPLATE (
    std::vector< double > ,
    get_counts ) const
```

Definition at line 521 of file support-meat.hpp.

**8.54.2.15 SUPPORT\_TEMPLATE() [5/17]**

```
SUPPORT_TEMPLATE (
    void ,
    add_counter )
```

**8.54.2.16 SUPPORT\_TEMPLATE() [6/17]**

```
SUPPORT_TEMPLATE (
    void ,
    add_rule )
```

**8.54.2.17 SUPPORT\_TEMPLATE() [7/17]**

```
SUPPORT_TEMPLATE (
    void ,
    add_rule_dyn )
```

**8.54.2.18 SUPPORT\_TEMPLATE() [8/17]**

```
SUPPORT_TEMPLATE (
    void ,
    calc )
```

Definition at line 373 of file support-meat.hpp.

**8.54.2.19 SUPPORT\_TEMPLATE() [9/17]**

```
SUPPORT_TEMPLATE (
    void ,
    calc_backend_dense )
```

**8.54.2.20 SUPPORT\_TEMPLATE() [10/17]**

```
SUPPORT_TEMPLATE (
    void ,
    calc_backend_sparse )
```

**8.54.2.21 SUPPORT\_TEMPLATE() [11/17]**

```
SUPPORT_TEMPLATE (
    void ,
    init_support )
```

Definition at line 16 of file support-meat.hpp.

**8.54.2.22 SUPPORT\_TEMPLATE() [12/17]**

```
SUPPORT_TEMPLATE (
    void ,
    print ) const
```

Definition at line 537 of file support-meat.hpp.

**8.54.2.23 SUPPORT\_TEMPLATE() [13/17]**

```
SUPPORT_TEMPLATE (
    void ,
    reset_array )
```

Definition at line 117 of file support-meat.hpp.

**8.54.2.24 SUPPORT\_TEMPLATE() [14/17]**

```
SUPPORT_TEMPLATE (
    void ,
    reset_array ) const &
```

Definition at line 123 of file support-meat.hpp.

**8.54.2.25 SUPPORT\_TEMPLATE() [15/17]**

```
SUPPORT_TEMPLATE (
    void ,
    set_counters )
```

**8.54.2.26 SUPPORT\_TEMPLATE() [16/17]**

```
SUPPORT_TEMPLATE (
    void ,
    set_rules )
```

**8.54.2.27 SUPPORT\_TEMPLATE() [17/17]**

```
SUPPORT_TEMPLATE (
    void ,
    set_rules_dyn )
```

**8.54.3 Variable Documentation**



### 8.54.3.1 array\_bank

```
std::vector< Array_Type > * array_bank
```

Definition at line 134 of file support-meat.hpp.

### 8.54.3.2 change\_stats\_different

```
unsigned int change_stats_different = hashes_initialized[pos] ? 0u : 1u
```

Definition at line 161 of file support-meat.hpp.

### 8.54.3.3 coord\_i

```
const size_t & coord_i = coordinates_free[pos * 2u]
```

Definition at line 147 of file support-meat.hpp.

### 8.54.3.4 coord\_j

```
const size_t & coord_j = coordinates_free[pos * 2u + 1u]
```

Definition at line 148 of file support-meat.hpp.

### 8.54.3.5 counters

```
counters = counters_
```

Definition at line 427 of file support-meat.hpp.

### 8.54.3.6 counters\_

```
Data_Counter_Type* counters_
```

**Initial value:**

```
{  
  
    if (delete_counters)  
        delete counters
```

Definition at line 420 of file support-meat.hpp.

#### 8.54.3.7 delete\_counters

```
delete_counters = false
```

Definition at line 426 of file support-meat.hpp.

#### 8.54.3.8 delete\_rules

```
delete_rules = false
```

Definition at line 460 of file support-meat.hpp.

#### 8.54.3.9 delete\_rules\_dyn

```
delete_rules_dyn = false
```

Definition at line 492 of file support-meat.hpp.

#### 8.54.3.10 else

```
else (  
    void )
```

##### Initial value:

```
{  
    if (change_stats_different > 0u)  
        hashes[pos] = data.add(current_stats, nullptr)
```

Definition at line 216 of file support-meat.hpp.

#### 8.54.3.11 f\_

```
Data_Rule_Dyn_Type f_
```

##### Initial value:

```
{  
    counters->add_counter(f_)
```

Definition at line 402 of file support-meat.hpp.

#### 8.54.3.12 hashes

```
& hashes
```

Definition at line 221 of file support-meat.hpp.

#### 8.54.3.13 return

```
return
```

Definition at line 255 of file support-meat.hpp.

#### 8.54.3.14 rules

```
rules = rules_
```

Definition at line 461 of file support-meat.hpp.

#### 8.54.3.15 rules\_

```
Data_Rule_Dyn_Type * rules_
```

**Initial value:**

```
{  
  
    if (delete_rules)  
        delete rules
```

Definition at line 454 of file support-meat.hpp.

#### 8.54.3.16 rules\_dyn

```
rules_dyn = rules_
```

Definition at line 493 of file support-meat.hpp.

### 8.54.3.17 stats\_bank

```
std::vector< Array_Type > std::vector< std::vector< double > > * stats_bank
```

#### Initial value:

```
{
    if (pos >= coordiantes_n_free)
        return
```

Definition at line 135 of file support-meat.hpp.

### 8.54.3.18 tmp\_chng

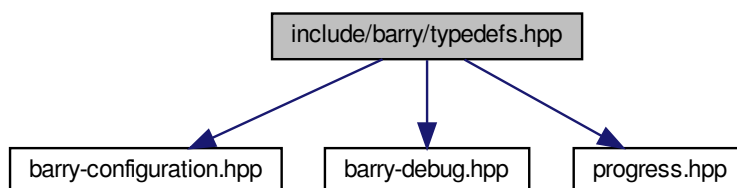
```
double tmp_chng
```

Definition at line 160 of file support-meat.hpp.

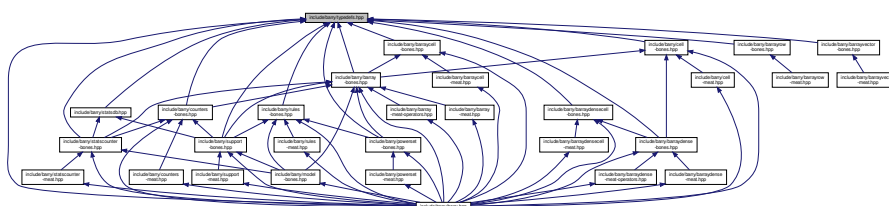
## 8.55 include/barry/typedefs.hpp File Reference

```
#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<> double [vec\\_inner\\_prod](#) ([const](#) std::vector< double > &a, [const](#) std::vector< double > &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-100)

## Variables

- `const int CHECK::BOTH = -1`
- `const int CHECK::NONE = 0`
- `const int CHECK::ONE = 1`
- `const int CHECK::TWO = 2`
- `const int EXISTS::BOTH = -1`
- `const int EXISTS::NONE = 0`
- `const int EXISTS::ONE = 1`
- `const int EXISTS::TWO = 1`
- `const int EXISTS::UNKNOWN = -1`
- `const int EXISTS::AS_ZERO = 0`
- `const int EXISTS::AS_ONE = 1`

### 8.55.1 Typedef Documentation

#### 8.55.1.1 Col\_type

```
template<typename Cell_Type >
using Col_type = Map< uint, Cell<Cell_Type>*>
```

Definition at line 71 of file typedefs.hpp.

#### 8.55.1.2 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 148 of file typedefs.hpp.

### 8.55.1.3 Counts\_type

```
typedef std::vector< std::pair< std::vector<double>, uint > > Counts_type
```

Definition at line 52 of file typedefs.hpp.

### 8.55.1.4 MapVec\_type

```
template<typename Ta = double, typename Tb = uint>  
using MapVec_type = std::unordered_map< std::vector< Ta >, Tb, vecHasher<Ta> >
```

Definition at line 129 of file typedefs.hpp.

### 8.55.1.5 Row\_type

```
template<typename Cell_Type >  
using Row_type = Map< uint, Cell<Cell_Type> >
```

Definition at line 68 of file typedefs.hpp.

### 8.55.1.6 Rule\_fun\_type

```
template<typename Array_Type , typename Data_Type >  
using Rule_fun_type = std::function<bool(const Array_Type &, uint, uint, Data_Type *)>
```

Definition at line 151 of file typedefs.hpp.

### 8.55.1.7 uint

```
typedef unsigned int uint
```

Definition at line 18 of file typedefs.hpp.

## 8.55.2 Function Documentation

### 8.55.2.1 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 162 of file `typedefs.hpp`.

#### 8.55.2.2 `vec_equal_approx()`

```
template<typename T >
bool vec_equal_approx (
    const std::vector< T > & a,
    const std::vector< T > & b,
    double eps = 1e-100 ) [inline]
```

Definition at line 180 of file `typedefs.hpp`.

#### 8.55.2.3 `vec_inner_prod()` [1/2]

```
template<>
double vec_inner_prod (
    const std::vector< double > & a,
    const std::vector< double > & b ) [inline]
```

Definition at line 223 of file `typedefs.hpp`.

#### 8.55.2.4 `vec_inner_prod()` [2/2]

```
template<typename T >
T vec_inner_prod (
    const std::vector< T > & a,
    const std::vector< T > & b ) [inline]
```

Definition at line 200 of file `typedefs.hpp`.

