



eXpress

0.9.5 BETA

<http://bio.math.berkeley.edu/eXpress>

Generated by Doxygen 1.7.3

Sun Apr 8 2012 11:48:58

Contents

1	Class Index	1
1.1	Class Hierarchy	1
2	Class Index	3
2.1	Class List	3
3	Class Documentation	5
3.1	BAMParser Class Reference	5
3.1.1	Detailed Description	5
3.1.2	Constructor & Destructor Documentation	6
3.1.2.1	BAMParser	6
3.1.2.2	~BAMParser	6
3.1.3	Member Function Documentation	6
3.1.3.1	header	6
3.1.3.2	next_fragment	6
3.1.3.3	reset	7
3.1.3.4	targ_index	7
3.1.3.5	targ_lengths	7
3.2	BAMWriter Class Reference	7
3.2.1	Detailed Description	8
3.2.2	Constructor & Destructor Documentation	8
3.2.2.1	BAMWriter	8
3.2.2.2	~BAMWriter	8
3.2.3	Member Function Documentation	8
3.2.3.1	write_fragment	8
3.3	BiasBoss Class Reference	9
3.3.1	Detailed Description	9
3.3.2	Constructor & Destructor Documentation	10
3.3.2.1	BiasBoss	10
3.3.3	Member Function Documentation	10
3.3.3.1	append_output	10
3.3.3.2	get_target_bias	10
3.3.3.3	normalize_expectations	10
3.3.3.4	to_string	11
3.3.3.5	update_expectations	11
3.3.3.6	update_observed	11
3.4	Bundle Class Reference	11
3.4.1	Detailed Description	12
3.4.2	Constructor & Destructor Documentation	12

3.4.2.1	Bundle	12
3.4.3	Member Function Documentation	12
3.4.3.1	counts	12
3.4.3.2	incr_counts	13
3.4.3.3	size	13
3.4.3.4	targets	13
3.5	BundleTable Class Reference	13
3.5.1	Detailed Description	14
3.5.2	Constructor & Destructor Documentation	14
3.5.2.1	BundleTable	14
3.5.2.2	~BundleTable	14
3.5.3	Member Function Documentation	14
3.5.3.1	bundles	14
3.5.3.2	create_bundle	14
3.5.3.3	merge	15
3.5.3.4	size	15
3.6	CovarTable Class Reference	15
3.6.1	Detailed Description	16
3.6.2	Member Function Documentation	16
3.6.2.1	get	16
3.6.2.2	increment	16
3.6.2.3	size	16
3.7	FLD Class Reference	17
3.7.1	Detailed Description	17
3.7.2	Constructor & Destructor Documentation	17
3.7.2.1	FLD	17
3.7.3	Member Function Documentation	18
3.7.3.1	add_val	18
3.7.3.2	append_output	18
3.7.3.3	cdf	18
3.7.3.4	max_val	18
3.7.3.5	mean	18
3.7.3.6	min_val	19
3.7.3.7	pdf	19
3.7.3.8	to_string	19
3.7.3.9	tot_mass	19
3.8	FragHit Struct Reference	20
3.8.1	Detailed Description	20
3.8.2	Member Function Documentation	21
3.8.2.1	length	21
3.8.2.2	pair_status	21
3.8.3	Member Data Documentation	21
3.8.3.1	deletes_l	21
3.8.3.2	deletes_r	21
3.8.3.3	inserts_l	22
3.8.3.4	inserts_r	22
3.8.3.5	left	22
3.8.3.6	left_first	22
3.8.3.7	mapped_targ	22
3.8.3.8	mate_l	22

3.8.3.9	name	22
3.8.3.10	right	23
3.8.3.11	seq_l	23
3.8.3.12	seq_r	23
3.8.3.13	targ_id	23
3.9	Fragment Class Reference	23
3.9.1	Detailed Description	23
3.9.2	Constructor & Destructor Documentation	24
3.9.2.1	~Fragment	24
3.9.3	Member Function Documentation	24
3.9.3.1	add_map_end	24
3.9.3.2	hits	24
3.9.3.3	name	24
3.9.3.4	num_hits	25
3.9.3.5	sample_hit	25
3.10	FrequencyMatrix Class Reference	25
3.10.1	Detailed Description	25
3.10.2	Constructor & Destructor Documentation	26
3.10.2.1	FrequencyMatrix	26
3.10.2.2	FrequencyMatrix	26
3.10.3	Member Function Documentation	26
3.10.3.1	arr	26
3.10.3.2	increment	27
3.10.3.3	increment	27
3.10.3.4	operator()	27
3.10.3.5	operator()	27
3.10.3.6	row	28
3.10.3.7	set_logged	28
3.11	Globals Struct Reference	28
3.11.1	Detailed Description	29
3.12	Indel Struct Reference	29
3.12.1	Detailed Description	29
3.12.2	Constructor & Destructor Documentation	29
3.12.2.1	Indel	29
3.12.3	Member Data Documentation	30
3.12.3.1	len	30
3.12.3.2	pos	30
3.13	MarkovModel Class Reference	30
3.13.1	Detailed Description	30
3.14	MismatchTable Class Reference	30
3.14.1	Detailed Description	31
3.14.2	Constructor & Destructor Documentation	31
3.14.2.1	MismatchTable	31
3.14.3	Member Function Documentation	31
3.14.3.1	activate	31
3.14.3.2	append_output	32
3.14.3.3	log_likelihood	32
3.14.3.4	to_string	32
3.14.3.5	update	32
3.15	Parser Class Reference	33

3.15.1	Detailed Description	33
3.15.2	Member Function Documentation	34
3.15.2.1	header	34
3.15.2.2	next_fragment	34
3.15.2.3	reset	34
3.15.2.4	targ_index	34
3.15.2.5	targ_lengths	34
3.16	ParseThreadSafety Struct Reference	35
3.16.1	Detailed Description	35
3.16.2	Member Data Documentation	35
3.16.2.1	cond	35
3.16.2.2	frag_clean	35
3.16.2.3	mut	36
3.16.2.4	next_frag	36
3.17	PosWeightTable Class Reference	36
3.17.1	Detailed Description	36
3.17.2	Constructor & Destructor Documentation	37
3.17.2.1	PosWeightTable	37
3.17.3	Member Function Documentation	37
3.17.3.1	append_output	37
3.17.3.2	get_weight	37
3.17.3.3	get_weight	37
3.17.3.4	increment_expected	38
3.17.3.5	increment_expected	38
3.17.3.6	increment_observed	38
3.17.3.7	increment_observed	39
3.17.3.8	normalize_expected	39
3.18	RoundParams Struct Reference	39
3.18.1	Detailed Description	39
3.18.2	Constructor & Destructor Documentation	40
3.18.2.1	RoundParams	40
3.18.3	Member Data Documentation	40
3.18.3.1	mass	40
3.18.3.2	mass_var	40
3.18.3.3	tot_ambig_mass	40
3.18.3.4	var_sum	40
3.19	SAMParser Class Reference	41
3.19.1	Detailed Description	41
3.19.2	Constructor & Destructor Documentation	41
3.19.2.1	SAMParser	41
3.19.3	Member Function Documentation	41
3.19.3.1	header	41
3.19.3.2	next_fragment	42
3.19.3.3	targ_index	42
3.19.3.4	targ_lengths	42
3.20	SAMWriter Class Reference	43
3.20.1	Detailed Description	43
3.20.2	Constructor & Destructor Documentation	43
3.20.2.1	SAMWriter	43
3.20.2.2	~SAMWriter	44

3.20.3	Member Function Documentation	44
3.20.3.1	write_fragment	44
3.21	Sequence Class Reference	44
3.21.1	Detailed Description	44
3.21.2	Constructor & Destructor Documentation	45
3.21.2.1	Sequence	45
3.21.2.2	Sequence	45
3.21.2.3	Sequence	45
3.21.2.4	~Sequence	45
3.21.3	Member Function Documentation	46
3.21.3.1	empty	46
3.21.3.2	length	46
3.21.3.3	operator=	46
3.21.3.4	operator[]	46
3.21.3.5	set	47
3.22	SeqWeightTable Class Reference	47
3.22.1	Detailed Description	47
3.22.2	Constructor & Destructor Documentation	48
3.22.2.1	SeqWeightTable	48
3.22.3	Member Function Documentation	48
3.22.3.1	append_output	48
3.22.3.2	copy_expected	48
3.22.3.3	copy_observed	48
3.22.3.4	get_weight	49
3.22.3.5	increment_expected	49
3.22.3.6	increment_observed	49
3.22.3.7	normalize_expected	49
3.22.3.8	to_string	50
3.23	Target Class Reference	50
3.23.1	Detailed Description	51
3.23.2	Constructor & Destructor Documentation	51
3.23.2.1	Target	51
3.23.2.2	~Target	51
3.23.3	Member Function Documentation	51
3.23.3.1	add_mass	51
3.23.3.2	bundle	52
3.23.3.3	bundle	52
3.23.3.4	cached_effective_length	52
3.23.3.5	est_effective_length	52
3.23.3.6	id	53
3.23.3.7	incr_counts	53
3.23.3.8	length	53
3.23.3.9	lock	53
3.23.3.10	log_likelihood	54
3.23.3.11	mass	54
3.23.3.12	mass_var	54
3.23.3.13	name	54
3.23.3.14	rho	55
3.23.3.15	round_reset	55
3.23.3.16	seq	55

3.23.3.17	solveable	55
3.23.3.18	solveable	55
3.23.3.19	tot_ambig_mass	56
3.23.3.20	tot_counts	56
3.23.3.21	uniq_counts	56
3.23.3.22	unlock	56
3.23.3.23	update_target_bias	56
3.23.3.24	var_sum	57
3.24	TargetTable Class Reference	57
3.24.1	Detailed Description	57
3.24.2	Constructor & Destructor Documentation	58
3.24.2.1	TargetTable	58
3.24.2.2	~TargetTable	58
3.24.3	Member Function Documentation	58
3.24.3.1	covar_size	58
3.24.3.2	get_covar	59
3.24.3.3	get_targ	59
3.24.3.4	merge_bundles	59
3.24.3.5	num_bundles	59
3.24.3.6	output_results	60
3.24.3.7	round_reset	60
3.24.3.8	size	60
3.24.3.9	threaded_bias_update	60
3.24.3.10	total_fpb	60
3.24.3.11	update_covar	61
3.24.3.12	update_total_fpb	61
3.25	ThreadedMapParser Class Reference	61
3.25.1	Detailed Description	62
3.25.2	Constructor & Destructor Documentation	62
3.25.2.1	ThreadedMapParser	62
3.25.2.2	~ThreadedMapParser	62
3.25.3	Member Function Documentation	63
3.25.3.1	reset_reader	63
3.25.3.2	targ_index	63
3.25.3.3	targ_lengths	63
3.25.3.4	threaded_parse	63
3.25.3.5	write_active	63
3.26	Writer Class Reference	64
3.26.1	Detailed Description	64
3.26.2	Member Function Documentation	64
3.26.2.1	write_fragment	64

Chapter 1

Class Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

BiasBoss	9
Bundle	11
BundleTable	13
CovarTable	15
FLD	17
FragHit	20
Fragment	23
FrequencyMatrix	25
Globals	28
Indel	29
MarkovModel	30
MismatchTable	30
Parser	33
BAMParser	5
SAMPParser	41
ParseThreadSafety	35
PosWeightTable	36
RoundParams	39
Sequence	44
SeqWeightTable	47
Target	50
TargetTable	57
ThreadedMapParser	61
Writer	64
BAMWriter	7
SAMWriter	43

Chapter 2

Class Index

2.1 Class List

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

BAMParser	5
BAMWriter	7
BiasBoss	9
Bundle	11
BundleTable	13
CovarTable	15
FLD	17
FragHit	20
Fragment	23
FrequencyMatrix	25
Globals	28
Indel	29
MarkovModel	30
MismatchTable	30
Parser	33
ParseThreadSafety	35
PosWeightTable	36
RoundParams	39
SAMParser	41
SAMWriter	43
Sequence	44
SeqWeightTable	47
Target	50
TargetTable	57
ThreadedMapParser	61
Writer	64

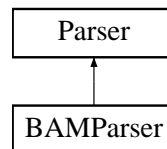
Chapter 3

Class Documentation

3.1 BAMParser Class Reference

```
#include <mapparser.h>
```

Inheritance diagram for BAMParser:



Public Member Functions

- [BAMParser](#) (BamTools::BamReader *reader)
- [~BAMParser](#) ()
- const std::string [header](#) () const
- const TransIndex & [targ_index](#) () const
- const TransIndex & [targ_lengths](#) () const
- bool [next_fragment](#) ([Fragment](#) &f)
- void [reset](#) ()

3.1.1 Detailed Description

The [BAMParser](#) class fills [Fragment](#) objects by parsing an input file in BAM format.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 102 of file mapparser.h.

3.1.2 Constructor & Destructor Documentation**3.1.2.1 BAMParser::BAMParser (BamTools::BamReader * *reader*)**

[BAMParser](#) constructor sets the reader

Definition at line 230 of file mapparser.cpp.

3.1.2.2 BAMParser::~BAMParser () [inline]

[BAMParser](#) destructor deletes the reader

Definition at line 140 of file mapparser.h.

3.1.3 Member Function Documentation**3.1.3.1 const std::string BAMParser::header () const [inline, virtual]**

a member function that returns a string version of the header

Returns

string version of the header

Implements [Parser](#).

Definition at line 146 of file mapparser.h.

3.1.3.2 bool BAMParser::next_fragment (Fragment & *f*) [virtual]

a member function that loads all mappings of the next fragment

Parameters

<i>f</i>	the empty Fragment to add mappings to
----------	---

Returns

true if more reads remain in the BAM file, false otherwise

Implements [Parser](#).

Definition at line 253 of file mapparser.cpp.

3.1.3.3 void BAMParser::reset () [virtual]

a member function that resets the parser and rewinds to the beginning of the BAM file

Implements [Parser](#).

Definition at line 310 of file mapparser.cpp.

3.1.3.4 const TransIndex& BAMParser::targ_index () const [inline, virtual]

a member function that returns the target-to-index map

Returns

the target-to-index map

Implements [Parser](#).

Definition at line 152 of file mapparser.h.

3.1.3.5 const TransIndex& BAMParser::targ_lengths () const [inline, virtual]

a member function that returns the target-to-length map

Returns

the target-to-length map

Implements [Parser](#).

Definition at line 158 of file mapparser.h.

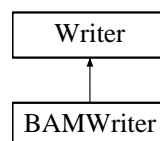
The documentation for this class was generated from the following files:

- src/mapparser.h
- src/mapparser.cpp

3.2 BAMWriter Class Reference

```
#include <mapparser.h>
```

Inheritance diagram for BAMWriter:



Public Member Functions

- [BAMWriter](#) (BamTools::BamWriter *writer, bool sample)
- [~BAMWriter](#) ()
- void [write_fragment](#) ([Fragment](#) &f)

3.2.1 Detailed Description

The [BAMWriter](#) class writes [Fragment](#) objects back to file (in BAM format) with per-mapping probabilistic assignments.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 179 of file mapparser.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 [BAMWriter::BAMWriter](#) ([BamTools::BamWriter](#) * *writer*, bool *sample*)

[BAMWriter](#) constructor stores a pointer to the BAM file

Parameters

<i>writer</i>	pointer to the BAM file object
<i>sample</i>	specifies if a single alignment should be sampled (true) or all output with their respective probabilities (false)

Definition at line 320 of file mapparser.cpp.

3.2.2.2 [BAMWriter::~~BAMWriter](#) ()

[BAMWriter](#) destructor flushes and deletes the Bamtools::BamWriter

Definition at line 324 of file mapparser.cpp.

3.2.3 Member Function Documentation

3.2.3.1 void [BAMWriter::write_fragment](#) ([Fragment](#) & *f*) [virtual]

a member function that writes all mappings of the fragment to the output file in BAM format along with their probabilities in the "XP" field

Parameters

<i>f</i>	the processed Fragment to output
----------	--

Implements [Writer](#).

Definition at line 330 of file mapparser.cpp.

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

- src/mapparser.h
- src/mapparser.cpp

3.3 BiasBoss Class Reference

```
#include <biascorrection.h>
```

Public Member Functions

- [BiasBoss](#) (double alpha)
- void **copy_observations** (const [BiasBoss](#) &other)
- void **copy_expectations** (const [BiasBoss](#) &other)
- void **update_expectations** (const [Target](#) &targ, double mass=0, const std::vector< double > &fl_cdf=std::vector< double >())
- void **normalize_expectations** ()
- void **update_observed** (const [FragHit](#) &hit, double mass)
- double **get_target_bias** (std::vector< float > &start_bias, std::vector< float > &end_bias, const [Target](#) &targ) const
- std::string **to_string** () const
- void **append_output** (std::ofstream &outfile) const

3.3.1 Detailed Description

The [BiasBoss](#) class keeps track of sequence-specific and positional bias. It allows for the bias associated with a given fragment end to be calculated, and for the bias parameters to be updated based on additional observations. All stored and returned values are in log space.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 224 of file biascorrection.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 BiasBoss::BiasBoss (double *alpha*)

a private [SeqWeightTable](#) that stores the 5' sequence-specific bias parameters (logged)
 a private [SeqWeightTable](#) that stores the 3' sequence-specific bias parameters (logged)
[BiasBoss](#) Constructor

Parameters

<i>alpha</i>	a double specifying the strength of the uniform prior (logged pseudo-counts for each parameter)
--------------	---

Definition at line 241 of file biascorrection.cpp.

3.3.3 Member Function Documentation

3.3.3.1 void BiasBoss::append_output (std::ofstream & *outfile*) const

a member function that outputs the positional and sequence-specific bias parameter matrices

Parameters

<i>outfile</i>	the file to append to
----------------	-----------------------

3.3.3.2 double BiasBoss::get_target_bias (std::vector< float > & *start_bias*, std::vector< float > & *end_bias*, const Target & *targ*) const

a member function that returns the 5' and 3' bias values at each position in a given target based on the current bias parameters

Parameters

<i>start_bias</i>	a vector containing the logged bias for each 5' start site in the target
<i>end_bias</i>	a vector containing the logged bias for each 3' end site in the target
<i>targ</i>	the target for which to calculate the logged bias

Returns

the product of the average 5' and 3' bias (logged)

Definition at line 294 of file biascorrection.cpp.

3.3.3.3 void BiasBoss::normalize_expectations ()

a member function that normalizes the expected counts and converts them to the log scale

Definition at line 270 of file biascorrection.cpp.

3.3.3.4 string BiasBoss::to_string () const

a member function that returns a string containing the observed positional nucleotide probabilities (non-logged) in column-major order (A,C,G,T)

Returns

the string representation of the observed probabilities

Definition at line 315 of file biascorrection.cpp.

3.3.3.5 void BiasBoss::update_expectations (const Target & targ, double mass = 0, const std::vector< double > & fl_cdf = std::vector<double>())

a member function that updates the expectation parameters (sequence-specific and positional) assuming uniform expression of and accross the target's sequence

Parameters

<i>targ</i>	the target to measure expected counts from
-------------	--

Definition at line 258 of file biascorrection.cpp.

3.3.3.6 void BiasBoss::update_observed (const FragHit & hit, double mass)

a member function that updates the observed parameters (sequence-specific and positional) given a fragment mapping to a target and its logged probabilistic assignment

Parameters

<i>hit</i>	the fragment hit (alignment)
<i>mass</i>	the logged probability of the mapping, which is the amount to update the observed counts by

Definition at line 276 of file biascorrection.cpp.

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

- src/biascorrection.h
- src/biascorrection.cpp

3.4 Bundle Class Reference

```
#include <bundles.h>
```

Public Member Functions

- [Bundle](#) (Target *targ)

- void [incr_counts](#) (size_t incr_amt=1)
- size_t [size](#) () const
- std::vector< [Target](#) * > & [targets](#) ()
- size_t [counts](#) () const

3.4.1 Detailed Description

The [Bundle](#) class keeps track of a group of targets that have shared ambiguous (multi-mapped) reads. Besides storing the target, it keeps track of the number of observed fragments, the total fragment mass, and the next fragment mass (which it also updates).

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 71 of file bundles.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 [Bundle::Bundle](#) ([Target](#) * *targ*)

[Bundle](#) Constructor.

Parameters

<i>targ</i>	a pointer to the initial Target object in the bundle
<i>fmt</i>	a pointer to the (global) FragMassTable

Definition at line 42 of file bundles.cpp.

3.4.3 Member Function Documentation

3.4.3.1 [size_t Bundle::counts](#) () const `[inline]`

a member function that returns the total number of observed fragments mapped to targets in the bundle

Returns

the total number of fragments mapped to targets in the bundle

Definition at line 115 of file bundles.h.

3.4.3.2 void Bundle::incr_counts (size_t *incr_amt* = 1)

a member function that increases the total bundle observed fragment counts by a given amount

Parameters

<i>incr_amt</i>	the amount to increase the counts by
-----------------	--------------------------------------

Definition at line 46 of file bundles.cpp.

3.4.3.3 size_t Bundle::size () const [inline]

a member function that returns the number of targets in the bundle

Returns

the number of targets in the bundle

Definition at line 103 of file bundles.h.

3.4.3.4 std::vector<Target*> & Bundle::targets () [inline]

a member function that returns a reference to the vector of pointers to targets in the bundle

Returns

reference to the vector pointing to bundle targets

Definition at line 109 of file bundles.h.

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

- src/bundles.h
- src/bundles.cpp

3.5 BundleTable Class Reference

```
#include <bundles.h>
```

Public Member Functions

- [BundleTable \(\)](#)
- [~BundleTable \(\)](#)
- const BundleSet & [bundles](#) () const
- size_t [size](#) () const
- [Bundle * create_bundle](#) (Target *targ)
- [Bundle * merge](#) (Bundle *b1, Bundle *b2)

3.5.1 Detailed Description

The [BundleTable](#) class keeps track of the [Bundle](#) objects for a given run. It has the ability to create, delete, and merge bundles. It also keeps track of the target covariances, since these are related to bundles in that all covariances outside of a bundle are nonzero.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 129 of file bundles.h.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 [BundleTable::BundleTable](#) () [inline]

[BundleTable](#) constructor.

Definition at line 141 of file bundles.h.

3.5.2.2 [BundleTable::~~BundleTable](#) ()

[BundleTable](#) deconstructor. Deletes all [Bundle](#) objects.

Definition at line 51 of file bundles.cpp.

3.5.3 Member Function Documentation

3.5.3.1 `const BundleSet& BundleTable::bundles () const` [inline]

a member function that returns the set of current [Bundle](#) objects

Returns

a reference to the unordered_set containing all current [Bundle](#) objects

Definition at line 152 of file bundles.h.

3.5.3.2 `Bundle * BundleTable::create_bundle (Target * targ)`

a member function that creates a new bundle, initially with only the single given [Target](#)

Parameters

<i>targ</i>	a pointer to the only Target initially contained in the Bundle
-------------	--

Returns

a pointer to the new [Bundle](#) object

Definition at line 59 of file bundles.cpp.

3.5.3.3 Bundle * BundleTable::merge (Bundle * *b1*, Bundle * *b2*)

a member function that merges two [Bundle](#) objects into one the Targets are all move to the larger bundles and the other is deleted

Parameters

<i>b1</i>	a pointer to one of the Bundle objects to merge
<i>b2</i>	a pointer to the other Bundle object to merge

Returns

a pointer to the merged [Bundle](#) object

Definition at line 66 of file bundles.cpp.