## 8.56 README.md File Reference



# Index

- ~BArray
  - BArray< Cell\_Type, Data\_Type >, [35](#)
- ~BArrayCell
  - BArrayCell< Cell\_Type, Data\_Type >, [46](#)
- ~BArrayCell\_const
  - BArrayCell\_const< Cell\_Type, Data\_Type >, [48](#)
- ~BArrayDense
  - BArrayDense< Cell\_Type, Data\_Type >, [54](#)
- ~BArrayDenseCell
  - BArrayDenseCell< Cell\_Type, Data\_Type >, [66](#)
- ~BArrayRow
  - BArrayRow< Cell\_Type, Data\_Type >, [79](#)
- ~BArrayRow\_const
  - BArrayRow\_const< Cell\_Type, Data\_Type >, [81](#)
- ~BArrayVector
  - BArrayVector< Cell\_Type, Data\_Type >, [84](#)
- ~BArrayVector\_const
  - BArrayVector\_const< Cell\_Type, Data\_Type >, [87](#)
- ~Cell
  - Cell< Cell\_Type >, [91](#)
- ~ConstBArrayRowIter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [96](#)
- ~Counter
  - Counter< Array\_Type, Data\_Type >, [99](#)
- ~Counters
  - Counters< Array\_Type, Data\_Type >, [103](#)
- ~Entries
  - Entries< Cell\_Type >, [108](#)
- ~Flock
  - Flock, [110](#)
- ~FreqTable
  - FreqTable< T >, [117](#)
- ~Geese
  - Geese, [122](#)
- ~Model
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [135](#)
- ~NetCounterData
  - NetCounterData, [146](#)
- ~NetworkData
  - NetworkData, [148](#)
- ~Node
  - Node, [151](#)
- ~PhyloRuleDynData
  - PhyloRuleDynData, [160](#)
- ~PowerSet
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [163](#)
- ~Progress
  - Progress, [168](#)
- ~Rule
  - Rule< Array\_Type, Data\_Type >, [170](#)
- ~Rules
  - Rules< Array\_Type, Data\_Type >, [172](#)
- ~StatsCounter
  - StatsCounter< Array\_Type, Data\_Type >, [176](#)
- ~Support
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [181](#)
- active
  - Cell< Cell\_Type >, [94](#)
- add
  - barray-meat.hpp, [210](#)
  - barraydense-meat.hpp, [237](#)
  - Cell< Cell\_Type >, [92](#), [93](#)
  - FreqTable< T >, [117](#)
- add\_array
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [135](#)
- add\_counter
  - Counters< Array\_Type, Data\_Type >, [104](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [135](#), [136](#)
  - StatsCounter< Array\_Type, Data\_Type >, [177](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [182](#)
- add\_data
  - Flock, [110](#)
- add\_rule
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [136](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [164](#)
  - Rules< Array\_Type, Data\_Type >, [173](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [182](#)
- add\_rule\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [136](#), [137](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [182](#)

- annotations
  - Node, [152](#)
- ans
  - barray-meat.hpp, [201](#), [210](#)
  - barraydense-meat.hpp, [227](#), [238](#)
- Array
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, [96](#)
- array
  - Node, [152](#)
- Array\_
  - barray-meat.hpp, [210](#)
- array\_bank
  - support-meat.hpp, [332](#)
- arrays
  - Node, [152](#)
- AS\_ONE
  - EXISTS, [29](#)
- as\_vector
  - FreqTable< T >, [117](#)
- AS\_ZERO
  - EXISTS, [29](#)
- at
  - PhyloCounterData, [157](#)
- BArray
  - BArray< Cell\_Type, Data\_Type >, [34](#), [35](#)
- BArray< Cell\_Type, Data\_Type >, [31](#)
  - ~BArray, [35](#)
  - BArray, [34](#), [35](#)
  - BArrayCell< Cell\_Type, Data\_Type >, [44](#)
  - BArrayCell\_const< Cell\_Type, Data\_Type >, [44](#)
  - clear, [35](#)
  - col, [35](#)
  - D, [36](#)
  - default\_val, [36](#)
  - flush\_data, [36](#)
  - get\_cell, [36](#)
  - get\_col\_vec, [36](#), [37](#)
  - get\_entries, [37](#)
  - get\_row\_vec, [37](#)
  - insert\_cell, [37](#), [38](#)
  - is\_dense, [38](#)
  - is\_empty, [38](#)
  - ncol, [38](#)
  - nnozero, [39](#)
  - nrow, [39](#)
  - operator\*=, [39](#)
  - operator(), [39](#)
  - operator+=, [39](#), [40](#)
  - operator-=, [40](#)
  - operator/=: [40](#)
  - operator=, [41](#)
  - operator==, [41](#)
  - out\_of\_range, [41](#)
  - print, [41](#)
  - reserve, [41](#)
  - resize, [42](#)
  - rm\_cell, [42](#)
  - row, [42](#)
  - set\_data, [42](#)
  - swap\_cells, [43](#)
  - swap\_cols, [43](#)
  - swap\_rows, [43](#)
  - toggle\_cell, [43](#)
  - toggle\_lock, [43](#)
  - transpose, [44](#)
  - visited, [45](#)
  - zero\_col, [44](#)
  - zero\_row, [44](#)
- barray-bones.hpp
  - BARRAY\_BONES\_HPP, [192](#)
- barray-meat-operators.hpp
  - BARRAY\_TEMPLATE, [194](#)–[196](#)
  - BARRAY\_TEMPLATE\_ARGS, [194](#), [196](#)
  - BARRAY\_TYPE, [194](#), [196](#)
  - BARRY\_BARRAY\_MEAT\_OPERATORS\_HPP, [194](#)
  - COL, [195](#)
  - for, [196](#)
  - operator(), [197](#)
  - rhs, [197](#)
  - ROW, [195](#)
  - this, [197](#)
- barray-meat.hpp
  - add, [210](#)
  - ans, [201](#), [210](#)
  - Array\_, [210](#)
  - BARRAY\_TEMPLATE, [200](#)–[205](#)
  - BARRAY\_TEMPLATE\_ARGS, [200](#)
  - BARRAY\_TYPE, [200](#)
  - check\_bounds, [210](#)
  - check\_exists, [210](#)
  - COL, [201](#), [205](#)
  - col0, [211](#)
  - const, [211](#)
  - copy\_data, [211](#)
  - data, [211](#)
  - delete\_data, [211](#)
  - delete\_data\_, [212](#)
  - else, [212](#)
  - false, [212](#)
  - first, [212](#)
  - for, [205](#), [206](#)
  - i1, [212](#)
  - if, [206](#)–[208](#)
  - j, [213](#)
  - j0, [213](#)
  - j1, [213](#)
  - M, [209](#), [213](#)
  - M\_, [213](#)
  - N, [214](#)
  - NCells, [214](#)
  - report, [214](#)
  - resize, [209](#)
  - return, [209](#), [214](#)
  - ROW, [201](#), [209](#)
  - row0, [214](#)

- search, 215
- source, 215
- target, 215
- v, 215
- value, 215
- BARRAY\_BONES\_HPP
  - barray-bones.hpp, 192
- BARRAY\_TEMPLATE
  - barray-meat-operators.hpp, 194–196
  - barray-meat.hpp, 200–205
- BARRAY\_TEMPLATE\_ARGS
  - barray-meat-operators.hpp, 194, 196
  - barray-meat.hpp, 200
- BARRAY\_TYPE
  - barray-meat-operators.hpp, 194, 196
  - barray-meat.hpp, 200
- BArrayCell
  - BArrayCell< Cell\_Type, Data\_Type >, 46
- BArrayCell< Cell\_Type, Data\_Type >, 45
  - ~BArrayCell, 46
  - BArray< Cell\_Type, Data\_Type >, 44
  - BArrayCell, 46
  - operator Cell\_Type, 46
  - operator\*=: 46
  - operator+=, 46
  - operator-=, 47
  - operator/=: 47
  - operator=: 47
  - operator==, 47
- BArrayCell\_const
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 48
- BArrayCell\_const< Cell\_Type, Data\_Type >, 48
  - ~BArrayCell\_const, 48
  - BArray< Cell\_Type, Data\_Type >, 44
  - BArrayCell\_const, 48
  - operator Cell\_Type, 49
  - operator!=, 49
  - operator<, 49
  - operator<=, 49
  - operator>, 49
  - operator>=, 50
  - operator==, 49
- BArrayDense
  - BArrayDense< Cell\_Type, Data\_Type >, 53, 54
- BArrayDense< Cell\_Type, Data\_Type >, 50
  - ~BArrayDense, 54
  - BArrayDense, 53, 54
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 64, 68
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 64, 71
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 64
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 64, 75
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 64
  - clear, 54
  - col, 54, 55
  - colsum, 55
  - D, 55
  - default\_val, 55
  - get\_cell, 55
  - get\_col\_vec, 56
  - get\_data, 56
  - get\_entries, 56
  - get\_row\_vec, 56, 57
  - insert\_cell, 57
  - is\_dense, 57
  - is\_empty, 57
  - ncol, 58
  - nnozero, 58
  - nrow, 58
  - operator\*=, 58
  - operator(), 58
  - operator+=, 59
  - operator-=, 59
  - operator/=: 60
  - operator=: 60
  - operator==, 60
  - out\_of\_range, 60
  - print, 60
  - reserve, 61
  - resize, 61
  - rm\_cell, 61
  - row, 61
  - rowsum, 61
  - set\_data, 62
  - swap\_cells, 62
  - swap\_cols, 62
  - swap\_rows, 62
  - toggle\_cell, 63
  - toggle\_lock, 63
  - transpose, 63
  - visited, 65
  - zero\_col, 63
  - zero\_row, 63
- barraydense-bones.hpp
  - BARRY\_BARRAYDENSE\_BONES\_HPP, 219
- barraydense-meat-operators.hpp
  - BARRY\_BARRAYDENSE\_MEAT\_OPERATORS\_HPP, 220
  - BDENSE\_TEMPLATE, 220–222
  - BDENSE\_TEMPLATE\_ARGS, 220, 222
  - BDENSE\_TYPE, 220, 222
  - COL, 221
  - POS, 221
  - POS\_N, 221
  - ROW, 221
- barraydense-meat.hpp
  - add, 237
  - ans, 227, 238
  - BDENSE\_TEMPLATE, 225, 227–234
  - BDENSE\_TEMPLATE\_ARGS, 225
  - BDENSE\_TYPE, 226
  - check\_bounds, 238
  - check\_exists, 238

- COL, 226
- col, 238
- const, 239
- copy\_data, 239
- data, 239
- delete\_data, 239
- delete\_data\_, 239
- el, 240
- el\_colsums, 240
- el\_rowsums, 240
- else, 240
- false, 240
- for, 234
- i1, 241
- if, 234
- insert\_cell, 235
- j, 241
- j0, 241
- j1, 241
- M, 235, 241
- M\_, 241
- N, 242
- POS, 226
- POS\_N, 226
- report, 242
- resize, 235, 236
- return, 242
- rm\_cell, 236, 237
- ROW, 226
- source, 242
- target, 242
- v, 243
- va\_end, 237
- va\_start, 237
- val0, 243
- val1, 243
- value, 243
- vprintf, 237
- ZERO\_CELL, 227
- BArrayDenseCell
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 66
- BArrayDenseCell< Cell\_Type, Data\_Type >, 65
  - ~BArrayDenseCell, 66
  - BArrayDense< Cell\_Type, Data\_Type >, 64, 68
  - BArrayDenseCell, 66
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 68, 71
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 68, 73
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 75
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 78
  - operator Cell\_Type, 66
  - operator\*=: 66
  - operator+=, 67
  - operator-=, 67
  - operator/=: 67
  - operator=, 67
  - operator==, 67
- barraydensecell-bones.hpp
  - POS, 244
- barraydensecell-meat.hpp
  - POS, 246
- BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 69
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 71
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 73
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 75
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 78
- BArrayDenseCol
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 69
- BArrayDenseCol< Cell\_Type, Data\_Type >, 69
  - BArrayDense< Cell\_Type, Data\_Type >, 64, 71
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 68, 71
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 71
  - BArrayDenseCol, 69
  - begin, 70
  - end, 70
  - operator(), 70
  - size, 70
- barraydensecol-bones.hpp
  - POS, 247
  - POS\_N, 247
  - ZERO\_CELL, 247
- BArrayDenseCol\_const
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 72
- BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 71
  - BArrayDense< Cell\_Type, Data\_Type >, 64
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 68, 73
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 73
  - BArrayDenseCol\_const, 72
  - begin, 72
  - end, 72
  - operator(), 72
  - size, 73
- BArrayDenseRow
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 74
- BArrayDenseRow< Cell\_Type, Data\_Type >, 73
  - BArrayDense< Cell\_Type, Data\_Type >, 64, 75
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 75
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 75
  - BArrayDenseRow, 74
  - begin, 74
  - end, 74
  - operator(), 75
  - size, 75
- barraydenserow-bones.hpp
  - POS, 248
  - POS\_N, 248
  - ZERO\_CELL, 248

- BArrayDenseRow\_const
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 76
- BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 76
  - BArrayDense< Cell\_Type, Data\_Type >, 64
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 78
  - BArrayDenseCell\_const< Cell\_Type, Data\_Type >, 78
  - BArrayDenseRow\_const, 76
  - begin, 77
  - end, 77
  - operator(), 77
  - size, 77
- BArrayRow
  - BArrayRow< Cell\_Type, Data\_Type >, 79
- BArrayRow< Cell\_Type, Data\_Type >, 78
  - ~BArrayRow, 79
  - BArrayRow, 79
  - operator BArrayRow< Cell\_Type, Data\_Type >, 79
  - operator\*=: 79
  - operator+=, 79
  - operator-=, 79
  - operator/=, 80
  - operator=, 80
  - operator==, 80
- barrayrow-meat.hpp
  - BARRY\_BARRAYROW\_MEAT\_HPP, 250
  - BROW\_TEMPLATE, 250–252
  - BROW\_TEMPLATE\_ARGS, 251
  - BROW\_TYPE, 251
- BArrayRow\_const
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 81
- BArrayRow\_const< Cell\_Type, Data\_Type >, 80
  - ~BArrayRow\_const, 81
  - BArrayRow\_const, 81
  - operator BArrayRow\_const< Cell\_Type, Data\_Type >, 81
  - operator!=, 81
  - operator<, 81
  - operator<=, 82
  - operator>, 82
  - operator>=, 82
  - operator==, 82
- BArrayVector
  - BArrayVector< Cell\_Type, Data\_Type >, 83
- BArrayVector< Cell\_Type, Data\_Type >, 82
  - ~BArrayVector, 84
  - BArrayVector, 83
  - begin, 84
  - end, 84
  - is\_col, 84
  - is\_row, 84
  - operator std::vector< Cell\_Type >, 85
  - operator\*=: 85
  - operator+=, 85
  - operator-=, 85
  - operator/=, 85
  - operator=, 86
  - operator==, 86
  - size, 86
- barrayvector-meat.hpp
  - BARRY\_BARRAYVECTOR\_MEAT\_HPP, 254
- BArrayVector\_const
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 87
- BArrayVector\_const< Cell\_Type, Data\_Type >, 86
  - ~BArrayVector\_const, 87
  - BArrayVector\_const, 87
  - begin, 88
  - end, 88
  - is\_col, 88
  - is\_row, 88
  - operator std::vector< Cell\_Type >, 88
  - operator!=, 88
  - operator<, 89
  - operator<=, 89
  - operator>, 89
  - operator>=, 89
  - operator==, 89
  - size, 90
- barry, 27
- barry-configuration.hpp
  - BARRY\_CHECK\_SUPPORT, 255
  - BARRY\_ISFINITE, 255
  - BARRY\_MAX\_NUM\_ELEMENTS, 255
  - BARRY\_SAFE\_EXP, 255
  - Map, 255
  - printf\_barry, 255
- barry-debug.hpp
  - BARRY\_DEBUG\_LEVEL, 256
- barry-macros.hpp
  - BARRY\_ONE, 257
  - BARRY\_ONE\_DENSE, 257
  - BARRY\_UNUSED, 257
  - BARRY\_ZERO, 257
  - BARRY\_ZERO\_DENSE, 257
- barry.hpp
  - BARRY\_HPP, 259
  - BARRY\_VERSION, 259
  - COUNTER\_FUNCTION, 259
  - COUNTER\_LAMBDA, 260
  - RULE\_FUNCTION, 260
  - RULE\_LAMBDA, 260
- barry::counters, 27
- barry::counters::network, 28
- barry::counters::phylo, 28
- BARRY\_BARRAY\_MEAT\_OPERATORS\_HPP
  - barray-meat-operators.hpp, 194
- BARRY\_BARRAYDENSE\_BONES\_HPP
  - barraydense-bones.hpp, 219
- BARRY\_BARRAYDENSE\_MEAT\_OPERATORS\_HPP
  - barraydense-meat-operators.hpp, 220
- BARRY\_BARRAYROW\_MEAT\_HPP
  - barrayrow-meat.hpp, 250
- BARRY\_BARRAYVECTOR\_MEAT\_HPP
  - barrayvector-meat.hpp, 254
- BARRY\_CHECK\_SUPPORT

- barry-configuration.hpp, 255
- BARRY\_DEBUG\_LEVEL
  - barry-debug.hpp, 256
- BARRY\_HPP
  - barry.hpp, 259
- BARRY\_ISFINITE
  - barry-configuration.hpp, 255
- BARRY\_MAX\_NUM\_ELEMENTS
  - barry-configuration.hpp, 255
- BARRY\_ONE
  - barry-macros.hpp, 257
- BARRY\_ONE\_DENSE
  - barry-macros.hpp, 257
- BARRY\_PROGRESS\_BAR\_WIDTH
  - progress.hpp, 313
- BARRY\_SAFE\_EXP
  - barry-configuration.hpp, 255
- BARRY\_SUPPORT\_MEAT\_HPP
  - support-meat.hpp, 326
- BARRY\_UNUSED
  - barry-macros.hpp, 257
- BARRY\_VERSION
  - barry.hpp, 259
- BARRY\_ZERO
  - barry-macros.hpp, 257
- BARRY\_ZERO\_DENSE
  - barry-macros.hpp, 257
- BARRY\_ZERO\_NETWORK
  - network.hpp, 286
- BARRY\_ZERO\_NETWORK\_DENSE
  - network.hpp, 287
- BDENSE\_TEMPLATE
  - barraydense-meat-operators.hpp, 220–222
  - barraydense-meat.hpp, 225, 227–234
- BDENSE\_TEMPLATE\_ARGS
  - barraydense-meat-operators.hpp, 220, 222
  - barraydense-meat.hpp, 225
- BDENSE\_TYPE
  - barraydense-meat-operators.hpp, 220, 222
  - barraydense-meat.hpp, 226
- begin
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 70
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 72
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 74
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 77
  - BArrayVector< Cell\_Type, Data\_Type >, 84
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 88
  - PhyloCounterData, 157
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 164
- blengths
  - NodeData, 155
- BOTH
  - CHECK, 28
  - EXISTS, 29
- BROW\_TEMPLATE
  - barrayrow-meat.hpp, 250–252
- BROW\_TEMPLATE\_ARGS
  - barrayrow-meat.hpp, 251
- BROW\_TYPE
  - barrayrow-meat.hpp, 251
- calc
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 164
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 183
- calc\_backend\_dense
  - support-meat.hpp, 327
- calc\_backend\_sparse
  - support-meat.hpp, 328
- calc\_reduced\_sequence
  - Geese, 123
- calc\_sequence
  - Geese, 123
- Cell
  - Cell< Cell\_Type >, 91, 92
- Cell< Cell\_Type >, 90
  - ~Cell, 91
  - active, 94
  - add, 92, 93
  - Cell, 91, 92
  - operator Cell\_Type, 93
  - operator!=, 93
  - operator=, 93, 94
  - operator==, 94
  - value, 94
  - visited, 94
- Cell\_const< Cell\_Type >, 95
- change\_stats
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 186
- change\_stats\_different
  - support-meat.hpp, 333
- CHECK, 28
  - BOTH, 28
  - NONE, 28
  - ONE, 28
  - TWO, 28
- check\_bounds
  - barray-meat.hpp, 210
  - barraydense-meat.hpp, 238
- check\_exists
  - barray-meat.hpp, 210
  - barraydense-meat.hpp, 238
- clear
  - BArray< Cell\_Type, Data\_Type >, 35
  - BArrayDense< Cell\_Type, Data\_Type >, 54
  - Counters< Array\_Type, Data\_Type >, 104
  - FreqTable< T >, 117
  - Rules< Array\_Type, Data\_Type >, 173
  - statscounter-meat.hpp, 319
- COL
  - barray-meat-operators.hpp, 195
  - barray-meat.hpp, 201, 205