3.5.3.4 size_t BundleTable::size () const [inline]

a member function that returns the size of the set of current [Bundle](#) objects, which is the current number of bundles

Returns

the current number of bundles

Definition at line 159 of file bundles.h.

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

- src/bundles.h
- src/bundles.cpp

3.6 CovarTable Class Reference

```
#include <bundles.h>
```

Public Member Functions

- void [increment](#) (TransID targ1, TransID targ2, double covar)
- double [get](#) (TransID targ1, TransID targ2)
- size_t [size](#) () const

3.6.1 Detailed Description

The [CovarTable](#) is a sparse matrix for storing and updating pairwise covariances between targets.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 26 of file bundles.h.

3.6.2 Member Function Documentation

3.6.2.1 `double CovarTable::get (TransID targ1, TransID targ2)`

a member function that returns the covariance between two targets these returned value will be the the negative of the true value (logged)

Parameters

<i>targ1</i>	one of the targets in the pair
<i>targ2</i>	the other target in the pair

Returns

the negative of the pair's covariance (logged)

Definition at line 29 of file bundles.cpp.

3.6.2.2 `void CovarTable::increment (TransID targ1, TransID targ2, double covar)`

a member function that increases the covariance between two targets by the specified amount (logged) these values are stored positive even though they are negative

Parameters

<i>targ1</i>	one of the targets in the pair
<i>targ2</i>	the other target in the pair
<i>covar</i>	a double specifying the amount to increase the pair's covariance by (logged)

Definition at line 15 of file bundles.cpp.

3.6.2.3 `size_t CovarTable::size () const` `[inline]`

a member function that returns the number of pairs of targets with non-zero covariance

Returns

the number of target pairs with non-zero covariance

Definition at line 60 of file bundles.h.

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

- src/bundles.h
- src/bundles.cpp

3.7 FLD Class Reference

```
#include <fld.h>
```

Public Member Functions

- [FLD](#) (double *alpha*, size_t *max_val*, size_t *mean*, size_t *std_dev*)
- size_t [max_val](#) () const
- size_t [min_val](#) () const
- double [mean](#) () const
- void [add_val](#) (size_t *len*, double *mass*)
- double [pdf](#) (size_t *len*) const
- std::vector< double > [cdf](#) () const
- double [tot_mass](#) () const
- std::string [to_string](#) () const
- void [append_output](#) (std::ofstream &*outfile*) const

3.7.1 Detailed Description

The [FLD](#) class keeps track of the observed fragment length distribution. It starts with a Gaussian prior with parameters specified by the arguments. A small "Gaussian" kernel is added for each observation. All mass values and probabilities are stored and returned in log space (except in [to_string](#)).

Definition at line 21 of file fld.h.

3.7.2 Constructor & Destructor Documentation**3.7.2.1 FLD::FLD (double *alpha*, size_t *max_val*, size_t *mean*, size_t *std_dev*)**

[FLD](#) Constructor

Parameters

<i>alpha</i>	double that sets the average pseudo-counts (logged)
<i>max_val</i>	an integer that sets the maximum allowable FragHit length
<i>mean</i>	a size_t for the mean of the prior gaussian dist
<i>std_dev</i>	a size_t for the std dev of the prior gaussian dist

Generated on Sun Apr 8 2012 11:48:58 for eXpress by Doxygen

Definition at line 21 of file fld.cpp.

3.7.3 Member Function Documentation

3.7.3.1 void FLD::add_val (size_t *len*, double *mass*)

a member function that updates the distribution based on a new [FragHit](#) observation

Parameters

<i>len</i>	an integer for the observed FragHit length
<i>mass</i>	a double for the mass (logged) of the observed FragHit

Definition at line 53 of file fld.cpp.

3.7.3.2 void FLD::append_output (std::ofstream & *outfile*) const

a member function that appends the [FLD](#) parameters to the end of the given file

Parameters

<i>outfile</i>	the file to append to
----------------	-----------------------

3.7.3.3 vector< double > FLD::cdf () const

a member function that returns a vector containing the cdf

Returns

cdf

Definition at line 79 of file fld.cpp.

3.7.3.4 size_t FLD::max_val () const

a member function that returns the maximum allowed [FragHit](#) length

Returns

max allowed [FragHit](#) length

Definition at line 41 of file fld.cpp.

3.7.3.5 double FLD::mean () const

a member function that returns the mean [FragHit](#) length

Returns

mean observed [FragHit](#) length

Definition at line 98 of file fld.cpp.

3.7.3.6 `size_t FLD::min_val () const`

a member function that returns the minimum observed [FragHit](#) length (1 initially)

Returns

minimum observed [FragHit](#) length

Definition at line 46 of file fld.cpp.

3.7.3.7 `double FLD::pdf (size_t len) const`

a member function that returns the (logged) probability of a given [FragHit](#) length

Parameters

<i>len</i>	an integer for the FragHit length to return the probability of
------------	--

Returns

(logged) probability of observing the given [FragHit](#) length

Definition at line 73 of file fld.cpp.

3.7.3.8 `string FLD::to_string () const`

a member function that returns a string containing the current distribution

Returns

space-separated string of probabilities ordered from length 0 to max_val (non-logged)

Definition at line 103 of file fld.cpp.

3.7.3.9 `double FLD::tot_mass () const`

a member function that returns the (logged) number of observed [FragHits](#) (including pseudo-counts)

Returns

number of observed fragments

Definition at line 93 of file fld.cpp.

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

- src/fld.h
- src/fld.cpp

3.8 FragHit Struct Reference

```
#include <fragments.h>
```

Public Member Functions

- `size_t` [length](#) () const
- `PairStatus` [pair_status](#) () const

Public Attributes

- `std::string` [name](#)
- `TransID` [targ_id](#)
- `Target *` [mapped_targ](#)
- `Sequence` [seq_l](#)
- `Sequence` [seq_r](#)
- `size_t` [left](#)
- `size_t` [right](#)
- `int` [mate_l](#)
- `bool` [left_first](#)
- `std::vector< Indel >` [inserts_l](#)
- `std::vector< Indel >` [deletes_l](#)
- `std::vector< Indel >` [inserts_r](#)
- `std::vector< Indel >` [deletes_r](#)
- `double` [probability](#)
- `BamTools::BamAlignment` [bam_l](#)
- `BamTools::BamAlignment` [bam_r](#)
- `std::string` [sam_l](#)
- `std::string` [sam_r](#)

3.8.1 Detailed Description

The [FragHit](#) struct stores the information for a single (multi-)mapping of a fragment.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 62 of file fragments.h.

3.8.2 Member Function Documentation**3.8.2.1** `size_t FragHit::length () const` `[inline]`

a member function returning the length of the fragment according to this mapping note, that this result will be invalid if the fragment is single-end

Returns

int length of fragment mapping

Definition at line 140 of file fragments.h.

3.8.2.2 `PairStatus FragHit::pair_status () const` `[inline]`

a member function returning whether the mapping is PAIRED, LEFT_ONLY, or RIGHT_ONLY LEFT_ONLY denotes that the single read is not reverse complemented => its left end is the left fragment end RIGHT_ONLY denotes that the single read is reverse complemented => its right end is the right fragment end

Returns

PairStatus the pair status of the mapping

Definition at line 152 of file fragments.h.

3.8.3 Member Data Documentation**3.8.3.1** `std::vector<Indel> FragHit::deletes_l`

a public vector of [Indel](#) objects storing all deletions from the reference in the left read (in order from left to right)

Definition at line 123 of file fragments.h.

3.8.3.2 `std::vector<Indel> FragHit::deletes_r`

a public vector of [Indel](#) objects storing all deletions from the reference in the left read (in order from right to left)

Definition at line 133 of file fragments.h.

3.8.3.3 `std::vector<Indel> FragHit::inserts_l`

a public vector of [Indel](#) objects storing all insertions to the reference in the left read (in order from left to right)

Definition at line 118 of file fragments.h.

3.8.3.4 `std::vector<Indel> FragHit::inserts_r`

a public vector of [Indel](#) objects storing all insertions to the reference in the right read (in order from right to left)

Definition at line 128 of file fragments.h.

3.8.3.5 `size_t FragHit::left`

a public `size_t` containing the 0-based leftmost coordinate mapped to in the target valid only if `PairStatus` is `PAIRED` or `LEFT_ONLY`

Definition at line 93 of file fragments.h.

3.8.3.6 `bool FragHit::left_first`

a public `bool` specifying that the "right" (second according to SAM flag) is reverse complemented when true and the "left" (first according to SAM flag) is reverse complemented when false in other words, the "left" read is truly left of the "right" read in target coordinate space when true

Definition at line 113 of file fragments.h.

3.8.3.7 `Target* FragHit::mapped_targ`

a public pointer to the target mapped to

Definition at line 77 of file fragments.h.

3.8.3.8 `int FragHit::mate_l`

a public `int` containing the left position for the mate of the first read read in from the SAM file 0 if single-end fragment this is temporarily used to help find the mate, but is not important later on

Definition at line 106 of file fragments.h.

3.8.3.9 `std::string FragHit::name`

a public string for the SAM "Query Template Name" (fragment name)

Definition at line 67 of file fragments.h.

3.8.3.10 `size_t FragHit::right`

a public `size_t` containing the position following the 0-based rightmost coordinate mapped to in the target valid only if `PairStatus` is `PAIRED` or `RIGHT_ONLY`

Definition at line 99 of file `fragments.h`.

3.8.3.11 `Sequence FragHit::seq_l`

a public string containing the "left" read sequence (first according to SAM flag)

Definition at line 82 of file `fragments.h`.

3.8.3.12 `Sequence FragHit::seq_r`

a public string containing the "right" read sequence (second according to SAM flag)

Definition at line 87 of file `fragments.h`.

3.8.3.13 `TransID FragHit::targ_id`

a public `TransID` for the target mapped to

Definition at line 72 of file `fragments.h`.

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

- `src/fragments.h`

3.9 Fragment Class Reference

```
#include <fragments.h>
```

Public Member Functions

- [~Fragment](#) ()
- [bool add_map_end](#) ([FragHit](#) *f)
- [const std::string & name](#) () const
- [const size_t num_hits](#) () const
- [const std::vector< FragHit * > & hits](#) () const
- [const FragHit * sample_hit](#) () const

3.9.1 Detailed Description

The [Fragment](#) class stores information for all multi-mappings of a single fragment. By design, only paired-end mappings of paired-end reads will be accepted. All mappings of single-end reads will be accepted.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 177 of file fragments.h.

3.9.2 Constructor & Destructor Documentation**3.9.2.1 `Fragment::~~Fragment ()`**[Fragment](#) destructor deletes all [FragHit](#) objects pointed to by the [Fragment](#)

Definition at line 17 of file fragments.cpp.

3.9.3 Member Function Documentation**3.9.3.1 `bool Fragment::add_map_end (FragHit * f)`**

a member function that adds a new [FragHit](#) (single read at this point) to the [Fragment](#) if it is the first [FragHit](#), it sets the [Fragment](#) name and is added to `_open_mates`, if the fragment is not paired, it is added to `_frag_hits`, otherwise, `add_open_mate` is called

Parameters

<i>f</i>	the FragHit to be added
----------	---

Definition at line 30 of file fragments.cpp.

3.9.3.2 `const std::vector<FragHit*>& Fragment::hits () const` `[inline]`a member function that returns [FragHit](#) multi-mappings of the fragment**Returns**a vector containing pointers to the [FragHit](#) multi-mappings

Definition at line 234 of file fragments.h.

3.9.3.3 `const std::string& Fragment::name () const` `[inline]`

a member function that returns the SAM "Query Template Name" (fragment name)

Returns

the string SAM "Query Template Name" (fragment name)

Definition at line 222 of file fragments.h.