- barraydense-meat-operators.hpp, 221
  - barraydense-meat.hpp, 226
- col
  - BArray< Cell\_Type, Data\_Type >, 35
  - BArrayDense< Cell\_Type, Data\_Type >, 54, 55
  - barraydense-meat.hpp, 238
- col0
  - barray-meat.hpp, 211
- Col\_type
  - typedefs.hpp, 338
- colnames
  - Flock, 111
  - Geese, 123
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 137
- colsum
  - BArrayDense< Cell\_Type, Data\_Type >, 55
- conditional\_prob
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 137
- const
  - barray-meat.hpp, 211
  - barraydense-meat.hpp, 239
- ConstBArrayRowIter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, 96
- ConstBArrayRowIter< Cell\_Type, Data\_Type >, 95
  - ~ConstBArrayRowIter, 96
  - Array, 96
  - ConstBArrayRowIter, 96
  - current\_col, 97
  - current\_row, 97
  - iter, 97
- coord\_i
  - support-meat.hpp, 333
- coord\_j
  - support-meat.hpp, 333
- cooriantes\_n\_free
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 186
- cooriantes\_n\_locked
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 186
- coordinates\_free
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 166
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 186
- coordinates\_locked
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 166
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 187
- copy\_data
  - barray-meat.hpp, 211
- barraydense-meat.hpp, 239
- count
  - Counter< Array\_Type, Data\_Type >, 100
- count\_all
  - StatsCounter< Array\_Type, Data\_Type >, 177
- count\_current
  - StatsCounter< Array\_Type, Data\_Type >, 177
- count\_fun
  - Counter< Array\_Type, Data\_Type >, 101
  - counters-meat.hpp, 267
- count\_fun\_
  - counters-meat.hpp, 272
- count\_init
  - StatsCounter< Array\_Type, Data\_Type >, 177
- Counter
  - Counter< Array\_Type, Data\_Type >, 98, 99
- counter
  - counters-meat.hpp, 273
  - statscounter-meat.hpp, 321
- Counter< Array\_Type, Data\_Type >, 97
  - ~Counter, 99
  - count, 100
  - count\_fun, 101
  - Counter, 98, 99
  - data, 101
  - delete\_data, 101
  - desc, 101
  - get\_description, 100
  - get\_name, 100
  - init, 100
  - init\_fun, 101
  - name, 102
  - operator=, 100
- counter\_
  - counters-meat.hpp, 273
- counter\_absdiff
  - Network counters, 13
- counter\_co\_opt
  - Phylo counters, 20
- counter\_cogain
  - Phylo counters, 21
- counter\_css\_census01
  - network-css.hpp, 279
- counter\_css\_census02
  - network-css.hpp, 280
- counter\_css\_census03
  - network-css.hpp, 280
- counter\_css\_census04
  - network-css.hpp, 280
- counter\_css\_census05
  - network-css.hpp, 280
- counter\_css\_census06
  - network-css.hpp, 281
- counter\_css\_census07
  - network-css.hpp, 281
- counter\_css\_census08
  - network-css.hpp, 281
- counter\_css\_census09

- network-css.hpp, 281
- counter\_css\_census10
  - network-css.hpp, 282
- counter\_css\_completely\_false\_recip\_comiss
  - network-css.hpp, 282
- counter\_css\_completely\_false\_recip\_omiss
  - network-css.hpp, 282
- counter\_css\_mixed\_recip
  - network-css.hpp, 282
- counter\_css\_partially\_false\_recip\_commi
  - network-css.hpp, 283
- counter\_css\_partially\_false\_recip\_omiss
  - network-css.hpp, 283
- counter\_ctriads
  - Network counters, 13, 14
- counter\_degree
  - Network counters, 14
- counter\_deleted
  - statscounter-meat.hpp, 321
- counter\_density
  - Network counters, 14
- counter\_diff
  - Network counters, 14
- counter\_edges
  - Network counters, 14
- Counter\_fun\_type
  - typedefs.hpp, 338
- COUNTER\_FUNCTION
  - barry.hpp, 259
- counter\_gains
  - Phylo counters, 21
- counter\_gains\_from\_0
  - Phylo counters, 21
- counter\_gains\_k\_offspring
  - Phylo counters, 21
- counter\_genes\_changing
  - Phylo counters, 22
- counter\_idegree
  - Network counters, 15
- counter\_idegree15
  - Network counters, 15
- counter\_isolates
  - Network counters, 15, 16
- counter\_istar2
  - Network counters, 16
- counter\_k\_genes\_changing
  - Phylo counters, 22
- COUNTER\_LAMBDA
  - barry.hpp, 260
- counter\_less\_than\_p\_prop\_genes\_changing
  - Phylo counters, 22
- counter\_longest
  - Phylo counters, 22
- counter\_loss
  - Phylo counters, 23
- counter\_maxfun
  - Phylo counters, 23
- counter\_mutual
  - Network counters, 16
- counter\_neofun
  - Phylo counters, 23
- counter\_neofun\_a2b
  - Phylo counters, 23
- counter\_nodecov
  - Network counters, 16
- counter\_nodeicov
  - Network counters, 17
- counter\_nodematch
  - Network counters, 17
- counter\_nodeocov
  - Network counters, 17
- counter\_odegree
  - Network counters, 17
- counter\_odegree15
  - Network counters, 18
- counter\_ostar2
  - Network counters, 18
- counter\_overall\_changes
  - Phylo counters, 24
- counter\_overall\_gains
  - Phylo counters, 24
- counter\_overall\_loss
  - Phylo counters, 24
- counter\_pairwise\_preserving
  - Phylo counters, 24
- counter\_prop\_genes\_changing
  - Phylo counters, 25
- counter\_subfun
  - Phylo counters, 25
- COUNTER\_TEMPLATE
  - counters-meat.hpp, 266–268
- COUNTER\_TEMPLATE\_ARGS
  - counters-meat.hpp, 266
- counter\_ttriads
  - Network counters, 18, 19
- COUNTER\_TYPE
  - counters-meat.hpp, 266
- Counters
  - Counters< Array\_Type, Data\_Type >, 103
- counters
  - statscounter-meat.hpp, 321
  - support-meat.hpp, 333
- Counters< Array\_Type, Data\_Type >, 102
  - ~Counters, 103
  - add\_counter, 104
  - clear, 104
  - Counters, 103
  - get\_descriptions, 105
  - get\_names, 105
  - operator=, 105
  - operator[], 106
  - size, 106
- counters-meat.hpp
  - count\_fun, 267
  - count\_fun\_, 272
  - counter, 273



- counter\_, 273
- COUNTER\_TEMPLATE, 266–268
- COUNTER\_TEMPLATE\_ARGS, 266
- COUNTER\_TYPE, 266
- COUNTERS\_TEMPLATE, 266, 268–270
- COUNTERS\_TEMPLATE\_ARGS, 266
- COUNTERS\_TYPE, 266
- data, 270
- data\_, 273
- delete\_data, 270
- delete\_data\_, 273
- delete\_to\_be\_deleted, 270, 271
- desc, 271
- desc\_, 274
- i, 274
- init\_fun, 271
- init\_fun\_, 274
- j, 274
- name, 272
- name\_, 274
- noexcept, 275
- push\_back, 272
- return, 275
- to\_be\_deleted, 272
- counters\_
  - statscounter-meat.hpp, 321
  - support-meat.hpp, 333
- COUNTERS\_TEMPLATE
  - counters-meat.hpp, 266, 268–270
- COUNTERS\_TEMPLATE\_ARGS
  - counters-meat.hpp, 266
- COUNTERS\_TYPE
  - counters-meat.hpp, 266
- Counting, 11
- counts
  - PhyloRuleDynData, 160
- Counts\_type
  - typedefs.hpp, 338
- CSS\_APPEND
  - network-css.hpp, 277
- CSS\_CASE\_ELSE
  - network-css.hpp, 277
- CSS\_CASE\_PERCEIVED
  - network-css.hpp, 278
- CSS\_CASE\_TRUTH
  - network-css.hpp, 278
- CSS\_CHECK\_SIZE
  - network-css.hpp, 278
- CSS\_CHECK\_SIZE\_INIT
  - network-css.hpp, 278
- CSS\_NET\_COUNTER\_LAMBDA\_INIT
  - network-css.hpp, 278
- CSS\_PERCEIVED\_CELLS
  - network-css.hpp, 279
- CSS\_SIZE
  - network-css.hpp, 279
- CSS\_TRUE\_CELLS
  - network-css.hpp, 279
- current\_col
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, 97
- current\_row
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, 97
- current\_stats
  - statscounter-meat.hpp, 321
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 187
- D
  - BArray< Cell\_Type, Data\_Type >, 36
  - BArrayDense< Cell\_Type, Data\_Type >, 55
  - Rule< Array\_Type, Data\_Type >, 171
- dat
  - Flock, 115
- data
  - barray-meat.hpp, 211
  - barraydense-meat.hpp, 239
  - Counter< Array\_Type, Data\_Type >, 101
  - counters-meat.hpp, 270
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 166
- data\_
  - counters-meat.hpp, 273
- DEFAULT\_DUPLICATION
  - phylo.hpp, 293
- default\_val
  - BArray< Cell\_Type, Data\_Type >, 36
  - BArrayDense< Cell\_Type, Data\_Type >, 55
- delete\_counters
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 187
  - support-meat.hpp, 333
- delete\_data
  - barray-meat.hpp, 211
  - barraydense-meat.hpp, 239
  - Counter< Array\_Type, Data\_Type >, 101
  - counters-meat.hpp, 270
- delete\_data\_
  - barray-meat.hpp, 212
  - barraydense-meat.hpp, 239
  - counters-meat.hpp, 273
- delete\_engine
  - Geese, 129
- delete\_rules
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 187
  - support-meat.hpp, 334
- delete\_rules\_dyn
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 187
  - support-meat.hpp, 334
- delete\_support
  - Geese, 130
- delete\_to\_be\_deleted
  - counters-meat.hpp, 270, 271

- desc
  - Counter< Array\_Type, Data\_Type >, 101
  - counters-meat.hpp, 271
- desc\_
  - counters-meat.hpp, 274
- directed
  - NetworkData, 148
- DUPL\_DUPL
  - phylo.hpp, 293
- DUPL\_EITH
  - phylo.hpp, 293
- DUPL\_SPEC
  - phylo.hpp, 293
- duplication
  - Node, 153
  - NodeData, 156
  - PhyloRuleDynData, 160
- el
  - barraydense-meat.hpp, 240
- el\_colsums
  - barraydense-meat.hpp, 240
- el\_rowsums
  - barraydense-meat.hpp, 240
- else
  - barray-meat.hpp, 212
  - barraydense-meat.hpp, 240
  - support-meat.hpp, 334
- empty
  - PhyloCounterData, 157
- EmptyArray
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 166
  - statscounter-meat.hpp, 322
- end
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 70
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 72
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 74
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 77
  - BArrayVector< Cell\_Type, Data\_Type >, 84
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 88
  - PhyloCounterData, 157
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 164
  - Progress, 169
- Entries
  - Entries< Cell\_Type >, 107
- Entries< Cell\_Type >, 107
  - ~Entries, 108
  - Entries, 107
  - resize, 108
  - source, 108
  - target, 108
  - val, 108
- eval\_rules\_dyn
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 183
- EXISTS, 29
- AS\_ONE, 29
- AS\_ZERO, 29
- BOTH, 29
- NONE, 30
- ONE, 30
- TWO, 30
- UNKNOWN, 30
- f\_
  - statscounter-meat.hpp, 322
  - support-meat.hpp, 334
- false
  - barray-meat.hpp, 212
  - barraydense-meat.hpp, 240
- first
  - barray-meat.hpp, 212
- Flock, 109
  - ~Flock, 110
  - add\_data, 110
  - colnames, 111
  - dat, 115
  - Flock, 110
  - get\_counters, 111
  - get\_model, 111
  - get\_stats\_support, 111
  - get\_stats\_target, 111
  - get\_support\_fun, 112
  - init, 112
  - initialized, 115
  - likelihood\_joint, 112
  - model, 115
  - nfunctions, 115
  - nfuns, 112
  - nleaves, 113
  - nnodes, 113
  - nterms, 113
  - ntrees, 113
  - operator(), 113
  - parse\_polytomies, 114
  - print, 114
  - rengine, 115
  - set\_seed, 114
  - support\_size, 114
- flush\_data
  - BArray< Cell\_Type, Data\_Type >, 36
- for
  - barray-meat-operators.hpp, 196
  - barray-meat.hpp, 205, 206
  - barraydense-meat.hpp, 234
  - statscounter-meat.hpp, 319
  - support-meat.hpp, 328
- FreqTable
  - FreqTable< T >, 116
- FreqTable< T >, 116
  - ~FreqTable, 117
  - add, 117
  - as\_vector, 117
  - clear, 117
  - FreqTable, 116

- get\_data, 117
- get\_index, 118
- make\_hash, 118
- print, 118
- reserve, 118
- size, 118
- Geese, 119
  - ~Geese, 122
  - calc\_reduced\_sequence, 123
  - calc\_sequence, 123
  - colnames, 123
  - delete\_engine, 129
  - delete\_support, 130
  - Geese, 122
  - get\_annotated\_nodes, 123
  - get\_counters, 123
  - get\_model, 123
  - get\_probabilities, 124
  - get\_engine, 124
  - get\_states, 124
  - get\_support\_fun, 124
  - inherit\_support, 124
  - init, 125
  - init\_node, 125
  - initialized, 130
  - likelihood, 125
  - likelihood\_exhaust, 125
  - map\_to\_nodes, 130
  - nannotations, 125
  - nfunctions, 130
  - nfuncs, 126
  - nleaves, 126
  - nnodes, 126
  - nodes, 130
  - nterms, 126
  - observed\_counts, 126
  - operator=, 127
  - parse\_polytomies, 127
  - predict, 127
  - predict\_backend, 127
  - predict\_exhaust, 128
  - predict\_exhaust\_backend, 128
  - predict\_sim, 128
  - print, 128
  - print\_observed\_counts, 128
  - pset\_loc, 130
  - reduced\_sequence, 131
  - sequence, 131
  - set\_seed, 129
  - simulate, 129
  - support\_size, 129
  - update\_annotations, 129
- geese-bones.hpp
  - INITIALIZED, 304
  - keygen\_full, 305
  - RULE\_FUNCTION, 305
  - vec\_diff, 305
  - vector\_caster, 305
- gen\_key
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 138
- get\_annotated\_nodes
  - Geese, 123
- get\_arrays2support
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 138
- get\_cell
  - BArray< Cell\_Type, Data\_Type >, 36
  - BArrayDense< Cell\_Type, Data\_Type >, 55
- get\_col\_vec
  - BArray< Cell\_Type, Data\_Type >, 36, 37
  - BArrayDense< Cell\_Type, Data\_Type >, 56
- get\_counters
  - Flock, 111
  - Geese, 123
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 138
  - PhyloCounterData, 158
  - StatsCounter< Array\_Type, Data\_Type >, 177
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 183
- get\_counts
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 183
- get\_current\_stats
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 184
- get\_data
  - BArrayDense< Cell\_Type, Data\_Type >, 56
  - FreqTable< T >, 117
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 165
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 184
- get\_data\_ptr
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 165
- get\_description
  - Counter< Array\_Type, Data\_Type >, 100
- get\_descriptions
  - Counters< Array\_Type, Data\_Type >, 105
  - StatsCounter< Array\_Type, Data\_Type >, 178
- get\_entries
  - BArray< Cell\_Type, Data\_Type >, 37
  - BArrayDense< Cell\_Type, Data\_Type >, 56
- get\_index
  - FreqTable< T >, 118
- get\_last\_name
  - phylo.hpp, 298
- get\_model
  - Flock, 111

- Geese, [123](#)
- get\_name
  - Counter< Array\_Type, Data\_Type >, [100](#)
- get\_names
  - Counters< Array\_Type, Data\_Type >, [105](#)
  - StatsCounter< Array\_Type, Data\_Type >, [178](#)
- get\_norm\_const
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [138](#)
- get\_parent
  - Node, [151](#)
- get\_probabilities
  - Geese, [124](#)
- get\_pset
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [138](#)
- get\_pset\_arrays
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [139](#)
- get\_pset\_probs
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [139](#)
- get\_pset\_stats
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [139](#)
- get\_rengine
  - Geese, [124](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [139](#)
- get\_row\_vec
  - BArray< Cell\_Type, Data\_Type >, [37](#)
  - BArrayDense< Cell\_Type, Data\_Type >, [56](#), [57](#)
- get\_rules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [140](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [184](#)
- get\_rules\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [140](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [184](#)
- get\_seq
  - Rules< Array\_Type, Data\_Type >, [173](#)
- get\_states
  - Geese, [124](#)
- get\_stats\_support
  - Flock, [111](#)
- Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [140](#)
- get\_stats\_target
  - Flock, [111](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [140](#)
- get\_support\_fun
  - Flock, [112](#)
  - Geese, [124](#)
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [140](#)
- hashes
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [188](#)
  - support-meat.hpp, [334](#)
- hashes\_initialized
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [188](#)
- i
  - counters-meat.hpp, [274](#)
- i1
  - barray-meat.hpp, [212](#)
  - barraydense-meat.hpp, [241](#)
- id
  - Node, [153](#)
- if
  - barray-meat.hpp, [206–208](#)
  - barraydense-meat.hpp, [234](#)
  - support-meat.hpp, [328](#)
- IF\_MATCHES
  - phylo.hpp, [293](#)
- IF\_NOTMATCHES
  - phylo.hpp, [294](#)
- include/barry/barray-bones.hpp, [191](#)
- include/barry/barray-iterator.hpp, [192](#)
- include/barry/barray-meat-operators.hpp, [193](#)
- include/barry/barray-meat.hpp, [198](#)
- include/barry/barraycell-bones.hpp, [216](#)
- include/barry/barraycell-meat.hpp, [216](#)
- include/barry/barraydense-bones.hpp, [217](#)
- include/barry/barraydense-meat-operators.hpp, [219](#)
- include/barry/barraydense-meat.hpp, [223](#)
- include/barry/barraydensecell-bones.hpp, [244](#)
- include/barry/barraydensecell-meat.hpp, [245](#)
- include/barry/barraydensecol-bones.hpp, [246](#)
- include/barry/barraydenserow-bones.hpp, [247](#)
- include/barry/barrayrow-bones.hpp, [249](#)
- include/barry/barrayrow-meat.hpp, [249](#)
- include/barry/barrayvector-bones.hpp, [252](#)
- include/barry/barrayvector-meat.hpp, [253](#)
- include/barry/barry-configuration.hpp, [254](#)
- include/barry/barry-debug.hpp, [256](#)

- include/barry/barry-macros.hpp, 256
- include/barry/barry.hpp, 258
- include/barry/cell-bones.hpp, 261
- include/barry/cell-meat.hpp, 261
- include/barry/col-bones.hpp, 262
- include/barry/counters-bones.hpp, 262
- include/barry/counters-meat.hpp, 264
- include/barry/counters/network-css.hpp, 276
- include/barry/counters/network.hpp, 284
- include/barry/counters/phylo.hpp, 291
- include/barry/model-bones.hpp, 298
- include/barry/model-meat.hpp, 300
- include/barry/models/geese.hpp, 302
- include/barry/models/geese/flock-bones.hpp, 303
- include/barry/models/geese/flock-meat.hpp, 303
- include/barry/models/geese/geese-bones.hpp, 304
- include/barry/models/geese/geese-meat-constructors.hpp, is\_col
  - 306
- include/barry/models/geese/geese-meat-likelihood.hpp,
  - 306
- include/barry/models/geese/geese-meat-likelihood\_exhaust.hpp,
  - 307
- include/barry/models/geese/geese-meat-predict.hpp,
  - 308
- include/barry/models/geese/geese-meat-predict\_exhaust.hpp,
  - 308
- include/barry/models/geese/geese-meat-predict\_sim.hpp,
  - 309
- include/barry/models/geese/geese-meat-simulate.hpp,
  - 309
- include/barry/models/geese/geese-meat.hpp, 310
- include/barry/models/geese/geese-node-bones.hpp,
  - 310
- include/barry/powerset-bones.hpp, 311
- include/barry/powerset-meat.hpp, 312
- include/barry/progress.hpp, 313
- include/barry/rules-bones.hpp, 313
- include/barry/rules-meat.hpp, 315
- include/barry/statscounter-bones.hpp, 315
- include/barry/statscounter-meat.hpp, 317
- include/barry/statsdb.hpp, 323
- include/barry/support-bones.hpp, 323
- include/barry/support-meat.hpp, 325
- include/barry/typedefs.hpp, 336
- indices
  - NetCounterData, 146
- inherit\_support
  - Geese, 124
- init
  - Counter< Array\_Type, Data\_Type >, 100
  - Flock, 112
  - Geese, 125
- init\_fun
  - Counter< Array\_Type, Data\_Type >, 101
  - counters-meat.hpp, 271
- init\_fun\_
  - counters-meat.hpp, 274
- init\_node
  - Geese, 125
- init\_support
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 165
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 184
- INITIALIZED
  - geese-bones.hpp, 304
- initialized
  - Flock, 115
  - Geese, 130
- insert\_cell
  - BArray< Cell\_Type, Data\_Type >, 37, 38
  - BArrayDense< Cell\_Type, Data\_Type >, 57
  - barraydense-meat.hpp, 235
  - support-meat.hpp, 329
- is\_col
  - BArrayVector< Cell\_Type, Data\_Type >, 84
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 88
- is\_dense
  - BArray< Cell\_Type, Data\_Type >, 38
  - BArrayDense< Cell\_Type, Data\_Type >, 57
- IS\_DUPLICATION
  - phylo.hpp, 294
- IS\_EITHER
  - phylo.hpp, 294
- is\_empty
  - BArray< Cell\_Type, Data\_Type >, 38
  - BArrayDense< Cell\_Type, Data\_Type >, 57
- is\_leaf
  - Node, 152
- is\_row
  - BArrayVector< Cell\_Type, Data\_Type >, 84
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 88
- IS\_SPECIATION
  - phylo.hpp, 294
- iter
  - ConstBArrayRowIter< Cell\_Type, Data\_Type >, 97
- j
  - barray-meat.hpp, 213
  - barraydense-meat.hpp, 241
  - counters-meat.hpp, 274
  - statscounter-meat.hpp, 322
- j0
  - barray-meat.hpp, 213
  - barraydense-meat.hpp, 241
- j1
  - barray-meat.hpp, 213
  - barraydense-meat.hpp, 241
- keygen\_default
  - model-bones.hpp, 299
- keygen\_full
  - geese-bones.hpp, 305
- lb
  - PhyloRuleDynData, 160
- likelihood