3.9.3.4 `const size_t Fragment::num_hits () const` `[inline]`

a member function that returns the number of multi-mappings for the fragment

Returns

number of multi-mappings for fragment

Definition at line 228 of file fragments.h.

3.9.3.5 `const FragHit * Fragment::sample_hit () const`

a member function that returns a single [FragHit](#) of the fragment sampled at random based on the probabilistic assignment

Returns

a randomly sampled [FragHit](#)

Definition at line 98 of file fragments.cpp.

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

- src/fragments.h
- src/fragments.cpp

3.10 FrequencyMatrix Class Reference

```
#include <frequencymatrix.h>
```

Public Member Functions

- [FrequencyMatrix](#) ()
- [FrequencyMatrix](#) (size_t m, size_t n, double alpha, bool logged=true)
- double [operator\(\)](#) (size_t i, size_t j) const
- double [operator\(\)](#) (size_t k) const
- void [increment](#) (size_t i, size_t j, double incr_amt)
- void [increment](#) (size_t k, double incr_amt)
- double [arr](#) (size_t k) const
- double [row](#) (size_t i) const
- void [set_logged](#) (bool logged)

3.10.1 Detailed Description

The [FrequencyMatrix](#) class keeps track of the frequency parameters in order to allow for constant-time probability look-ups and updates. The table is rectangular to allow for multiple distributions to be stored in one [FrequencyMatrix](#). Rows are distributions. Values are in log space by default.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 24 of file frequencymatrix.h.

3.10.2 Constructor & Destructor Documentation**3.10.2.1 `FrequencyMatrix::FrequencyMatrix ()` [inline]**

dummy constructor

Definition at line 56 of file frequencymatrix.h.

3.10.2.2 `FrequencyMatrix::FrequencyMatrix (size_t m, size_t n, double alpha, bool logged = true)`[FrequencyMatrix](#) constructor initializes the matrix based on the log of the given pseudo-counts**Parameters**

<i>m</i>	a size_t specifying the number of distributions (rows)
<i>n</i>	a size_t specifying the number of values in each distribution (columns)
<i>alpha</i>	a double specifying the initial pseudo-counts (un-logged)
<i>logged</i>	bool that specifies if the table is in log space

Definition at line 16 of file frequencymatrix.cpp.

3.10.3 Member Function Documentation**3.10.3.1 `double FrequencyMatrix::arr (size_t k) const` [inline]**

a member function that returns the raw value stored at a given position of the flattened matrix

Parameters

<i>k</i>	the array position
----------	--------------------

Returns

a double specifying the raw value stored at the given position of the flattened matrix

Definition at line 102 of file frequencymatrix.h.

3.10.3.2 void FrequencyMatrix::increment (size_t *i*, size_t *j*, double *incr_amt*)

a member function to increase the mass of a given position in the matrix

Parameters

<i>i</i>	the distribution (row)
<i>j</i>	the value (column)
<i>incr_amt</i>	the amount to increase the mass by (logged if table is logged)

Definition at line 39 of file frequencymatrix.cpp.

3.10.3.3 void FrequencyMatrix::increment (size_t *k*, double *incr_amt*)

a member function to increase the mass of a given position in the flattened matrix (logged if table is logged)

Parameters

<i>k</i>	the array position
<i>incr_amt</i>	the amount to increase the mass by (logged if table is logged)

Definition at line 56 of file frequencymatrix.cpp.

3.10.3.4 double FrequencyMatrix::operator() (size_t *k*) const

a member function to extract the probability of a given position in the flattened matrix (logged if table is logged)

Parameters

<i>k</i>	the array position
----------	--------------------

Returns

a double specifying the probability of the given position in the flattened matrix (logged if table is logged)

Definition at line 33 of file frequencymatrix.cpp.

3.10.3.5 double FrequencyMatrix::operator() (size_t *i*, size_t *j*) const

a member function to extract the probability of a given position in the matrix (logged if table is logged)

Parameters

<i>i</i>	the distribution (row)
<i>j</i>	the value (column)

Returns

a double specifying the probability of the given value in the given distribution (logged if table is logged)

Definition at line 24 of file frequencymatrix.cpp.

3.10.3.6 double FrequencyMatrix::row (size_t *i*) const [inline]

a member function that returns the raw row sum

Parameters

<i>i</i>	the distribution (row)
----------	------------------------

Returns

a double specifying the raw row sum for the given distribution

Definition at line 109 of file frequencymatrix.h.

3.10.3.7 void FrequencyMatrix::set_logged (bool *logged*)

a member function that converts the table between log-space and non-log space

Parameters

<i>logged</i>	bool specifying if the table should be converted to logged or non-logged space
---------------	--

Definition at line 61 of file frequencymatrix.cpp.

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

- src/frequencymatrix.h
- src/frequencymatrix.cpp

3.11 Globals Struct Reference

```
#include <main.h>
```

Public Attributes

- [FLD](#) * **fld**
- [MismatchTable](#) * **mismatch_table**
- [BiasBoss](#) * **bias_table**
- [TargetTable](#) * **targ_table**

3.11.1 Detailed Description

a struct for holding pointers to the global parameter tables (bias_table, mismatch_table, fld)

Definition at line 27 of file main.h.

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

- src/main.h

3.12 Indel Struct Reference

```
#include <fragments.h>
```

Public Member Functions

- [Indel](#) (size_t p, size_t l)

Public Attributes

- size_t [pos](#)
- size_t [len](#)

3.12.1 Detailed Description

The [Indel](#) struct stores the information for a single insertion or deletion.

Author

Adam Roberts

Date

2012 Artistic License 2.0

Definition at line 38 of file fragments.h.

3.12.2 Constructor & Destructor Documentation

3.12.2.1 [Indel::Indel](#) (size_t p, size_t l) [inline]

[Indel](#) constructor

Definition at line 53 of file fragments.h.

3.12.3 Member Data Documentation

3.12.3.1 `size_t Indel::len`

a public `size_t` for the length of the [Indel](#) in the read

Definition at line 48 of file `fragments.h`.

3.12.3.2 `size_t Indel::pos`

a public `size_t` for the position of the [Indel](#) in the read

Definition at line 43 of file `fragments.h`.

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

- `src/fragments.h`

3.13 MarkovModel Class Reference

Public Member Functions

- **MarkovModel** (`size_t` order, `size_t` window_size, `size_t` num_pos, double alpha, bool logged=true)
- double **transition_prob** (`size_t` p, `size_t` cond, `size_t` curr) const
- double **seq_prob** (const [Sequence](#) &seq, int left) const
- double **marginal_prob** (`size_t` w, `size_t` nuc) const
- void **update** (const [Sequence](#) &seq, int left, double mass)
- void **fast_learn** (const [Sequence](#) &seq, double mass, const std::vector< double > &fl_cdf)
- void **calc_marginals** ()
- void **set_logged** (bool logged)

3.13.1 Detailed Description

Definition at line 19 of file `markovmodel.h`.

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

- `src/markovmodel.h`
- `src/markovmodel.cpp`

3.14 MismatchTable Class Reference

```
#include <mismatchmodel.h>
```

Public Member Functions

- [MismatchTable](#) (double *alpha*)
- void [activate](#) (bool *active*=true)
- double [log_likelihood](#) (const [FragHit](#) &*f*) const
- void [update](#) (const [FragHit](#) &, double *mass*)
- std::string [to_string](#) () const
- void [append_output](#) (std::ofstream &*outfile*) const

3.14.1 Detailed Description

The [MismatchTable](#) class is used to store and update mismatch (error) parameters using a first-order Markov model based on nucleotide and position in a read and to return likelihoods of mismatches in given reads. All values are stored and returned in log space.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 24 of file mismatchmodel.h.

3.14.2 Constructor & Destructor Documentation

3.14.2.1 MismatchTable::MismatchTable (double *alpha*)

[MismatchTable](#) constructor initializes the model parameters using the specified (non-logged) pseudo-counts.

Parameters

<i>alpha</i>	a double containing the non-logged pseudo-counts for parameter initialization
--------------	---

Definition at line 19 of file mismatchmodel.cpp.

3.14.3 Member Function Documentation

3.14.3.1 void MismatchTable::activate (bool *active* = true) [inline]

member function that 'activates' the table to allow its values to be used in calculating log-likelihoods when it is sufficiently burned-in

Parameters

<i>active</i>	a boolean specifying whether to activate (true) or deactivate (false)
---------------	---

Definition at line 69 of file mismatchmodel.h.

3.14.3.2 void MismatchTable::append_output (std::ofstream & outfile) const

a member function that outputs the final model parameters in a tab-separated file the file has 1 row for each read position and the parameters are in columns indexed as (ref, prev, obs) in base 4 with A,C,G,T encoded as 0,1,2,3.

Parameters

<i>file</i>	stream to append to
-------------	---------------------

Definition at line 70 of file biascorrection.cpp.

3.14.3.3 double MismatchTable::log_likelihood (const FragHit & f) const

member function returns the log likelihood of mismatches in the mapping given the current error model parameters

Parameters

<i>f</i>	the fragment mapping to calculate the log likelihood for
----------	--

Returns

the log likelihood of the mapping based on mismatches

Definition at line 28 of file mismatchmodel.cpp.

3.14.3.4 string MismatchTable::to_string () const

member function that returns a string containing a collapsed confusion matrix based on the model parameters for the first read

Returns

a space-separated string for the flattened, collapsed confusion matrix in row-major format (observed value as rows)

Definition at line 250 of file mismatchmodel.cpp.

3.14.3.5 void MismatchTable::update (const FragHit & f, double mass)

member function that updates the error model parameters based on a mapping and its (logged) mass

Parameters

<i>f</i>	the fragment mapping
<i>mass</i>	the logged mass to increase the parameters by

Definition at line 143 of file mismatchmodel.cpp.

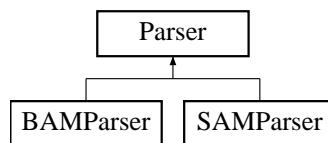
The documentation for this class was generated from the following files:

- src/mismatchmodel.h
- src/biascorrection.cpp
- src/fld.cpp
- src/mismatchmodel.cpp

3.15 Parser Class Reference

```
#include <mapparser.h>
```

Inheritance diagram for Parser:

**Public Member Functions**

- virtual const std::string [header](#) () const =0
- virtual const TransIndex & [targ_index](#) () const =0
- virtual const TransIndex & [targ_lengths](#) () const =0
- virtual bool [next_fragment](#) ([Fragment](#) &f)=0
- virtual void [reset](#) ()=0

3.15.1 Detailed Description

The [Parser](#) class is an abstract class that can be a [SAMParser](#) or [BAMParser](#). It fills [Fragment](#) objects by parsing an input file in SAM/BAM format.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 33 of file mapparser.h.

3.15.2 Member Function Documentation

3.15.2.1 `virtual const std::string Parser::header () const` [pure virtual]

a member function that returns a string version of the header

Returns

string version of the header

Implemented in [BAMParser](#), and [SAMParser](#).

3.15.2.2 `virtual bool Parser::next_fragment (Fragment & f)` [pure virtual]

a member function that loads all mappings of the next fragment

Parameters

<i>f</i>	the empty Fragment to add mappings to
----------	---

Returns

true if more reads remain in the SAM/BAM file/stream, false otherwise

Implemented in [BAMParser](#), and [SAMParser](#).

3.15.2.3 `virtual void Parser::reset ()` [pure virtual]

a member function that resets the parser and rewinds to the beginning of the input

Implemented in [BAMParser](#).

3.15.2.4 `virtual const TransIndex& Parser::targ_index () const` [pure virtual]

a member function that returns the target-to-index map

Returns

the target-to-index map

Implemented in [BAMParser](#), and [SAMParser](#).

3.15.2.5 `virtual const TransIndex& Parser::targ_lengths () const` [pure virtual]

a member function that returns the target-to-length map

Returns

the target-to-length map

Implemented in [BAMParser](#), and [SAMPParser](#).

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

- `src/mapparser.h`

3.16 ParseThreadSafety Struct Reference

```
#include <threadsafety.h>
```

Public Attributes

- [Fragment](#) * `next_frag`
- `boost::mutex` `mut`
- `boost::condition_variable` `cond`
- `bool` `frag_clean`

3.16.1 Detailed Description

The [ParseThreadSafety](#) struct stores objects to allow for parsing to safely occur on a separate thread from processing.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 23 of file `threadsafety.h`.

3.16.2 Member Data Documentation

3.16.2.1 `boost::condition_variable ParseThreadSafety::cond`

a conditional variable where the processor waits for a new [Fragment](#) and the parser waits for the [Fragment](#) pointer to be copied by the processor

Definition at line 39 of file `threadsafety.h`.

3.16.2.2 `bool ParseThreadSafety::frag_clean`

a `bool` specifying the condition that the current `next_frag` pointer is clean, meaning that it hasn't been copied by the processor

Definition at line 45 of file `threadsafety.h`.

3.16.2.3 `boost::mutex ParseThreadSafety::mut`

a mutex for the conditional variable

Definition at line 33 of file threadsafety.h.

3.16.2.4 `Fragment* ParseThreadSafety::next_frag`

a pointer to the next [Fragment](#) to be processed by the main thread

Definition at line 28 of file threadsafety.h.

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

- `src/threadsafety.h`

3.17 `PosWeightTable` Class Reference

```
#include <biascorrection.h>
```

Public Member Functions

- [PosWeightTable](#) (const std::vector< size_t > &len_bins, const std::vector< double > &pos_bins, double alpha)
- const std::vector< size_t > & **len_bins** () const
- const std::vector< double > & **pos_bins** () const
- void [increment_expected](#) (size_t len, double pos)
- void [increment_expected](#) (size_t l, size_t p)
- void [normalize_expected](#) ()
- void [increment_observed](#) (size_t len, double pos, double normalized_mass)
- void [increment_observed](#) (size_t l, size_t p, double normalized_mass)
- double [get_weight](#) (size_t len, double pos) const
- double [get_weight](#) (size_t l, size_t p) const
- void [append_output](#) (std::ofstream &outfile) const

3.17.1 Detailed Description

The [PosWeightTable](#) class keeps track of fractional position bias parameters in log space. It allows for the bias associated with a given fractional position to be calculated, and for the bias parameters to be updated based on additional fragment observations.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 118 of file biascorrection.h.

3.17.2 Constructor & Destructor Documentation

3.17.2.1 `PosWeightTable::PosWeightTable (const std::vector< size_t > & len_bins, const std::vector< double > & pos_bins, double alpha)`

[PosWeightTable](#) Constructor

Parameters

<i>len_bins</i>	a vector of unsigned integers specifying the bin ranges for target lengths
<i>pos_bins</i>	a vector of doubles specifying the bin ranges for fractional positions
<i>alpha</i>	a double specifying the strength of the uniform prior (logged pseudo-counts for each parameter)

Definition at line 146 of file biascorrection.cpp.

3.17.3 Member Function Documentation

3.17.3.1 `void PosWeightTable::append_output (std::ofstream & outfile) const`

a member function that outputs the fractional position probabilities in matrix format with length bins as rows and fractional position bins as columns

Parameters

<i>outfile</i>	the file to append to
----------------	-----------------------

3.17.3.2 `double PosWeightTable::get_weight (size_t len, double pos) const`

a member function that return the bias weight (logged) of a fractional target position

Parameters

<i>len</i>	the target length
<i>pos</i>	the fractional target position

Returns

the logged bias weight for the fractional target position

Definition at line 182 of file biascorrection.cpp.

3.17.3.3 `double PosWeightTable::get_weight (size_t l, size_t p) const`

a member function that return the bias weight (logged) of a fractional target position bin

Parameters

<i>l</i>	the target length bin
<i>p</i>	the fractional target position bin

Returns

the logged bias weight for the fractional target position

Definition at line 189 of file biascorrection.cpp.

3.17.3.4 void PosWeightTable::increment_expected (size_t *l*, size_t *p*)

a member function that increments the expected counts for the given fractional position bin by 1 (logged)

Parameters

<i>l</i>	the target length bin
<i>p</i>	the fractional target position bin

Definition at line 46 of file biascorrection.cpp.

3.17.3.5 void PosWeightTable::increment_expected (size_t *len*, double *pos*)

a member function that increments the expected counts for the given fractional position by 1 (logged)

Parameters

<i>len</i>	the target length
<i>pos</i>	the fractional target position

Definition at line 153 of file biascorrection.cpp.

3.17.3.6 void PosWeightTable::increment_observed (size_t *l*, size_t *p*, double *normalized_mass*)

a member function that increments the observed counts for the given fragment position bin by some mass (logged)

Parameters

<i>l</i>	the target length bin
<i>p</i>	the fractional target position bin
<i>normalized_mass</i>	the mass (logged probabilistic assignment) of the fragment normalized by its estimated expression

Definition at line 177 of file biascorrection.cpp.

3.17.3.7 `void PosWeightTable::increment_observed (size_t len, double pos, double normalized_mass)`

a member function that increments the observed counts for the given fragment position by some mass (logged)

Parameters

<i>len</i>	the target length
<i>pos</i>	the fractional target position
<i>normalized_-mass</i>	the mass (logged probabilistic assignment) of the fragment normalized by its estimated expression

Definition at line 170 of file biascorrection.cpp.

3.17.3.8 `void PosWeightTable::normalize_expected ()`

a member function that normalizes the expected counts and converts them to the log scale

Definition at line 165 of file biascorrection.cpp.

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

- src/biascorrection.h
- src/biascorrection.cpp

3.18 RoundParams Struct Reference

```
#include <targets.h>
```

Public Member Functions

- [RoundParams](#) ()

Public Attributes

- double [mass](#)
- double [tot_ambig_mass](#)
- double [mass_var](#)
- double [var_sum](#)

3.18.1 Detailed Description

The [RoundParams](#) struct stores the target parameters unique to a given round (iteration) of EM

Author

Adam Roberts

Date

2012 Artistic License 2.0

Definition at line 34 of file targets.h.

3.18.2 Constructor & Destructor Documentation**3.18.2.1 RoundParams::RoundParams () [inline]**

[RoundParams](#) constructor sets initial values for parameters

Definition at line 59 of file targets.h.

3.18.3 Member Data Documentation**3.18.3.1 double RoundParams::mass**

a public double that stores the (logged) assigned mass based on observed fragment mapping probabilities

Definition at line 39 of file targets.h.

3.18.3.2 double RoundParams::mass_var

a public double that stores the (logged) variance due to uncertainty on p

Definition at line 49 of file targets.h.

3.18.3.3 double RoundParams::tot_ambig_mass

a public double that stores the (logged) total mass of ambiguous fragments mapping to the target

Definition at line 44 of file targets.h.

3.18.3.4 double RoundParams::var_sum

a public double that stores the (logged) weighted sum of the variance on the assignments

Definition at line 54 of file targets.h.

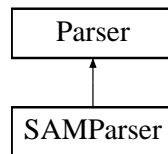
The documentation for this struct was generated from the following file:

- src/targets.h

3.19 SAMParser Class Reference

```
#include <mapparser.h>
```

Inheritance diagram for SAMParser:



Public Member Functions

- [SAMParser](#) (std::istream *in)
- const std::string [header](#) () const
- const TransIndex & [targ_index](#) () const
- const TransIndex & [targ_lengths](#) () const
- bool [next_fragment](#) ([Fragment](#) &f)

3.19.1 Detailed Description

The [SAMParser](#) class fills [Fragment](#) objects by parsing an input in SAM format. The input may come from a file or stdin.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 223 of file mapparser.h.

3.19.2 Constructor & Destructor Documentation

3.19.2.1 SAMParser::SAMParser (std::istream * in)

[SAMParser](#) constructor removes the header, and parses the first line

Definition at line 350 of file mapparser.cpp.

3.19.3 Member Function Documentation

3.19.3.1 const std::string SAMParser::header () const [inline, virtual]

a member function that returns a string version of the header

Returns

string version of the header

Implements [Parser](#).

Definition at line 271 of file mapparser.h.

3.19.3.2 bool SAMParser::next_fragment (Fragment & f) [virtual]

a member function that loads all mappings of the next fragment when the next fragment is reached, the current alignment is left in the `_frag_buff` for the next call

Parameters

<code>f</code>	the empty Fragment to add mappings to
----------------	---

Returns

true if more reads remain in the SAM file, false otherwise

Implements [Parser](#).

Definition at line 406 of file mapparser.cpp.

3.19.3.3 const TransIndex& SAMParser::targ_index () const [inline, virtual]

a member function that returns the target-to-index map

Returns

the target-to-index map

Implements [Parser](#).

Definition at line 277 of file mapparser.h.

3.19.3.4 const TransIndex& SAMParser::targ_lengths () const [inline, virtual]

a member function that returns the target-to-length map

Returns

the target-to-length map

Implements [Parser](#).

Definition at line 283 of file mapparser.h.

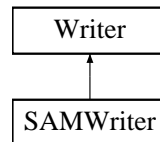
The documentation for this class was generated from the following files:

- `src/mapparser.h`
- `src/mapparser.cpp`

3.20 SAMWriter Class Reference

```
#include <mapparser.h>
```

Inheritance diagram for SAMWriter:



Public Member Functions

- [SAMWriter](#) (std::ostream *out, bool sample)
- [~SAMWriter](#) ()
- void [write_fragment](#) (Fragment &f)

3.20.1 Detailed Description

The [SAMWriter](#) class writes [Fragment](#) objects back to file (in SAM format) with per-mapping probabilistic assignments.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 301 of file mapparser.h.

3.20.2 Constructor & Destructor Documentation

3.20.2.1 SAMWriter::SAMWriter (std::ostream * out, bool sample)

[SAMWriter](#) constructor stores a pointer to the output stream

Parameters

<i>out</i>	SAM output stream
<i>sample</i>	specifies if a single alignment should be sampled (true) or all output with their respective probabilities (false)

Definition at line 556 of file mapparser.cpp.

3.20.2.2 SAMWriter::~~SAMWriter ()

[SAMWriter](#) destructor flushes and deletes output stream

Definition at line 560 of file mapparser.cpp.

3.20.3 Member Function Documentation

3.20.3.1 void SAMWriter::write_fragment ([Fragment](#) & *f*) [virtual]

a member function that writes all mappings of the fragment to the output file in SAM format along with their probabilities in the "XP" field

Parameters

<i>f</i>	the processed Fragment to output
----------	--

Implements [Writer](#).

Definition at line 566 of file mapparser.cpp.

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

- src/mapparser.h
- src/mapparser.cpp

3.21 Sequence Class Reference

```
#include <sequence.h>
```

Public Member Functions

- [Sequence](#) ()
- [Sequence](#) (const std::string &seq, bool rev)
- [Sequence](#) (const [Sequence](#) &other)
- [Sequence](#) & operator= (const [Sequence](#) &other)
- ~[Sequence](#) ()
- void [set](#) (const std::string &seq, bool rev)
- bool [empty](#) () const
- size_t [operator\[\]](#) (const size_t index) const
- size_t [length](#) () const

3.21.1 Detailed Description

The [Sequence](#) class is used to store and access encoded nucleotide sequences.

Author

Adam Roberts

Date

2012 Artistic License 2.0

Definition at line 20 of file sequence.h.

3.21.2 Constructor & Destructor Documentation**3.21.2.1 Sequence::Sequence ()**

dummy constructor

Definition at line 58 of file sequence.cpp.