- Geese, [125](#)
- Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [141](#)
- likelihood\_
  - model-meat.hpp, [301](#)
- likelihood\_exhaust
  - Geese, [125](#)
- likelihood\_joint
  - Flock, [112](#)
- likelihood\_total
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [141](#)
- M
  - barray-meat.hpp, [209](#), [213](#)
  - barraydense-meat.hpp, [235](#), [241](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [167](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [188](#)
- M\_
  - barray-meat.hpp, [213](#)
  - barraydense-meat.hpp, [241](#)
- MAKE\_DUPL\_VARS
  - phylo.hpp, [294](#)
- make\_hash
  - FreqTable< T >, [118](#)
- Map
  - barry-configuration.hpp, [255](#)
- map\_to\_nodes
  - Geese, [130](#)
- MapVec\_type
  - typedefs.hpp, [339](#)
- max\_num\_elements
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [188](#)
- Model
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [134](#)
- model
  - Flock, [115](#)
- Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [131](#)
- ~Model, [135](#)
- add\_array, [135](#)
- add\_counter, [135](#), [136](#)
- add\_rule, [136](#)
- add\_rule\_dyn, [136](#), [137](#)
- colnames, [137](#)
- conditional\_prob, [137](#)
- gen\_key, [138](#)
- get\_arrays2support, [138](#)
- get\_counters, [138](#)
- get\_norm\_const, [138](#)
- get\_pset, [138](#)
- get\_pset\_arrays, [139](#)
- get\_pset\_probs, [139](#)
- get\_pset\_stats, [139](#)
- get\_engine, [139](#)
- get\_rules, [140](#)
- get\_rules\_dyn, [140](#)
- get\_stats\_support, [140](#)
- get\_stats\_target, [140](#)
- get\_support\_fun, [140](#)
- likelihood, [141](#)
- likelihood\_total, [141](#)
- Model, [134](#)
- nterms, [142](#)
- operator=, [142](#)
- print, [142](#)
- print\_stats, [142](#)
- sample, [142](#), [143](#)
- set\_counters, [143](#)
- set\_keygen, [143](#)
- set\_engine, [143](#)
- set\_rules, [143](#)
- set\_rules\_dyn, [144](#)
- set\_seed, [144](#)
- size, [144](#)
- size\_unique, [144](#)
- store\_psets, [144](#)
- support\_size, [145](#)
- model-bones.hpp
  - keygen\_default, [299](#)
- model-meat.hpp
  - likelihood\_, [301](#)
  - MODEL\_TEMPLATE, [300](#), [301](#)
  - MODEL\_TEMPLATE\_ARGS, [300](#)
  - MODEL\_TYPE, [301](#)
  - update\_normalizing\_constant, [302](#)
- MODEL\_TEMPLATE
  - model-meat.hpp, [300](#), [301](#)
- MODEL\_TEMPLATE\_ARGS
  - model-meat.hpp, [300](#)
- MODEL\_TYPE
  - model-meat.hpp, [301](#)
- N
  - barray-meat.hpp, [214](#)
  - barraydense-meat.hpp, [242](#)
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [167](#)
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [188](#)
- n\_counters
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, [189](#)
- n\_free
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [167](#)
- n\_locked
  - PowerSet< Array\_Type, Data\_Rule\_Type >, [167](#)
- name
  - Counter< Array\_Type, Data\_Type >, [102](#)

- counters-meat.hpp, 272
- name\_
  - counters-meat.hpp, 274
- nannotations
  - Geese, 125
- narray
  - Node, 153
- NCells
  - barray-meat.hpp, 214
- ncol
  - BArray< Cell\_Type, Data\_Type >, 38
  - BArrayDense< Cell\_Type, Data\_Type >, 58
- NET\_C\_DATA\_IDX
  - network.hpp, 287
- NET\_C\_DATA\_NUM
  - network.hpp, 287
- NetCounter
  - network.hpp, 289
- NetCounterData, 145
  - ~NetCounterData, 146
  - indices, 146
  - NetCounterData, 145, 146
  - numbers, 146
- NetCounters
  - network.hpp, 289
- NetModel
  - network.hpp, 289
- NetRule
  - network.hpp, 289
- NetRules
  - network.hpp, 289
- NetStatsCounter
  - network.hpp, 289
- NetSupport
  - network.hpp, 290
- Network
  - network.hpp, 290
- Network counters, 12
  - counter\_absdiff, 13
  - counter\_ctriads, 13, 14
  - counter\_degree, 14
  - counter\_density, 14
  - counter\_diff, 14
  - counter\_edges, 14
  - counter\_iddegree, 15
  - counter\_iddegree15, 15
  - counter\_isolates, 15, 16
  - counter\_istar2, 16
  - counter\_mutual, 16
  - counter\_nodecov, 16
  - counter\_nodeicov, 17
  - counter\_nodematch, 17
  - counter\_nodeocov, 17
  - counter\_odegree, 17
  - counter\_odegree15, 18
  - counter\_ostar2, 18
  - counter\_ttriads, 18, 19
  - NETWORK\_COUNTER, 19
- network-css.hpp
  - counter\_css\_census01, 279
  - counter\_css\_census02, 280
  - counter\_css\_census03, 280
  - counter\_css\_census04, 280
  - counter\_css\_census05, 280
  - counter\_css\_census06, 281
  - counter\_css\_census07, 281
  - counter\_css\_census08, 281
  - counter\_css\_census09, 281
  - counter\_css\_census10, 282
  - counter\_css\_completely\_false\_recip\_comiss, 282
  - counter\_css\_completely\_false\_recip\_omiss, 282
  - counter\_css\_mixed\_recip, 282
  - counter\_css\_partially\_false\_recip\_commi, 283
  - counter\_css\_partially\_false\_recip\_omiss, 283
  - CSS\_APPEND, 277
  - CSS\_CASE\_ELSE, 277
  - CSS\_CASE\_PERCEIVED, 278
  - CSS\_CASE\_TRUTH, 278
  - CSS\_CHECK\_SIZE, 278
  - CSS\_CHECK\_SIZE\_INIT, 278
  - CSS\_NET\_COUNTER\_LAMBDA\_INIT, 278
  - CSS\_PERCEIVED\_CELLS, 279
  - CSS\_SIZE, 279
  - CSS\_TRUE\_CELLS, 279
- network.hpp
  - BARRY\_ZERO\_NETWORK, 286
  - BARRY\_ZERO\_NETWORK\_DENSE, 287
  - NET\_C\_DATA\_IDX, 287
  - NET\_C\_DATA\_NUM, 287
  - NetCounter, 289
  - NetCounters, 289
  - NetModel, 289
  - NetRule, 289
  - NetRules, 289
  - NetStatsCounter, 289
  - NetSupport, 290
  - Network, 290
  - NETWORK\_COUNTER, 287
  - NETWORK\_COUNTER\_LAMBDA, 287
  - NETWORK\_RULE, 288
  - NETWORK\_RULE\_LAMBDA, 288
  - NetworkDense, 290
  - NETWORKDENSE\_COUNTER\_LAMBDA, 288
  - rules\_zerodiag, 290
- NETWORK\_COUNTER
  - Network counters, 19
  - network.hpp, 287
- NETWORK\_COUNTER\_LAMBDA
  - network.hpp, 287
- NETWORK\_RULE
  - network.hpp, 288
- NETWORK\_RULE\_LAMBDA
  - network.hpp, 288
- NetworkData, 147
  - ~NetworkData, 148
  - directed, 148



- NetworkData, 147, 148
- vertex\_attr, 148
- NetworkDense
  - network.hpp, 290
- NETWORKDENSE\_COUNTER\_LAMBDA
  - network.hpp, 288
- next
  - Progress, 169
- nfunctions
  - Flock, 115
  - Geese, 130
- nfuncs
  - Flock, 112
  - Geese, 126
- nleaves
  - Flock, 113
  - Geese, 126
- nnodes
  - Flock, 113
  - Geese, 126
- nnozero
  - BArray< Cell\_Type, Data\_Type >, 39
  - BArrayDense< Cell\_Type, Data\_Type >, 58
- Node, 149
  - ~Node, 151
  - annotations, 152
  - array, 152
  - arrays, 152
  - duplication, 153
  - get\_parent, 151
  - id, 153
  - is\_leaf, 152
  - narray, 153
  - Node, 150, 151
  - noffspring, 152
  - offspring, 153
  - ord, 153
  - parent, 154
  - probability, 154
  - subtree\_prob, 154
  - visited, 154
- NodeData, 155
  - blengths, 155
  - duplication, 156
  - NodeData, 155
  - states, 156
- nodes
  - Geese, 130
- noexcept
  - counters-meat.hpp, 275
- noffspring
  - Node, 152
- NONE
  - CHECK, 28
  - EXISTS, 30
- nrow
  - BArray< Cell\_Type, Data\_Type >, 39
  - BArrayDense< Cell\_Type, Data\_Type >, 58
- nterms
  - Flock, 113
  - Geese, 126
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 142
- ntrees
  - Flock, 113
- numbers
  - NetCounterData, 146
- observed\_counts
  - Geese, 126
- offspring
  - Node, 153
- ONE
  - CHECK, 28
  - EXISTS, 30
- operator BArrayRow< Cell\_Type, Data\_Type >
  - BArrayRow< Cell\_Type, Data\_Type >, 79
- operator BArrayRow\_const< Cell\_Type, Data\_Type >
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 81
- operator Cell\_Type
  - BArrayCell< Cell\_Type, Data\_Type >, 46
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 49
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 66
  - Cell< Cell\_Type >, 93
- operator std::vector< Cell\_Type >
  - BArrayVector< Cell\_Type, Data\_Type >, 85
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 88
- operator!=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 49
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 81
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 88
  - Cell< Cell\_Type >, 93
- operator<
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 49
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 81
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 89
- operator<=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 49
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 82
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 89
- operator>
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 49
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 82
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 89
- operator>=
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 50
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 82
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 89
- operator\*=
  - BArray< Cell\_Type, Data\_Type >, 39
  - BArrayCell< Cell\_Type, Data\_Type >, 46
  - BArrayDense< Cell\_Type, Data\_Type >, 58
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 66
  - BArrayRow< Cell\_Type, Data\_Type >, 79
  - BArrayVector< Cell\_Type, Data\_Type >, 85
- operator()