3.21.2.2 Sequence::Sequence (const std::string & seq, bool rev)

[Sequence](#) constructor encodes and stores the given nucleotide sequence

Parameters

<i>seq</i>	the nucleotide sequence to encode and store
<i>rev</i>	a boolean if the sequence should be reverse complemented before encoding

Definition at line 60 of file sequence.cpp.

3.21.2.3 Sequence::Sequence (const Sequence & other)

[Sequence](#) copy constructor

Parameters

<i>other</i>	the Sequence object to copy
--------------	---

Definition at line 65 of file sequence.cpp.

3.21.2.4 Sequence::~~Sequence ()

[Sequence](#) destructor. Deletes the char array.

Definition at line 88 of file sequence.cpp.

3.21.3 Member Function Documentation

3.21.3.1 `bool Sequence::empty () const` `[inline]`

a member function that returns true iff the encoded sequence has zero length

Returns

true iff the encoded sequence has zero length

Definition at line 76 of file `sequence.h`.

3.21.3.2 `size_t Sequence::length () const` `[inline]`

a member function that returns the length of the encoded sequence

Returns

the length of the encoded sequence

Definition at line 90 of file `sequence.h`.

3.21.3.3 `Sequence & Sequence::operator= (const Sequence & other)`

[Sequence](#) assignment constructor

Parameters

<i>other</i>	the Sequence object to copy
--------------	---

Definition at line 75 of file `sequence.cpp`.

3.21.3.4 `size_t Sequence::operator[] (const size_t index) const`

a member function that returns the encoded character at the given index

Parameters

<i>index</i>	the index of the encoded character to return (assumed to be $< _len$)
--------------	---

Returns

the encoded character at the given index

Definition at line 119 of file `sequence.cpp`.

3.21.3.5 void Sequence::set (const std::string & seq, bool rev)

a member function that encodes the given sequence and overwrites the current stored sequence with it

Parameters

<i>seq</i>	the nucleotide sequence to encode and store
<i>rev</i>	a boolean if the sequence should be reverse complemented before encoding

Definition at line 95 of file sequence.cpp.

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

- src/sequence.h
- src/sequence.cpp

3.22 SeqWeightTable Class Reference

```
#include <biascorrection.h>
```

Public Member Functions

- [SeqWeightTable](#) (size_t window_size, double alpha)
- void [copy_observed](#) (const [SeqWeightTable](#) &other)
- void [copy_expected](#) (const [SeqWeightTable](#) &other)
- void [increment_expected](#) (const [Sequence](#) &seq, double mass, const std::vector<double> &fl_cdf)
- void [normalize_expected](#) ()
- void [increment_observed](#) (const [Sequence](#) &seq, size_t i, double mass)
- double [get_weight](#) (const [Sequence](#) &seq, size_t i) const
- std::string [to_string](#) () const
- void [append_output](#) (std::ofstream &outfile) const

3.22.1 Detailed Description

The [SeqWeightTable](#) class keeps track of sequence-specific bias parameters. It allows for the bias associated with a given sequence to be calculated, and for the bias parameters to be updated based on additional observations. All values stored in log space.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 30 of file biascorrection.h.

3.22.2 Constructor & Destructor Documentation

3.22.2.1 SeqWeightTable::SeqWeightTable (size_t *window_size*, double *alpha*)

[SeqWeightTable](#) Constructor

Parameters

<i>window_size</i>	an unsigned integer specifying the size of the bias window surrounding fragment ends
<i>alpha</i>	a double specifying the strength of the uniform prior (logged pseudo-counts for each parameter)

Definition at line 31 of file biascorrection.cpp.

3.22.3 Member Function Documentation

3.22.3.1 void SeqWeightTable::append_output (std::ofstream & *outfile*) const

a member function that outputs the positional nucleotide probabilities in matrix format with nucleotides (A,C,G,T) as rows and window position as columns

Parameters

<i>outfile</i>	the file to append to
----------------	-----------------------

3.22.3.2 void SeqWeightTable::copy_expected (const SeqWeightTable & *other*)

a member function that overwrites the "expected" parameters with those from another [SeqWeightTable](#)

Parameters

<i>other</i>	another SeqWeightTable from which to copy the parameters
--------------	--

Definition at line 41 of file biascorrection.cpp.

3.22.3.3 void SeqWeightTable::copy_observed (const SeqWeightTable & *other*)

a member function that overwrites the "observed" parameters with those from another [SeqWeightTable](#)

Parameters

<i>other</i>	another SeqWeightTable from which to copy the parameters
--------------	--

Definition at line 36 of file biascorrection.cpp.

3.22.3.4 double SeqWeightTable::get_weight (const Sequence & seq, size_t i) const

a member function that calculates the bias weight (logged) of a bias window

Parameters

<i>seq</i>	the target sequence the fragment hits to
<i>i</i>	the fragment end point (the central point of the bias window)

Returns

the bias weight for the bias window which is the product of the individual nucleotide bias weights

Definition at line 63 of file biascorrection.cpp.

3.22.3.5 void SeqWeightTable::increment_expected (const Sequence & seq, double mass, const std::vector< double > & fl_cdf)

a member function that increments the expected counts for a sliding window through the given target sequence by some mass

Parameters

<i>seq</i>	the target sequence
<i>mass</i>	the amount of used to weight the target's sequence in the parameter table

3.22.3.6 void SeqWeightTable::increment_observed (const Sequence & seq, size_t i, double mass)

a member function that increments the observed counts for the given fragment sequence by some mass (logged)

Parameters

<i>seq</i>	the target sequence (possibly reverse complemented) to which the fragment end maps
<i>i</i>	the index into the sequence at which to center the bias window (where the fragment starts/ends)
<i>mass</i>	the fragment's mass

Definition at line 57 of file biascorrection.cpp.

3.22.3.7 void SeqWeightTable::normalize_expected ()

a member function that normalizes the expected counts and converts them to the log scale

Definition at line 51 of file biascorrection.cpp.

3.22.3.8 std::string SeqWeightTable::to_string () const

a member function that returns a string containing the positional nucleotide probabilities in column-major order (A,C,G,T)

Returns

the string representation of the positional nucleotide probabilities

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

- src/biascorrection.h
- src/biascorrection.cpp

3.23 Target Class Reference

```
#include <targets.h>
```

Public Member Functions

- [Target](#) (const size_t id, const std::string &name, const std::string &seq, double alpha, const [Globals](#) *globs)
- [~Target](#) ()
- void [lock](#) ()
- void [unlock](#) ()
- const std::string & [name](#) () const
- TransID [id](#) () const
- const [Sequence](#) & [seq](#) (bool rev) const
- size_t [length](#) () const
- double [rho](#) () const
- double [mass](#) (bool with_pseudo=true) const
- double [mass_var](#) (bool with_pseudo=true) const
- double [var_sum](#) () const
- double [tot_ambig_mass](#) () const
- void [round_reset](#) ()
- size_t [tot_counts](#) () const
- size_t [uniq_counts](#) () const
- [Bundle](#) * [bundle](#) ()
- void [bundle](#) ([Bundle](#) *b)
- void [add_mass](#) (double p, double v, double mass)
- void [incr_counts](#) (bool uniq, size_t incr_amt=1)
- double [log_likelihood](#) (const [FragHit](#) &frag, bool with_pseudo) const
- double [est_effective_length](#) ([FLD](#) *fld=NULL, bool with_bias=true) const
- double [cached_effective_length](#) (bool with_bias=true) const
- void [update_target_bias](#) ([BiasBoss](#) *bias_table=NULL, [FLD](#) *fld=NULL)
- bool [solveable](#) ()
- void [solveable](#) (bool s)

3.23.1 Detailed Description

The [Target](#) class is used to store objects for the targets being mapped to. Besides storing basic information about the object (id, length), it also stores a mass based on the number of fragments mapping to the object. To help with updating this number, it returns the likelihood that a given fragment originated from it. These values are stored and returned in log space.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 76 of file targets.h.

3.23.2 Constructor & Destructor Documentation

3.23.2.1 `Target::Target (const size_t id, const std::string & name, const std::string & seq, double alpha, const Globals * globs)`

[Target](#) Constructor

Parameters

<i>name</i>	a string that stores the target name
<i>seq</i>	a string that stores the target sequence
<i>alpha</i>	a double that specifies the intial pseudo-counts (non-logged)
<i>globs</i>	a pointer to the struct containing pointers to the global parameter tables (bias_table, mismatch_table, fld)

Definition at line 23 of file targets.cpp.

3.23.2.2 `Target::~Target () [inline]`

[Target](#) Destructor deletes bias vectors

Definition at line 185 of file targets.h.

3.23.3 Member Function Documentation

3.23.3.1 `void Target::add_mass (double p, double v, double mass)`

a member function that increases the expected fragment counts and variance by a given (logged) fragment mass

Parameters

<i>p</i>	a double for the (logged) probability that the fragment was generated by this target
<i>v</i>	a double for the (logged) approximate variance (uncertainty) on the probability
<i>mass</i>	a double specifying the (logged) mass of the fragment being mapped

Definition at line 50 of file targets.cpp.

3.23.3.2 Bundle* Target::bundle () [inline]

a member function that returns the [Bundle](#) this [Target](#) is a member of

Returns

a pointer to the [Bundle](#) this target is a member of

Definition at line 278 of file targets.h.

3.23.3.3 void Target::bundle (Bundle * b) [inline]

a member function that set the [Bundle](#) this [Target](#) is a member of

Parameters

<i>b</i>	a pointer to the Bundle to set this Target as a member of
----------	---

Definition at line 284 of file targets.h.

3.23.3.4 double Target::cached_effective_length (bool with_bias = true) const

a member function that returns the most recently estimated effective length (logged) as calculated by the bias updater thread

Returns

the cached effective length of the target calculated

Definition at line 149 of file targets.cpp.

3.23.3.5 double Target::est_effective_length (FLD * fld = NULL, bool with_bias = true) const

a member function that calculates and returns the estimated effective length of the target (logged) using the avg bias

Parameters

<i>fld</i>	an optional pointer to a different FLD than the global one, for thread-safety
<i>with_bias</i>	a boolean specifying whether or not the average bias should be used in the calculation

Returns

the estimated effective length of the target calculated as $\tilde{l} = \bar{bias} \sum_{l=1}^{L(t)} D(l)(L(t) - l + 1)$

Definition at line 127 of file targets.cpp.

3.23.3.6 TransID Target::id () const [inline]

a member function that returns the target id

Returns

TransID target ID

Definition at line 213 of file targets.h.

3.23.3.7 void Target::incr_counts (bool uniq, size_t incr_amt = 1) [inline]

a member function that increases the count of fragments mapped to this target

Parameters

<i>uniq</i>	a bool specifying whether or not the fragment uniquely maps to this target
<i>incr_amt</i>	a size_t to increase the counts by

Definition at line 299 of file targets.h.

3.23.3.8 size_t Target::length () const [inline]

a member function that returns the target length

Returns

target length

Definition at line 224 of file targets.h.

3.23.3.9 void Target::lock () [inline]

a member function that locks the target mutex to provide thread safety

Definition at line 196 of file targets.h.

3.23.3.10 double Target::log_likelihood (const FragHit & frag, bool with_pseudo) const

a member function that returns (a value proportional to) the log likelihood the given fragment originated from this target

Parameters

<i>frag</i>	a FragHit to return the likelihood of being originated from this target
<i>with_pseudo</i>	a FragHit specifying whether or not pseudo-counts should be included in the calculation

Returns

(a value proportional to) the log likelihood the given fragment originated from this target

Definition at line 96 of file targets.cpp.

3.23.3.11 double Target::mass (bool with_pseudo = true) const

a member function that returns the current (logged) probabilistically assigned fragment mass

Parameters

<i>with_pseudo</i>	a boolean specifying whether pseudo-counts should be included in returned mass
--------------------	--

Returns

logged mass

Definition at line 82 of file targets.cpp.