- BArray< Cell\_Type, Data\_Type >, 39
- barray-meat-operators.hpp, 197
- BArrayDense< Cell\_Type, Data\_Type >, 58
- BArrayDenseCol< Cell\_Type, Data\_Type >, 70
- BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 72
- BArrayDenseRow< Cell\_Type, Data\_Type >, 75
- BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 77
- Flock, 113
- PhyloCounterData, 158
- Rule< Array\_Type, Data\_Type >, 171
- Rules< Array\_Type, Data\_Type >, 174
- vecHasher< T >, 189
- operator+=
  - BArray< Cell\_Type, Data\_Type >, 39, 40
  - BArrayCell< Cell\_Type, Data\_Type >, 46
  - BArrayDense< Cell\_Type, Data\_Type >, 59
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 67
  - BArrayRow< Cell\_Type, Data\_Type >, 79
  - BArrayVector< Cell\_Type, Data\_Type >, 85
- operator-=
  - BArray< Cell\_Type, Data\_Type >, 40
  - BArrayCell< Cell\_Type, Data\_Type >, 47
  - BArrayDense< Cell\_Type, Data\_Type >, 59
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 67
  - BArrayRow< Cell\_Type, Data\_Type >, 79
  - BArrayVector< Cell\_Type, Data\_Type >, 85
- operator/=
  - BArray< Cell\_Type, Data\_Type >, 40
  - BArrayCell< Cell\_Type, Data\_Type >, 47
  - BArrayDense< Cell\_Type, Data\_Type >, 60
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 67
  - BArrayRow< Cell\_Type, Data\_Type >, 80
  - BArrayVector< Cell\_Type, Data\_Type >, 85
- operator=
  - BArray< Cell\_Type, Data\_Type >, 41
  - BArrayCell< Cell\_Type, Data\_Type >, 47
  - BArrayDense< Cell\_Type, Data\_Type >, 60
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 67
  - BArrayRow< Cell\_Type, Data\_Type >, 80
  - BArrayVector< Cell\_Type, Data\_Type >, 86
  - Cell< Cell\_Type >, 93, 94
  - Counter< Array\_Type, Data\_Type >, 100
  - Counters< Array\_Type, Data\_Type >, 105
  - Geese, 127
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 142
  - Rules< Array\_Type, Data\_Type >, 174
- operator==
  - BArray< Cell\_Type, Data\_Type >, 41
  - BArrayCell< Cell\_Type, Data\_Type >, 47
  - BArrayCell\_const< Cell\_Type, Data\_Type >, 49
  - BArrayDense< Cell\_Type, Data\_Type >, 60
  - BArrayDenseCell< Cell\_Type, Data\_Type >, 67
  - BArrayRow< Cell\_Type, Data\_Type >, 80
  - BArrayRow\_const< Cell\_Type, Data\_Type >, 82
- BArrayVector< Cell\_Type, Data\_Type >, 86
- BArrayVector\_const< Cell\_Type, Data\_Type >, 89
- Cell< Cell\_Type >, 94
- operator[]
  - Counters< Array\_Type, Data\_Type >, 106
  - PhyloCounterData, 158
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 165
- ord
  - Node, 153
- out\_of\_range
  - BArray< Cell\_Type, Data\_Type >, 41
  - BArrayDense< Cell\_Type, Data\_Type >, 60
- parent
  - Node, 154
- parse\_polytomies
  - Flock, 114
  - Geese, 127
- Phylo counters, 19
  - counter\_co\_opt, 20
  - counter\_cogain, 21
  - counter\_gains, 21
  - counter\_gains\_from\_0, 21
  - counter\_gains\_k\_offspring, 21
  - counter\_genes\_changing, 22
  - counter\_k\_genes\_changing, 22
  - counter\_less\_than\_p\_prop\_genes\_changing, 22
  - counter\_longest, 22
  - counter\_loss, 23
  - counter\_maxfun, 23
  - counter\_neofun, 23
  - counter\_neofun\_a2b, 23
  - counter\_overall\_changes, 24
  - counter\_overall\_gains, 24
  - counter\_overall\_loss, 24
  - counter\_pairwise\_preserving, 24
  - counter\_prop\_genes\_changing, 25
  - counter\_subfun, 25
- Phylo rules, 25
  - rule\_dyn\_limit\_changes, 26
- phylo.hpp
  - DEFAULT\_DUPLICATION, 293
  - DUPL\_DUPL, 293
  - DUPL\_EITH, 293
  - DUPL\_SPEC, 293
  - get\_last\_name, 298
  - IF\_MATCHES, 293
  - IF\_NOTMATCHES, 294
  - IS\_DUPLICATION, 294
  - IS\_EITHER, 294
  - IS\_SPECIATION, 294
  - MAKE\_DUPL\_VARS, 294
  - PHYLO\_CHECK\_MISSING, 295
  - PHYLO\_COUNTER\_LAMBDA, 295
  - PHYLO\_RULE\_DYN\_LAMBDA, 295
  - PhyloArray, 296
  - PhyloCounter, 296
  - PhyloCounters, 296
  - PhyloModel, 296

- PhyloPowerSet, 296
- PhyloRule, 296
- PhyloRuleData, 297
- PhyloRuleDyn, 297
- PhyloRules, 297
- PhyloRulesDyn, 297
- PhyloStatsCounter, 297
- PhyloSupport, 297
- PHYLO\_CHECK\_MISSING
  - phylo.hpp, 295
- PHYLO\_COUNTER\_LAMBDA
  - phylo.hpp, 295
- PHYLO\_RULE\_DYN\_LAMBDA
  - phylo.hpp, 295
- PhyloArray
  - phylo.hpp, 296
- PhyloCounter
  - phylo.hpp, 296
- PhyloCounterData, 156
  - at, 157
  - begin, 157
  - empty, 157
  - end, 157
  - get\_counters, 158
  - operator(), 158
  - operator[], 158
  - PhyloCounterData, 157
  - push\_back, 158
  - reserve, 158
  - shrink\_to\_fit, 158
  - size, 159
- PhyloCounters
  - phylo.hpp, 296
- PhyloModel
  - phylo.hpp, 296
- PhyloPowerSet
  - phylo.hpp, 296
- PhyloRule
  - phylo.hpp, 296
- PhyloRuleData
  - phylo.hpp, 297
- PhyloRuleDyn
  - phylo.hpp, 297
- PhyloRuleDynData, 159
  - ~PhyloRuleDynData, 160
  - counts, 160
  - duplication, 160
  - lb, 160
  - PhyloRuleDynData, 159
  - pos, 160
  - ub, 161
- PhyloRules
  - phylo.hpp, 297
- PhyloRulesDyn
  - phylo.hpp, 297
- PhyloStatsCounter
  - phylo.hpp, 297
- PhyloSupport
  - phylo.hpp, 297
- POS
  - barraydense-meat-operators.hpp, 221
  - barraydense-meat.hpp, 226
  - barraydensecell-bones.hpp, 244
  - barraydensecell-meat.hpp, 246
  - barraydensecol-bones.hpp, 247
  - barraydenserow-bones.hpp, 248
- pos
  - PhyloRuleDynData, 160
- POS\_N
  - barraydense-meat-operators.hpp, 221
  - barraydense-meat.hpp, 226
  - barraydensecol-bones.hpp, 247
  - barraydenserow-bones.hpp, 248
- PowerSet
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 163
- PowerSet< Array\_Type, Data\_Rule\_Type >, 161
  - ~PowerSet, 163
  - add\_rule, 164
  - begin, 164
  - calc, 164
  - coordinates\_free, 166
  - coordinates\_locked, 166
  - data, 166
  - EmptyArray, 166
  - end, 164
  - get\_data, 165
  - get\_data\_ptr, 165
  - init\_support, 165
  - M, 167
  - N, 167
  - n\_free, 167
  - n\_locked, 167
  - operator[], 165
  - PowerSet, 163
  - reset, 165
  - rules, 167
  - rules\_deleted, 167
  - size, 166
- predict
  - Geese, 127
- predict\_backend
  - Geese, 127
- predict\_exhaust
  - Geese, 128
- predict\_exhaust\_backend
  - Geese, 128
- predict\_sim
  - Geese, 128
- print
  - BArray< Cell\_Type, Data\_Type >, 41
  - BArrayDense< Cell\_Type, Data\_Type >, 60
  - Flock, 114
  - FreqTable< T >, 118
  - Geese, 128
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 118

- 142
- Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
185
- print\_observed\_counts  
Geese, 128
- print\_stats  
Model< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
142
- printf\_barry  
barry-configuration.hpp, 255
- probability  
Node, 154
- Progress, 168  
~Progress, 168  
end, 169  
next, 169  
Progress, 168
- progress.hpp  
BARRY\_PROGRESS\_BAR\_WIDTH, 313
- pset\_loc  
Geese, 130
- push\_back  
counters-meat.hpp, 272  
PhyloCounterData, 158
- README.md, 340
- reduced\_sequence  
Geese, 131
- rengine  
Flock, 115
- report  
barry-meat.hpp, 214  
barrydense-meat.hpp, 242
- reserve  
BArray< Cell\_Type, Data\_Type >, 41  
BArrayDense< Cell\_Type, Data\_Type >, 61  
FreqTable< T >, 118  
PhyloCounterData, 158
- reset  
PowerSet< Array\_Type, Data\_Rule\_Type >, 165
- reset\_array  
StatsCounter< Array\_Type, Data\_Type >, 178  
Support< Array\_Type, Data\_Counter\_Type,  
Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >,  
185
- resize  
BArray< Cell\_Type, Data\_Type >, 42  
barry-meat.hpp, 209  
BArrayDense< Cell\_Type, Data\_Type >, 61  
barrydense-meat.hpp, 235, 236  
Entries< Cell\_Type >, 108  
statscounter-meat.hpp, 319
- return  
barry-meat.hpp, 209, 214  
barrydense-meat.hpp, 242  
counters-meat.hpp, 275  
statscounter-meat.hpp, 322
- support-meat.hpp, 335
- rhs  
barry-meat-operators.hpp, 197
- rm\_cell  
BArray< Cell\_Type, Data\_Type >, 42  
BArrayDense< Cell\_Type, Data\_Type >, 61  
barrydense-meat.hpp, 236, 237  
support-meat.hpp, 329
- ROW  
barry-meat-operators.hpp, 195  
barry-meat.hpp, 201, 209  
barrydense-meat-operators.hpp, 221  
barrydense-meat.hpp, 226
- row  
BArray< Cell\_Type, Data\_Type >, 42  
BArrayDense< Cell\_Type, Data\_Type >, 61
- row0  
barry-meat.hpp, 214
- Row\_type  
typedefs.hpp, 339
- rowsum  
BArrayDense< Cell\_Type, Data\_Type >, 61
- Rule  
Rule< Array\_Type, Data\_Type >, 170
- Rule< Array\_Type, Data\_Type >, 169  
~Rule, 170  
D, 171  
operator(), 171  
Rule, 170
- rule\_dyn\_limit\_changes  
Phylo rules, 26
- rule\_fun\_default  
rules-bones.hpp, 314
- Rule\_fun\_type  
typedefs.hpp, 339
- RULE\_FUNCTION  
barry.hpp, 260  
geese-bones.hpp, 305
- RULE\_LAMBDA  
barry.hpp, 260
- Rules  
Rules< Array\_Type, Data\_Type >, 172
- rules  
PowerSet< Array\_Type, Data\_Rule\_Type >, 167  
support-meat.hpp, 335
- Rules< Array\_Type, Data\_Type >, 171  
~Rules, 172  
add\_rule, 173  
clear, 173  
get\_seq, 173  
operator(), 174  
operator=, 174  
Rules, 172  
size, 174
- rules-bones.hpp  
rule\_fun\_default, 314
- rules\_  
support-meat.hpp, 335