3.23.3.12 double Target::mass_var (bool with_pseudo = true) const

a member function that returns the total (logged) variance on mass

Returns

the total (logged) variance on mass

Definition at line 89 of file targets.cpp.

3.23.3.13 const std::string& Target::name () const [inline]

a member function that returns the target name

Returns

string containing target name

Definition at line 207 of file targets.h.

3.23.3.14 `double Target::rho () const`

a member function that returns the current estimated rho (logged, w/ pseudo-counts) for the target

Returns

the current estimated rho

Definition at line 73 of file targets.cpp.

3.23.3.15 `void Target::round_reset ()`

a member function that readies the target object for the next round of batch EM

Definition at line 66 of file targets.cpp.

3.23.3.16 `const Sequence& Target::seq (bool rev) const` `[inline]`

a member function that returns the target sequence

Returns

string containing target sequence

Definition at line 218 of file targets.h.

3.23.3.17 `bool Target::solveable ()` `[inline]`

a member function that returns the `_solveable` flag

Returns

a boolean specifying whether or not the target has a unique solution

Definition at line 341 of file targets.h.

3.23.3.18 `void Target::solveable (bool s)` `[inline]`

a member function that sets the `_solveable` flag

Parameters

<i>a</i>	boolean specifying whether or not the target has a unique solution
----------	--

Definition at line 347 of file targets.h.

3.23.3.19 double Target::tot_ambig_mass () const [inline]

a member function that returns the (logged) total mass of ambiguous fragments mapping to the target

Returns

the (logged) total mass of ambiguous fragments mapping to the target

Definition at line 255 of file targets.h.

3.23.3.20 size_t Target::tot_counts () const [inline]

a member function that returns the current count of fragments mapped to this target (uniquely or ambiguously)

Returns

total fragment count

Definition at line 266 of file targets.h.

3.23.3.21 size_t Target::uniq_counts () const [inline]

a member function that returns the current count of fragments uniquely mapped to this target

Returns

unique fragment count

Definition at line 272 of file targets.h.

3.23.3.22 void Target::unlock () [inline]

a member function that unlocks the target mutex to provide thread safety

Definition at line 201 of file targets.h.

3.23.3.23 void Target::update_target_bias (BiasBoss * *bias_table* = NULL, FLD * *fld* = NULL)

a member function that causes the target bias to be re-calculated by the `_bias_table` based on current parameters

Parameters

<i>bias_table</i>	an optional pointer to a different BiasBoss than the global one, for thread-safety
<i>fld</i>	an optional pointer to a different FLD than the global one, for thread-safety

Definition at line 157 of file targets.cpp.

3.23.3.24 double Target::var_sum () const [inline]

a member function that returns the (logged) weighted sum of the variance on the assignments

Returns

the (logged) weighted sum of the variance on the assignments

Definition at line 249 of file targets.h.

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

- src/targets.h
- src/targets.cpp

3.24 TargetTable Class Reference

```
#include <targets.h>
```

Public Member Functions

- [TargetTable](#) (const std::string &targ_fasta_file, const TransIndex &targ_index, const TransIndex &targ_lengths, double alpha, const AlphaMap *alpha_map, [Globals](#) *globs)
- [~TargetTable](#) ()
- [Target](#) * [get_targ](#) (TransID id)
- void [round_reset](#) ()
- size_t [size](#) () const
- double [total_fpb](#) () const
- void [update_total_fpb](#) (double incr_amt)
- void [update_covar](#) (TransID targ1, TransID targ2, double covar)
- double [get_covar](#) (TransID targ1, TransID targ2)
- size_t [covar_size](#) () const
- [Bundle](#) * [merge_bundles](#) ([Bundle](#) *b1, [Bundle](#) *b2)
- size_t [num_bundles](#) ()
- void [output_results](#) (std::string output_dir, size_t tot_counts, bool output_varcov)
- void [threaded_bias_update](#) (boost::mutex *mut)

3.24.1 Detailed Description

The [TargetTable](#) class is used to keep track of the [Target](#) objects for a run. The constructor parses a fasta file to generate the [Target](#) objects and store them in a map that allows them to be looked up based on their string id.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 364 of file targets.h.

3.24.2 Constructor & Destructor Documentation

3.24.2.1 `TargetTable::TargetTable (const std::string & targ_fasta_file, const TransIndex & targ_index, const TransIndex & targ_lengths, double alpha, const AlphaMap * alpha_map, Globals * globs)`

[TargetTable](#) Constructor

Parameters

<i>targ_fasta_file</i>	a string storing the path to the fasta file from which to load targets
<i>targ_index</i>	the target-to-index map from the alignment file
<i>targ_lengths</i>	the target-to-length map from the alignment file
<i>alpha</i>	a double that specifies the initial pseudo-counts for each bp of the targets (non-logged)
<i>alpha_map</i>	an optional pointer to a map object that specifies proportional weights of pseudo-counts for each target
<i>globs</i>	a pointer to the struct containing pointers to the global parameter tables (bias_table, mismatch_table, fld)

Definition at line 176 of file targets.cpp.

3.24.2.2 `TargetTable::~~TargetTable ()`

[TargetTable](#) Destructor deletes all of the target objects in the table

Definition at line 265 of file targets.cpp.

3.24.3 Member Function Documentation

3.24.3.1 `size_t TargetTable::covar_size () const` `[inline]`

a member function that returns the number of pairs of targets with non-zero covariance

Returns

the number of target pairs with non-zero covariance

Definition at line 477 of file targets.h.

3.24.3.2 `double TargetTable::get_covar (TransID targ1, TransID targ2)` `[inline]`

a member function that returns the covariance between two targets these returned value will be the log of the negative of the true value

Parameters

<i>targ1</i>	one of the targets in the pair
<i>targ2</i>	the other target in the pair

Returns

the negative of the pair's covariance (logged)

Definition at line 471 of file targets.h.

3.24.3.3 `Target * TargetTable::get_targ (TransID id)`

a member function that returns a pointer to the target with the given id

Parameters

<i>id</i>	of the target queried
-----------	-----------------------

Returns

pointer to the target wit the given id

Definition at line 295 of file targets.cpp.

3.24.3.4 `Bundle * TargetTable::merge_bundles (Bundle * b1, Bundle * b2)`

a member function that merges the given Bundles

Parameters

<i>b1</i>	a pointer to the first Bundle to merge
<i>b2</i>	a pointer to the second Bundle to merge

Returns

a pointer to the merged [Bundle](#)

Definition at line 300 of file targets.cpp.

3.24.3.5 `size_t TargetTable::num_bundles ()`

a member function that returns the number of bundles in the partition

Returns

the number of bundles in the partition

Definition at line 309 of file targets.cpp.

3.24.3.6 void TargetTable::output_results (std::string output_dir, size_t tot_counts, bool output_varcov)

a member function that outputs the final expression data in a file called 'results.xprs' and (optionally) the variance-covariance matrix in 'varcov.xprs' in the given output directory

Parameters

<i>output_dir</i>	the directory to output the expression file to
<i>tot_counts</i>	the total number of observed mapped fragments
<i>output_varcov</i>	boolean specifying whether to also output the variance-covariance matrix

Definition at line 375 of file targets.cpp.

3.24.3.7 void TargetTable::round_reset ()

a member function that readies all [Target](#) objects in the table for the next round of batch EM

Definition at line 314 of file targets.cpp.

3.24.3.8 size_t TargetTable::size () const [inline]

a member function that returns the number of targets in the table

Returns

number of targets in the table

Definition at line 441 of file targets.h.

3.24.3.9 void TargetTable::threaded_bias_update (boost::mutex * mut)

a member function for driving a thread that continuously updates the target bias values

Definition at line 538 of file targets.cpp.

3.24.3.10 double TargetTable::total_fpb () const

a member function that returns the (logged) total mass per base (including pseudo-counts)

Returns

the (logged) total mass per base (including pseudo-counts)

Definition at line 526 of file targets.cpp.

3.24.3.11 `void TargetTable::update_covar (TransID targ1, TransID targ2, double covar)`
`[inline]`

a member function that increases the covariance between two targets by the specified amount these values are stored positive even though they are negative (logged)

Parameters

<i>targ1</i>	one of the targets in the pair
<i>targ2</i>	the other target in the pair
<i>covar</i>	a double specifying the amount to increase the pair's covariance by (logged)

Definition at line 462 of file targets.h.

3.24.3.12 `void TargetTable::update_total_fpb (double incr_amt)`

a member function that increments the (logged) total mass per base

Parameters

<i>incr_amt</i>	the (logged) amount to increment by
-----------------	-------------------------------------

Definition at line 532 of file targets.cpp.

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

- src/targets.h
- src/targets.cpp

3.25 ThreadedMapParser Class Reference

```
#include <mapparser.h>
```

Public Member Functions

- [ThreadedMapParser](#) (std::string input_file, std::string output_file, bool write_active)
- [~ThreadedMapParser](#) ()
- void [threaded_parse](#) (ParseThreadSafety *thread_safety, [TargetTable](#) *targ_table)
- const TransIndex & [targ_index](#) ()
- const TransIndex & [targ_lengths](#) ()

- void [write_active](#) (bool b)
- void [reset_reader](#) ()

3.25.1 Detailed Description

The [ThreadedMapParser](#) class is meant to be run on as a separate thread from the main processing. Once started, this thread will read input from a file or stream in SAM/BAM format, parse, and collect read alignments into fragment alignments, and fragment alignments into fragments, which are placed on a buffer for the processing thread. Once the processing thread copies the fragment address from the buffer, the parser is unlocked to load the next fragment. The process stops when EOF is reached

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 348 of file mapparser.h.

3.25.2 Constructor & Destructor Documentation

3.25.2.1 ThreadedMapParser::ThreadedMapParser (*std::string input_file*, *std::string output_file*, *bool write_active*)

[ThreadedMapParser](#) constructor determines what format the input is in and initializes the correct parser.

Parameters

<i>input_file</i>	string containing the path to the input SAM/BAM file
<i>output_file</i>	string containing the path the output file less its extension (empty if writing is to be disabled)

Definition at line 97 of file mapparser.cpp.

3.25.2.2 ThreadedMapParser::~~ThreadedMapParser ()

[ThreadedMapParser](#) destructor deletes the parser and writer (if it exists).

Definition at line 161 of file mapparser.cpp.

3.25.3 Member Function Documentation

3.25.3.1 void ThreadedMapParser::reset_reader () [inline]

a member function that resets the input parser

Definition at line 409 of file mapparser.h.

3.25.3.2 const TransIndex& ThreadedMapParser::targ_index () [inline]

a member function that returns the target-to-index map

Returns

the target-to-index map

Definition at line 391 of file mapparser.h.

3.25.3.3 const TransIndex& ThreadedMapParser::targ_lengths () [inline]

a member function that returns the target-to-length map

Returns

the target-to-length map

Definition at line 397 of file mapparser.h.

3.25.3.4 void ThreadedMapParser::threaded_parse (ParseThreadSafety * *thread_safety*, TargetTable * *targ_table*)

a member function that drives the parse thread when all valid mappings of a fragment have been parsed, its mapped targets are found and the information is passed in a [Fragment](#) object to the processing thread through the [ParseThreadSafety](#) struct

Parameters

<i>thread_safety</i>	a pointer to the struct containing shared locks and data with the processing thread
<i>targ_table</i>	a pointer to the table of Target objects to lookup the mapped targets

Definition at line 168 of file mapparser.cpp.

3.25.3.5 void ThreadedMapParser::write_active (bool *b*) [inline]

a member function that sets the write-active status of the parser this specifies whether or not the alignments (sampled or with probs) could be output

Parameters

<i>b</i>	updated write-active status
----------	-----------------------------

Definition at line 404 of file mapparser.h.

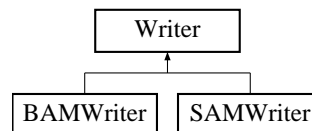
The documentation for this class was generated from the following files:

- src/mapparser.h
- src/mapparser.cpp

3.26 Writer Class Reference

```
#include <mapparser.h>
```

Inheritance diagram for Writer:

**Public Member Functions**

- virtual void [write_fragment](#) ([Fragment](#) &*f*)=0

3.26.1 Detailed Description

The [Writer](#) class is an abstract class that can be a [SAMWriter](#) or [BAMWriter](#). It writes [Fragment](#) objects back to file (in SAM/BAM format) with per-mapping probabilistic assignments.

Author

Adam Roberts

Date

2011 Artistic License 2.0

Definition at line 76 of file mapparser.h.

3.26.2 Member Function Documentation**3.26.2.1 virtual void Writer::write_fragment ([Fragment](#) & *f*) [pure virtual]**

a member function that writes all mappings of the fragment to the output file along with their probabilities in the "XP" field

Parameters

f	the processed Fragment to output
-----	--

Implemented in [BAMWriter](#), and [SAMWriter](#).

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

- `src/mapparser.h`

Index

- ~BAMParser
 - BAMParser, [6](#)
- ~BAMWriter
 - BAMWriter, [8](#)
- ~BundleTable
 - BundleTable, [14](#)
- ~Fragment
 - Fragment, [24](#)
- ~SAMWriter
 - SAMWriter, [43](#)
- ~Sequence
 - Sequence, [45](#)
- ~Target
 - Target, [51](#)
- ~TargetTable
 - TargetTable, [58](#)
- ~ThreadedMapParser
 - ThreadedMapParser, [62](#)
- activate
 - MismatchTable, [31](#)
- add_map_end
 - Fragment, [24](#)
- add_mass
 - Target, [51](#)
- add_val
 - FLD, [18](#)
- append_output
 - BiasBoss, [10](#)
 - FLD, [18](#)
 - MismatchTable, [32](#)
 - PosWeightTable, [37](#)
 - SeqWeightTable, [48](#)
- arr
 - FrequencyMatrix, [26](#)
- BAMParser, [5](#)
 - ~BAMParser, [6](#)
 - BAMParser, [6](#)
 - header, [6](#)
 - next_fragment, [6](#)
 - reset, [6](#)
 - targ_index, [7](#)
 - targ_lengths, [7](#)
- BAMWriter, [7](#)
 - ~BAMWriter, [8](#)
 - BAMWriter, [8](#)
 - write_fragment, [8](#)
- BiasBoss, [9](#)
 - append_output, [10](#)
 - BiasBoss, [10](#)
 - get_target_bias, [10](#)
 - normalize_expectations, [10](#)
 - to_string, [10](#)
 - update_expectations, [11](#)
 - update_observed, [11](#)
- Bundle, [11](#)
 - Bundle, [12](#)
 - counts, [12](#)
 - incr_counts, [12](#)
 - size, [13](#)
 - targets, [13](#)
- bundle
 - Target, [52](#)
- bundles
 - BundleTable, [14](#)
- BundleTable, [13](#)
 - ~BundleTable, [14](#)
 - bundles, [14](#)
 - BundleTable, [14](#)
 - create_bundle, [14](#)
 - merge, [15](#)
 - size, [15](#)
- cached_effective_length
 - Target, [52](#)
- cdf
 - FLD, [18](#)
- cond
 - ParseThreadSafety, [35](#)
- copy_expected
 - SeqWeightTable, [48](#)

- copy_observed
 - SeqWeightTable, 48
- counts
 - Bundle, 12
- covar_size
 - TargetTable, 58
- CovarTable, 15
 - get, 16
 - increment, 16
 - size, 16
- create_bundle
 - BundleTable, 14
- deletes_l
 - FragHit, 21
- deletes_r
 - FragHit, 21
- empty
 - Sequence, 46
- est_effective_length
 - Target, 52
- FLD, 17
 - add_val, 18
 - append_output, 18
 - cdf, 18
 - FLD, 17
 - max_val, 18
 - mean, 18
 - min_val, 19
 - pdf, 19
 - to_string, 19
 - tot_mass, 19
- frag_clean
 - ParseThreadSafety, 35
- FragHit, 20
 - deletes_l, 21
 - deletes_r, 21
 - inserts_l, 21
 - inserts_r, 22
 - left, 22
 - left_first, 22
 - length, 21
 - mapped_targ, 22
 - mate_l, 22
 - name, 22
 - pair_status, 21
 - right, 22
 - seq_l, 23
 - seq_r, 23
 - targ_id, 23
- Fragment, 23
 - ~Fragment, 24
 - add_map_end, 24
 - hits, 24
 - name, 24
 - num_hits, 24
 - sample_hit, 25
- FrequencyMatrix, 25
 - arr, 26
 - FrequencyMatrix, 26
 - increment, 26, 27
 - operator(), 27
 - row, 28
 - set_logged, 28
- get
 - CovarTable, 16
- get_covar
 - TargetTable, 58
- get_targ
 - TargetTable, 59
- get_target_bias
 - BiasBoss, 10
- get_weight
 - PosWeightTable, 37
 - SeqWeightTable, 48
- Globals, 28
- header
 - BAMParser, 6
 - Parser, 34
 - SAMParser, 41
- hits
 - Fragment, 24
- id
 - Target, 53
- incr_counts
 - Bundle, 12
 - Target, 53
- increment
 - CovarTable, 16
 - FrequencyMatrix, 26, 27
- increment_expected
 - PosWeightTable, 38
 - SeqWeightTable, 49
- increment_observed
 - PosWeightTable, 38

- SeqWeightTable, 49
- Indel, 29
 - Indel, 29
 - len, 30
 - pos, 30
- inserts_l
 - FragHit, 21
- inserts_r
 - FragHit, 22
- left
 - FragHit, 22
- left_first
 - FragHit, 22
- len
 - Indel, 30
- length
 - FragHit, 21
 - Sequence, 46
 - Target, 53
- lock
 - Target, 53
- log_likelihood
 - MismatchTable, 32
 - Target, 53
- mapped_targ
 - FragHit, 22
- MarkovModel, 30
- mass
 - RoundParams, 40
 - Target, 54
- mass_var
 - RoundParams, 40
 - Target, 54
- mate_l
 - FragHit, 22
- max_val
 - FLD, 18
- mean
 - FLD, 18
- merge
 - BundleTable, 15
- merge_bundles
 - TargetTable, 59
- min_val
 - FLD, 19
- MismatchTable, 30
 - activate, 31
 - append_output, 32
 - log_likelihood, 32
 - MismatchTable, 31
 - to_string, 32
 - update, 32
- mut
 - ParseThreadSafety, 35
- name
 - FragHit, 22
 - Fragment, 24
 - Target, 54
- next_frag
 - ParseThreadSafety, 36
- next_fragment
 - BAMParser, 6
 - Parser, 34
 - SAMParser, 42
- normalize_expectations
 - BiasBoss, 10
- normalize_expected
 - PosWeightTable, 39
 - SeqWeightTable, 49
- num_bundles
 - TargetTable, 59
- num_hits
 - Fragment, 24
- operator()
 - FrequencyMatrix, 27
- operator=
 - Sequence, 46
- output_results
 - TargetTable, 60
- pair_status
 - FragHit, 21
- Parser, 33
 - header, 34
 - next_fragment, 34
 - reset, 34
 - targ_index, 34
 - targ_lengths, 34
- ParseThreadSafety, 35
 - cond, 35
 - frag_clean, 35
 - mut, 35
 - next_frag, 36
- pdf
 - FLD, 19
- pos

- Indel, 30
- PosWeightTable, 36
 - append_output, 37
 - get_weight, 37
 - increment_expected, 38
 - increment_observed, 38
 - normalize_expected, 39
 - PosWeightTable, 37
- reset
 - BAMParser, 6
 - Parser, 34
- reset_reader
 - ThreadedMapParser, 63
- rho
 - Target, 55
- right
 - FragHit, 22
- round_reset
 - Target, 55
 - TargetTable, 60
- RoundParams, 39
 - mass, 40
 - mass_var, 40
 - RoundParams, 40
 - tot_ambig_mass, 40
 - var_sum, 40
- row
 - FrequencyMatrix, 28
- SAMParser, 41
 - header, 41
 - next_fragment, 42
 - SAMParser, 41
 - targ_index, 42
 - targ_lengths, 42
- sample_hit
 - Fragment, 25
- SAMWriter, 43
 - ~SAMWriter, 43
 - SAMWriter, 43
 - write_fragment, 44
- seq
 - Target, 55
- seq_l
 - FragHit, 23
- seq_r
 - FragHit, 23
- Sequence, 44
 - ~Sequence, 45
- empty, 46
- length, 46
- operator=, 46
- Sequence, 45
- set, 46
- SeqWeightTable, 47
 - append_output, 48
 - copy_expected, 48
 - copy_observed, 48
 - get_weight, 48
 - increment_expected, 49
 - increment_observed, 49
 - normalize_expected, 49
 - SeqWeightTable, 48
 - to_string, 49
- set
 - Sequence, 46
- set_logged
 - FrequencyMatrix, 28
- size
 - Bundle, 13
 - BundleTable, 15
 - CovarTable, 16
 - TargetTable, 60
- solveable
 - Target, 55
- targ_id
 - FragHit, 23
- targ_index
 - BAMParser, 7
 - Parser, 34
 - SAMParser, 42
 - ThreadedMapParser, 63
- targ_lengths
 - BAMParser, 7
 - Parser, 34
 - SAMParser, 42
 - ThreadedMapParser, 63
- Target, 50
 - ~Target, 51
 - add_mass, 51
 - bundle, 52
 - cached_effective_length, 52
 - est_effective_length, 52
 - id, 53
 - incr_counts, 53
 - length, 53
 - lock, 53
 - log_likelihood, 53

- mass, 54
- mass_var, 54
- name, 54
- rho, 55
- round_reset, 55
- seq, 55
- solveable, 55
- Target, 51
- tot_ambig_mass, 55
- tot_counts, 56
- uniq_counts, 56
- unlock, 56
- update_target_bias, 56
- var_sum, 57
- targets
 - Bundle, 13
- TargetTable, 57
 - ~TargetTable, 58
 - covar_size, 58
 - get_covar, 58
 - get_targ, 59
 - merge_bundles, 59
 - num_bundles, 59
 - output_results, 60
 - round_reset, 60
 - size, 60
 - TargetTable, 58
 - threaded_bias_update, 60
 - total_fpb, 60
 - update_covar, 61
 - update_total_fpb, 61
- threaded_bias_update
 - TargetTable, 60
- threaded_parse
 - ThreadedMapParser, 63
- ThreadedMapParser, 61
 - ~ThreadedMapParser, 62
 - reset_reader, 63
 - targ_index, 63
 - targ_lengths, 63
 - threaded_parse, 63
 - ThreadedMapParser, 62
 - write_active, 63
- to_string
 - BiasBoss, 10
 - FLD, 19
 - MismatchTable, 32
 - SeqWeightTable, 49
- tot_ambig_mass
 - RoundParams, 40
 - Target, 55
- tot_counts
 - Target, 56
- tot_mass
 - FLD, 19
- total_fpb
 - TargetTable, 60
- uniq_counts
 - Target, 56
- unlock
 - Target, 56
- update
 - MismatchTable, 32
- update_covar
 - TargetTable, 61
- update_expectations
 - BiasBoss, 11
- update_observed
 - BiasBoss, 11
- update_target_bias
 - Target, 56
- update_total_fpb
 - TargetTable, 61
- var_sum
 - RoundParams, 40
 - Target, 57
- write_active
 - ThreadedMapParser, 63
- write_fragment
 - BAMWriter, 8
 - SAMWriter, 44
 - Writer, 64
- Writer, 64
 - write_fragment, 64