- rules\_deleted
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 167
- rules\_dyn
  - support-meat.hpp, 335
- rules\_zerodiag
  - network.hpp, 290
- sample
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 142, 143
- search
  - barray-meat.hpp, 215
- sequence
  - Geese, 131
- set\_counters
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 143
  - StatsCounter< Array\_Type, Data\_Type >, 178
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 185
- set\_data
  - BArray< Cell\_Type, Data\_Type >, 42
  - BArrayDense< Cell\_Type, Data\_Type >, 62
- set\_keygen
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 143
- set\_engine
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 143
- set\_rules
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 143
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 185
- set\_rules\_dyn
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 144
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 186
- set\_seed
  - Flock, 114
  - Geese, 129
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 144
- shrink\_to\_fit
  - PhyloCounterData, 158
- simulate
  - Geese, 129
- size
  - BArrayDenseCol< Cell\_Type, Data\_Type >, 70
  - BArrayDenseCol\_const< Cell\_Type, Data\_Type >, 73
  - BArrayDenseRow< Cell\_Type, Data\_Type >, 75
  - BArrayDenseRow\_const< Cell\_Type, Data\_Type >, 77
  - BArrayVector< Cell\_Type, Data\_Type >, 86
  - BArrayVector\_const< Cell\_Type, Data\_Type >, 90
  - Counters< Array\_Type, Data\_Type >, 106
  - FreqTable< T >, 118
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 144
  - PhyloCounterData, 159
  - PowerSet< Array\_Type, Data\_Rule\_Type >, 166
  - Rules< Array\_Type, Data\_Type >, 174
- size\_unique
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 144
- source
  - barray-meat.hpp, 215
  - barraydense-meat.hpp, 242
  - Entries< Cell\_Type >, 108
- states
  - NodeData, 156
- Statistical Models, 11
- stats\_bank
  - support-meat.hpp, 335
- StatsCounter
  - StatsCounter< Array\_Type, Data\_Type >, 176
- StatsCounter< Array\_Type, Data\_Type >, 175
  - ~StatsCounter, 176
  - add\_counter, 177
  - count\_all, 177
  - count\_current, 177
  - count\_init, 177
  - get\_counters, 177
  - get\_descriptions, 178
  - get\_names, 178
  - reset\_array, 178
  - set\_counters, 178
  - StatsCounter, 176
- statscounter-meat.hpp
  - clear, 319
  - counter, 321
  - counter\_deleted, 321
  - counters, 321
  - counters\_, 321
  - current\_stats, 321
  - EmptyArray, 322
  - f\_, 322
  - for, 319
  - j, 322
  - resize, 319
  - return, 322
  - STATSCOUNTER\_TEMPLATE, 318–320
  - STATSCOUNTER\_TEMPLATE\_ARGS, 318

- STATSCOUNTER\_TYPE, 318
- STATSCOUNTER\_TEMPLATE
  - statscounter-meat.hpp, 318–320
- STATSCOUNTER\_TEMPLATE\_ARGS
  - statscounter-meat.hpp, 318
- STATSCOUNTER\_TYPE
  - statscounter-meat.hpp, 318
- store\_psets
  - Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 144
- subtree\_prob
  - Node, 154
- Support
  - Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 181
- Support< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 179
  - ~Support, 181
  - add\_counter, 182
  - add\_rule, 182
  - add\_rule\_dyn, 182
  - calc, 183
  - change\_stats, 186
  - cooriantes\_n\_free, 186
  - cooriantes\_n\_locked, 186
  - coordinates\_free, 186
  - coordinates\_locked, 187
  - current\_stats, 187
  - delete\_counters, 187
  - delete\_rules, 187
  - delete\_rules\_dyn, 187
  - eval\_rules\_dyn, 183
  - get\_counters, 183
  - get\_counts, 183
  - get\_current\_stats, 184
  - get\_data, 184
  - get\_rules, 184
  - get\_rules\_dyn, 184
  - hashes, 188
  - hashes\_initialized, 188
  - init\_support, 184
  - M, 188
  - max\_num\_elements, 188
  - N, 188
  - n\_counters, 189
  - print, 185
  - reset\_array, 185
  - set\_counters, 185
  - set\_rules, 185
  - set\_rules\_dyn, 186
  - Support, 181
- support-meat.hpp
  - array\_bank, 332
  - BARRY\_SUPPORT\_MEAT\_HPP, 326
  - calc\_backend\_dense, 327
  - calc\_backend\_sparse, 328
  - change\_stats\_different, 333
  - coord\_i, 333
  - coord\_j, 333
  - counters, 333
  - counters\_, 333
  - delete\_counters, 333
  - delete\_rules, 334
  - delete\_rules\_dyn, 334
  - else, 334
  - f\_, 334
  - for, 328
  - hashes, 334
  - if, 328
  - insert\_cell, 329
  - return, 335
  - rm\_cell, 329
  - rules, 335
  - rules\_, 335
  - rules\_dyn, 335
  - stats\_bank, 335
  - SUPPORT\_TEMPLATE, 327, 329–332
  - SUPPORT\_TEMPLATE\_ARGS, 327
  - SUPPORT\_TYPE, 327
  - tmp\_chng, 336
- support\_size
  - Flock, 114
  - Geese, 129
- Model< Array\_Type, Data\_Counter\_Type, Data\_Rule\_Type, Data\_Rule\_Dyn\_Type >, 145
- SUPPORT\_TEMPLATE
  - support-meat.hpp, 327, 329–332
- SUPPORT\_TEMPLATE\_ARGS
  - support-meat.hpp, 327
- SUPPORT\_TYPE
  - support-meat.hpp, 327
- swap\_cells
  - BArray< Cell\_Type, Data\_Type >, 43
  - BArrayDense< Cell\_Type, Data\_Type >, 62
- swap\_cols
  - BArray< Cell\_Type, Data\_Type >, 43
  - BArrayDense< Cell\_Type, Data\_Type >, 62
- swap\_rows
  - BArray< Cell\_Type, Data\_Type >, 43
  - BArrayDense< Cell\_Type, Data\_Type >, 62
- target
  - barray-meat.hpp, 215
  - barraydense-meat.hpp, 242
  - Entries< Cell\_Type >, 108
- this
  - barray-meat-operators.hpp, 197
- tmp\_chng
  - support-meat.hpp, 336
- to\_be\_deleted
  - counters-meat.hpp, 272
- toggle\_cell
  - BArray< Cell\_Type, Data\_Type >, 43
  - BArrayDense< Cell\_Type, Data\_Type >, 63

toggle\_lock  
     BArray< Cell\_Type, Data\_Type >, [43](#)  
     BArrayDense< Cell\_Type, Data\_Type >, [63](#)  
 transpose  
     BArray< Cell\_Type, Data\_Type >, [44](#)  
     BArrayDense< Cell\_Type, Data\_Type >, [63](#)  
 TWO  
     CHECK, [28](#)  
     EXISTS, [30](#)  
 typedefs.hpp  
     Col\_type, [338](#)  
     Counter\_fun\_type, [338](#)  
     Counts\_type, [338](#)  
     MapVec\_type, [339](#)  
     Row\_type, [339](#)  
     Rule\_fun\_type, [339](#)  
     uint, [339](#)  
     vec\_equal, [339](#)  
     vec\_equal\_approx, [340](#)  
     vec\_inner\_prod, [340](#)  
 ub  
     PhyloRuleDynData, [161](#)  
 uint  
     typedefs.hpp, [339](#)  
 UNKNOWN  
     EXISTS, [30](#)  
 update\_annotations  
     Geese, [129](#)  
 update\_normalizing\_constant  
     model-meat.hpp, [302](#)  
 v  
     barray-meat.hpp, [215](#)  
     barraydense-meat.hpp, [243](#)  
 va\_end  
     barraydense-meat.hpp, [237](#)  
 va\_start  
     barraydense-meat.hpp, [237](#)  
 val  
     Entries< Cell\_Type >, [108](#)  
 val0  
     barraydense-meat.hpp, [243](#)  
 val1  
     barraydense-meat.hpp, [243](#)  
 value  
     barray-meat.hpp, [215](#)  
     barraydense-meat.hpp, [243](#)  
     Cell< Cell\_Type >, [94](#)  
 vec\_diff  
     geese-bones.hpp, [305](#)  
 vec\_equal  
     typedefs.hpp, [339](#)  
 vec\_equal\_approx  
     typedefs.hpp, [340](#)  
 vec\_inner\_prod  
     typedefs.hpp, [340](#)  
 vecHasher< T >, [189](#)  
     operator(), [189](#)  
 vector\_caster  
     geese-bones.hpp, [305](#)  
 vertex\_attr  
     NetworkData, [148](#)  
 visited  
     BArray< Cell\_Type, Data\_Type >, [45](#)  
     BArrayDense< Cell\_Type, Data\_Type >, [65](#)  
     Cell< Cell\_Type >, [94](#)  
     Node, [154](#)  
 vprintf  
     barraydense-meat.hpp, [237](#)  
 ZERO\_CELL  
     barraydense-meat.hpp, [227](#)  
     barraydensecol-bones.hpp, [247](#)  
     barraydenserow-bones.hpp, [248](#)  
 zero\_col  
     BArray< Cell\_Type, Data\_Type >, [44](#)  
     BArrayDense< Cell\_Type, Data\_Type >, [63](#)  
 zero\_row  
     BArray< Cell\_Type, Data\_Type >, [44](#)  
     BArrayDense< Cell\_Type, Data\_Type >, [63](#)